Prompting Robot Teams with Natural Language

Anonymous Authors

Abstract— **This paper presents a framework towards prompting multi-robot teams with high-level tasks using natural language expressions. Our objective is to use the reasoning capabilities demonstrated by recent language models in understanding and decomposing human expressions of intent, and repurpose these for multi-robot collaboration and decision-making. The key challenge is that an individual's behavior in a collective can be hard to specify and interpret, and must continuously adapt to actions from others. This necessitates a framework that possesses the representational capacity required by the logic and semantics of a task, and yet supports decentralized and interactive real-time operation. We solve this dilemma by recognizing that a task can be represented as a deterministic finite automaton (DFA), and that recurrent neural networks (RNNs) can encode numerous automata. This allows us to *distill* the logic and sequential decompositions of sub-tasks obtained from a language model into an RNN, and align its internal states with the semantics of a given task. By training a graph neural network (GNN) control policy that is conditioned on the hidden states of the RNN and the language embeddings, our method enables robots to execute task-relevant actions in a decentralized manner. We present evaluations of this single light-weight interpretable model on various simulated and real-world multi-robot tasks that require sequential and collaborative behavior by the team —** `sites.google.com/view/prompting-teams`.

I. Introduction

Teams of robots have a natural application in scenarios that benefit from strength in numbers [1]–[4]. When tasks involve robots supporting human operators, we anticipate the operators to provide abstract commands for the task (e.g., "explore this zone and find the missing people"), and supply semantically relevant hints (e.g., "there are likely ten people on the other side of the damaged bridge"). Given such information, human teams excel at collaborating and reasoning through language by decomposing, distributing and sequencing sub-tasks, which is a highly sought-after capability for teams of robots. The goal of this paper is to develop a single, light-weight model that enables distributed coordination and reasoning in teams of robots commanded through natural language specifications (Fig. 1).

A fundamental challenge in realizing such multi-robot autonomy is the gap between the *natural-language* expression of intent that human operators can convey, and its corresponding representation as an *algorithm* that robots can execute [5]. Early work involved extensive manual design for establishing one-to-one correspondences with algorithmic primitives [6], [7], and thus had limited reasoning capabilities. In contrast, language models today can side-step such design by encapsulating vast amounts of human language data into one model. Large multi-modal and language models (LMMs/LLMs) have recently shown excellent reasoning, disambiguation and even planning for single- and multi-robot tasks [8]–[10]. However, their computational demands still make them inaccessible to practical, deployable robot platforms. While some recent work has developed methods to effectively distill or extract the relevant capabilities from language models into more succinct representations [11]–[14], they still depend on a distillation phase that can be hard to adapt to real-time, dynamic and unforeseen settings.

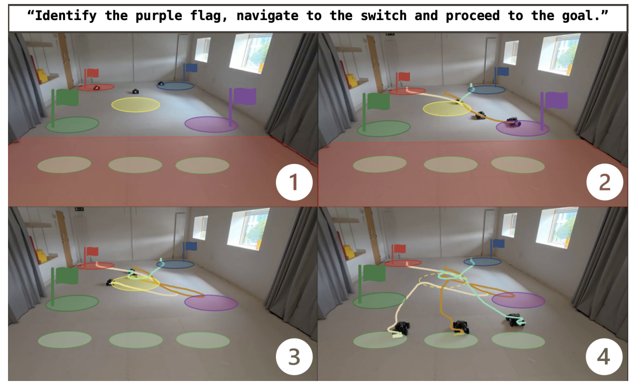


Fig. 1: A deployment of our method on RoboMasters [15]. The robots are tasked with identifying and retrieving the purple flag. Once the flag has been placed on the yellow switch, the access to the second part of the arena is activated, and the robots navigate to their respective final destinations.

In this work, we bridge the gap between high-level reasoning via natural language and its real-time distributed multi-robot implementation. Our approach leverages the observation that all robot tasks are *algorithms*: algorithms can be represented using automata, which can, in turn, be encoded by recurrent neural networks (RNNs). We train this RNN offline in a supervised fashion using task decompositions obtained from an LLM, distilling its reasoning capabilities and aligning its internal states with the tasks' semantics. This formulation encodes *multiple* automata into one RNN, encapsulating solutions to a variety of multi-robot tasks. The internal state of the RNN is used to condition a graph neural network-based policy trained using multi-agent reinforcement learning, achieving distributed collaboration. We evaluate via simulations and real experiments with multi-robot problems that require reasoning and sequential decision-making. Our results demonstrate a first in achieving onboard, real-time, decentralized multi-robot reasoning and collaboration—all from a natural language command.

II. Background

Early work on interpretable natural-language commands for robotics [6], [7], [16]–[18] focused on developing ontologies and corpora that had one-to-one correspondences with algorithmic primitives. However, these approaches require intense manual design and cannot accommodate ambiguous intent and reasoning. LLMs [19]–[21] and LMMs [22]–[24] overcome these limitations by encapsulating a comprehensive subset of human knowledge within a single data-driven model. Henceforth, they exhibit effective reasoning and flexibility to handle ambiguous yet expressive language prompts in both single- and multi-robot problems [8]–[10], [25]–[28]. For instance, LLMs can be used to enable algorithmic functions grounded in semantic information [29] or create complex formations [30] hard to specify in a metric space. Although effective, these works assume an infrastructure to connect the robots with the LLM/LMM in real-time.

To alleviate this, recent works [11]–[14], [31], [32] propose computationally efficient alternatives that rely on reasoning distillation and distributed coordination. Ravichandran et al. [11] distill a LLM into a smaller model that captures the relevant skills for robot task decomposition; however, the decomposition cannot integrate feedback in real-time. Meanwhile, solutions based on reinforcement learning allow onboard deployment but do not handle team-level prompting and credit assignment deconfliction [12]. Similar to them, our proposal distills LLMs into smaller models; nonetheless, we draw on the connections between automata theory and recurrent models [33] to achieve real-time reasoning. In this sense, LLMs have shown surprising skills in translating human commands into linear temporal logic [34], signal temporal logic [35] or planning domain definition language [36] formulas. However, formal-based approaches need (i) an initial time-consuming offline phase to compute the formula to be solved and (ii) low-level policies that actually *solve* the Partially Observable Markov Decision Processes (POMDP) corresponding to each sub-task. Indeed, the latter is often neglected but of key importance since it is in the process of solving the sub-tasks when real-time reasoning can handle unexpected phenomena. Our solution deals with both issues by learning to generate in real-time the automata associated to the tasks and, simultaneously, the low-level policy that solves each sub-task.

III. Preliminaries

In this work, a team of N robots is given a task as a natural-language instruction $\mathcal{W}$ (e.g., "Guide all people back to base"), which is typically a composition of some logically distinct 'sub-tasks', w (e.g., "search wide", "navigate group to base", "herd target" ...). The team has local connectivity, formalized as a time-varying, undirected graph $\mathcal{G}_k = (V, E_k)$, where $V = \{1, \dots, \mathsf{N}\}$ is the set of robots and $E_k \subseteq V \times V$ is the set of edges, such that $(r, r') \in E_k$ implies connectivity between robots r and r' at discrete time instant $k \in \mathbb{N}$. We define the set of neighbors of robot r at time k as $N_{r,k}$. The objective is to train control policies for the team such that the robots complete the task $\mathcal{W}$, relying only on local information available through $\mathcal{G}_k$. Solving $\mathcal{W}$ involves reasoning and solving sub-tasks sequentially, which we formalize as a Deterministic Finite Automata (DFA). To learn to solve each sub-task effectively, we assume that only a reward signal is available, which we formalize as a Decentralized POMDP (Dec-POMDP).

Deterministic Finite Automata. A DFA is a tuple $\mathcal{A} = (\mathcal{H}, \Sigma, \delta, h_0, \mathcal{F})$, where $\mathcal{H}$ is a finite set of sub-tasks to be solved; Σ is a finite alphabet of inputs, with each proposition $p \in \Sigma$ a boolean that describes an event that can trigger a transition between states; $\delta : \mathcal{H} \times \Sigma \to \mathcal{H}$ is a transition function such that $h_{k+1} = \delta(h_k, p_k)$, with $h_k, h_{k+1} \in \mathcal{H}$ and input $p_k \in \Sigma$; $h_0 \in \mathcal{H}$ is an initial state; and, $\mathcal{F} \subseteq \mathcal{H}$ is the set of accepting states modelling the completion of the task. Note that individuals in a multi-robot team completing the same task may step through $\mathcal{A}$ separately. We develop $\mathcal{A}$ as a computational representation of $\mathcal{W}$, and thus its states, $h \in \mathcal{H}$, are representations of sub-tasks w. The typical challenge is that $\mathcal{A}$ is unknown, and constructing it from $\mathcal{W}$ potentially involves deduction and logic reasoning. While the sub-tasks may be describable as Markov processes, switching between them requires memory—DFAs allow us to capture this algorithm.

Decentralized POMDP. Given a DFA, the team must also be able to collaborate and complete the sub-tasks within it. We define each sub-task $h \in \mathcal{H}$ as a Dec-POMDP $\mathcal{M}_h = (V, S, \{A_r\}_{r\in V}, T_h, R_h, \{\Omega_r\}_{r\in V}, O, b_h, \gamma_h)$, with a finite set of robot states S, and actions $\{A_r\}_{r\in V}$, such that $A = \times_{r\in V} A_r$ is the joint action space. Functions $T_h = P(s_{k+1} = s' | h, s_k = s, a_k = a)$ and $R_h : S \times A \to \mathbb{R}$ describe the dynamics and the reward of the sub-task h. Observations are described by the finite set $\{\Omega_r\}_{r\in V}$. Finally, the function $O = P(o_{k+1} = o | s_{k+1} = s', a_k = a)$ describes the measurement model of the team, b_h defines an initial probability distribution of the sub-task, and γ_h is a discount factor. In a Dec-POMDP, the goal is to find a policy π_h,

$$\pi_h = \arg\max_{\pi} \ \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma_h R_h | b_h, \pi\right],$$

such that the sub-task h is solved.

Problem formulation. Our goal is to find a *translation* of a language instruction into an automaton, and then learn a policy that *implements* the logic of the automaton. The policy must enable decentralized collaboration, and employ real-time feedback to compute appropriate control actions for a robot, solving each sub-task $\mathcal{M}_h$ needed to complete the task $\mathcal{A}$. Thus, the task is modeled as the set-product $\mathcal{A} \times \{\mathcal{M}_h\}_{h\in\mathcal{H}}$, leading to the following problem:

Problem 1. *Find a policy π that solves the task specified by natural-language specification $\mathcal{W}$ and modeled by the set-product $\mathcal{A} \times \{\mathcal{M}_h\}_{h\in\mathcal{H}}$, such that π relies only on the information available through $\mathcal{G}_k$ $\forall k$.*

We recognize (and indeed use) the fact that LLMs/LMMs today broadly possess the representational capacity to solve Problem 1 by, for instance, autoregressively generating action sequences for each robot at each time-step, adapting them with new information. This requires persistent low-latency connection to servers, which is wasteful, uneconomical and often infeasible. Our objective instead is to produce a small neural network π that implements a solution to $\mathcal{W}$ as a robust real-time feedback system easily distributed onto each robot.

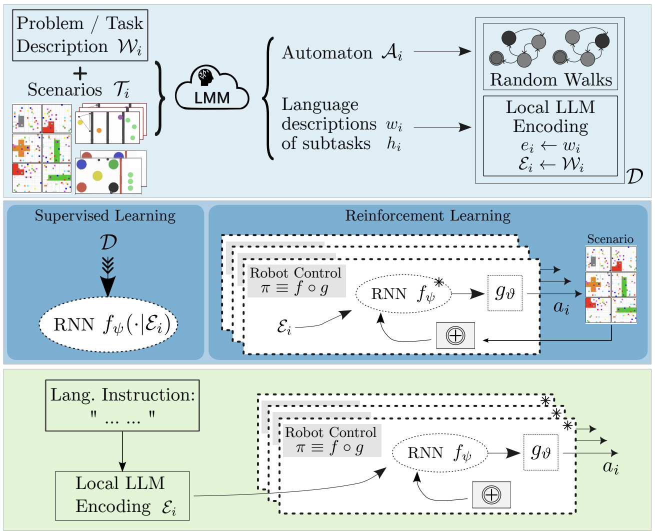


Fig. 2: A schematic overview of our framework. We begin with an offline generation phase that compiles a dataset $\mathcal{D}$, composed of sequences of sub-tasks and associated language embeddings for *different tasks*. This is done by prompting an LLM with diverse natural-language specifications covering all the tasks. Next, two offline training stages follow: (left) an RNN is trained with a supervised loss, using $\mathcal{D}$, to learn a set of semantic-aware automata; (right) a GNN-based policy is trained with an RL loss, using the frozen $(*)$ RNN locally on each robot, to learn a low-level policy that solves all sub-tasks. Finally, during online deployment, the robots only have to translate a language command into an embedding, and execute the RNN and GNN policy to complete the task.

IV. Automata-based Reasoning and Collaboration for Robot Teams

To solve Problem 1, we propose a novel methodology that combines *(i)* an automated dataset generation phase that queries an LLM with multi-robot tasks/problems, and compiles its logic and reasoning skills in solving them as state automata, *(ii)* a supervised learning phase that distills those capabilities into an RNN that encapsulates the desired automata, and, *(iii)* a reinforcement learning phase that learns decentralized control policies that solve the sub-tasks encoded in the automata. Fig. 2 depicts a schematic of the proposed methodology, which we now elaborate in the following three subsections.

A. Automatic generation of reasoning data

Our first step is to generate a dataset $\mathcal{D}$ that captures diverse tasks and their underlying logic. We will denote the set of tasks as $\mathcal{T}$, with $i \in \mathcal{T}$ a task. To generate $\mathcal{D}$, we need a representation that is conducive to learning a relational map between the language specification of a task and its computational representation as sub-tasks and events in the underlying automata. We clarify the different representations and their symbols in Table I.

TABLE I: Representations of task i and corresponding sub-tasks, where l indexes the sequence associated to a random walk.

Representation	Task	Sub-tasks	Format
Natural-Language Expression	$\mathcal{W}_i^l$	$w_{i,k}^l$	string
Automata Representation	$\mathcal{A}_i$	$h_{i,k}^l$	one-hot vector
Sentence Embedding	$\mathcal{E}_i^l$	$e_{i,k}^l$	N-bit vector
Atomic Propositions/Events	—	$p_{i,k}^l$	N-bit vector

For each task i, we first prompt an LLM with a task description. The LLM then generates a formal decomposition for each of these tasks as scripts or, e.g., LTL formulas. For each task in $\mathcal{T}$, we use its formal decomposition to perform random walks and collect L sequences of length K comprised of sub-tasks and the events that trigger transitions. Specifically, we collect one-hot encoded representations of the sub-tasks, $h_{i,k}^l$, their corresponding sentence embeddings, $e_{i,k}^l$, and the atomic propositions $p_{i,k}^l$. We also collect $\mathcal{E}_i^l$, the sentence embedding associated to the high-level natural-language command $\mathcal{W}_i^l$. The dataset is, therefore, a collection of successful steps that fulfill the tasks, defined as

$$\mathcal{D} = \{\{\mathcal{E}_i^l, \{h_{i,k}^l, e_{i,k}^l, p_{i,k}^l\}_{k=1}^{\mathsf{K}}\}_{l=1}^{\mathsf{L}}\}_{i\in\mathcal{T}}. \tag{1}$$

As shown in Fig. 2 (top), we automate the dataset generation by grounding the natural-language specifications $\mathcal{W}_i^l$ in the actual environment that will be used to train the policies. That is, instead of manually handcrafting each $\mathcal{W}_i^l$, we procedurally generate $\mathsf{L} \cdot \mathsf{M}$ scenarios of the tasks to be solved. We then feed their features, including visual information, to the LLM/LMM with a prompt that requests natural-language specifications $\mathcal{W}_i^l$ that match the scenarios. A typical prompt is of the following form:

```
You are leading a team of autonomous robots tasked with
exploring and finding {n_target} targets. The image below
represents a simplified map of the environment. It
highlights a {c_target} region of interest - this region does
not show the targets directly but suggests where they are
likely located and their color. The region covers ≈{size}%
of the map, i.e. it is {size_label}. Your mission:
- Describe the location of the region in relation to the
environment (e.g. top-left, south-east, center).
- Estimate its size (you already know it is {size_label}).
- Mention the number of targets.
- After identifying the targets, instruct the robots to
return to base.
Please respond in a {tone_style} tone. Be concise.
Additionally, generate the automaton representation of the
task provided the scenario event "found all the targets"
and the following DFA generator template:
# Python description of DFA generator
```

This not only automates the data generation step, but also grounds it in the actual environment that the robots will operate in. This avoids human bias and follows a more natural way of specifying a command, where an operator builds a general notion of the problem after observing a scenario and conveys it to the robot team. The dataset generation process is conducted offline and only once. We now detail the process by which a neural network can be trained to *distill* the task-relevant logic and reasoning contained in $\mathcal{D}$, so that it can interpolate and generalize across tasks and unseen scenarios.

B. Distilling reasoning with RNNs

Recent work has established a theoretical connection between automata and a second-order RNNs to prove that a network with sufficient capacity can represent any automaton [33]. This connection can also be applied to classical recurrent models like gated recurrent units, long short-term memory models or attention-based networks [14]. Consequently, we can train a sufficiently expressive recurrent

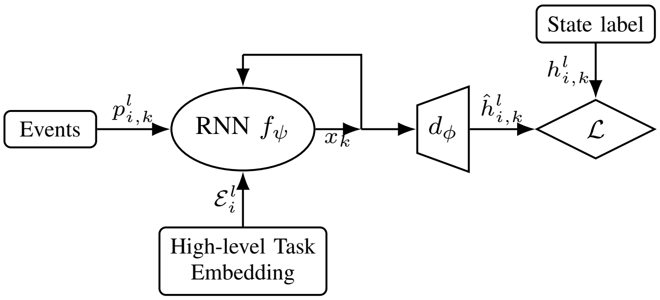


Fig. 3: Supervised training pipeline for the RNN task model.

model, in a supervised manner, to represent an automaton from $\mathcal{D}$—thus building a compact translation from a natural-language specification into an algorithm. This key insight underpins the model described in this section.

Fig. 3 illustrates the pipeline that trains an RNN f_ψ. Formally, given a dataset $\mathcal{D}$, we train the parameters ψ of the RNN that operates according to

$$x_k = f_\psi\big(x_{k-1}, p^l_{i,k} \bigm| \mathcal{E}^l_i\big), \tag{2}$$

where x_k is the internal state of the model at time k. The goal of training f_ψ is to encode the sub-tasks and transition logic of all automata $\{\mathcal{A}_i\}_{i\in\mathcal{T}}$ contained in $\mathcal{D}$. By including $\mathcal{E}^l_i$ at every step, the model is explicitly conditioned on the structure and constraints of the automaton of a given task, enabling it to interpret incoming events in the right context and guide state transitions accordingly. This conditioning also ensures that latent states corresponding to different tasks remain separated in the hidden space: even if two sequences from distinct tasks pass through identical event patterns, the task embedding $\mathcal{E}^l_i$ shifts their trajectories in the latent space so that x_k for one task cannot overlap with x'_k from another.

To learn ψ, a decoder projects x_k into a predicted one-hot state representation

$$\hat{h}^l_{i,k} = d_\phi\left(x_k\right), \tag{3}$$

which is trained to approximate the ground-truth label $h^l_{i,k}$. The decoder d_ϕ is implemented as a single-layer MLP, deliberately kept shallow to limit its capturing of sequential and semantic dynamics, while all temporal reasoning and semantic grounding remain encoded in the hidden dynamics of f_θ. Finally, the loss $\mathcal{L}(h^l_{i,k}, \hat{h}^l_{i,k})$ is a multi-class cross entropy loss penalizes incorrect state trajectories.

C. Decentralized collaboration with GNNs

Now that we have distilled the reasoning capabilities of LLMs into a single light-weight model, we turn to execution policies that solve the Dec-POMDPs associated with each of the sub-tasks. Recall that this step is especially relevant to grounding an automaton's logic to multi-robot interactions and transitions. Specifically, to solve Problem 1 we propose a policy π that is a composition of the RNN f_ψ and a language-conditioned multi-task policy g_ϑ,

$$\pi = f_\psi \circ g_\vartheta. \tag{4}$$

We model g_ϑ using reinforcement learning. Unlike existing multi-task policies that operate over entire episodes, we train π to act over short temporal segments corresponding to individual sub-tasks. At execution time, an appropriate sub-task policy is selected by conditioning on the hidden state of the RNN, which encodes the sub-task in the task automaton.

Formally, at time-step k, the action of a robot r is

$$a_{r,k} \sim g_\vartheta\big(a_{r,k} \bigm| \{o_{r',k}\}_{r',\in N_{r,k}},\ x_{r,k}\big), \tag{5}$$

where $x_{r,k}$ is the RNN's hidden state (i.e., an encoding of the current sub-task), and $\{o_{r',,k}\}_{r',\in N_{r,k}}$ is an observation vector from robot r and its neighbors at time k. We denote the hidden state with a subscript r because, for a decentralized implementation, each robot carries its own local copy of f_ψ. This not only enables decentralized execution, but also enables different robots to address different sub-tasks in parallel when needed, relying purely on their local information. On the other hand, the policy only depends on the local observation of robot r and information received through local communication. Specifically, g_ϑ is implemented as a GNN that processes the local observations o_r of the robot, together with the observations communicated by neighboring robots.

In learning ϑ, we take advantage of the dataset, where the RL scenarios are automatically paired with sentence embeddings. This lets us apply standard multi-agent reinforcement learning algorithms like Multi-Agent Proximity Policy Optimization [37] to train the multi-task policy, provided that every episode randomizes the sub-tasks and the reward signal $R_{i,h}$ accordingly. With a sufficiently expressive model g_ϑ we can capture the solution of all Dec-POMDPs $\{\mathcal{M}_{i,h}\}_{i\in\mathcal{T}}$; by restricting training to individual sub-tasks, the learning process becomes more efficient and substantially reduces the computational load. High-level decision-making is delegated entirely to the RNN, allowing the multi-task policy to focus on low-level execution in the context of its current sub-task.

Lastly, o_r contains information from sub-task related measurements (e.g., position of obstacles, velocity of targets) and event information related to the atomic propositions of the automata (e.g., a switch turns on, a flag is captured). The latter can be shared among neighbors to improve the reactivity of the local RNN at each robot and the efficiency of the team overall. We therefore modify the input to the RNN from the observed $p^l_{i,k}$ to an aggregated $\bar{p}^l_{i,k}$ that includes information from neighbors as well. The vector $\bar{p}^l_{i,k}$ is obtained from a general aggregation operation $\bigoplus$ (see Fig. 2 (middle and bottom)) that can be implemented in multiple ways. In the most general case, $\bigoplus$ can be learned, for instance, as a GNN. For some use-cases, it can be reduced to simple boolean logic such as `AND` or `XOR`; or it can even be extracted automatically from the language command, although we leave this option as future work.

V. Evaluations

The framework presented here can generate multiple automata for various tasks, and encode them into a single RNN that captures the logic and state-flow. Additionally, the final policy is trained to execute in real-time, with robust feedback control for each robot in a decentralized manner. The product is a complete pipeline that accepts as input a task

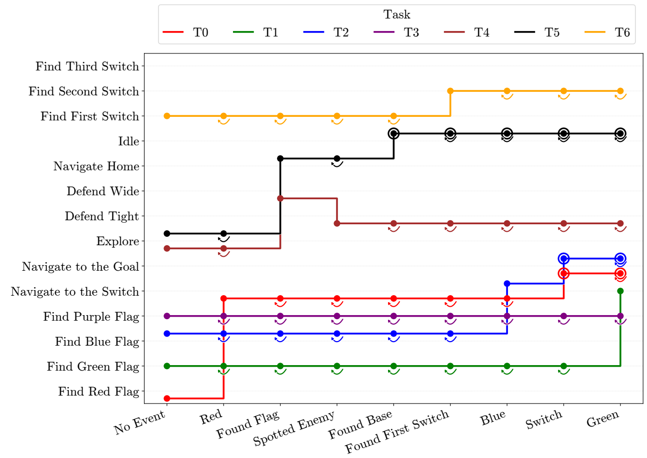


Fig. 4: Rollouts of the same RNN on seven different tasks ($\mathbf{T}\{0..6\}$) over the available sub-tasks (y-axis) with relevant/irrelevant events presented to it (x-axis). Training and test achieve 100% of accuracy. Absorbing states are denoted by ◉ . The model exhibits correct transitions while rejecting events that belong to a potentially different automaton. For instance, **T1** requires the team to *"Search for the green flag, then the switch, and navigate to the goal"*, and events such as "spotted enemy" have no impact on the model. On the other hand, the same event in **T4**, *"Locate the mission flag and defend the position; adapt your defense."*, causes the team to change its formation from 'wide' to 'tight'. Other tasks are listed on our website.

as a natural-language command, and causes the robot team to complete the task. We now evaluate our framework for *(i)* its correctness in representing the various task automata, and, *(ii)* its ability to complete a given multi-robot task, even when presented with disruptions, potentially long/repeated sub-task sequences, and real-world noise. Note that, as a key characteristic of our method, we use the *same* RNN model to encode all the tasks presented here.

A. Automata Representation

The first requirement from our framework is its ability to correctly encode various different task automata. This means that given a language input, $\mathcal{W}_i$, the model must get initialized into the correct state, $h_{i,0}$, of the corresponding automata $\mathcal{A}_i$. Subsequent state transitions at each step k must follow the logic of the task, and never produce a state $h_{i',k}$ in a task $i \neq i'$ even when presented with an event from i'.

Fig. 4 shows an evaluation sequence of the model for seven different tasks, $\mathbf{T}\{0..6\}$, and their sub-tasks listed on the vertical axis. The illustration shows how the model steps through the automata pool given the various states/events observed on the horizontal axis. For instance, in **T2**: "Find the blue flag, spot the switch, and head for the target", the sub-task is 'Find blue flag' and the RNN remains in this state until the observation 'Blue'. At this point it transitions to 'Navigate to the switch', and remains in this state until the observation 'Switch', where it transitions into 'Navigate to Goal', which is an absorbing state in this automaton. We observe that for each task, the initialization of the correct sub-task follows the semantics of the task, i.e., given "No Event", the first sub-task always corresponds to the relevant one according to $\mathcal{W}_i$.

We also observe that the rollout for each task follows a logically correct and semantically relevant sequence of sub-tasks. In other words, the RNN transitions when a relevant event occurs, and is able to reject events and observations that do not have any meaning in a given automaton. For instance, events such as 'Found Base' and 'Red' only produce self-loops in **T2**. The RNN model has essentially captured the reasoning and decision-making necessary for completing a given task, which is a key skill we distilled from the language model. Indeed, our supervised train and test accuracy is a $100\,\%$, observed with 500 sequences per task with an 80-20 split. Note that the sequence of events along the horizontal axis has no particular order, and can continue indefinitely. We list the full set of tasks and events on the website.

B. Language Initialization of Tasks

We now focus on evaluating the advantage offered by a language-based initialization in a complex task that requires team cooperation in addition to a correctly sequenced automaton. The scenario we use is shown in Fig. 5(left), where a team of three robots is initialized at random locations in an arena that is sub-divided into four 'rooms'. Each of the leftmost three rooms has a switch that a robot needs to hit to open the rightmost wall, so that it can navigate to the adjacent room on the right and, ultimately, towards its corresponding goal. The team receives as language input not only the natural language description of the overall task, but we can also initialize each agent at a specific state of the RNN by providing a natural language description of its current sub-task. For example, the task may be stated as: "Agents, unlock the first, second, and third switches in sequence, then advance to the objective room and reach the target." An individual agent, however, may be further initialized with a sub-task such as: "Head to the second trigger in the second room from the left."

In Fig. 5 (middle), we see a comparison (averaged over 500 runs) of our method against two candidate baselines: a vanilla RL approach that rewards the team when navigating towards the goals, and a reward-tuning RL approach, where we manually hand-craft a multi-stage reward so that each robot is first rewarded for hitting the switch and then for navigating to the adjacent room. Unsurprisingly, the vanilla RL performs well only in hitting the very first switch, after which, the successive rooms are never unlocked. The reward-tuned RL method performs much better, since the rewards are decomposed according to the stages in the task. This decomposition is manual, and thus laborious, error-prone, and infeasible to scale and apply different scenarios. Our proposed method sidesteps this exact problem by encoding the appropriate sub-tasks in the RNN, and delegating the sub-task switching to it. In other words, the automaton (RNN) handles the room and switch transition, which makes it easy for an RL policy to then complete only the relevant sub-task.

The proposed method also enables robots to be initialized non-uniformly, both physically as well as in their initial RNN states through natural language. This can be seen in the three samples shown in Fig. 5(left), where we allow robots to be initialized in same/different rooms. We show quantitative analysis in Fig. 5(right) by initializing robots in all six

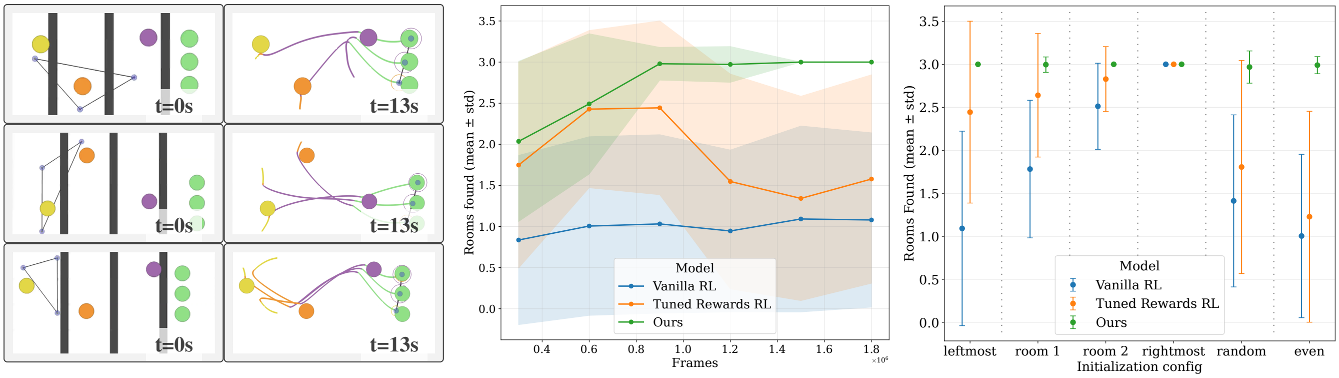


Fig. 5: Analysis of the impact of reasoning and language initialization in a four-room scenario. Three robots are placed in any of the three leftmost rooms, which are divided by solid black walls. Each room has a differently coloured switch (• • •), and the robots need to hit them in order to open a passage and navigate to their goals (•). (left) Snapshots of different robot and RNN initializations, showing that our solution successfully accomplishes the task irrespective of the configuration of robots, switches and goals. Traces are coloured according to the sub-tasks deducted by their local RNNs. (middle) Training curves for the three benchmarked methods: a vanilla RL policy, a reward-tuning RL policy, and our policy; success is measured as the average number of rooms found by the team of robots in 500 episodes. Our policy consistently outperforms the other methods by achieving perfect task completion at the end of the training. (right) After training, we evaluate all six possible initial configurations with the three methods.

unique configurations for each method, evaluated over 500 runs. Clearly, when the team is initialized in the rightmost room, all methods perform equally well ($= 100\,\%$), since the only sub-task then is to navigate to their respective goals. In the same vein, initialization in 'room-2' is only slightly worse since it adds only one additional sub-task. For all other initializations, only our language-driven RNN method succeeds in completing the entire task reliably, especially in the hardest configuration where the robots are spread evenly (one in each room). For evaluation, we report the best policy obtained in 1.6×10^6 frames, which suffices for our method to reach near-perfect accuracy. Although baseline, language-free, RL methods eventually learn under predictable initializations, their catastrophic failures under "random" and "even" initializations persist, as these require asymmetric behaviors that are considerably harder to learn.

This success notwithstanding, our method does exhibit some susceptibility in the 'random' and 'even' configurations. We attribute this to noise and ambiguity in the language initialization step, such that a sufficiently incorrect expression (with very little correspondence to the task) can produce an incorrect mapping into an automaton's state. An example from our evaluations is, "*Exploration drones, activate the three room switches from left to right; then move into room four and finalize at the goal*" with a sub-task "*Units, make your way smoothly to the second release plate in the room after the left-most bay near the middle divider gate; signal ready when the ring glows*". We provide more details for such cases on the website.

C. Long Sequences, Disruptions and Zero-shot Deployment

Finally, we evaluate our framework in deploying it on a team of robots in simulations and in real-world. Our objective here is to demonstrate the robustness of the full pipeline against disturbances that are modeled (i.e., events that may naturally occur during the course of a mission) as well as unmodelled external noise (such as what we expect in typical real-world robot deployments). Note that

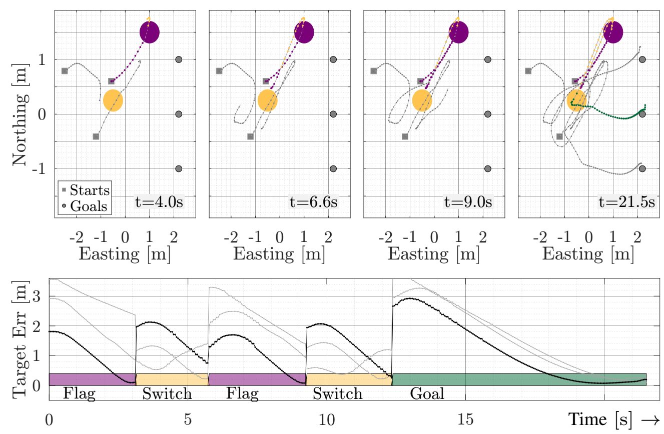
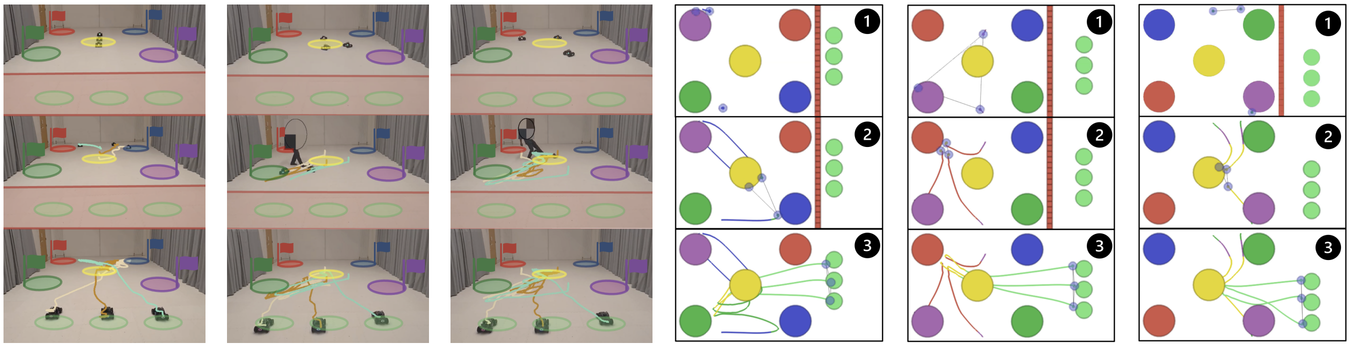


Fig. 6: Evolution of team trajectories from an instance of 'retrieve-the-flag' scenario (Fig. 1), with the team initialized at random locations and assigned fixed goals. The task requires them to "*Find a purple flag, and bring it to the switch*", which then permits to cross to their goal locations. **(top)** Four snapshots of a top-down view at different times show the spatial evolution of the task. **(bottom)** A temporal view of the task shown as a function of robot distances to their current targets (Flag, Switch or Goal). The task faces a disruption ('Flag lost') at $\approx 6\,$s before they reach the switch, causing the team to double back to the flag location a second time.

the former is possible primarily due to the automata we encode and their ability to model explicit state machines; we covered this in Section V-A. The latter is obtained during the RL policy training step. However, it is the combination of the two that makes for a powerful recipe—it bridges the gap between language-driven reasoning and its real-time distributed implementation as a feedback control.

Fig. 1 already shows four snapshots from a zero-shot physical deployment using a team of three RoboMaster ground robots [15] for the task, *"Identify the purple flag, navigate to the switch and proceed to the goal"*. In Fig. 6, we additionally present spatial (top-down) and temporal trajectories for the team executing a different run of this task. However, this time we emulate an additional event, 'Flag lost', moments after the flag was found ($\approx 6\,$s), which causes the team to double back and recover the flag before heading to the switch a second time. Fig. 6 (top) codes the trajectory



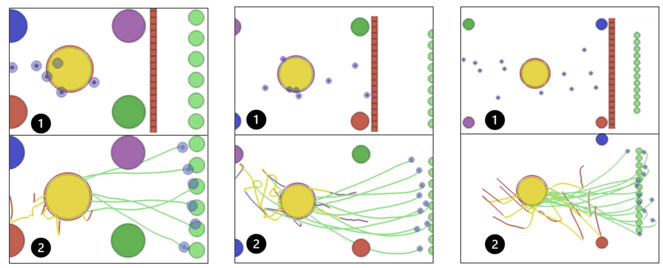
(a) "Search for the blue flag, then the switch, then head to the goal room."
(b) "First search for the green flag, then the actuator, then proceed to goal."
(c) Same as (b), with major disruptions.
(d) "Head for the blue flag, then the green flag, and finish at the target."
(e) "Locate the purple flag, then the red flag, and proceed to the goal."
(f) "Find the green flag, identify the purple one, then reach the goal."

Fig. 7: Examples of zero-shot multi-robot deployments in the retrieve-the-flag scenario, on physical robots (**a-c**) and in simulations (**d-f**). Each task requires the team to locate the specified flag, and then 'bring' it to the switch (yellow) which opens a virtual gate separating their goal region. The team can handle a variety of natural-language expressions, objectives and even unmodeled disturbances (a person interferes with the team in (b) and (c), middle panel). Simulated examples add more complexity: instead of locating a single flag, the agents must find and retrieve a series of flags in a given sequence.

of one of the agents with the color of team's current target (the other two are shown in light gray). We observe the behavior switch in the form of two 'loops' that connect the purple flag and the yellow switch, corresponding to the two trips the team has to make towards the flag. The temporal view in Fig. 6 (bottom) sheds more light on the evolution of the team's trajectory towards the different targets. The robots start by heading towards the purple flag; upon obtaining it ($\approx 3\,\mathrm{s}$), they change their target to the yellow switch. At the emulated 'Flag lost' event ($\approx 6\,\mathrm{s}$), the state machine resets to heading to the purple flag. The flag is found again ($\approx 9\,\mathrm{s}$), and the team continues to the switch and then finally make their way to the goal.

In Fig. 7, we show snapshots from six additional evaluations on other language instructions. We showcase variations on the task, and in particular, with the deployments on the physical robot team, we also test the robustness of the policies when faced with external (unmodeled) disturbances. For instance, Fig. 7(b,c) show a person (masked for anonymity) manually displacing random robots in the team during different sub-tasks. Note that this disruption might happen in addition to the 'Flag lost' event. The team gracefully recovers from such events, thanks to both the robustness of the control policies and the automaton adhering to a global task sequence. Fig. 7(d–f) show evaluations where the task requires them to follow a longer sequence. Similar to the multi-room problem in Section V-B, our method is successful due to the sub-task sequencing enforced by the corresponding task automaton.

Finally, we evaluate how well the framework scales to larger teams. Because the policy uses a graph-based architecture, it generalizes naturally to variable team sizes. Fig. 8 shows snapshots from the same task in deployments with 6, 9, and 12 agents in simulation, using a policy that was trained with 3 agents. We observe that the team completes the tasks successfully, while the inference time (per-loop) remains fairly manageable and well within realistic constraints for feedback control (under $1\,\mathrm{ms}$/loop/robot).



(a) N = 6, Inference: 2.17 ms.
(b) N = 9, Inference: 3.24 ms.
(c) N = 12, Inference: 9.37 ms.

Fig. 8: Retrieve-the-flag scenario. The policy is trained with 3 agents but deployed with 6, 9, and 12 agents in simulation, while maintaining real-time inference (measured on a MacBook Air M3). The task is completed successfully, though the team performance drops gradually at larger disparities.

VI. Conclusions

In this work, we presented a novel framework that enables teams of robots to perform complex, collaborative tasks from high-level natural language commands. Our approach successfully addresses the challenges of real-time reasoning, decentralized coordination, and the interpretation of ambiguous yet expressive semantic instructions in natural language. By leveraging the connection between automata theory and RNNs, we effectively distill the reasoning capabilities of a large language model into a single, light-weight RNN. This model, paired with a graph neural network-based policy, allows robots to reason about task sequences and collaborate on sub-tasks in a fully distributed manner. Simulations as well as real-world experiments demonstrated that our method empowers robot teams to display interactive, onboard reasoning, effectively leveraging semantic cues from human language to achieve their goals. We plan to extend this work to teams with arbitrary robot capabilities. Furthermore, we are interested in integrating off-the-shelf logic checkers and then addressing more nuanced and unexpected state machines for various physical tasks.

References

[1] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon Multi-robot Rearrangement Planning for Construction

Assembly," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, 2022.

[2] V. Edwards, T. C. Silva, B. Mehta, J. Dhanoa, and M. A. Hsieh, "On Collaborative Robot Teams for Environmental Monitoring: A Macroscopic Ensemble Approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2023, pp. 11 148–11 153.

[3] S. Kim, M. Corah, J. Keller, G. Best, and S. Scherer, "Multi-robot Multi-room Exploration with Geometric Cue Extraction and Circular Decomposition," *IEEE Robotics and Automation Letters*, vol. 9, no. 2, pp. 1190–1197, 2023.

[4] J. Liu, P. Li, Y. Wu, G. S. Sukhatme, V. Kumar, and L. Zhou, "Multi-robot Target Tracking with Sensing and Communication Danger Zones," *arXiv preprint arXiv:2404.07880*, 2024.

[5] V. Cohen, J. X. Liu, R. Mooney, S. Tellex, and D. Watkins, "A survey of robotic language grounding: tradeoffs between symbols and embeddings," in *International Joint Conference on Artificial Intelligence*, 2024, pp. 7999–8009.

[6] G. Bugmann, E. Klein, S. Lauria, and T. Kyriacou, "Corpus-based Robotics: A Route Instruction Example," in *Proceedings of Intelligent Autonomous Systems*, 2004, pp. 96–103.

[7] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward Understanding Natural Language Directions," in *ACM/IEEE International Conference on Human-Robot Interaction*, 2010, pp. 259–266.

[8] H. Biggie, A. N. Mopidevi, D. Woods, and C. Heckman, "Tell Me Where to Go: A Composable Framework for Context-Aware Embodied Robot Navigation," *arXiv preprint arXiv:2306.09523*, 2023.

[9] H. Li, H. N. Mahjoub, B. Chalaki, V. Tadiparthi, K. Lee, E. Moradi-Pari, C. M. Lewis, and K. P. Sycara, "Language Grounded Multi-agent Reinforcement Learning with Human-interpretable Communication," *arXiv preprint arXiv:2409.17348*, 2024.

[10] K. Garg, S. Zhang, J. Arkin, and C. Fan, "Foundation Models to the Rescue: Deadlock Resolution in Connected Multi-Robot Systems," *arXiv preprint arXiv:2404.06413*, 2024.

[11] Z. Ravichandran, I. Hounie, F. Cladera, A. Ribeiro, G. J. Pappas, and V. Kumar, "Distilling On-device Language Models for Robot Planning with Minimal Human Intervention," *arXiv preprint arXiv:2506.17486*, 2025.

[12] S. Morad, A. Shankar, J. Blumenkamp, and A. Prorok, "Language-Conditioned Offline RL for Multi-Robot Navigation," *arXiv preprint arXiv:2407.20164*, 2024.

[13] Y. Qu, A. Singh, Y. Lee, A. Setlur, R. Salakhutdinov, C. Finn, and A. Kumar, "Learning to Discover Abstractions for LLM Reasoning," in *Workshop on Programmatic Representations for Agent Learning, International Conference on Machine Learning*, 2025.

[14] W. Zhan, Q. Dong, E. Sebastián, and N. Atanasov, "LATMOS: Latent Automaton Task Model from Observation Sequences," *arXiv preprint arXiv:2503.08090*, 2025.

[15] J. Blumenkamp, A. Shankar, M. Bettini, J. Bird, and A. Prorok, "The Cambridge RoboMaster: An Agile Multi-Robot Research Platform," *arXiv preprint arXiv:2405.02198*, 2024.

[16] M. Levit and D. Roy, "Interpretation of Spatial Language in a Map Navigation Task," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 3, pp. 667–679, 2007.

[17] M. MacMahon, B. Stankiewicz, and B. Kuipers, "Walk the talk: connecting language, knowledge, and action in route instructions," in *National Conference on Artificial Intelligence*, vol. 2, 2006, pp. 1475–1482.

[18] T. M. Howard, E. Stump, J. Fink, J. Arkin, R. Paul, D. Park, S. Roy, D. Barber, R. Bendell, K. Schmeckpeper, J. Tian, J. Oh, M. Wigness, L. Quang, B. Rothrock, J. Nash, M. R. Walter, F. Jentsch, and N. Roy, "An Intelligence Architecture for Grounded Language Communication with Field Robots," *Field Robotics*, vol. 2, pp. 468–512, 2022.

[19] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[20] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, "Tidybot: Personalized Robot Assistance with Large Language Models," *Autonomous Robots*, vol. 47, no. 8, pp. 1087–1102, 2023.

[21] Z. Ravichandran, V. Murali, M. Tzes, G. J. Pappas, and V. Kumar, "SPINE: Online Semantic Planning for Missions with Incomplete Natural Language Specifications in Unstructured Environments," *arXiv preprint arXiv:2410.03035*, 2025.

[22] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, *et al.*, "Open X-embodiment: Robotic Learning Datasets and RT-X Models: Open x-embodiment Collaboration 0," in *IEEE International Conference on Robotics and Automation*, 2024, pp. 6892–6903.

[23] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, *et al.*, "Gemini Robotics: Bringing AI into the Physical World," *arXiv preprint arXiv:2503.20020*, 2025.

[24] B. Yu, Q. Yuan, K. Li, H. Kasaei, and M. Cao, "Co-NavGPT: Multi-Robot Cooperative Visual Semantic Navigation Using Vision Language Models," *arXiv preprint arXiv:2310.07937*, 2025.

[25] A. Rajvanshi, K. Sikka, X. Lin, B. Lee, H.-P. Chiu, and A. Velasquez, "SayNav: Grounding Large Language Models for Dynamic Planning to Navigation in New Environments," *arXiv preprint arXiv:2309.04077*, 2024.

[26] S. Liao, X. Lv, Y. Cao, J. Lew, W. Wu, and G. Sartoretti, "HELM: Human-Preferred Exploration with Language Models," *arXiv preprint arXiv:2503.07006*, 2025.

[27] Y. Qu, B. Wang, Y. Jiang, J. Shao, Y. Mao, C. Wang, C. Liu, and X. Ji, "Choices are More Important than Efforts: LLM Enables Efficient Multi-Agent Exploration," *arXiv preprint arXiv:2410.02511*, 2024.

[28] Y. Han, M. Yang, Y. Ren, and W. Li, "Large Language Model Guided Reinforcement Learning Based Six-Degree-of-Freedom Flight Control," *IEEE Access*, vol. 12, pp. 89 479–89 492, 2024.

[29] F. Cladera, Z. Ravichandran, J. Hughes, V. Murali, C. Nieto-Granda, M. A. Hsieh, G. J. Pappas, C. J. Taylor, and V. Kumar, "Air-Ground Collaboration for Language-Specified Missions in Unknown Environments," *arXiv preprint arXiv:2505.09108*, 2025.

[30] V. L. N. Venkatesh and B.-C. Min, "ZeroCAP: Zero-Shot Multi-Robot Context Aware Pattern Formation via Large Language Models," *arXiv preprint arXiv:2404.02318*, 2025.

[31] I. Nematollahi, B. DeMoss, A. L. Chandra, N. Hawes, W. Burgard, and I. Posner, "LUMOS: Language-Conditioned Imitation Learning with World Models," *arXiv preprint arXiv:2503.10370*, 2025, arXiv:2503.10370.

[32] T. Godfrey, W. Hunt, and M. D. Soorati, "MARLIN: Multi-Agent Reinforcement Learning Guided by Language-Based Inter-Robot Negotiation," *arXiv preprint arXiv:2410.14383*, 2025.

[33] T. Li, D. Precup, and G. Rabusseau, "Connecting weighted automata, tensor networks and recurrent neural networks through spectral learning," *Machine Learning*, vol. 113, no. 5, pp. 2619–2653, 2024.

[34] Z. Dai, A. Asgharivaskasi, T. Duong, S. Lin, M.-E. Tzes, G. Pappas, and N. Atanasov, "Optimal Scene Graph Planning with Large Language Model Guidance," in *IEEE International Conference on Robotics and Automation*, 2024, pp. 14 062–14 069.

[35] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "AutoTAMP: Autoregressive task and motion planning with llms as translators and checkers," in *IEEE International Conference on Robotics and Automation*, 2024, pp. 6695–6702.

[36] J. Strader, A. Ray, J. Arkin, M. B. Peterson, Y. Chang, N. Hughes, C. Bradley, Y. X. Jia, C. Nieto-Granda, R. Talak, C. Fan, L. Carlone, J. P. How, and N. Roy, "Language-Grounded Hierarchical Planning and Execution with Multi-Robot 3D Scene Graphs," *arXiv preprint arXiv:2506.07454*, 2025.

[37] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.