SPECMAGIC: SPECIFICATION-GUIDED MULTI-TASK GENERALIZATION IN MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-task generalization is an important yet underexplored problem in multi-agent reinforcement learning (MARL). Many real-world applications require multi-agent systems to generalize across complex, temporally extended tasks with diverse inter-agent coordination requirements. We introduce a specification language that combines temporal-logic task specifications with operators for expressing such requirements. We propose SpecMAGIC, a hierarchical framework that leverages the automaton structure of task specifications for subtask decomposition. A centralized high-level planner identifies and allocates compatible subtask sequences across agents, while decentralized low-level policies execute individual subtasks. This enables zero-shot generalization to new tasks by composing reusable subtask policies according to new specifications, without retraining. We establish soundness and probabilistic completeness of the high-level planner. Experiments on multi-agent navigation tasks show that SpecMAGIC substantially outperforms existing approaches on unseen task specifications and maintains strong performance as the number of agents increases.

1 INTRODUCTION

Multi-task multi-agent reinforcement learning (MTMARL) aims to learn multi-agent policies that generalize across diverse tasks (Omidshafiei et al., 2017; Chai et al., 2025). Recent methods have explored this setting in applications including multi-robot systems (Wang et al., 2023; Li et al., 2024a; Zhu et al., 2025) and autonomous driving (Berahman et al., 2025; Wu et al., 2026). However, zero-shot generalization to complex, temporally extended tasks with diverse inter-agent coordination requirements, as arise in many real-world applications, remains largely underexplored. To capture this setting, we consider multi-agent systems where each agent is assigned individual tasks, while the team must additionally satisfy coordination requirements such as dependency and count constraints. Moreover, at test time, both the individual tasks and the coordination structure among agents can differ from those encountered during training. For example, warehouse robots that have learned to deliver packages need to follow different user-specified delivery sequences, where some deliveries must precede others and certain deliveries require a specified number of robots.

Temporal logic provides a principled framework for specifying multi-agent behaviors over time. In particular, linear temporal logic (LTL) has been widely used to express complex temporally extended tasks and to guide the learning of policies that satisfy them (León & Belardinelli, 2020; Liu et al., 2025; Yalcinkaya et al., 2025). Recent approaches exploit the structure of automata corresponding to LTL specifications (Vardi & Wolper, 1986) to decompose them into subtasks or subtask sequences (Vaezipoor et al., 2021; Qiu et al., 2023; Jackermeier & Abate, 2025), enabling generalization to unseen specifications by composing previously learned subtask policies according to the structure of a new specification. This paradigm can, in principle, be extended to multi-agent specifications by combining individual tasks and inter-agent coordination requirements into a global specification and applying the same decomposition procedure. However, the automaton corresponding to a global specification can be exponential in the size of the specification, and composing agent-specific requirements can further induce a combinatorial blow-up in the joint state space. Consequently, explicit automaton construction and planning become increasingly intractable as the number of agents and specification complexity grow.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this work, we propose SpecMAGIC, a hierarchical framework for zero-shot **Spec**ification-guided **M**ulti-**A**gent task **G**eneralization with **I**nter-agent **C**oordination requirements. We introduce specification operators for dependency and count constraints that can be translated into standard LTL formulas. allowing individual tasks and coordination requirements to be represented within a unified specification framework. To avoid constructing a global automaton over all agents and coordination requirements, SpecMAGIC decomposes the specifications into *reach-avoid* subtask sequences and plans directly over the resulting subtask space. A centralized sampling-based high-level planner then identifies, allocates and sequences compatible subtasks across agents. The resulting plan is executed by decentralized subtask-conditioned RL policies, compatible with a variety of goal-conditioned RL policies with subtask embeddings (Vaezipoor et al., 2021; Qiu et al., 2023; Jackermeier & Abate, 2025; Guo et al., 2026a). This hierarchical approach enables learned subtask policies to be recombined at test time to satisfy unseen individual tasks and coordination constraints, including both finite- and infinite-horizon specifications. Our main contributions are summarized as follows:

- We study zero-shot generalization to complex multi-agent LTL specifications that combine individual tasks with inter-agent coordination requirements, and introduce new specification operators to conveniently express these coordination constraints.
- We develop a scalable task-allocation approach that avoids the exponential complexity associated with explicit global product-automaton construction, and establish soundness and probabilistic completeness of the proposed planner.
- We conduct comprehensive experiments on diverse multi-agent task specifications with varying numbers of agents in a navigation environment, demonstrating consistent improvements over state-of-the-art baselines in zero-shot task generalization and scalability.

2 Related Work

Specification-driven RL. There has been growing interest in learning RL policies that satisfy tasks expressed through formal specifications. Early work primarily focuses on a single fixed specification (Hasanbeig et al., 2018; Shao & Kwiatkowska, 2023; Shah et al., 2024a). More recently, zero-shot generalization to unseen specifications has been studied through direct specification or automaton encoding (Vaezipoor et al., 2021; Yalcinkaya et al., 2024) and through subtask decomposition (Qiu et al., 2023; Liu et al., 2024a; Jackermeier & Abate, 2025; Guo et al., 2026a). In multi-agent settings, formal specifications have been used to coordinate multiple agents toward a global/system specification through hand-designed (Neary et al., 2020), heuristic (Smith et al., 2023), or learned (Shah et al., 2025) decompositions. Recent work extends formal specifications to MTMARL over a *given* collection of tasks (Hammond et al., 2021; Liu et al., 2025). We instead focus on zero-shot generalization to *unseen* specifications; the closest work, ACC-MARL (Yalcinkaya et al., 2025), also considers unseen specifications but is limited to finite-horizon single-agent tasks, whereas we consider both finite- and infinite-horizon specifications with explicit inter-agent coordination requirements.

Multi-agent task decomposition and allocation. In multi-agent systems with temporal-logic specifications, task decomposition and allocation are widely used to synthesize coordinated plans, ranging from independently assigned local specifications to global tasks with temporal dependencies, collaborative execution, and capability constraints (Guo & Dimarogonas, 2015; Liu et al., 2024b; Li et al., 2025; Luo & Liu, 2025). These methods primarily address planning for a given specification and typically assume known system dynamics. To improve planning efficiency, sampling-based approaches incrementally explore the product-automaton state space rather than exhaustively searching it (Kantaros & Zavlanos, 2018; 2020). Hierarchical methods have also been developed for task decomposition and allocation in MTMARL (Wang et al., 2021; Iqbal et al., 2022; Li et al., 2024b), but do not consider temporal-logic specifications. In contrast, our high-level planner samples directly over reach-avoid subtask sequences derived from the component specifications, avoiding construction of the global product automaton, while decentralized policies learn to execute reusable subtasks under unknown dynamics.

3 Preliminaries

Markov Games. In this work, we consider partially observable Markov games (POMGs) (Littman, 1994). A POMG with N agents is defined as the tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, \gamma, d_0)$, where $\mathcal{S} = \mathcal{S}_1 \times$

$\cdots \times \mathcal{S}_N$ denotes the joint state space, $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$ denotes the joint action space, $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$ denotes the joint observation space. The transition dynamics are given by $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, $\gamma \in [0, 1)$ is the discount factor, and $d_0 \in \Delta(\mathcal{S})$ is the initial state distribution. Each agent i receives its local observation correlated with the state $o_i : \mathcal{S} \mapsto \mathcal{O}_i$ and selects an action according to a policy $\pi_i : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$. The joint policy is denoted by $\pi := (\pi_1, \pi_2, \ldots, \pi_N)$. Unlike standard formulations, we do not explicitly define a reward function as part of the POMG since the reward is induced by the task specification provided at test time.

Linear Temporal Logic. LTL (Pnueli, 1977) provides a formalism for expressing temporal properties over system behaviors. Given a finite set of atomic propositions AP, an LTL formula is recursively constructed using Boolean operators negation $\neg$, conjunction $\wedge$, and disjunction $\vee$, together with the temporal operators "until" (U), "eventually" (F), and "always" (G):

$$\varphi := a \mid \neg\varphi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \mathsf{F}\,\varphi \mid \mathsf{G}\,\varphi \mid \phi\ \mathsf{U}\ \varphi$$

Intuitively, the formula $\varphi_1\ \mathsf{U}\ \varphi_2$ holds if φ_1 is satisfied at all time steps prior to the first occurrence of φ_2. The operator $\mathsf{F}\,\varphi$ holds if φ is satisfied at some future time, while $\mathsf{G}\,\varphi$ holds if φ is satisfied from the current step onward. For each agent i, let AP_i denote its set of atomic propositions, and let $AP = \bigcup_{i=1}^{N} AP_i$ denote the overall proposition set. Each agent has access to a known labeling function $L_i : \mathcal{S}_i \mapsto 2^{AP_i}$ that maps its local state to the set of atomic propositions. We write $L(s) = \bigcup_{i=1}^{N} L_i(s_i)$ for the propositions satisfied at a joint state $s = (s_1, \ldots, s_N)$. For a trajectory $\tau = s_0 s_1 \ldots$, its trace is given by $\mathrm{Tr}(\tau) = L(s_0)L(s_1)\ldots$, and $\tau \models \varphi$ iff this trace satisfies φ. Accordingly, the probability of satisfying φ under policy π is $\Pr(\pi \models \varphi) = \mathbb{E}_{\tau\sim\pi}[\mathbb{1}[\tau \models \varphi]]$. We are interested in learning policies that maximize this satisfaction probability for arbitrary φ.

Büchi automata and subtasks. We represent an LTL specification φ using a (non-deterministic) Büchi automaton (Büchi, 1966) $\mathcal{B}_\varphi = (\mathcal{Q}, \Sigma, \delta, \mathcal{F}, q_0)$, where $\mathcal{Q}$ is a finite set of automaton states, $\Sigma = 2^{AP}$ is the alphabet, $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is the transition relation, $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states, and q_0 is the initial state. For an infinite word $\omega = \alpha_0\alpha_1 \ldots \in \Sigma^\omega$, a run $r = q_0 q_1 \ldots$ satisfies $(q_i, \alpha_i, q_{i+1}) \in \delta$ for every $i \geq 0$ and is accepting iff it visits an accepting state infinitely often. To keep track of progress toward specification satisfaction, *reach-avoid subtasks* (Jackermeier & Abate, 2025), denoted by $\sigma = (\alpha^+, \alpha^-)$, are proposed to represent transitions over the automaton. The reach part $\alpha^+ \in \Sigma$ corresponds to a transition $(q, \alpha^+, q') \in \delta$, where q and q' are consecutive states along an accepting run and $q' \neq q$ when $q \notin \mathcal{F}$. The avoid part $\alpha^- \in \Sigma$ corresponds to a transition $(q, \alpha^-, q') \in \delta$, where q and q' are not consecutive states along any accepting run. Intuitively, the reach part specifies progress toward satisfying the specification, whereas the avoid part captures transitions that can lead to a violation.

4 METHOD

In this section, we present our hierarchical framework, with an overview shown in Figure 1. We first state the generalization problem considered in this work (Section 4.1) and then introduce the different classes of task specifications (Section 4.2). Next, we present the centralized sampling-based high-level planner for task allocation over reach-avoid subtask sequences and establish its soundness and probabilistic completeness (Section 4.3). Finally, we present the decentralized low-level subtask-conditioned policies (Section 4.4).

4.1 PROBLEM STATEMENT

We consider multi-agent tasks in which each agent is assigned an *individual-agent* specification, together with additional *inter-agent dependency* and *count* requirements. A global specification combines all these specification components $\varphi_{\texttt{global}} := \bigwedge_{i\in\mathcal{I}} \varphi_i$, where $I := \{1, \ldots, N\}$ is the agent index set, $\mathcal{I} := I \cup \{\texttt{dep}, \texttt{cnt}\}$, $\varphi_i \sim \Phi_{\texttt{ind}}$ for $i \in I$, $\varphi_{\texttt{dep}} \sim \Phi_{\texttt{dep}}$, and $\varphi_{\texttt{cnt}} \sim \Phi_{\texttt{cnt}}$, where $\Phi_{\texttt{ind}}$, $\Phi_{\texttt{dep}}$, and $\Phi_{\texttt{cnt}}$ denote arbitrary distributions over these three types of specifications. Let $\Phi_{\texttt{global}}$ denote the distribution over such global specifications. Our goal is to learn a multi-agent policy π that maximizes the probability of satisfying specifications drawn from $\Phi_{\texttt{global}}$

$$\pi^\star = \arg\max_{\pi} \mathbb{E}_{\varphi_{\texttt{global}}\sim\Phi_{\texttt{global}},\tau\sim\pi} \left[\mathbb{1}[\tau \models \varphi_{\texttt{global}}]\right].$$

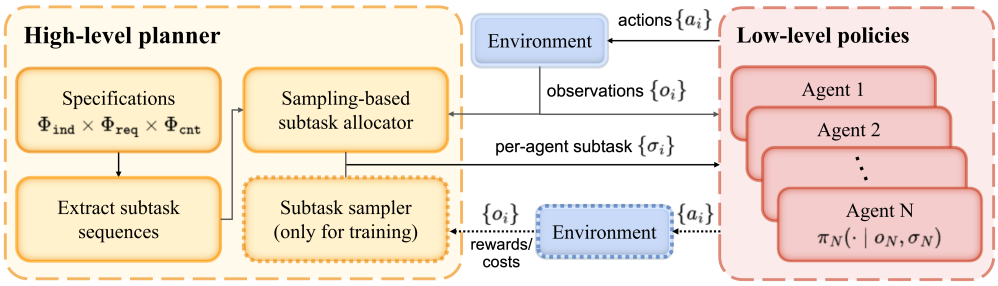


Figure 1: Overview of the proposed hierarchical framework. Specifications are decomposed into reach-avoid subtasks, from which a centralized sampling-based planner constructs a feasible subtask sequence for execution by decentralized low-level policies. Solid and dotted blocks and arrows indicate components used at test time and during training, respectively.

4.2 Specification Language

We now introduce the specification language for the components of $\varphi_{\texttt{global}}$. We use standard LTL to express individual-agent specifications, allowing each φ_i, $i \in I$, to be an arbitrary LTL formula over the agent-specific atomic propositions AP_i. To express inter-agent coordination, we augment these temporal-logic task specifications with operators for dependency and count requirements.

Dependency requirements. To capture dependency relations among agents, we define the operators `Req` and `EachReq`, and specify dependency requirements by the following syntax:

$$\varphi_{\text{dep}} ::= (\alpha \texttt{ Req } \beta) \mid (\alpha \texttt{ EachReq } \beta) \mid \phi_{\text{dep}} \wedge \varphi_{\text{dep}} \mid \phi_{\text{dep}} \vee \varphi_{\text{dep}}$$

where α and β are Boolean formulas over atomic propositions. We say $\tau \models \alpha \texttt{ Req } \beta$ iff there exists a time $t \geq 0$ such that $\tau, t \models \beta$ and $\forall t' \in [0, t)$, $\tau, t' \not\models \alpha$ and $\tau \models \alpha \texttt{ EachReq } \beta$ iff for every $t \geq 0$ such that $\tau, t \models \alpha$ and either $t = 0$ or $\tau, t-1 \not\models \alpha$, there exists $t' \in [0, t)$ such that $\tau, t' \models \beta$ and $\tau, t'' \not\models \alpha$ for all $t'' \in [t', t)$. Intuitively, `Req` requires the prerequisite β before the first occurrence of α, whereas `EachReq` requires β before every non-consecutive occurrence of α. In contrast, the negation of a dependency relation, e.g., $\neg(\alpha \texttt{ Req } \beta)$ or $\neg(\alpha \texttt{ EachReq } \beta)$, does not correspond to a meaningful dependency requirement and is therefore excluded from the syntax.

Count requirements. We also introduce operators `Sync` and `Async` to express count requirements among agents, such as how many agents must satisfy a given condition. The corresponding specifications are defined by the following syntax:

$$\varphi_{\text{cnt}} ::= \texttt{Sync}[I, \alpha, m] \mid \texttt{Async}[I, \alpha, m] \mid \neg\varphi_{\text{cnt}} \mid \phi_{\text{cnt}} \wedge \varphi_{\text{cnt}} \mid \phi_{\text{cnt}} \vee \varphi_{\text{cnt}}$$

where α is a Boolean formulas over atomic propositions, I is the set of agents $I \subseteq [N]$, and $1 \leq m \leq |I|$. Satisfaction of φ by τ is defined as $\tau \models \texttt{Sync}[I, \alpha, m] \Leftrightarrow \exists t : |\{i \in I \mid L_i(s_{i,t}) \models \alpha\}| \geq m$ and $\tau \models \texttt{Async}[I, \alpha, m] \Leftrightarrow |\{i \in I \mid \exists t : L_i(s_{i,t}) \models \alpha\}| \geq m$. Intuitively, `Sync` requires at least m agents to satisfy α at the same (discrete) timestep, and `Async` only requires at least m agents to satisfy α, possibly at different timesteps.

Connections to real-world applications. These operators provide a convenient way to express coordination requirements among agents that naturally arise in a wide range of multi-agent applications. For example, in surveillance and security, a guard robot may need to inspect an area before another robot enters, which can be expressed using `Req` or `EachReq`, while `Async` can specify that a certain number of areas are visited for monitoring coverage. In warehouse automation, `Req` or `EachReq` can specify that inventory availability must be confirmed before items are retrieved, while $\neg$`Async` can prevent a certain number of robots from retrieving out-of-stock items. In multi-robot manufacturing, `Req` or `EachReq` can require an object to be stabilized before manipulation, while $\neg$`Sync` can limit the number of robots operating on the same object simultaneously. These operators also serve as a convenient specification interface and can be translated into standard LTL formulas, allowing standard automata-theoretic constructions and operations for LTL to be directly applied; the corresponding translations are provided in Appendix B.2.

4.3 Task Decomposition and Allocation

To obtain a feasible multi-agent task plan, a straightforward approach is to construct the automaton of the global specification $\varphi_{\texttt{global}}$, or equivalently the product automaton of its component specifications, and search for feasible subtask sequences. However, the resulting state space grows exponentially with the number of agents and specification complexity (Kantaros & Zavlanos, 2018; 2020; Abate et al., 2026), making explicit construction and search intractable at scale. Previous works (Qiu et al., 2023; Jackermeier & Abate, 2025; Guo et al., 2026a) have shown that leveraging equivalent subtask sequences derived from specifications can improve learning efficiency and generalization. Motivated by this, we develop a sampling-based task-allocation method that operates on the reach-avoid subtask sequences of the component specifications, identifies compatible combinations of subtasks across components while respecting their coordination requirements, and incrementally constructs a feasible multi-agent plan without explicitly constructing the product automaton. Under unknown dynamics and partial observability, we focus on finding feasible solutions and establish the soundness and probabilistic completeness of the planner.

Algorithm 1 Centralized sampling-based planner

Require: Subtask-sequence sets $\{\mathcal{G}_i\}_{i\in\mathcal{I}}$ for specifications $\{\varphi_i\}_{i\in\mathcal{I}}$, maximum sampling iterations K

Initialize tree $\mathcal{T}$ with the nominal (empty) root x_{root}
for $k = 0, 1, \ldots, K$ **do**
  Sample $x_{\text{parent}} \sim \text{Unif}(\mathcal{T})$
  $x_{\text{new}} \leftarrow \textsc{Extend}(x_{\text{parent}}, \{\mathcal{G}_i\}_{i\in\mathcal{I}})$
  if x_{new} is INVALID, **then continue**
  if $x_{\text{new}} \notin \mathcal{T}$, **then** add x_{new} with parent x_{parent}
  if x_{new} is terminal, **then return** the root-to-x_{new} subtask sequence
end for
return FAILURE

Algorithm 2 Extend

Require: Node x_{parent} and $\{\mathcal{G}_i\}_{i\in\mathcal{I}}$

Identify expandable components $\mathcal{I}_{\text{exp}}$
Sample a nonempty subset $\mathcal{I}_{\text{s}} \subseteq \mathcal{I}_{\text{exp}}$
$x' \leftarrow x_{\text{parent}}$, $A^+ \leftarrow \{\}$, $A^- \leftarrow \{\}$
for $i \in \mathcal{I}$ **do**
  Determine the current subtask for component i, update A^+ and A^- with its reach and avoid conditions, and update x' accordingly
end for
if x' is valid, **then return** x'
return INVALID

Sampling-based task allocation. The overall process of the proposed method is summarized in Algorithm 1. For each φ_i, $i \in \mathcal{I}$, let $\mathcal{G}_i = \{g_i^1, g_i^2, \ldots, g_i^{M_i}\}$ denote the set of candidate subtask sequences, where $M_i = |\mathcal{G}_i|$. Each $g_i^j \in \mathcal{G}_i$ is a finite sequence $g_i^j = [\sigma_{i,0}^j, \sigma_{i,1}^j, \ldots]$, where $\sigma_{i,k}^j = (A_{i,k}^{+,j}, A_{i,k}^{-,j})$ denotes a reach-avoid subtask. The planner constructs a search tree $\mathcal{T}$ over the subtask space, where each node x represents the progress of all specification components as a set of index tuples (i, j, k), with exactly one tuple for each $i \in \mathcal{I}$. For each tuple, i identifies the specification component, j identifies the selected sequence g_i^j, and k identifies the current subtask $\sigma_{i,k}^j$ in that sequence, with $k = |g_i^j|$ indicating that the planner has reached the end of the sequence. We use $(i, 0, 0)$ when no sequence has yet been selected for component i, such that the root node is $x_{\text{root}} = \{(i, 0, 0) \mid i \in \mathcal{I}\}$. At each iteration, a node x_{parent} is uniformly sampled from $\mathcal{T}$ and extended using Algorithm 2 to obtain a successor x_{new} with compatible next subtasks. If such a successor is found, it is added to $\mathcal{T}$. A node is terminal when, for every tuple (i, j, k), a sequence has been selected and the planner has reached its end, i.e., $j \neq 0$ and $k = |g_i^j|$ for all $i \in \mathcal{I}$. The path from x_{root} to a terminal node then yields a subtask sequence satisfying all specifications. More generally, our method also applies when the specifications are provided as a collection of Büchi automata, as the planner only requires the corresponding reach-avoid subtask sequences.

Subtask combination compatibility. Algorithm 2 identifies a compatible combination of next subtasks for a sampled node. Given x_{parent}, we define the expandable components as $\mathcal{I}_{\text{exp}} := \{i \in \mathcal{I} \mid j = 0 \text{ or } k < |g_i^j| \text{ for the tuple } (i, j, k) \in x_{\text{parent}}\}$, i.e., components for which no sequence has yet been selected, or the planner has not yet reached the end of the selected sequence. We then sample a nonempty subset $\mathcal{I}_{\text{s}} \subseteq \mathcal{I}_{\text{exp}}$ of components to consider at the current planning step, since enumerating all possible subsets can be intractable. For each $i \in \mathcal{I}_{\text{s}}$, if its tuple is $(i, 0, 0)$, we sample a sequence $g_i^j \in \mathcal{G}_i$ and select its first subtask; otherwise, for tuple (i, j, k), we select the next subtask in g_i^j. The corresponding tuples are updated to record this planning progress, while the tuples for $i \notin \mathcal{I}_{\text{s}}$ remain unchanged. To check compatibility, for each $i \in \mathcal{I}_{\text{s}}$, one conjunctive clause $\tilde{A}_i^+$ is sampled from the reach condition of its selected subtask, while $\tilde{A}_i^+ = \emptyset$ for $i \in \mathcal{I} \setminus \mathcal{I}_{\text{s}}$. The

joint reach condition is then $A^+_{\text{joint}} := \bigcup_{i \in \mathcal{I}} \tilde{A}^+_i$. In contrast, every component contributes its full avoid condition A^-_i, yielding $A^-_{\text{joint}} := \bigcup_{i \in \mathcal{I}} A^-_i$, since omitting an unselected component's avoid condition could allow it to be satisfied during execution, after which the corresponding specification can no longer be satisfied. The sampled successor is invalid if there exists a conjunctive clause $B \in A^-_{\text{joint}}$ such that $B \subseteq A^+_{\text{joint}}$. More details of the planner can be found in Appendix C.

Soundness and (probabilistic) completeness. Many automata-based search methods (Guo et al., 2013; Guo & Dimarogonas, 2015; Qiu et al., 2023; Shah et al., 2024b; Jackermeier & Abate, 2025) have been developed to find accepting paths that satisfy a given LTL specification. These methods search for a finite prefix reaching an accepting state together with a cyclic suffix, and the resulting prefix-suffix paths, i.e., subtask sequences, are guaranteed to satisfy the given specification. We leverage these methods to obtain candidate subtask sequences for each specification component over which our planner operates. Given this property, we have the following soundness guarantee:

Theorem 1 (Soundness)**.** *Let τ be a path returned by the proposed sampling-based planner. Then τ is an accepting path satisfying the global specification $\varphi_{\texttt{global}}$, i.e., $\tau \models \varphi_{\texttt{global}}$.*

Soundness guarantees that any path returned by the proposed method satisfies the global specification. We next show probabilistic completeness, i.e., if a feasible path exists, the probability of finding one converges to one given sufficient iterations. The proofs are provided in Appendix A.

Theorem 2 (Probabilistic completeness)**.** *Suppose that there exists a feasible subtask sequence that satisfies $\varphi_{\texttt{global}}$. Then, the probability that Algorithm 1 finds a feasible subtask sequence converges to one as the number of sampling iterations approaches infinity.*

4.4 Policy Learning

The goal is to learn a decentralized goal-conditioned policy $\pi_i(a_i \mid o_i, \sigma_i)$ for each agent i, which takes as input its local observation $o_i \in \mathcal{O}_i$ and current reach-avoid subtask σ_i. We assume that all agents are homogeneous. The challenge is that the same set of local subtasks can correspond to different coordination requirements under different specifications. For example, two agents may have the same local subtasks of reaching their respective target regions, while one specification allows them to complete these subtasks independently and another requires them to do so simultaneously. Thus, the global coordination requirement cannot be determined from the local subtasks alone. Consequently, commonly adopted cooperative MARL schemes based on shared team rewards and centralized critics (Sunehag et al., 2017; Foerster et al., 2018; Rashid et al., 2020; Yu et al., 2022) are not well suited to our setting. Instead, we use a subtask-dependent reward and cost function evaluated with respect to each agent's own subtask, and follow an IPPO-style approach (De Witt et al., 2020) to learn decentralized policies and critics.

During training, the high-level planner described in Section 4.3 is not required. We independently sample a reach-avoid subtask for each agent and train the low-level policies to complete their respective subtasks. Although the subtasks are sampled independently, agents still interact and must account for one another, allowing the policies to learn interaction-aware behaviors; for example, multiple agents may sample subtasks that require reaching the same region and must avoid collisions with other agents. Each agent learns to satisfy its subtask by receiving positive rewards for subtask satisfaction while being penalized for collisions and subtask violations. High-level coordination requirements, such as synchronized or ordered subtask execution, are instead handled at test time by the centralized planner, which controls each agent's current subtask and its progression. The learned subtask-conditioned policies are then reused to execute these planned subtasks across different specifications, enabling zero-shot generalization to unseen specifications without retraining. We instantiate the low-level policies by adapting GenZ-LTL (Guo et al., 2026a) because it explicitly models safety constraints, which are common in our specifications, while our framework is also compatible with other goal-conditioned RL methods capable of learning reach-avoid policies (Qiu et al., 2023; Yalcinkaya et al., 2024). Further implementation details are provided in Appendix C.

5 Experiments

In this section, we evaluate our method, SpecMAGIC, in a navigation environment across a wide range of multi-agent task specifications to answer the following questions: **Q1:** Can SpecMAGIC generalize zero-shot to unseen task specifications? **Q2:** How well does SpecMAGIC scale as the

Table 1: Evaluation results of finite-horizon tasks: success rate η_s, violation rate η_v, and average steps μ to satisfy the complex specifications listed in Table 3. ↑: higher is better; ↓: lower is better. **Bold**: the best performance for each metric. Each value is reported as mean ± standard deviation over 5 seeds, with 100 trajectories evaluated per seed.

	Specifications	ACC-MARL			Ours		
		$\eta_s \uparrow$	$\eta_v \downarrow$	$\mu \downarrow$	$\eta_s \uparrow$	$\eta_v \downarrow$	$\mu \downarrow$
IND	$\Phi_{\texttt{ind}}$	$0.21_{\pm 0.09}$	$0.66_{\pm 0.08}$	$510.29_{\pm 105.01}$	$\mathbf{0.95}_{\pm 0.02}$	$\mathbf{0.04}_{\pm 0.02}$	$\mathbf{290.33}_{\pm 36.64}$
	$\Phi_{\texttt{ind/dep}}$				$\mathbf{0.94}_{\pm 0.04}$	$\mathbf{0.05}_{\pm 0.03}$	$\mathbf{379.73}_{\pm 60.20}$
	$\Phi_{\texttt{ind/cnt}}$		N/A		$\mathbf{0.95}_{\pm 0.02}$	$\mathbf{0.05}_{\pm 0.02}$	$\mathbf{380.11}_{\pm 49.64}$
	$\Phi_{\texttt{ind/dep/cnt}}$				$\mathbf{0.91}_{\pm 0.04}$	$\mathbf{0.07}_{\pm 0.04}$	$\mathbf{460.72}_{\pm 75.88}$
	Average	$0.21_{\pm 0.09}$	$0.66_{\pm 0.08}$	$510.29_{\pm 105.01}$	$\mathbf{0.94}_{\pm 0.03}$	$\mathbf{0.05}_{\pm 0.03}$	$\mathbf{377.72}_{\pm 82.62}$
OOD	$\Phi_{\texttt{ind}}$	$0.04_{\pm 0.02}$	$0.74_{\pm 0.09}$	$832.44_{\pm 23.00}$	$\mathbf{0.96}_{\pm 0.02}$	$\mathbf{0.04}_{\pm 0.02}$	$\mathbf{379.79}_{\pm 33.89}$
	$\Phi_{\texttt{ind/dep}}$				$\mathbf{0.84}_{\pm 0.04}$	$\mathbf{0.10}_{\pm 0.04}$	$\mathbf{633.58}_{\pm 47.89}$
	$\Phi_{\texttt{ind/cnt}}$		N/A		$\mathbf{0.94}_{\pm 0.04}$	$\mathbf{0.06}_{\pm 0.04}$	$\mathbf{433.42}_{\pm 37.13}$
	$\Phi_{\texttt{ind/dep/cnt}}$				$\mathbf{0.83}_{\pm 0.05}$	$\mathbf{0.08}_{\pm 0.04}$	$\mathbf{659.28}_{\pm 43.10}$
	Average	$0.04_{\pm 0.02}$	$0.74_{\pm 0.09}$	$832.44_{\pm 23.00}$	$\mathbf{0.89}_{\pm 0.07}$	$\mathbf{0.07}_{\pm 0.04}$	$\mathbf{526.52}_{\pm 129.06}$

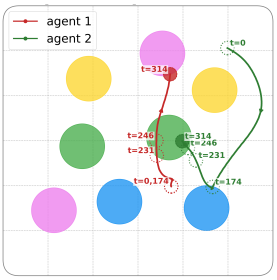


Figure 2: Example trajectories of 2 agents in the `Zone` environment: $\varphi_1 : \neg \mathsf{b}\ \mathsf{U}\ \mathsf{m}, \varphi_2 : \neg \mathsf{y}\ \mathsf{U}\ \mathsf{b}$, $\varphi_{\texttt{dep}} : \mathsf{m}_1\ \texttt{Req}\ \mathsf{b}_2$, $\varphi_{\texttt{cnt}} : \mathsf{Sync}[\{1, 2\}, \mathsf{g}, 2]$.

number of agents increases? **Q3:** What is the impact of the proposed sampling-based planner on SpecMAGIC versus a reference product-automaton-based planner?

Environments. We consider a multi-agent `Zone` environment based on prior work (Jackermeier & Abate, 2025; Guo et al., 2026a;b), where multiple agents with continuous action spaces and high-dimensional lidar observations navigate among randomly positioned colored regions to satisfy the given task specifications. The `Zone` environment is similar to the commonly used Multi-Agent Particle Environment (MPE) for MARL (Lowe et al., 2017), but involves complex long-horizon tasks, including reaching sequences of regions with varying lengths, satisfying diverse coordination constraints among agents, and avoiding inter-agent collisions at all times. An illustration of the environment with an example trajectory is shown in Figure 2; more details are provided in Appendix B.1.

Baselines. We compare our method with ACC-MARL (Yalcinkaya et al., 2025), the only existing method that, to our knowledge, targets generalization across temporal-logic specifications in MARL. ACC-MARL first learns embeddings of compositions of deterministic finite automata (DFAs). It then learns decentralized policies, with each agent's policy conditioned on the global state information together with the DFA embeddings of the specifications assigned to all agents. Further details of the implementation are provided in Appendix C.

LTL specifications. For individual tasks $\Phi_{\texttt{ind}}$, we consider reach-avoid specifications of varying complexity, constructed from nested properties of the form $\neg\phi\ \mathsf{U}\ \varphi$ and combinations with F, $\wedge$, and $\vee$. For multi-agent tasks, we consider specifications that capture both independent and joint behaviors, including inter-agent dependency requirements $\Phi_{\texttt{dep}}$ and count requirements $\Phi_{\texttt{cnt}}$, both of which are constructed to involve all agents in the team rather than a subset of agents. We then evaluate four types of specifications: $\Phi_{\texttt{ind}}$, $\Phi_{\texttt{ind/dep}}$, $\Phi_{\texttt{ind/cnt}}$, and $\Phi_{\texttt{ind/dep/cnt}}$. The specifications used for evaluation are listed in Table 3.

Evaluation metrics. For each LTL specification, we measure the success rate η_s and violation rate η_v, corresponding to the fractions of trajectories that satisfy and violate the specification, respectively. The remaining trajectories are captured by $\eta_o = 1 - (\eta_s + \eta_v)$. We also report the average number of steps μ required for successful trajectories to satisfy the specification, treating it as a secondary metric since efficiency is less informative when few trajectories succeed.

5.1 Can SpecMAGIC generalize zero-shot to unseen task specifications (Q1)?

Finite-horizon tasks. We first evaluate finite-horizon tasks, i.e., specifications that can be satisfied by finite-length trajectories. We consider 3 agents, each assigned an individual task, together with additional inter-agent dependency $\Phi_{\texttt{dep}}$ and count requirements $\Phi_{\texttt{cnt}}$. We evaluate both in-distribution (IND) and out-of-distribution (OOD) specifications, following the specification distribution used to train ACC-MARL. ACC-MARL is trained on specifications represented as DFAs and sampled from a fixed distribution; IND specifications are drawn from this distribution, whereas OOD specifications fall outside it and require longer subtask sequences to satisfy. The same IND/OOD specification sets used to evaluate ACC-MARL are used to evaluate SpecMAGIC.

As shown in Table 1, our method consistently outperforms the baseline across all evaluated specifications, achieving higher success rates, lower violation rates, and better efficiency, as indicated by fewer steps required for satisfaction. ACC-MARL supports only $\Phi_{\texttt{ind}}$, where each agent has an individual task and access to the task automata of the other agents. Extending it to $\Phi_{\texttt{dep}}$ and $\Phi_{\texttt{cnt}}$ would require additional training distributions over these specifications, potentially biasing zero-shot evaluation. ACC-MARL also assumes access to agent state information s_i, whereas agents in our environment receive only partial observations o_i through LiDAR sensing. Moreover, ACC-MARL pre-trains automaton embeddings using automata sampled from a fixed distribution, which limits its generalization when encountering unseen automata. In contrast, our method decomposes specifications into reach-avoid subtasks and learns reusable subtask-conditioned policies, enabling unseen specifications to be handled by composing previously learned subtask behaviors and thereby improving OOD generalization. Finally, performance generally decreases with increasing specification complexity, such as from $\Phi_{\texttt{ind}}$ to $\Phi_{\texttt{ind/dep/cnt}}$. Inter-agent dependency and count requirements introduce additional coordination constraints, while OOD specifications further increase the difficulty by requiring each agent to complete more subtasks and imposing more constraints in $\Phi_{\texttt{dep}}$ and $\Phi_{\texttt{cnt}}$. These factors make successful coordination more challenging and can increase the likelihood of inter-agent collisions in practice. Nevertheless, our method maintains high success rates and low violation rates across these more complex and unseen specifications.

Infinite-horizon tasks. Our method can also handle infinite-horizon specifications by repeatedly applying the same planning and execution procedure. ACC-MARL is excluded from this evaluation since it relies on DFAs, which cannot represent such specifications. We therefore report only the performance of our method, with the results provided in Appendix D as a reference for future comparisons when suitable baselines become available.

5.2 How well does SpecMAGIC scale as the number of agents increases? (Q2)

We evaluate scalability with respect to the number of agents using specifications from $\Phi_{\texttt{ind/dep/cnt}}$, with $N \in \{3, 5, 10\}$. Note that the complexity of $\Phi_{\texttt{dep}}$ and $\Phi_{\texttt{cnt}}$ increases with N, since these specifications impose coordination requirements involving an increasing number of agents: more `Req` operators are introduced in $\Phi_{\texttt{dep}}$ to ensure that each agent has at least one dependency with another agent, while the agent index sets of the count operators grow accordingly to include all agents (see Appendix B.2 for details). We first evaluate the scalability of the high-level planner separately and then the overall framework.

Evaluation of the high-level planner. We measure specification complexity by the number of states in the automaton corresponding to the global specification $\varphi_{\texttt{global}}$; this automaton is used only as a reference measure and is not required by our planner. For the proposed sampling-based planner, we report the number of sampling iterations required to find a feasible path and the resulting tree size $|\mathcal{T}|$, which counts all explored nodes, including those not on the final feasible path. We also report the average planning time $t_{\texttt{path}}$, excluding automaton-construction time; $t_{\texttt{path}}$ includes both finding feasible paths for the component specifications and the subsequent sampling process for the final multi-agent subtask allocation. As shown in Table 2, the automaton size grows rapidly with the number of agents and becomes intractable to construct for 10 agents (see Appendix C for details), while the sampling-based planner maintains a relatively small tree, with only a moderate increase in $|\mathcal{T}|$ and sampling iterations.

Table 2: High-level planner scalability. We report the product-automaton size $|\mathcal{B}_{\varphi_{\texttt{global}}}|$, the constructed tree size $|\mathcal{T}|$, sampling iterations, and planning time $t_{\texttt{path}}$, averaged over 10 seeds.

N	$\lvert\mathcal{B}_{\varphi_{\texttt{global}}}\rvert$	$\lvert\mathcal{T}\rvert$	Iterations	$t_{\texttt{path}}$
3	$98.7_{\pm 42.5}$	$7.9_{\pm 3.3}$	$64.7_{\pm 100.5}$	$0.02_{\pm 0.02}$
5	$1789.3_{\pm 1267.3}$	$10.4_{\pm 5.2}$	$320.4_{\pm 511.7}$	$0.17_{\pm 0.24}$
10	> 118814.3	$12.6_{\pm 5.2}$	$477.4_{\pm 530.2}$	$8.64_{\pm 0.77}$

Evaluation of the overall framework. We next evaluate the overall framework by training policies with $N_{\texttt{train}} \in \{3, 5, 10\}$ agents and evaluating each policy with $N_{\texttt{eval}} \in \{3, 5, 10\}$ agents without retraining. We report the success rate η_s, violation rate η_v, and average number of steps μ to satisfy the specifications, with results shown in Figure 3. When $N_{\texttt{eval}} = N_{\texttt{train}}$, performance slightly decreases as the number of agents increases, as indicated by lower success rates and higher violation rates, due to more frequent inter-agent interactions and the increasing complexity of $\Phi_{\texttt{dep}}$ and $\Phi_{\texttt{cnt}}$. When $N_{\texttt{eval}} > N_{\texttt{train}}$, the performance drop becomes more noticeable, as the larger number of agents makes the environment more congested and shifts the observation distribution, making safety constraints more difficult to satisfy. For a fixed $N_{\texttt{eval}}$, policies trained with more agents tend to perform better; for example, when $N_{\texttt{eval}} = 10$, the policy trained with $N_{\texttt{train}} = 5$

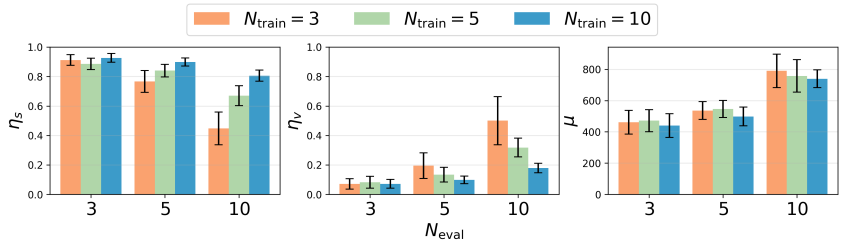


Figure 3: Performance under different numbers of agents during training and evaluation. We report the average success rate η_s, violation rate η_v, and number of steps μ to specification satisfaction for $N_{\texttt{train}}, N_{\texttt{eval}} \in \{3, 5, 10\}$. Error bars indicate one standard deviation.

outperforms that trained with $N_{\texttt{train}} = 3$. In contrast, when $N_{\texttt{eval}} < N_{\texttt{train}}$, the policies maintain or slightly improve their performance, as training with more agents exposes the policies to more complex inter-agent interactions and leads to more robust execution when fewer agents are present. Overall, these results show that our framework can be directly applied to different numbers of agents without retraining, while improving the robustness of the low-level policies could further enhance performance when scaling to larger multi-agent systems.

5.3 Ablation Study: What is the impact of the proposed sampling-based planner on SpecMAGIC versus a reference search-based planner? (Q3)

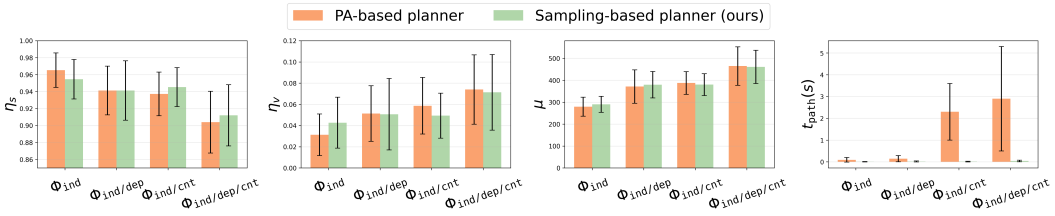


Figure 4: Average success rate η_s, violation rate η_v, and average number of steps μ to satisfy the IND specifications $\Phi_{\texttt{ind}}$, $\Phi_{\texttt{ind/dep}}$, $\Phi_{\texttt{ind/cnt}}$, and $\Phi_{\texttt{ind/dep/cnt}}$ listed in Table 3.

We compare the proposed sampling-based planner with a product-automaton (PA)-based planner, which explicitly constructs the product automaton of the component specifications as discussed in Section 4.3 and performs graph search to find a feasible subtask sequence. The resulting subtasks are executed sequentially using the same trained low-level policies as our sampling-based planner. We evaluate both approaches on the 3-agent IND specifications in Table 3, with results shown in Figure 4. We report the average planning time $t_{\texttt{path}}$ as defined in Section 5.2, excluding automaton-construction time for both methods. Our sampling-based method achieves similar success rates, violation rates, and average steps to specification satisfaction as the search-based planner across all evaluated specifications, empirically validating the proposed sampling technique. Meanwhile, our method requires substantially less planning time than the search-based planner, with the gap becoming more pronounced as specification complexity increases. For specifications involving four or more agents evaluated in Section 5.2, the search-based planner fails to find a feasible path within a 10-minute time limit; thus, these results are excluded from the comparison.

6 Conclusion and Future Work

This work develops a hierarchical framework for zero-shot generalization across diverse LTL specifications in multi-agent systems. The framework consists of a novel centralized sampling-based high-level planner with soundness and probabilistic completeness guarantees for planning over the reach-avoid subtask space, and decentralized subtask-conditioned policies for low-level execution. Empirical results demonstrate that our method achieves strong generalization across diverse finite- and infinite-horizon specifications and remains effective as the number of agents increases. A limitation of our framework is that the high-level planner remains centralized, which may limit scalability as the number of agents further increases. Future work will investigate fully decentralized planning mechanisms to further improve scalability and integrate alternative safe RL methods (Kushwaha et al., 2025) into the low-level policies that may better handle safety constraints.

AI USE STATEMENT

In this work, we used generative AI tools for grammar checking and assistance with parts of the code. All AI-assisted code was reviewed, verified, and tested for correctness by two authors. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not introduce specific ethical concerns beyond those generally associated with autonomous multi-agent systems. However, care should be taken when designing task specifications to ensure that they do not induce unsafe or undesirable behaviors.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide the source code and implementation details necessary to reproduce the experiments and results reported in this work in the supplementary materials.

REFERENCES

Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Kretínskỳ, Maximilian Prokop, and Christoph Weinhuber. Semantically labelled automata for multi-task reinforcement learning with ltl instructions. *arXiv preprint arXiv:2602.06746*, 2026.

Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*, volume 735. MIT press Cambridge, 2008.

Mehran Berahman, Athanasia Karalakou, Majid Rostami-Shahrbabaki, and Klaus Bogenberger. Multi-task lane-free driving strategy for connected and automated vehicles: A multi-agent deep reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 154:110797, 2025.

J Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Studies in Logic and the Foundations of Mathematics*, volume 44, pp. 1–11. Elsevier, 1966.

Jiajun Chai, Zijie Zhao, Yuanheng Zhu, and Dongbin Zhao. A survey of cooperative multi-agent reinforcement learning for multi-task scenarios. *Artificial Intelligence Science and Engineering*, 1(2):98–121, 2025.

Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Meng Guo and Dimos V Dimarogonas. Multi-agent plan reconfiguration under local ltl specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

Meng Guo, Karl H Johansson, and Dimos V Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *2013 IEEE international conference on robotics and automation*, pp. 5025–5032. IEEE, 2013.

Zijian Guo, İlker Işık, HM Ahmad, and Wenchao Li. One subgoal at a time: Zero-shot generalization to arbitrary linear temporal logic requirements in multi-task reinforcement learning. *Advances in Neural Information Processing Systems*, 38:77500–77529, 2026a.

Zijian Guo, İlker Işık, HM Ahmad, and Wenchao Li. Specrlbench: A benchmark for generalization in specification-guided reinforcement learning. *arXiv preprint arXiv:2604.24729*, 2026b.

Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael Wooldridge. Multi-agent reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:2102.00582*, 2021.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

Shariq Iqbal, Robby Costales, and Fei Sha. Alma: Hierarchical learning for composite multi-agent tasks. *Advances in neural information processing systems*, 35:7155–7166, 2022.

Mathias Jackermeier and Alessandro Abate. Deepltl: Learning to efficiently satisfy complex ltl specifications for multi-task rl. In *International Conference on Learning Representations*, volume 2025, pp. 14000–14028, 2025.

Yiannis Kantaros and Michael M Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control*, 64(5):1916–1931, 2018.

Yiannis Kantaros and Michael M Zavlanos. Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research*, 39(7):812–836, 2020.

Ankita Kushwaha, Kiran Ravish, Preeti Lamba, and Pawan Kumar. A survey of safe reinforcement learning and constrained mdps: A technical survey on single-agent and multi-agent safety. *arXiv preprint arXiv:2505.17342*, 2025.

Borja G León and Francesco Belardinelli. Extended markov games to learn multiple tasks in multi-agent reinforcement learning. In *ECAI 2020: 24th European Conference on Artificial Intelligence, 29 August–8 September 2020, Santiago de Compostela, Spain–Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, pp. 139–146. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2020.

Chao Li, Shaokang Dong, Shangdong Yang, Yujing Hu, Tianyu Ding, Wenbin Li, and Yang Gao. Multi-task multi-agent reinforcement learning with interaction and task representations. *IEEE transactions on neural networks and learning systems*, 36(7):13431–13445, 2024a.

Lin Li, Ziyang Chen, Hao Wang, and Zhen Kan. Task allocation of heterogeneous robots under temporal logic specifications with inter-task constraints and variable capabilities. *IEEE Transactions on Automation Science and Engineering*, 22:14030–14047, 2025.

Xianglong Li, Yuan Li, Jieyuan Zhang, Xinhai Xu, and Donghong Liu. A hierarchical multi-agent allocation-action learning framework for multi-subtask games. *Complex & Intelligent Systems*, 10(2):1985–1995, 2024b.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Chanjuan Liu, Jinmiao Cong, Bingcai Chen, Yaochu Jin, and Enqiang Zhu. Guiding multiagent multitask reinforcement learning by a hierarchical framework with logical reward shaping. *IEEE Transactions on Cybernetics*, 2025.

Jason Xinyu Liu, Ankit Shah, Eric Rosen, Mingxi Jia, George Konidaris, and Stefanie Tellex. Skill Transfer for Temporal Task Specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2535–2541, 2024a. doi: 10.1109/ICRA57147.2024.10611432.

Zesen Liu, Meng Guo, Weimin Bao, and Zhongkui Li. Fast and adaptive multi-agent planning under collaborative temporal logic tasks via poset products. *Research*, 7:0337, 2024b.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Xusheng Luo and Changliu Liu. Simultaneous task allocation and planning for multi-robots under hierarchical temporal logic specifications. *IEEE Transactions on Robotics*, 2025.

Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. Reward machines for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2007.01962*, 2020.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International conference on machine learning*, pp. 2681–2690. PMLR, 2017.

Amir Pnueli. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pp. 46–57. ieee, 1977.

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. *Advances in Neural Information Processing Systems*, 36:39147–39175, 2023.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

Kenneth H Rosen and Kamala Krithivasan. *Discrete mathematics and its applications*, volume 6. McGraw-hill New York, 1999.

Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A. Seshia. LTL-Constrained Policy Optimization with Cycle Experience Replay. In *arXiv*, 2024a. doi: 10.48550/arXiv.2404.11578.

Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A Seshia. Ltl-constrained policy optimization with cycle experience replay. *arXiv preprint arXiv:2404.11578*, 2024b.

Ameesh Shah, Niklas Lauffer, Thomas Chen, Nikhil Pitta, and Sanjit A Seshia. Learning symbolic task decompositions for multi-agent teams. *arXiv preprint arXiv:2502.13376*, 2025.

Daqian Shao and Marta Kwiatkowska. Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 4180–4189, 2023. doi: 10.24963/IJCAI.2023/465.

Sophia Smith, Cyrus Neary, and Ufuk Topcu. Automatic decomposition of reward machines for decentralized multiagent reinforcement learning. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 5423–5430. IEEE, 2023.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pp. 10497–10508. PMLR, 2021.

Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.

Jiawei Wang, Jian Zhao, Zhengtao Cao, Ruili Feng, Rongjun Qin, and Yang Yu. Multi-task multi-agent shared layers are universal cognition of multi-agent coordination. *arXiv preprint arXiv:2312.15674*, 2023.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. {RODE}: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TTUVg6vkNjK.

Sifan Wu, Xuting Duan, Jianshan Zhou, Kaige Qu, Ivan Wang-Hei Ho, and Daxin Tian. Risk-aware multi-task hybrid-attention multi-agent reinforcement learning for decision-making at unsignalized intersections. *IEEE Transactions on Vehicular Technology*, 2026.

Beyazit Yalcinkaya, Niklas Lauffer, Marcell Vazquez-Chanlatte, and Sanjit Seshia. Compositional automata embeddings for goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 37:72933–72963, 2024.

Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, Ameesh Shah, Hanna Krasowski, and Sanjit A Seshia. Automata-conditioned cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2511.02304*, 2025.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

Guobin Zhu, Rui Zhou, Wenkang Ji, Hongyin Zhang, Donglin Wang, and Shiyu Zhao. Multi-task multi-agent reinforcement learning via skill graphs. *IEEE Robotics and Automation Letters*, 10 (9):8650–8657, 2025.

A Theoretical Analysis

Soundness and (probabilistic) completeness. We first establish soundness, showing that any path returned by the proposed planner satisfies the global specification.

Theorem 1 (Soundness)**.** *Let τ be a path returned by the proposed sampling-based planner. Then τ is an accepting path satisfying the global specification $\varphi_{\texttt{global}}$, i.e., $\tau \models \varphi_{\texttt{global}}$.*

Proof. Let $\tau = [\sigma_0, \sigma_1, \ldots]$ be a path returned by Algorithm 1, constructed from the selected feasible subtask sequences $g_i^{j_i} \in \mathcal{G}_i$, $i \in \mathcal{I}$. By construction, the subtasks σ in τ preserve their original order in each $g_i^{j_i}$, and each subtask in τ is compatible with its subsequent subtask. Therefore, $\tau \models \varphi_i$ for all $i \in \mathcal{I}$, and consequently, $\tau \models \bigwedge_{i\in\mathcal{I}} \varphi_i = \varphi_{\texttt{global}}$. □

Having established that every returned path satisfies the global specification, we next consider whether the planner can find such a path whenever one exists. Specifically, we show probabilistic completeness: if a feasible subtask sequence exists, the probability of finding one converges to one as the number of sampling iterations increases.

Theorem 2 (Probabilistic completeness)**.** *Suppose that there exists a feasible subtask sequence that satisfies $\varphi_{\texttt{global}}$. Then, the probability that Algorithm 1 finds a feasible subtask sequence converges to one as the number of sampling iterations approaches infinity.*

Proof. Let M denote the maximum number of distinct tree nodes and C the total number of possible subtask combinations across all specifications. Since $\{\mathcal{G}_i\}_{i\in\mathcal{I}}$ is finite, both M and C are finite constants determined by the specifications. Suppose that a node along a feasible path has been added to the tree. By Algorithm 1, at the current iteration, this node is sampled with probability at least $1/M$, and a particular next subtask combination is sampled with probability $1/C$. If this combination is compatible, the corresponding successor node is added to the tree. Hence, the probability of making the corresponding successful extension in one iteration is at least $p = 1/(MC) > 0$. Let m denote the number of successful extensions required to reach a terminal node, and let X_K denote the number of successes after K iterations. Then $\Pr[X_K < m] = \sum_{j=0}^{m-1} \binom{K}{j} p^j (1-p)^{K-j} \leq \sum_{j=0}^{m-1} \binom{K}{m-1} p^j (1-p)^{K-j} \leq \binom{K}{m-1} \sum_{j=0}^{m-1} (1-p)^K \leq \binom{K}{m-1} m e^{-pK} \leq \frac{m}{(m-1)!} K^m e^{-pK} \to 0$ as $K \to \infty$, where the transitions rely on (1) $m \ll K$, (2) $p < \frac{1}{2}$, and (3) $(1-p) \leq e^{-p}$. Therefore, the probability that Algorithm 1 fails to reach a terminal node converges to zero, and hence Algorithm 1 is probabilistically complete. □

B Experimental Settings

B.1 Environments

We extend the `Zone` environment used in prior work (Jackermeier & Abate, 2025; Guo et al., 2026a;b) to a multi-agent setting. The environment contains colored regions associated with the base

atomic propositions blue, green, magenta, yellow. Each colored region appears twice in the environment. For the 10-agent setting, we use a slightly larger environment and increase the radius of the colored regions to accommodate the larger number of agents. For agent i, the corresponding atomic propositions are indexed as $AP_i = \{\mathsf{blue}_i, \mathsf{green}_i, \mathsf{magenta}_i, \mathsf{yellow}_i\}$, representing agent i reaching the corresponding colored regions, and the overall proposition set is given by $AP = \bigcup_i AP_i$. Each agent is modeled as a `point` robot with continuous acceleration and steering actions. We modify the observation space to include LiDAR measurements of neighboring agents in addition to the colored regions and ego-centric state information from onboard sensors, enabling interactions among multiple agents. The initial positions of all agents and colored regions are randomly sampled at the beginning of each episode. We construct environments with $N \in \{3, 5, 10\}$ agents to evaluate scalability with respect to the number of agents. The maximum episode length is $T = 2000$. Illustrations of the resulting environments with 3, 5, and 10 agents are shown in Figure 5.



Figure 5: Visualizations of the `Zone` environments with $N = 3$, 5, and 10 agents, from left to right. The positions of agents and zones are randomly sampled each episode.

B.2 Specifications

Finite- and infinite-horizon tasks. We evaluate the methods on specifications with different types and levels of complexity, as summarized in Table 3. While many more specifications can be constructed, exhaustive evaluation is intractable since the space of possible specifications is infinite. We therefore select a representative set of specifications covering different types and levels of complexity. For each agent, we first define a set of single-agent `sa` specifications that describe its individual task. We then define inter-agent requirement `dep` specifications to impose temporal dependencies between agents and count `cnt` specifications to impose synchronous or asynchronous constraints on the number of agents satisfying a proposition. Based on these specifications, we construct four classes of multi-agent tasks: Φ_{indep} contains only independent single-agent specifications, $\Phi_{\text{indep/dep}}$ additionally includes requirement constraints, $\Phi_{\text{indep/cnt}}$ includes count constraints, and $\Phi_{\text{indep/dep/cnt}}$ combines all three types. In comparison, the OOD specifications are structurally more complex, involving longer temporal dependencies and more compositional constraints than those encountered during training.

Scalability experiments. For the scalability experiments with different numbers of agents, we randomly select single-agent specifications from φ_1-φ_9 for each agent and construct new `dep` and `cnt` specifications according to the number of agents, such that every agent is involved in the inter-agent constraints. We then combine the single-agent, `dep`, and `cnt` specifications to form $\Phi_{\text{indep/dep/cnt}}$ for evaluation. We also retrain the policies for each $N_{\text{train}} \in \{3, 5, 10\}$ and evaluate each trained policy with $N_{\text{eval}} \in \{3, 5, 10\}$ agents. This procedure allows us to evaluate the scalability of the methods as the number of agents and the resulting specification complexity increase.

Ablation study of the search-based method. For the ablation study, we evaluate the three IND global specifications in $\Phi_{\text{indep/dep/cnt}}$ for the three-agent setting, i.e., $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{10} \wedge \varphi_{13}$, $\varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_{11} \wedge \varphi_{14}$, and $\varphi_7 \wedge \varphi_8 \wedge \varphi_9 \wedge \varphi_{12} \wedge \varphi_{15}$. The resulting automata contain an average of 99.67 ± 34.72 states and 1851.00 ± 908.17 transitions, while the corresponding individual specifications have an average of 6.53 ± 3.30 states and 21.47 ± 19.23 transitions. We use these specifications to compare the proposed sampling-based method with the search-based method and demonstrate that sampling provides a more practical approach for handling complex global specifications with large automata.

Translation of `dep` and `cnt` into LTL formulas. The proposed `dep` and `cnt` operators provide compact notation for inter-agent coordination requirements and can be translated into standard LTL formulas. The operator $\alpha \,\texttt{Req}\, \beta$ requires β to occur before the first occurrence of α

and can be translated as $\neg\alpha \ \mathsf{U} \ \beta$. The operator $\alpha \, \texttt{EachReq} \, \beta$ requires this prerequisite relation to hold for every non-consecutive occurrence of α and can be translated as $(\neg\alpha \ \mathsf{U} \ \beta) \wedge \mathsf{G}\,((\alpha \wedge \mathsf{X}\neg\alpha) \rightarrow \mathsf{X}(\neg\alpha \ \mathsf{W} \ \beta))$, where $\mathsf{X}\,\varphi$ means that φ holds at the next time step, and weak until is defined as $\varphi_1 \ \mathsf{W} \ \varphi_2 \equiv (\varphi_1 \ \mathsf{U} \ \varphi_2) \vee \mathsf{G}\,\varphi_1$ (Baier et al., 2008). For the `cnt` operators, let $\mathcal{I}$ denote the set of involved agents, α_i the proposition associated with agent $i \in \mathcal{I}$, and m the required number of agents. $\texttt{Sync}[\mathcal{I}, \alpha, m]$ requires at least m agents in $\mathcal{I}$ to satisfy their corresponding α_i simultaneously, and can be translated as the disjunction over all subsets $\mathcal{K} \subseteq \mathcal{I}$ with $|\mathcal{K}| = m$ of $\mathsf{F}\,(\bigwedge_{i\in\mathcal{K}} \alpha_i)$. Similarly, $\texttt{Async}[\mathcal{I}, \alpha, m]$ requires at least m agents to each satisfy their corresponding α_i without requiring simultaneous satisfaction, and can be translated as the disjunction over all such subsets of $\bigwedge_{i\in\mathcal{K}} \mathsf{F}\,\alpha_i$. Boolean combinations of `dep` and `cnt` operators are translated compositionally using standard LTL connectives. Thus, these operators do not extend the expressiveness of LTL, but provide convenient notation for expressing inter-agent dependency and count requirements.

Table 3: Specifications used for evaluation. b, g, m, and y denote blue, green, magenta, and yellow, respectively. sa and ma denote specifications for single-agent and multi-agent behaviors, respectively. Top: finite-horizon specifications with 3 agents, with results in Table 1. Bottom left: finite-horizon specifications with 5 and 10 agents for the scalability study, with results in Figure 3. Bottom right: infinite-horizon specifications with 3 agents, with results in Table 5.

Specification types			IND		OOD
Finite-horizon (3 agents)	sa	φ_1	$(\neg \mathsf{m} \ \mathsf{U} \ (\mathsf{b} \vee \mathsf{g})) \wedge (\ \mathsf{F}\ \mathsf{m})$	φ_{16}	$\neg \mathsf{m} \ \mathsf{U} \ (\mathsf{b} \wedge \ \mathsf{F}\ (\neg \mathsf{y} \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ \mathsf{m})))$
	sa	φ_2	$(\ \mathsf{F}\ \mathsf{b}) \wedge (\neg \mathsf{b} \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ \mathsf{y}))$	φ_{17}	$\neg \mathsf{b} \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ (\neg \mathsf{m} \ \mathsf{U} \ (\mathsf{y} \wedge \ \mathsf{F}\ \mathsf{b})))$
	sa	φ_3	$\neg(\mathsf{m} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{b} \wedge \ \mathsf{F}\ \mathsf{g})$	φ_{18}	$\neg(\mathsf{m} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{b} \wedge \ \mathsf{F}\ (\neg \mathsf{b} \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ \mathsf{m})))$
	sa	φ_4	$\neg \mathsf{m} \ \mathsf{U} \ ((\mathsf{b} \vee \mathsf{y}) \wedge \ \mathsf{F}\ \mathsf{g})$	φ_{19}	$\neg \mathsf{m} \ \mathsf{U} \ ((\mathsf{b} \vee \mathsf{y}) \wedge \ \mathsf{F}\ (\neg \mathsf{b} \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ \mathsf{m})))$
	sa	φ_5	$\neg(\mathsf{b} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ \mathsf{m})$	φ_{20}	$\neg(\mathsf{b} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{g} \wedge \ \mathsf{F}\ (\neg \mathsf{g} \ \mathsf{U} \ (\mathsf{m} \wedge \ \mathsf{F}\ \mathsf{y})))$
	sa	φ_6	$(\ \mathsf{F}\ \mathsf{m}) \wedge (\neg \mathsf{y} \ \mathsf{U} \ (\mathsf{b} \vee \mathsf{g}))$	φ_{21}	$\neg \mathsf{y} \ \mathsf{U} \ ((\mathsf{b} \vee \mathsf{g}) \wedge \ \mathsf{F}\ (\neg \mathsf{m} \ \mathsf{U} \ (\mathsf{y} \wedge \ \mathsf{F}\ \mathsf{b})))$
	sa	φ_7	$\neg \mathsf{g} \ \mathsf{U} \ (\mathsf{y} \wedge \ \mathsf{F}\ (\mathsf{b} \vee \mathsf{m}))$	φ_{22}	$\neg \mathsf{g} \ \mathsf{U} \ (\mathsf{y} \wedge \ \mathsf{F}\ (\neg(\mathsf{y} \vee \mathsf{m}) \ \mathsf{U} \ (\mathsf{b} \wedge \ \mathsf{F}\ \mathsf{m})))$
	sa	φ_8	$\neg \mathsf{m} \ \mathsf{U} \ ((\mathsf{g} \vee \mathsf{b}) \wedge \ \mathsf{F}\ \mathsf{y})$	φ_{23}	$\neg \mathsf{m} \ \mathsf{U} \ ((\mathsf{g} \vee \mathsf{b}) \wedge \ \mathsf{F}\ (\neg \mathsf{y} \ \mathsf{U} \ (\mathsf{m} \wedge \ \mathsf{F}\ \mathsf{g})))$
	sa	φ_9	$\neg(\mathsf{g} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{m} \wedge \ \mathsf{F}\ \mathsf{b})$	φ_{24}	$\neg(\mathsf{g} \vee \mathsf{y}) \ \mathsf{U} \ (\mathsf{m} \wedge \ \mathsf{F}\ (\neg \mathsf{m} \ \mathsf{U} \ (\mathsf{b} \wedge \ \mathsf{F}\ \mathsf{y})))$
	dep	φ_{10}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{b}_1 \ \mathsf{Req}\ \mathsf{g}_2) \wedge (\mathsf{b}_2 \ \mathsf{Req}\ \mathsf{m}_0)$	φ_{25}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{b}_2) \wedge (\mathsf{g}_1 \ \mathsf{Req}\ \mathsf{g}_0) \wedge (\mathsf{b}_2 \ \mathsf{Req}\ \mathsf{y}_1)$
	dep	φ_{11}	$(\mathsf{g}_0 \ \mathsf{Req}\ \mathsf{m}_1) \wedge (\mathsf{m}_1 \ \mathsf{Req}\ \mathsf{b}_2) \wedge (\mathsf{m}_2 \ \mathsf{Req}\ \mathsf{g}_1)$	φ_{26}	$(\mathsf{g}_1 \ \mathsf{Req}\ \mathsf{g}_0) \wedge (\mathsf{g}_1 \ \mathsf{Req}\ \mathsf{y}_2) \wedge (\mathsf{g}_0 \ \mathsf{Req}\ \mathsf{m}_1) \wedge (\mathsf{y}_2 \ \mathsf{Req}\ \mathsf{m}_0)$
	dep	φ_{12}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{y}_1 \ \mathsf{Req}\ \mathsf{m}_2) \wedge (\mathsf{b}_2 \ \mathsf{Req}\ \mathsf{y}_0)$	φ_{27}	$(\mathsf{y}_0 \ \mathsf{Req}\ \mathsf{m}_2) \wedge (\mathsf{y}_0 \ \mathsf{Req}\ \mathsf{m}_1) \wedge (\mathsf{m}_2 \ \mathsf{Req}\ \mathsf{b}_0) \wedge (\mathsf{m}_1 \ \mathsf{Req}\ \mathsf{b}_2)$
	cnt	φ_{13}	$\texttt{Sync}\,[\{0, 2\}, \mathsf{y}, 2] \wedge \ \texttt{Async}\,[\{0, 1, 2\}, \mathsf{g}, 3]$	φ_{28}	$\texttt{Sync}\,[\{0, 1, 2\}, \mathsf{g}, 2] \wedge \ \texttt{Async}\,[\{0, 1\}, \mathsf{y}, 1] \wedge \ \texttt{Async}\,[\{1, 2\}, \mathsf{m}, 1]$
	cnt	φ_{14}	$\texttt{Sync}\,[\{0, 1, 2\}, \mathsf{m}, 2] \wedge \ \texttt{Async}\,[\{0, 2\}, \mathsf{y}, 2]$	φ_{29}	$\texttt{Async}\,[\{0, 1, 2\}, \mathsf{m}, 3] \wedge \ \texttt{Sync}\,[\{0, 2\}, \mathsf{y}, 1] \wedge \ \texttt{Sync}\,[\{0, 1, 2\}, \mathsf{b}, 2]$
	cnt	φ_{15}	$\texttt{Sync}\,[\{1, 2\}, \mathsf{g}, 2] \wedge \ \texttt{Async}\,[\{0, 1, 2\}, \mathsf{b}, 2]$	φ_{30}	$\texttt{Sync}\,[\{1, 2\}, \mathsf{g}, 1] \wedge \ \texttt{Async}\,[\{0, 1, 2\}, \mathsf{b}, 2] \wedge \ \texttt{Sync}\,[\{0, 1\}, \mathsf{m}, 1]$
	ma	Φ_{indep}	$\{\{\varphi_1, \varphi_2, \varphi_3\}, \{\varphi_4, \varphi_5, \varphi_6\}, \{\varphi_7, \varphi_8, \varphi_9\}\}$	Φ_{indep}	$\{\{\varphi_{16}, \varphi_{17}, \varphi_{18}\}, \{\varphi_{19}, \varphi_{20}, \varphi_{21}\}, \{\varphi_{22}, \varphi_{23}, \varphi_{24}\}\}$
	ma	$\Phi_{\text{indep/dep}}$	$\{\{\varphi_1, \varphi_2, \varphi_3, \varphi_{10}\}, \{\varphi_4, \varphi_5, \varphi_6, \varphi_{11}\}, \{\varphi_7, \varphi_8, \varphi_9, \varphi_{12}\}\}$	$\Phi_{\text{indep/dep}}$	$\{\{\varphi_{16}, \varphi_{17}, \varphi_{18}, \varphi_{25}\}, \{\varphi_{19}, \varphi_{20}, \varphi_{21}, \varphi_{26}\}, \{\varphi_{22}, \varphi_{23}, \varphi_{24}, \varphi_{27}\}\}$
	ma	$\Phi_{\text{indep/cnt}}$	$\{\{\varphi_1, \varphi_2, \varphi_3, \varphi_{13}\}, \{\varphi_4, \varphi_5, \varphi_6, \varphi_{14}\}, \{\varphi_7, \varphi_8, \varphi_9, \varphi_{15}\}\}$	$\Phi_{\text{indep/cnt}}$	$\{\{\varphi_{16}, \varphi_{17}, \varphi_{18}, \varphi_{28}\}, \{\varphi_{19}, \varphi_{20}, \varphi_{21}, \varphi_{29}\}, \{\varphi_{22}, \varphi_{23}, \varphi_{24}, \varphi_{30}\}\}$
	ma	$\Phi_{\text{indep/dep/cnt}}$	$\{\{\varphi_1, \varphi_2, \varphi_3, \varphi_{10}, \varphi_{13}\}, \{\varphi_4, \varphi_5, \varphi_6, \varphi_{11}, \varphi_{14}\}, \{\varphi_7, \varphi_8, \varphi_9, \varphi_{12}, \varphi_{15}\}\}$	$\Phi_{\text{indep/dep/cnt}}$	$\{\{\varphi_{16}, \varphi_{17}, \varphi_{18}, \varphi_{25}, \varphi_{28}\}, \{\varphi_{19}, \varphi_{20}, \varphi_{21}, \varphi_{26}, \varphi_{29}\}, \{\varphi_{22}, \varphi_{23}, \varphi_{24}, \varphi_{27}, \varphi_{30}\}\}$

Finite-horizon (5 agents)	dep	φ_{46}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{b}_1 \ \mathsf{Req}\ \mathsf{g}_2) \wedge (\mathsf{g}_2 \ \mathsf{Req}\ \mathsf{y}_3) \wedge (\mathsf{g}_3 \ \mathsf{Req}\ \mathsf{g}_4)$	Infinite-horizon	sa	φ_{31}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{g}) \wedge (\mathsf{G}\,\neg\mathsf{y})$
	dep	φ_{47}	$(\mathsf{g}_0 \ \mathsf{Req}\ \mathsf{m}_1) \wedge (\mathsf{b}_1 \ \mathsf{Req}\ \mathsf{b}_2) \wedge (\mathsf{y}_2 \ \mathsf{Req}\ \mathsf{g}_3) \wedge (\mathsf{b}_3 \ \mathsf{Req}\ \mathsf{y}_4)$		sa	φ_{32}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{y}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{g}) \wedge (\mathsf{G}\,\neg\mathsf{b})$
	dep	φ_{48}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{b}_1) \wedge (\mathsf{g}_1 \ \mathsf{Req}\ \mathsf{g}_2) \wedge (\mathsf{m}_2 \ \mathsf{Req}\ \mathsf{y}_3) \wedge (\mathsf{y}_3 \ \mathsf{Req}\ \mathsf{m}_4)$		sa	φ_{33}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{m}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\neg\mathsf{g})$
	cnt	φ_{49}	$\texttt{Async}\,[\{0, 1, 2, 3, 4\}, \mathsf{y}, 3] \wedge \texttt{Sync}\,[\{0, 1, 2, 3, 4\}, \mathsf{b}, 2]$		sa	φ_{34}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{y}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{m}) \wedge (\mathsf{G}\,\neg\mathsf{g})$
	cnt	φ_{50}	$\texttt{Sync}\,[\{0, 1, 2, 3, 4\}, \mathsf{y}, 2] \wedge \texttt{Async}\,[\{0, 1, 2, 3, 4\}, \mathsf{g}, 4]$		sa	φ_{35}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{m}) \wedge (\mathsf{G}\,\neg\mathsf{y})$
	cnt	φ_{51}	$\texttt{Sync}\,[\{0, 1, 2, 3, 4\}, \mathsf{m}, 2] \wedge \texttt{Sync}\,[\{0, 1, 2, 3, 4\}, \mathsf{g}, 2]$		sa	φ_{36}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{g}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{y}) \wedge (\mathsf{G}\,\neg\mathsf{b})$
	ma	$\Phi_{\text{indep/dep/cnt}}$	$\{\{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_{46}, \varphi_{49}\}, \{\varphi_5, \varphi_6, \varphi_7, \varphi_8, \varphi_9, \varphi_{47}, \varphi_{50}\}, \{\varphi_1, \varphi_3, \varphi_5, \varphi_7, \varphi_9, \varphi_{48}, \varphi_{51}\}\}$		sa	φ_{37}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\neg\mathsf{m})$
					sa	φ_{38}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{m}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{y}) \wedge (\mathsf{G}\,\neg\mathsf{g})$
Finite-horizon (10 agents)	dep	φ_{52}	$(\mathsf{b}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{b}_1 \ \mathsf{Req}\ \mathsf{g}_2) \wedge (\mathsf{b}_2 \ \mathsf{Req}\ \mathsf{m}_3) \wedge (\mathsf{b}_3 \ \mathsf{Req}\ \mathsf{b}_4) \wedge (\mathsf{g}_4 \ \mathsf{Req}\ \mathsf{m}_5) \wedge (\mathsf{b}_5 \ \mathsf{Req}\ \mathsf{g}_6) \wedge (\mathsf{b}_6 \ \mathsf{Req}\ \mathsf{g}_7) \wedge (\mathsf{m}_7 \ \mathsf{Req}\ \mathsf{b}_8) \wedge (\mathsf{y}_8 \ \mathsf{Req}\ \mathsf{b}_9)$		sa	φ_{39}	$(\mathsf{G}\,\mathsf{F}\,\mathsf{m}) \wedge (\mathsf{G}\,\mathsf{F}\,\mathsf{b}) \wedge (\mathsf{G}\,\neg\mathsf{m})$
	dep	φ_{53}	$(\mathsf{y}_0 \ \mathsf{Req}\ \mathsf{g}_1) \wedge (\mathsf{y}_1 \ \mathsf{Req}\ \mathsf{m}_2) \wedge (\mathsf{b}_2 \ \mathsf{Req}\ \mathsf{b}_3) \wedge (\mathsf{g}_3 \ \mathsf{Req}\ \mathsf{b}_4) \wedge (\mathsf{b}_4 \ \mathsf{Req}\ \mathsf{b}_5) \wedge (\mathsf{g}_5 \ \mathsf{Req}\ \mathsf{g}_6) \wedge (\mathsf{y}_6 \ \mathsf{Req}\ \mathsf{b}_7) \wedge (\mathsf{g}_7 \ \mathsf{Req}\ \mathsf{g}_8) \wedge (\mathsf{m}_8 \ \mathsf{Req}\ \mathsf{g}_9)$		dep	φ_{40}	$(\mathsf{b}_0 \ \texttt{EachReq}\ \mathsf{g}_1) \wedge (\mathsf{y}_1 \ \texttt{EachReq}\ \mathsf{b}_2)$
	dep	φ_{54}	$(\mathsf{m}_0 \ \mathsf{Req}\ \mathsf{b}_1) \wedge (\mathsf{m}_1 \ \mathsf{Req}\ \mathsf{y}_2) \wedge (\mathsf{y}_2 \ \mathsf{Req}\ \mathsf{b}_3) \wedge (\mathsf{b}_3 \ \mathsf{Req}\ \mathsf{m}_4) \wedge (\mathsf{b}_4 \ \mathsf{Req}\ \mathsf{y}_5) \wedge (\mathsf{y}_5 \ \mathsf{Req}\ \mathsf{m}_6) \wedge (\mathsf{m}_6 \ \mathsf{Req}\ \mathsf{b}_7) \wedge (\mathsf{g}_7 \ \mathsf{Req}\ \mathsf{y}_8) \wedge (\mathsf{y}_8 \ \mathsf{Req}\ \mathsf{m}_9)$		dep	φ_{41}	$(\mathsf{y}_0 \ \texttt{EachReq}\ \mathsf{m}_1) \wedge (\mathsf{b}_1 \ \texttt{EachReq}\ \mathsf{y}_2)$
	cnt	φ_{55}	$\texttt{Async}\,[\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \mathsf{b}, 9]$		dep	φ_{42}	$(\mathsf{g}_0 \ \texttt{EachReq}\ \mathsf{y}_1) \wedge (\mathsf{m}_1 \ \texttt{EachReq}\ \mathsf{b}_2)$
	cnt	φ_{56}	$\texttt{Async}\,[\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \mathsf{m}, 7]$		cnt	φ_{43}	$\neg\ \texttt{Async}\,[\{0, 1, 2\}, \mathsf{m}, 2]$
	cnt	φ_{57}	$\texttt{Async}\,[\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \mathsf{g}, 9]$		cnt	φ_{44}	$\neg\ \texttt{Sync}\,[\{0, 1, 2\}, \mathsf{b}, 2]$
	ma	$\Phi_{\text{indep/dep/cnt}}$	$\{\{\varphi_3, \varphi_8, \varphi_1, \varphi_6, \varphi_4, \varphi_9, \varphi_2, \varphi_5, \varphi_7, \varphi_3, \varphi_{52}, \varphi_{55}\}, \{\varphi_7, \varphi_2, \varphi_9, \varphi_4, \varphi_1, \varphi_6, \varphi_8, \varphi_3, \varphi_5, \varphi_8, \varphi_{53}, \varphi_{56}\}, \{\varphi_5, \varphi_1, \varphi_7, \varphi_3, \varphi_9, \varphi_2, \varphi_6, \varphi_4, \varphi_8, \varphi_1, \varphi_{54}, \varphi_{57}\}\}$		cnt	φ_{45}	$\neg\ \texttt{Async}\,[\{0, 1, 2\}, \mathsf{g}, 2]$
					ma	Φ_{indep}	$\{\{\varphi_{31}, \varphi_{32}, \varphi_{33}\}, \{\varphi_{34}, \varphi_{35}, \varphi_{36}\}, \{\varphi_{37}, \varphi_{38}, \varphi_{39}\}\}$
					ma	$\Phi_{\text{indep/dep}}$	$\{\{\varphi_{31}, \varphi_{32}, \varphi_{33}, \varphi_{40}\}, \{\varphi_{34}, \varphi_{35}, \varphi_{36}, \varphi_{41}\}, \{\varphi_{37}, \varphi_{38}, \varphi_{39}, \varphi_{42}\}\}$
					ma	$\Phi_{\text{indep/cnt}}$	$\{\{\varphi_{31}, \varphi_{32}, \varphi_{33}, \varphi_{43}\}, \{\varphi_{34}, \varphi_{35}, \varphi_{36}, \varphi_{44}\}, \{\varphi_{37}, \varphi_{38}, \varphi_{39}, \varphi_{45}\}\}$
					ma	$\Phi_{\text{indep/dep/cnt}}$	$\{\{\varphi_{31}, \varphi_{32}, \varphi_{33}, \varphi_{40}, \varphi_{43}\}, \{\varphi_{34}, \varphi_{35}, \varphi_{36}, \varphi_{41}, \varphi_{44}\}, \{\varphi_{37}, \varphi_{38}, \varphi_{39}, \varphi_{42}, \varphi_{45}\}\}$

C Implementation Details

We use the official codebase of the baseline ACC-MARL[1]. The low-level policy used in our framework is adapted from GenZ-LTL[2] by extending it to the multi-agent setting, while we implement the proposed high-level sampling-based planner. For a fair comparison, all methods use the same network architectures: the actor and critic are fully connected networks with three hidden layers of size $[64, 64, 64]$, and the value function network has two hidden layers of size $[64, 64]$. We retain the original state/observation preprocessing of each method. All methods use the `Adam` optimizer

[1] https://github.com/rad-dfa/acc-marl

[2] https://github.com/BU-DEPEND-Lab/GenZ-LTL

with a learning rate of 3×10^{-4} and are trained for 15M environment interactions. The discount factor is set to $\gamma = 0.998$ for `ZoneEnv`. Additional hyperparameter details are provided in the code included in the supplementary materials. All methods are trained on a 16-core AMD CPU and an NVIDIA GeForce RTX 4090 GPU, with our model taking approximately 6 hours to train.

Training and evaluation. For each agent i, we construct a feasible reach-avoid subtask set by considering every assignment $\alpha_i \in 2^{AP_i}$ as a candidate reach condition A_i^+. For each candidate, assignments that conflict with A_i^+ are discarded, and all combinations of the remaining assignments are enumerated to construct the avoid condition A_i^-. This yields the subtask set $\xi_i = \{(A_{i,m}^+, A_{i,m}^-)\}_{m=1}$. During training, a reach-avoid subtask $\sigma_i$is uniformly sampled from the corresponding ξ_i for each agent. We define a binary reward $r_i = 1$ when the current true propositions, obtained according to L_i, satisfy the reach condition of σ_i, and $r_i = 0$ otherwise. Similarly, we define a binary cost $c_i = 1$ when the current true propositions satisfy the avoid condition of σ_i or agent i collides with another agent, and $c_i = 0$ otherwise. If any $c_i > 0$, the episode is terminated immediately, since the corresponding specification can no longer be satisfied. Otherwise, whenever $r_i = 1$, a new subtask is re-sampled for agent i. We train GenZ-LTL using the rewards/costs collected during interactions, and the policy and critics are shared across agents because the agents are homogeneous in the `Zone` environment. The 5 seeds used for training and evaluation are $[1, 11, 22, 33, 44]$. During evaluation, the agent and zone positions are randomly sampled for each episode, while the same sampled configurations are used across different methods under each seed for fair comparison.

Specification satisfaction evaluation. For each rollout, we evaluate its trace against every specification component φ_i, $i \in \mathcal{I}$. For finite-horizon tasks, a trajectory is considered successful only if all specification components are satisfied, whereas it is considered a violation if any component is violated. For infinite-horizon tasks, violation is determined in the same way. For each non-violating trajectory, we count the visits to accepting states for every specification component and take the minimum of these counts as the number of accepting-state visits reported in Table 5.

Product automaton. We use the Spot[3] library to construct an automaton for each specification component φ_i and the corresponding product automaton for the global specification $\varphi_{\texttt{global}} = \bigwedge_{i \in \mathcal{I}} \varphi_i$. We then record the number of states in the resulting global automaton. For the specifications in Section 5.2, we perform this analysis for $\Phi_{\texttt{ind/dep/cnt}}$ with $N \in \{3, 5, 10\}$ and report the mean and standard deviation of the automaton-state counts in Table 2. For $N = 10$, the complete product automata could not be constructed within tractable computational limits (1 hour limit), as each global specification contains 12 component specifications, including 10 individual-agent specifications together with one `dep` and one `cnt` specification with $|AP| = 40$. We therefore report the largest observed state number as a lower bound, denoted by $> \text{num}$ in Table 2.

Details of the sampling-based planner. For each φ_i, $i \in \mathcal{I}$, let $\mathcal{G}_i = \{g_i^1, g_i^2, \ldots, g_i^{M_i}\}$ denote the set of feasible subtask sequences, where M_i is the number of feasible subtask sequences for φ_i. Each $g_i^j \in \mathcal{G}_i$ is a sequence $g_i^j = [\sigma_{i,1}^j, \sigma_{i,2}^j, \ldots]$, where $\sigma_{i,k}^j = (A_{i,k}^{+,j}, A_{i,k}^{-,j})$ denotes a reach-avoid subtask. We assume without loss of generality that each reach condition $A_{i,k}^{+,j}$ and avoid condition $A_{i,k}^{-,j}$ is expressed in disjunctive normal form (DNF) (Rosen & Krithivasan, 1999), where each conjunctive clause is represented by the set of atomic propositions it contains; for example, $(a \wedge b) \vee (c \wedge d)$ is represented as $\{\{a, b\}, \{c, d\}\}$. Algorithm 2 extends a sampled node by finding compatible next subtasks. We first identify the expandable components $\mathcal{I}_{\text{exp}}$ such that, for each $i \in \mathcal{I}_{\text{exp}}$, none of its subtask sequences has reached its end, and then sample a nonempty subset $\mathcal{I}_{\text{s}} \subseteq \mathcal{I}_{\text{exp}}$. Now we explain the main loop of Algorithm 2 (line 5). For each $i \in \mathcal{I}_{\text{s}}$, if no subtask sequence has been selected, we uniformly sample $g_i^j \in \mathcal{G}_i$, retrieve its first subtask, and update the corresponding tuple to $(i, j, 1)$. Otherwise, given the tuple (i, j, k) stored in x_{parent}, we retrieve the next subtask $\sigma_{i,k+1}^j$ and update the corresponding tuple to $(i, j, k + 1)$.

We then check whether the resulting node is valid by verifying the compatibility of the selected subtasks based on their reach and avoid conditions. For each component $i \in \mathcal{I}_{\text{s}}$ whose current subtask is $\sigma_{i,k}^j = (A_{i,k}^{+,j}, A_{i,k}^{-,j})$, one conjunctive clause is sampled from $A_{i,k}^{+,j}$ and denoted by $\tilde{A}_i^+$. Thus $\tilde{A}_i^+ = \emptyset$ for $i \in \mathcal{I} \setminus \mathcal{I}_{\text{s}}$ since no new reach condition is considered in the current extension.

[3] `https://spot.lre.epita.fr/`

The centralized planner then constructs the joint reach condition $A^+_{\text{joint}} := \bigcup_{i\in\mathcal{I}} \tilde{A}^+_i$. To construct the joint avoid condition, we consider every component $i \in \mathcal{I}$. For $i \in \mathcal{I}_{\text{s}}$, we use the avoid condition of the newly retrieved subtask. For $i \in \mathcal{I} \setminus \mathcal{I}_{\text{s}}$, given its stored tuple (i, j, k), if $j = 0$, we uniformly sample a sequence $g^j_i \in \mathcal{G}_i$ and use the avoid condition of its first subtask; otherwise, we use the avoid condition of the current subtask $\sigma^j_{i,k}$. Let $\tilde{A}^-_i$ denote the resulting avoid condition for component i. Since each $\tilde{A}^-_i$ is represented in DNF as a set of conjunctive clauses, with each clause itself represented as a set of atomic propositions, the joint avoid condition is also a set of such clauses and is defined as $A^-_{\text{joint}} := \bigcup_{i\in\mathcal{I}} \tilde{A}^-_i$. The joint reach and avoid conditions are incompatible if any conjunctive clause $B \in A^-$ satisfies $B \subseteq A^+$, since satisfying the reach condition would necessarily violate the avoid condition. Otherwise, propositions already contained in A^+ are removed from each avoid clause, yielding the final avoid condition $\hat{A}^- := \{B \setminus A^+ \mid B \in A^-\}$. For example, if $A^+ = \{a, b\}$ and $A^- = \{\{a, b, c\}, \{d\}\}$, the conditions are compatible and reduce to $A^+ = \{a, b\}$ and $\hat{A}^- = \{\{c\}, \{d\}\}$. In contrast, if $A^+ = \{a, b\}$ and $A^- = \{\{a\}, \{c\}\}$, they are incompatible because $\{a\} \subseteq A^+$. This compatibility check depends only on the Boolean structure of the reach and avoid conditions.

Replanning and value-guided sampling. During execution, we evaluate the current active propositions $L(s)$ and use them to update the automaton state of each specification component. If none of the automaton states changes, the agents continue following the current plan. Whenever the automaton state of at least one component changes, we recompute the feasible reach-avoid subtask sequences from the updated automaton states and invoke the planner again to generate a new plan from the current specification progress. For infinite-horizon specifications, we construct each candidate subtask sequence from its finite prefix followed by n_{cyc} repetitions of its cyclic suffix, and apply the same planning and replanning procedure to these finite sequences. On the other hand, at test time, learned value functions can in principle be used to guide subtask selection, e.g., by choosing the candidate subtask σ with the largest value $V(s, \sigma)$ at the current state s. However, evaluating a multi-step plan also requires estimating the values of subsequent subtasks at future states that have not yet been observed. One possible approach is to train an additional model to predict a future state $\tilde{s}$ and then evaluate $V(\tilde{s}, \sigma')$ for the next subtask σ'. However, the planning horizon can be long, and prediction errors can accumulate over multiple subtasks, making accurate future-state estimates difficult to obtain. Therefore, value-function-based lookahead becomes challenging for long-horizon task allocation considered in our setting. Moreover, under partial observability, optimizing the planned subtask sequence is challenging, and value-guided sampling does not offer an obvious advantage. We leave a systematic investigation of this direction to future work.

D Additional Results

Performance on simple specifications. We additionally evaluate our method and ACC-MARL on simple specifications consisting of reach-only tasks, i.e., $\mathsf{F}\,\alpha$, and reach-avoid tasks, i.e., $\neg\alpha\ \mathsf{U}\ \beta$, where $\alpha, \beta \in AP$ and $\alpha \neq \beta$. The results are shown in Table 4. Note that for reach-only specifications, violations arise solely from inter-agent collisions. ACC-MARL achieves substantially higher success rates on these simple specifications than on the more complex specifications considered in the main experiments, suggesting that its performance degradation is mainly due to the increased specification complexity. In contrast, our method maintains consistently high success rates and low violation rates across both simple and complex specifications.

Table 4: Evaluation results on simple specifications: success rate η_s, violation rate η_v, and average number of steps μ. ↑: higher is better; ↓: lower is better. Each value is averaged over 5 seeds, with 100 trajectories per seed.

Method	Specifications	$\eta_s \uparrow$	$\eta_v \downarrow$	$\mu \downarrow$
ACC-MARL	Reach-Only	$0.77_{\pm 0.08}$	$0.19_{\pm 0.07}$	$261.27_{\pm 39.58}$
	Reach-Avoid	$0.60_{\pm 0.09}$	$0.37_{\pm 0.08}$	$226.06_{\pm 38.71}$
Ours	Reach-Only	$0.97_{\pm 0.02}$	$0.02_{\pm 0.02}$	$139.92_{\pm 14.16}$
	Reach-Avoid	$0.98_{\pm 0.02}$	$0.01_{\pm 0.01}$	$148.40_{\pm 15.47}$

Infinite-horizon tasks. We also evaluate infinite-horizon tasks, i.e., specifications whose satisfaction requires infinite-length trajectories. We focus on specifications involving $\mathsf{G}\,\mathsf{F}\,\varphi$, which induce continual navigation and inter-agent interactions, rather than $\mathsf{F}\,\mathsf{G}\,\varphi$, where agents remain in specific regions after reaching them. ACC-MARL is the only existing baseline applicable to our setting, but it is excluded from this evaluation because DFAs cannot represent infinite-horizon specifications. We therefore report the performance of our method without baseline comparison, providing a reference for future evaluation when suitable baselines become available. The evaluated specifications are listed in Table 3, with results reported in Table 5. We report the violation rate η_v and the average number of visits to accepting states among non-violating trajectories, $\mu_{\texttt{acc}}$. Our method naturally extends to this setting, as the same sampling-based planning procedure can be repeatedly applied to the subtask sequences obtained from each specification. The learned policy is conditioned only on the current subtask and can therefore be directly reused throughout infinite-horizon execution. Our method can obtain low violation rates and frequent visits to accepting states across all evaluated specifications, although $\mu_{\texttt{acc}}$ decreases and η_v increases slightly as dependency and count requirements are introduced due to the increased coordination difficulty.

Table 5: Evaluation results of infinite-horizon tasks: violation rate η_v and average number of visits to accepting states μ_{acc} on infinite-horizon tasks listed in Table 3. ↑: higher is better; ↓: lower is better. Each value is averaged over 5 seeds, with 100 trajectories per seed.

Method	Metrics	Φ_{indep}	$\Phi_{\text{indep/req}}$	$\Phi_{\text{indep/cnt}}$	$\Phi_{\text{indep/req/cnt}}$	Average
Ours	$\mu_{\texttt{acc}}$ ↑	$10.71_{\pm 1.86}$	$5.98_{\pm 0.87}$	$10.48_{\pm 1.72}$	$5.55_{\pm 0.84}$	$8.18_{\pm 2.80}$
	η_v ↓	$0.05_{\pm 0.03}$	$0.15_{\pm 0.03}$	$0.05_{\pm 0.02}$	$0.16_{\pm 0.04}$	$0.10_{\pm 0.06}$

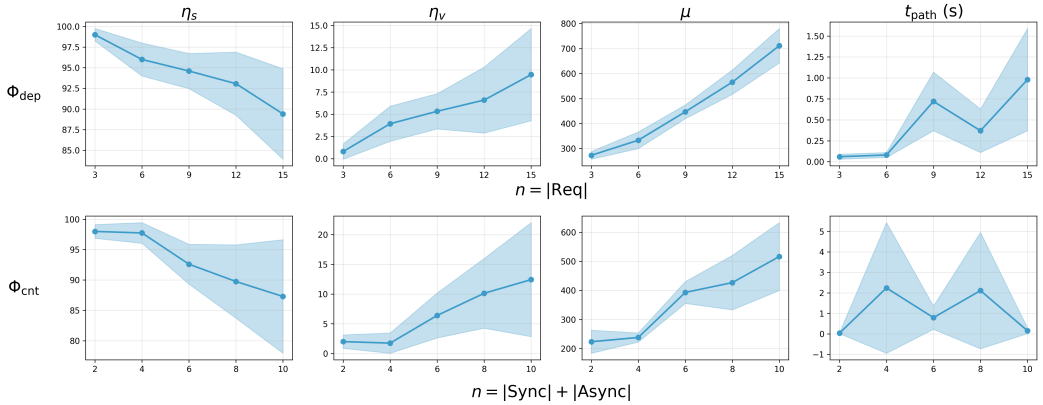


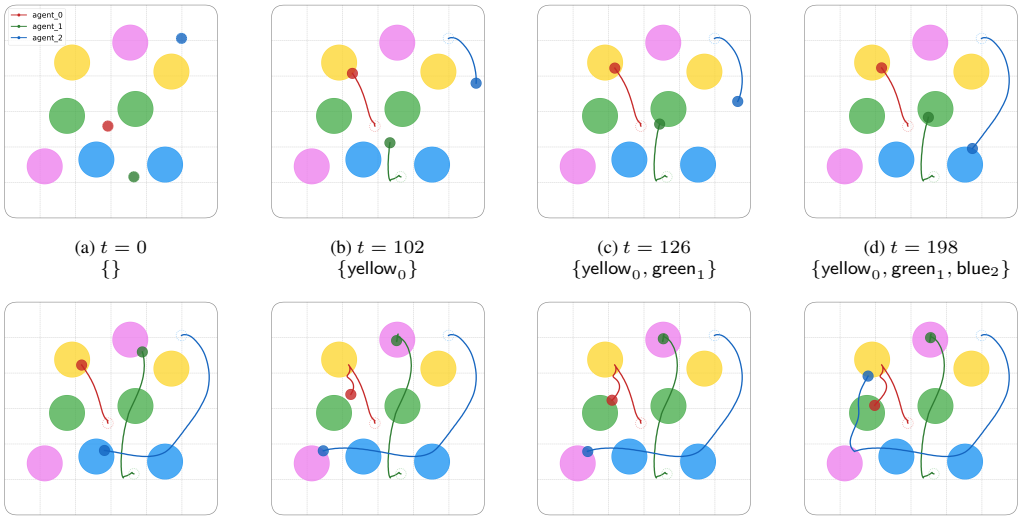
Figure 6: Performance as the complexity of inter-agent coordination specifications increases. The top row varies the number of dependency operators $n = |\texttt{Req}|$ in $\varphi_{\texttt{dep}}$, while the bottom row varies the number of count operators $n = |\texttt{Sync}| + |\texttt{Async}|$ in $\varphi_{\texttt{cnt}}$. We report the success rate η_s, violation rate η_v, average number of steps μ to satisfy a specification, and planning time $t_{\texttt{path}}$. Solid lines show the mean, and shaded regions indicate one standard deviation.

Coordination-specification complexity. We also evaluate how performance changes as the complexity of the inter-agent coordination specifications increases in the 3-agent setting. To isolate the effects of $\varphi_{\texttt{dep}}$ and $\varphi_{\texttt{cnt}}$, each agent is assigned a simple individual task of the form $\mathsf{F}\,\alpha$, while the coordination-specification complexity is varied separately. For dependency requirements, we construct $\varphi_{\texttt{dep}}$ as a conjunction of $\texttt{Req}$ clauses and increase $n = |\texttt{Req}|$, introducing more inter-agent dependency relations. Similarly, for count requirements, we construct $\varphi_{\texttt{cnt}}$ as a conjunction of $\texttt{Sync}$ and $\texttt{Async}$ clauses and increase $n = |\texttt{Sync}| + |\texttt{Async}|$, introducing more count requirements. Although dependency and count requirements can be combined within the same global specification, we vary them separately here to isolate the effect of each type of coordination complexity. As shown in Figure 6, increasing either type of specification complexity generally decreases the success rate η_s and increases the violation rate η_v and the number of steps μ required to satisfy the specification, as agents must satisfy a larger number of inter-agent coordination requirements. Meanwhile, the planning time $t_{\texttt{path}}$ remains tractable as the number of coordination operators in-

creases, since the sampling-based planner operates directly on subtask sequences without explicitly constructing the product automaton.

E QUALITATIVE RESULTS

We conduct a qualitative analysis of our method using specification $\{\varphi_4, \varphi_5, \varphi_6, \varphi_{11}, \varphi_{14}\}$, with the visualization of the trajectories shown in Figure 7 and the planning process of the high-level planner in Figure 8. In natural language, the specifications mean that $\texttt{agent}_0$ is required to avoid the magenta region until reaching either the blue or yellow region, followed by reaching the green region; $\texttt{agent}_1$ must avoid the blue and yellow regions until reaching the green region, followed by the magenta region; and $\texttt{agent}_2$ must reach the magenta region and avoid the yellow region until reaching either the blue or green region. The dep specification requires $\texttt{agent}_1$ to reach the magenta region before $\texttt{agent}_0$ reaches the green region, $\texttt{agent}_2$ to reach the blue region before $\texttt{agent}_1$ reaches the magenta region, and $\texttt{agent}_1$ to reach the green region before $\texttt{agent}_2$ reaches the magenta region. The cnt specification requires that at least two agents must reach the magenta region simultaneously, and both $\texttt{agent}_0$ and $\texttt{agent}_2$ must reach the yellow region, without requiring simultaneous visits.

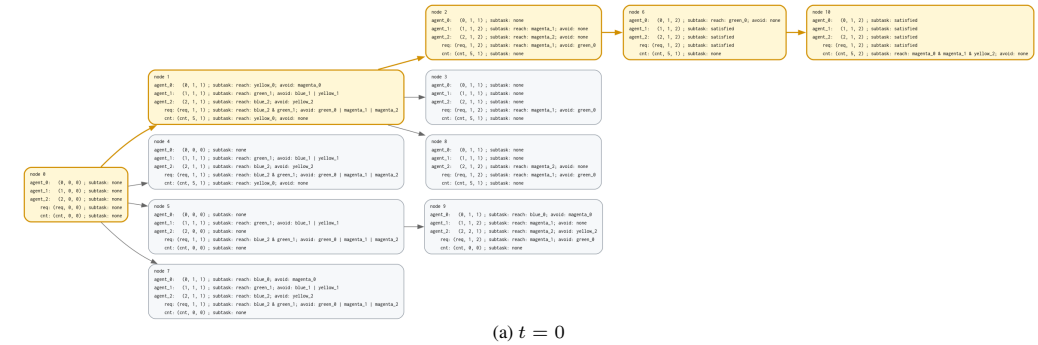


(a) $t = 0$ $\{\}$ (b) $t = 102$ $\{\text{yellow}_0\}$ (c) $t = 126$ $\{\text{yellow}_0, \text{green}_1\}$ (d) $t = 198$ $\{\text{yellow}_0, \text{green}_1, \text{blue}_2\}$

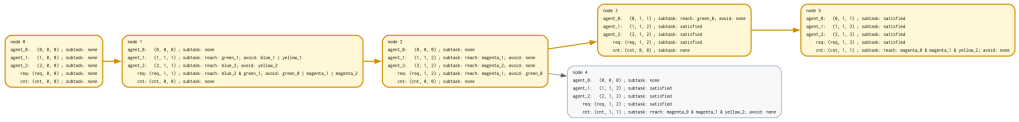
(e) $t = 289$ $\{\text{yellow}_0, \text{magenta}_1, \text{blue}_2\}$ (f) $t = 349$ $\{\text{magenta}_1, \text{magenta}_2\}$ (g) $t = 358$ $\{\text{green}_0, \text{magenta}_1, \text{magenta}_2\}$ (h) $t = 471$ $\{\text{green}_0, \text{magenta}_1, \text{yellow}_2\}$

Figure 7: Visualization of the trajectory of specification $\{\varphi_4, \varphi_5, \varphi_6, \varphi_{11}, \varphi_{14}\}$. For each subplot, t denotes the timestep, and the set denotes the propositions that are true at the current timestep.

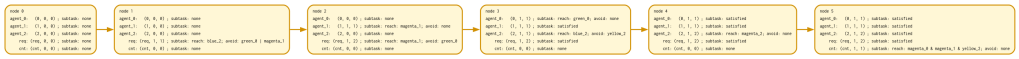
At the beginning of execution, the planner assigns $\sigma_0 = (\{\text{yellow}_0\}, \{\text{green}_0, \text{magenta}_0\})$ (i.e., reach yellow_0 and avoid green_0 or magenta_0), $\sigma_1 = (\{\text{green}_1\}, \{\text{blue}_1, \text{magenta}_1, \text{yellow}_1\})$, and $\sigma_2 = (\{\text{blue}_2\}, \{\text{magenta}_2, \text{yellow}_2\})$. After $\texttt{agent}_0$ reaches yellow at $t = 102$, its individual task permits progression toward green, but green_0 remains avoided until the prerequisite magenta_1 is satisfied; meanwhile, $\texttt{agent}_1$ and $\texttt{agent}_2$ continue their current subtasks. At $t = 126$, $\texttt{agent}_1$ reaches green, after which magenta_1 remains avoided until $\texttt{agent}_2$ reaches blue, as required by the corresponding dep constraint. Once $\texttt{agent}_2$ reaches blue at $t = 198$, the planner updates the subtasks to $\sigma_1 = (\{\text{magenta}_1\}, \{\emptyset\})$ and $\sigma_2 = (\{\text{magenta}_2\}, \{\emptyset\})$, while $\texttt{agent}_0$ continues to avoid green. After $\texttt{agent}_1$ first reaches magenta at $t = 289$, the prerequisite for green_0 is satisfied, and the planner assigns $\sigma_0 = (\{\text{green}_0\}, \{\emptyset\})$ while retaining magenta_1 and magenta_2 as targets to satisfy the synchronous count requirement. At $t = 349$, $\texttt{agent}_1$ and $\texttt{agent}_2$ reach magenta simultaneously, after which the remaining subtasks are $\sigma_0 = (\{\text{green}_0\}, \{\emptyset\})$ and $\sigma_2 = (\{\text{yellow}_2\}, \{\emptyset\})$. Finally, $\texttt{agent}_0$ reaches green at $t = 358$ and $\texttt{agent}_2$ reaches yellow at $t = 471$, and the specifications are satisfied.



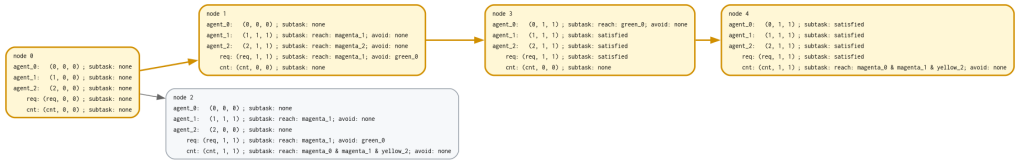
(a) $t = 0$



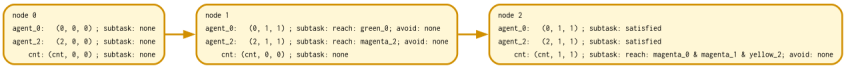
(b) $t = 102$



(c) $t = 126$



(d) $t = 198$



(e) $t = 289$



(f) $t = 349$ (g) $t = 358$

Figure 8: Visualization of the planning process of the high-level sampling-based planner for specification $\{\varphi_4, \varphi_5, \varphi_6, \varphi_{11}, \varphi_{14}\}$. Each subplot shows the planning state at timestep t. Yellow blocks denote the selected subtask sequence, while gray blocks denote sampled nodes that are not part of the selected sequence.