ETH zürich

PROROKLAB

UNIVERSITY OF CAMBRIDGE

Prof. Amanda Prorok

Master Thesis

Multi-robot Coordination and Reasoning through Natural Language

Spring Term 2025

Supervised by:
Dr. Eduardo Sebastián
Dr. Ajay Shankar

Author:
Nicolas Pfitzer

Preface

This thesis was conducted as part of my Master's studies at ETH Zürich and carried out during a research stay at the ProrokLab, University of Cambridge, under the supervision of Prof. Amanda Prorok.

I would like to express my sincere gratitude to Dr. Eduardo Sebastián and Dr. Ajay Shankar (ProrokLab, University of Cambridge) for their close mentorship, guidance, and encouragement throughout this project. I also thank Prof. Stelian Coros (ETH Zürich) for enabling this collaboration between ETH Zürich and the University of Cambridge.

Finally, I am deeply grateful to Prof. Amanda Prorok for welcoming me into the lab and involving me in her research during my stay.

Contents

Abstract

Humans communicate instructions most naturally through language, yet robot teams typically require structured algorithmic specifications to act. This thesis investigates how natural language can serve as a direct, lightweight, and expressive control interface for decentralized multi-robot systems. We first demonstrate that sentence embeddings provide an effective high-level bias for reinforcement learning policies, enabling teams of robots to execute shared instructions and coordinate exploration without a central planner. Building on this foundation, we address tasks that demand temporal reasoning and structured progression. We propose a hybrid framework that translates language into deterministic finite automata and distills their logic into compact recurrent models, which condition multi-agent policies in real time. This approach allows robots to reason over contingencies, dependencies, and multi-stage tasks while maintaining decentralization and on-device efficiency. By combining semantic grounding in language with automata-based reasoning, we provide a unified and deployable framework that narrows the gap between human communication and coordinated robot team execution.

Symbols

Symbols

π	Policy
a	Action
r/R	Reward function
θ, ϕ, ψ, ξ	learnable parameters
o	Observation
s	Natural language instruction
g	Sentence embedding
h	RNN latent vector
c	CNN latent vector
z	GNN latent vector
p	Position
v	Velocity
i	Agent index
k	Discrete time step
d	Sentence embedding dimension
I	Set of agents
S	Set of robot states
A_i	Action space of agent i
T	Transition function
S	Set of robot states
A_i	Action space of agent i
O_i	Observation space of agent i
γ	Discount factor
$\mathcal{G}$	Proximity Graph
E	Sentence encoder
$\mathcal{F}_\theta$	RNN Automaton Model
$\mathcal{D}_\phi$	Sentence embedding decoder
$\mathcal{D}_\psi$	RNN latent decoder
$\mathcal{G}_\xi$	Team-event GNN
L_{BCE}	Binary Cross Entropy Loss
L_{CE}	Cross Entropy Loss

Acronyms and Abbreviations

RL	Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
LLM	Large Language Model
VLM	Vision Language Model
NLP	Natural Language Processing
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GNN	Graph Neural Network
GAT	Graph Attention Network
ReLU	Rectified Linear Unit
LTL	Linear Temporal Logic
STL	Signal Temporal Logic
PDDL	Planning Domain Definition Language
DFA	Deterministic Finite Automaton
ROI	Region Of Interest

Chapter 1

Introduction

1.1 Motivation

Teams of robots are well-suited for tasks where distributed effort, redundancy, and parallelization bring major benefits [1–4], ranging from search and rescue and environmental monitoring to industrial inspection and transportation. In such scenarios, robots typically face tight energy budgets, limited sensor ranges, and uncertain localization, making exhaustive, unguided missions long and costly. Moreover, real-world deployments are rarely predictable: noise, unexpected changes in the environment, and layers of complexity make it impractical to train and deploy policies that account for every possible scenario. Yet in many missions, human operators or upstream systems might possess partial but valuable knowledge. Such higher-level insights might include rough GPS fixes from emergency calls, semantic knowledge from previous experiences, operator intuition about likely target sites, or domain heuristics (e.g., "cracks often start near weld seams on the stern of the ship"). These insights can help condition the robot team's behavior by narrowing the operational space, focusing attention on specific regions, or injecting priors that would otherwise be inaccessible. They are typically qualitative and easier to convey in natural language than through formal constraints or hard-coded programming.

Autonomous and collaborative robot exploration exemplifies these challenges: robot teams must communicate and reason about where to move next in order to efficiently reveal unknown parts of an environment or locate specific targets. In real-world deployments, simply maximizing information gain is not enough. Robots must also account for limited energy, communication constraints, and safety risks, while incorporating high-level goals and soft heuristics provided by human supervisors. Crucially, these human operators would rarely interact with the entire team through technical, low-level commands or numerical specifications. Instead, expressing intent and partial knowledge of the environment in natural, intuitive ways, much like they would when coordinating with other people, would reduce the interaction burden. Traditional methods that encode guidance as fixed way-points, constraint maps, or cost functions are not only brittle and hard to scale, but also incompatible with the way humans naturally communicate.

This thesis explores coordinating robot team behaviors through natural-language inputs. Language is inherently expressive, intuitive, and compositional. It allows

humans to specify approximate goals (e.g., "explore the far side of the river"), inject domain heuristics ("targets are likely near the power plant"), or suggest constraints ("avoid entering flooded zones"). Recent work [5] has shown that sentence embeddings from sentence transformers make it feasible to project natural language instructions into semantic vector spaces that can guide control policies, and in this work, we show that in decentralized multi-agent settings, where no central planner coordinates robot actions, such language-conditioned policies can serve as a powerful interface between high-level intent and robot teams.

Yet robot tasks often go beyond simple semantic constraints such as those described above. Consider the following scenario:

"Agents must survey the southern sector for survivors. There may be people trapped near the collapsed tunnel. Robots should avoid the chemical spill on the eastern side. The road south might be blocked, in which case the team will need to clear out a path."

Such instructions require robots not only to understand spatial and temporal priorities, but also to reason about conditional actions, inter-agent dependencies, and dynamic environmental factors. Humans naturally do this by parsing language, identifying dependencies (e.g., "check for debris before crossing the road"), and decomposing tasks into subgoals. For robot teams to match this flexibility, they need lightweight but expressive models that can emulate such reasoning within a decentralized execution model. In this way, the thesis traces a path from language as bias for control to language as a medium for reasoning and coordination.

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 [6]. Early work on developing interpretable natural-language commands for robots involved extensive manual design for establishing one-to-one correspondences with algorithmic primitives [7, 8], 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 multimodal and language models (LMMs/LLMs) have recently shown excellent reasoning, disambiguation and even planning for single- and multi-robot tasks [9–11], demonstrating that they posses sufficient expressive power to solve typical multi-robot problems. 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 [5, 12–14], they still depend on a distillation phase that can be hard to adapt to real-time, dynamic and unforeseen settings.

1.2 Contributions

Bridging the gap between the expressiveness of natural language and the structured representations required for robot execution requires approaches that retain the expressive benefits of natural language while remaining lightweight, adaptable, and compatible with decentralized multi-robot execution. This thesis addresses that need by exploring how compact, semantically meaningful representations of language can guide robot team behaviors, first in simple tasks that involve a single step or stage, and then in progressively richer scenarios that involve multiple steps

and demand reasoning and coordination. In doing so, we aim to move closer to deployable systems that combine the intuitiveness of human communication with the efficiency and robustness of decentralized robot control.

Our first contribution focuses on demonstrating that natural language serves as an effective high-level control modality in tasks where a robot team can act directly on an instruction without needing to reason over internal task decomposition or temporal sequences. We train reinforcement learning (RL) policies conditioned on sentence embeddings derived from Sentence Transformer models or sentence embedding LLMs. We aim to learn a single shared policy $\pi_\theta(a_i \mid o_i, g)$, which each agent executes independently using its own local observation o_i and the shared sentence embedding g. That is, each agent samples actions from:

$$a_i \sim \pi_\theta(a_i \mid o_i, g)$$

The policy π with learnable parameters θ is shared across all agents and is trained to maximize the team's cumulative reward in a decentralized, partially observable setting.

We focus on decentralized multi-robot exploration in unknown environments as the target domain for this formulation. Exploration is chosen because it lends itself naturally to language based control. Unlike temporally extended tasks that require explicit sequencing or sub-goal planning, exploration can often be directed by abstract intent ("search the left wing," "look for 4 green targets," "the targets are tightly grouped together")–all of which are easily and compactly expressed in natural language. Moreover, the uncertainty inherent in unfamiliar environments makes it difficult to define reliable scalar rewards or hard coded constraints. Language, by contrast, allows human operators to flexibly express intent or environmental priors.

We evaluate how well the robots interpret various types of prompts, showing that language enables more flexible, efficient, and semantically aligned behavior. Our goal is to understand how natural language can be used to coordinate multi-agent exploration and search at the team level. Specifically, we investigate how LLMs can enable distributed teams to interpret and execute shared instructions, fostering emergent coordination strategies. We show that language-conditioned exploration policies can incorporate high-level human insights to significantly reduce exploration effort without over-constraining autonomy. We compare the results against the same policy without language input and a baseline trained without any language conditioning.

While these single-stage tasks demonstrate the power of language-conditioned policies for exploration, the second part of this thesis builds upon the framework and addresses tasks that require temporal reasoning, instruction decomposition, and event-driven transitions. These are referred to as multi-stage tasks, where successful execution depends not only on interpreting a high-level instruction but also on following its logical progression across time. To address this, we develop a hybrid framework that combines automata theory with recurrent neural networks (RNNs) to model and execute structured temporal logic in a decentralized multi-robot setting.

Our core insight is that all such robot tasks can be described as algorithms that map naturally onto automata. Specifically, any semantically structured instruction can be translated into a deterministic finite automaton (DFA), which represents the sequence of sub-tasks and transitions required for execution. We then encode this automaton in the internal state dynamics of an RNN, trained to model all

valid transitions between sub-tasks. Because RNNs have the capacity to maintain a latent representation of the current subtask and its dependencies, they serve as lightweight, expressive mechanisms for capturing the state logic of task execution.

To make this practical for decentralized control, we train the RNN offline in a supervised manner using task decompositions automatically generated by a large language model. This distills the reasoning capabilities of the LLM into a compact model that can be deployed efficiently on-device. The RNN is conditioned on the same sentence embeddings used in the single-stage setting, grounding the reasoning process in a shared semantic space. As a result, the interface between instruction parsing and execution remains unified across both single-stage and multi-stage tasks.

To enable decentralized collaboration, each robot maintains its own RNN instance, initialized at potentially different states of the automaton. This allows for diverse interpretations of the task depending on local observations or roles, while still adhering to the global instruction structure. Crucially, the initialization of the RNN is also performed through natural language, meaning that each instruction not only defines the automaton but also specifies its hidden state in a semantically meaningful way. The RNN's current hidden state is then used to condition a graph neural network (GNN)-based policy, trained using multi-agent reinforcement learning similarly to the single-stage setting. While the policy is still shared across agents, it's conditioned on the evolving hidden state h_i of the RNN automaton model:

$$a_i \sim \pi_\theta(a_i \mid o_i, h_i)$$

This policy enables agents to act in a decentralized manner while respecting both the team topology and the temporal structure of the task.

We further show that a single RNN can learn to model multiple distinct automata, each representing a different high-level task. This yields a generalizable framework where robots can execute a broad range of complex instructions using a single, language-grounded controller. Because the RNN is trained on sentence embeddings, it inherits many of the benefits of language conditioning that we demonstrate in the first part of the thesis, such as adaptability and semantic alignment with human intent.

Finally, we emphasize that the full policy pipeline is lightweight and fully executable in real time. Since both the RNN and the GNN components are compact and efficient, the entire system is suitable for deployment on embedded hardware without requiring centralized computation or external coordination. By grounding the entire process in language, we directly map observations and human intent to coordinated robot actions. We refer the reader to Chapter 4 for a detailed explanation of the RNN training process and automaton extraction.

1.3 Background & Related Work

Sentence representations are a critical component in natural language processing (NLP) tasks such as retrieval and question answering [15–18]. Early approaches such as bag-of-words failed to capture semantic meaning, leading to neural sentence encoders based on convolutional neural networks (CNNs), RNNs, and later Transformers. Models like InferSent [19], USE [15], and Sentence-BERT [16] established transferable embeddings, with contrastive and instruction-tuned variants (e.g., SimCSE [20], INSTRUCTOR) further aligning representations with task intent. These

sentence embeddings now form the standard way of mapping free-form text into compact semantic vectors.

Beyond natural language processing (NLP) benchmarks, sentence embeddings have also been applied directly to robot learning. These approaches treat natural language as a compact, semantic input that conditions policies without requiring heavy symbolic parsing or full-scale reasoning. For example, *Do As I Can, Not As I Say* [21] uses frozen sentence embeddings to condition multitask policies, allowing robots to generalize across a wide range of affordances specified in free-form language. Similarly, Morad et al. [5] demonstrate that sentence embeddings can bias decentralized policies for multi-robot navigation, enabling teams to prioritize relevant regions of the environment based on linguistic guidance.

To our knowledge, however, there is no demonstration that sentence embeddings alone directly induce *team-level* behaviors in decentralized settings, where one natural-language description must yield coordination across multiple robots without a centralized planner. This gap motivates the first stage of our thesis: we evaluate how far simple language conditioning can go as a semantic bias for decentralized exploration. We then turn to instructions that encode contingencies, temporal order, and inter-agent dependencies, which require compact reasoning mechanisms that track task progress and coordinate actions across the team.

Early work on interpretable natural-language commands for robotics [7, 8, 22–24] 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 [25–27] and LMMs [28–30] overcome these limitations by encapsulating a comprehensive subset of human knowledge within a single data-driven model. As a consequence, they exhibit effective reasoning and flexibility to handle ambiguous yet expressive language prompts in both single- and multi-robot problems [9–11, 31–34]. For instance, LLMs can be used to enable algorithmic functions grounded in semantic information [35] or create complex formations [36] hard to specify, purely, in a metric space. Furthermore, LLMs have the ability to solve credit assignment in ambiguous settings [37–39], an extension of the classic Hungarian method to semantic metrics. Although effective, these works assume an infrastructure to connect the robots with the LLM/LMM in real-time, and some of them delegate the calculation of the low-level action to the LLM/LMM, neglecting potential unpredictable events.

To alleviate this, recent works [5, 12–14, 40, 41] propose computationally efficient alternatives that rely on reasoning distillation and distributed coordination. Ravichandran et al. [12] distil 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 [5]. Similar to them, our proposal distils LLMs into smaller models; nonetheless, we draw on the connections between automata theory and recurrent models [42] to achieve real-time reasoning. In this sense, LLMs have shown surprising skills in translating human commands into linear temporal logics [43], signal temporal logics [44] or planning domain definition language [45] 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 un-

expected 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.

Chapter 2

Problem Definition

A team of N robots is commanded to solve a task specified by a natural-language instruction $\mathcal{W}$. The robots have local connectivity, formalized as a time-varying, undirected graph

$$\mathcal{G}_k = (\mathcal{V}, \mathcal{E}_k),$$

where

$$\mathcal{V} = \{1, \ldots, \mathsf{N}\}$$

is the set of robots, and

$$\mathcal{E}_k \subseteq V \times V$$

is the set of edges at discrete time instant $k \in \mathbb{N}$. An edge $(i, j) \in \mathcal{E}_k$ implies connectivity between robots i and j. The set of neighbors of robot i at time k is defined as

$$N_{i,k} = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}_k\}.$$

The objective is to train control policies for the team such that the robots complete the task $\mathcal{A}$, relying only on local information available through $\mathcal{G}_k$.

This thesis addresses two levels of language-driven multi-robot control. First, single-stage tasks, where instructions can be executed directly without reasoning over temporal structures. Second, multi-stage tasks that require reasoning over subtasks, sequencing, and dependencies. In the latter case, the challenge extends beyond interpreting a high-level instruction: the team must also collaborate to complete the constituent subtasks and respect the logical progression encoded in the instruction.

However, we can capture both settings within a common mathematical framework: the decentralized partially observable Markov decision process (Dec-POMDP). A Dec-POMDP provides a natural formulation for cooperative multi-robot control under partial observability, where agents act based on local information while pursuing a shared objective. Formally, it is defined by the tuple

$$\mathcal{M} = (I, S, \{A_i\}, T, \{O_i\}, O, R, \gamma),$$

where

- $I = \{1, \ldots, N\}$ is the set of agents,

- S is the set of robot states,
- A_i is the action space for agent i,
- $T : S \times A_1 \times \cdots \times A_n \to \mathcal{P}(S)$ is the transition function,
- O_i is the observation space of agent i,
- $O : S \times I \to \mathcal{P}(O_i)$ is the observation function,
- $R : S \times A_1 \times \cdots \times A_n \to \mathbb{R}$ is the shared reward function,
- $\gamma \in [0, 1)$ is the discount factor.

In the single-stage setting, the entire instruction can be treated as a single Dec-POMDP instance, and policies are trained directly on this formulation. In the multi-stage setting, however, each subtask generated by the automaton is itself modeled as a Dec-POMDP.

2.1 Single-stage Problem

In the single-stage scenario, the task is for agents to cooperatively explore and locate T static targets in a continuous, two-dimensional environment defined on the bounded square domain $[0, D_{\text{map}}] \times [0, D_{\text{map}}] \subset \mathbb{R}^2$. Agents must locate every target while avoiding collisions with obstacles and each other.

From an instruction provided at the team level, agents must coordinate to efficiently cover the area while avoiding collisions. Search tasks may vary in complexity and requirements:

- Search targets defined by attributes like an index or color (e.g., "Locate the blue target").
- Conduct searches within a specific region (e.g., "Look for the target in the south-east corner").
- Find a specified number of targets in a given area (e.g., "Locate three targets in Room A").

Although motion is continuous, the exploration environment and all perceptual information is quantized onto a fixed $G_{\text{map}} \times G_{\text{map}}$ lattice with cell width

$$w_{cell} \;=\; \frac{D_{\text{map}}}{G_{\text{map}}}.$$

A continuous point (x, y) is mapped to the grid cell according to

$$\lfloor x/w_{cell} \rfloor,\ \lfloor y/w_{cell} \rfloor.$$

Each agent carries two sensors, that both yield accurate yet strictly local observations; no global or long-range sensing is available:

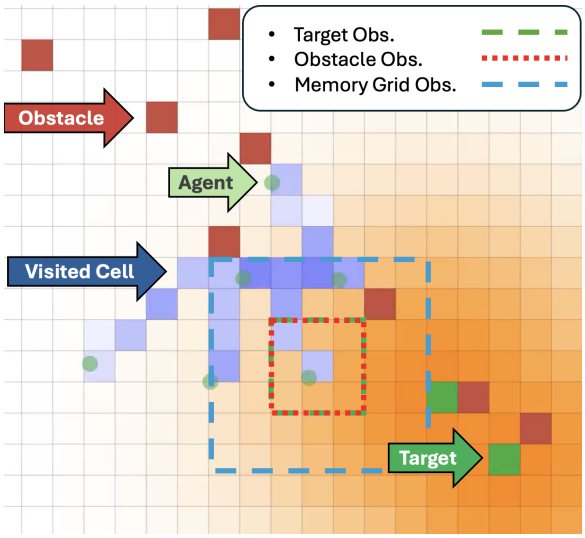


Figure 2.1: Visualization of the exploration environment.

- **Obstacle sensor**: outputs a binary occupancy map $O_i \in \{0,1\}^{L\times L}$, where a value of 1 marks cells containing an obstacle within the $L \times L$ window centred on the agent.
- **Target sensor**: outputs a binary occupancy map $T_i \in \{0,1\}^{L\times L}$, where a value of 1 marks cells containing a target within the same window.

Additionally, every agent maintains an internal occupancy grid

$$U_i \in \{0,1\}^{2\times G_{\text{map}}\times G_{\text{map}}},$$

on the same lattice. The two layers correspond to a *visitation memory* and a *target memory*. Specifically, $U_i^{(1)}(x,y) = 1$ denotes that cell (x,y) has been visited, while $U_i^{(2)}(x,y) = 1$ flags that a target has been discovered in that cell. The map is updated online after every observation and provides memory for exploration and coordination.

At each time step, agent i has access to a local observation of its internal map in an $L' \times L'$ window ,

$$u_i \in \{0,1\}^{2\times L'\times L'},$$

Under the Dec-POMDP assumption, such local observations of the memory grid are sufficient for coordinated team exploration, while yielding a significantly lighter and more tractable policy optimization problem.

Agents, targets, and obstacles are initialized at random while ensuring non-overlapping positions. An episode terminates once all targets have been located or a fixed horizon $k_{\max}$ is reached.

2.2 Multi-stage Problem

In the multi-stage setting, solving the task involves reasoning and solving sub-tasks sequentially, which we formalize as a DFA. A DFA consists of a finite set of states,

a start state, an input alphabet, and a transition function that maps state-input pairs to new states. In our context, the DFA is defined as the tuple:

$$\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, F)$$

where:

- $\mathcal{Q}$ is the finite set of automaton states,
- Σ is the input alphabet,
- $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is the transition function,
- $q_0 \in \mathcal{Q}$ is the initial state,
- $F \subseteq \mathcal{Q}$ is the set of accepting (or terminal) states.

While the above describes the construction of a single automaton for a given task, we aim to generate an entire family of such automata, one for each task specification. All automata in $\mathcal{A}$ share a common state space $\mathcal{Q}$ and input alphabet Σ, but may differ in their initial state, accepting states, and transition functions. Therefore, let $\mathcal{A}$ denote the set of automata generated in this framework as:

$$\mathcal{A} = \Big\{ (\mathcal{Q}, \Sigma, \delta_j, q_{0,j}, F_j) \;\Big|\; \delta_j : \mathcal{Q} \times \Sigma \to \mathcal{Q},\ q_{0,j} \in \mathcal{Q},\ F_j \subseteq \mathcal{Q} \Big\}.$$

In this thesis, we limit ourselves to tasks where all automata share a common state space $\mathcal{Q}$ and input alphabet Σ, differing only in their initial state, accepting states, and transition functions. This assumption ensures consistency of the symbolic structures that the RNN operates over and is sufficient for the tasks we aim to solve. In reality, however, there is no requirement that different automata must share the same state or event spaces: as long as the inputs and states can be embedded into vectors of equal dimension, the RNN could in principle learn to process entirely unrelated automata.

In our context, the "input" is derived from environment observations, and the "state" represents the current subtask within the instruction. We aim to train an RNN that emulates this set of automatons, allowing it to serve as a data-driven task model capable of interpreting and executing complex instruction flows.

In other words, given a set of tasks, our goal is to find a translation of their natural language specification into an automaton and then learn a policy that captures the logic behind the automaton, such that the robots can collaborate and reason about the steps of the task to compute the appropriate low-level actions that lead to its resolution. We require the model to be a lightweight neural network that is easily deployable on low-cost robot platforms without the need for online connectivity to LLM servers, or intense hardware to run large models.

Chapter 3

Language-Conditioned Exploration via Sentence Embeddings

In this chapter, we introduce a framework for training and deploying autonomous multi-agent exploration policies conditioned on natural language. The objective is for agents to navigate and cooperatively explore unknown environments for targets based on high-level instructions provided in natural language. These instructions are encoded into fixed-length vectors using a sentence embedding model, enabling direct conditioning of the agents' policies on human language. To achieve this, we leverage sentence transformers, which produce semantically meaningful vectors that capture the intent behind an instruction, even when phrased in different ways. For instance, the commands *"Explore the room until all corners have been seen."* and *"Keep moving until every part of the space is covered."* differ in wording but share the same meaning; a sentence transformer embeds them into nearby points in latent space, enabling the policy to generalize across such natural-language variations during training.

The exploration policies are trained entirely in simulation using a multi-agent reinforcement learning (MARL) algorithm. They are subsequently deployed in a zero-shot manner on real-world robot platforms without additional fine-tuning. This chapter outlines the reward signals used during training and the architecture of the model. To ensure robust generalization across diverse linguistic inputs, we incorporate a large-scale synthetic data generation pipeline using a language model. This component expands the agents' exposure to varied phrasings and semantic structures during training, enhancing their language understanding capabilities.

We also detail the mechanism for conditioning the policy on natural language: from embedding raw instructions to integrating them into the agent's observation and decision-making loop. Together, these components enable agents to perform language-grounded decentralized exploration tasks with minimal human intervention at deployment time.

3.1 Language-Conditioned Goals

Let $\mathcal{S}$ denote a set of natural-language commands or mission statements. We introduce an encoder

$$E : \mathcal{S} \longrightarrow G, \tag{3.1}$$

which maps a sentence $s \in \mathcal{S}$ to a dense embedding vector $g = E(s) \in G \subseteq \mathbb{R}^d$, where d is the output dimension of the encoder. The agent therefore learns a *language-conditioned policy*

$$\pi_\theta\big(a_i \mid o_i, g = E(s)\big), \tag{3.2}$$

capable of generalizing across a diverse set of linguistic tasks while leveraging shared environment dynamics.

Following our earlier discussion, E is implemented with a sentence transformer [46]. Concretely, each command $s \in \mathcal{S}$ is fed through the pre-trained model and the token-level outputs are mean-pooled to yield a fixed-length embedding g. However, embeddings alone provide no explicit signal about how to act or what constitutes success. To train effectively, we pair each sentence with labels that encode the desired behavior. This pairing bridges the gap between natural language and policy learning by anchoring the reward landscape on the extracted label. At environment reset, a sentence-label pair is drawn at random from the pre-generated dataset.

All sentences in this dataset have been pre-embedded using the chosen sentence transformer, so the environment directly retrieves the corresponding embedding vector g without recomputation. This embedding is fixed and shared by all agents for the entire episode, while the associated label parameters are used to configure the reward function for that episode. We will revisit the reward parametrization in section 3.3.

3.2 Data Generation

This section describes our process for generating the sentence–label pairs used to train the exploration policies. While the present case study focuses on exploration tasks, the same methodology applies to any natural-language instruction with a well-defined semantic goal.

Manual collection of such data is both slow and prone to bias. Prior work [5] has attempted to increase variety through hand-crafted vocabulary permutations, but these typically yield limited linguistic diversity and often produce unnatural phrasing. To overcome these limitations, we adopt a scalable, automated approach based on a Vision–Language Model (VLM). Given a rendered snapshot of the exploration environment and a structured prompt, the VLM generates a concise, natural-sounding mission description. Each image is paired with (i) a high-quality natural-language description of the task and (ii) a corresponding ground-truth label suitable for consumption by a reinforcement learning agent.

We therefore adopt an automated pipeline with the following three steps:

1. *Sample* a random grid representation of the environment converted to an image format and with the target region of interest clearly highlighted.

2. *Fill* a prompt template with these sampled values (color, target count, confidence, size, location).
3. *Pass* the filled prompt and image to the VLM, which produces a short, plain-text mission description.

Because the template slots are randomized, the VLM produces diverse sentence structures and tone while remaining consistent with the label. The result is a high-volume, high-variety corpus of sentence-label tuples that can be fed directly into supervised or reinforcement-learning pipelines to parametrize rewards. Figure 3.1 summarizes the three automated steps.

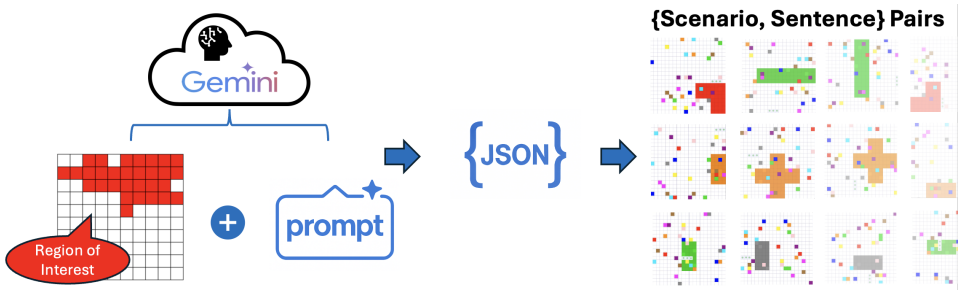


Figure 3.1: End-to-end pipeline for generating labeled language data. A random grid map is rendered, a compact patch is highlighted, and a vision–language model is prompted to produce a concise mission description. The boldface variables in the template 3.2 are sampled randomly to maximize linguistic diversity while keeping the ground-truth labels consistent.

Image and Prompt assembly. This paragraph is to help understand what is given to the VLM and why.

The image is generated by randomly sampling and rendering a 10×10 binary grid with a highlighted compact patch of m cells ($2 \leq m \leq 20$). This patch represents a **Region Of Interest (ROI)**, a region of the environment where agents are most likely to find a target. Additionally, we randomly sample semantic labels relevant to the exploration effort:

- a target colour $c_{\text{target}} \in \{\text{red}, \text{blue}, \text{green}, \dots\}$;
- a target count $n_{\text{target}} \in \{1, 2, 3\}$, which represents the number of targets agents should be looking for;
- a size label obtained from the size of the ROI,
- and a confidence level $q_{\text{lvl}} \in \{\text{low}, \text{moderate}, \text{high}\}$. The confidence level indicates how confident we are that the instruction is accurate

We then substitute all boldface tokens in the prompt template below with the sampled variables:

```
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.
-- Mention the color of the targets.

Please respond in a {tone_style} tone. Do not use any formatting such as bold or italics. Write plain-text sentences only. Be brief.

Provide an assessment of your confidence in the region as a likely location for the targets. You are {q_{lvl}}ly confident.

The completed prompt and the rendered grid image are sent to the VLM, which returns a short textual mission statement. Because the boldface placeholders are randomized, the pipeline produces a broad range of sentence structures, vocabulary and tones, giving us a rich, labeled corpus at virtually zero manual cost. This synthetic-data engine lets us scale to hundreds of thousands of language–label pairs, enabling robust training of our goal-conditioned policies.

3.3 Parameterizing the Reward Landscape on Sentence Labels

The availability of sentence–label pairs from the data generation pipeline (§3.2) enables direct conditioning of the reward function on the semantics of the natural-language instruction. This allows us to train exploration policies that align their behavior with the spatial and semantic cues contained in the command, despite the policy receiving only the sentence embedding as input.

Each sentence is associated with a structured label that specifies (i) the spatial location of the ROI in the map, (ii) the expected target attributes (color and count), and (iii) a confidence score. These labels serve as parameters for constructing reward functions that bias exploration toward the instructed areas and encourage correct target identification.

Instructions containing spatial information, such as "targets are most likely situated in the north-east," are linked to the ROI grid produced during data generation. Mapping the ROI to the environment grid enables the construction of spatially-biased rewards: agents receive higher rewards for visiting cells inside or near the ROI, especially when those cells have not been visited before.

Formally, the internal occupancy grid tracks visit counts per cell and computes a sigmoid visit level:

$$\nu_{i,k} \;=\; \sigma\big(v_{i,k}^{cell} - \theta_{\text{visit}}\big), \quad \sigma(x) = \frac{1}{1+e^{-x}}, \tag{3.3}$$

where $v_{i,k}^{cell}$ is the visit count of the cell visited by agent i at time k and θ_{visit} is the visit threshold. A cell is considered *new* if $\nu_k < \nu_{\min}$. The exploration reward

combines penalties for revisiting well-known cells with bonuses for visiting new ones:

$$r_{i,k}^{\text{explore}} \;=\; -\alpha_{\text{exp}}\,\nu_{i,k} \;+\; \alpha_{\text{new}}\,\mathbf{1}\big[\nu_{i,k} < \nu_{\min}\big]. \tag{3.4}$$

Here, $\mathbf{1}[\cdot]$ denotes the indicator function, which equals 1 if the condition inside holds and 0 otherwise.

When language conditioning is active, a heading bonus r_k^{head} increases rewards inside the ROI:

$$r_{i,k}^{\text{head}} \;=\; \alpha_{\text{head}}\,h_k\,\mathbf{1}\big[\nu_{i,k} < \nu_{\min}\big], \tag{3.5}$$

where $h_{i,k} \in [0,1]$ is the heading potential of the cell currently visited by agent i. This term densifies rewards in promising areas, making ROI exploration more likely.

The ROI also serves as a spatial prior for target spawning, strengthening the link between instruction semantics and exploration behavior. Confidence labels modulate this prior: sentences marked as "low confidence" broaden the sampling distribution, increasing the likelihood of targets appearing outside the ROI. This allows the policy to learn that spatial instructions with lower confidence are less reliable than those with high confidence.

In addition to spatial cues, sentence labels specify the target's color and expected count. The policy is rewarded for correctly covering targets matching these attributes.

Let $t_c^\star$ be the instructed target class, $n_{\max}$ the total number of such targets, and $n_{\text{team},k}$ the number of correctly covered targets by the team at time k. An exponential shaping function

$$A_{\text{exp},k} \;=\; \exp\!\Big(a\,\frac{n_{\text{team},k+1}}{n_{\max}}\Big) + b \tag{3.6}$$

increases the marginal reward for each additional discovery, reflecting the increasing difficulty of finding remaining targets.

Per-agent coverage rewards are given by:

$$r_{i,k}^{\text{cover}} \;=\; n_{i,k,t_c^\star}\,A_{\text{exp},k} \;-\; \beta_{\text{false}} \sum_{g \neq g^\star} n_{i,k,t_c}, \tag{3.7}$$

where n_{i,k,t_c} is the number of targets in class t_c covered by agent i, and β_{false} penalizes covering incorrect targets. Rewards are shared across the team:

$$r_k^{\text{cover}} \;=\; \frac{1}{N}\sum_j r_{j,k}^{\text{cover}}. \tag{3.8}$$

When the team has covered all targets of class $t_c^\star$ ($n_{\text{team}}^{\text{covered}} \geq n_{\max}$), agents receive a terminal bonus and are penalized for unnecessary movement:

$$r_{i,k}^{\text{term}} \;=\; \rho_{\text{term}}\,(1 - s_k \;-\; \rho_{\text{move}}\,\|v_{i,k}\|_2^2, \tag{3.9}$$

where s_k indicates episode termination for the team. Exploration and time-penalty terms are masked after termination.

Although the policy never observes the raw label, it receives a sentence embedding that encodes the same semantic structures. Over training, it learns to associate

patterns in this embedding space with specific reward structures: for example, recognizing that "green targets" and "north-east" correlate with high reward in certain spatial and attribute configurations. This embedding-to-reward alignment enables robust grounding of natural-language instructions in actionable multi-agent behaviors.

3.4 Model Architecture

In this section we describe the policy architecture and the rationale behind its components. The model has three stages: (i) a per-agent CNN head that encodes the internal map into a compact vector, (ii) a Graph Attention Network-based team encoder that aggregates information over a position-based communication graph, and (iii) a policy based on a multi-layer perceptron (MLP) that combines the team-conditioned embedding with the sentence embedding to produce per-agent actions.

3.4.1 Architecture Overview

Each agent encodes its two-channel internal map with a convolution, flattens the result, and projects it to a hidden vector h_i. Local observations are defined as

$$o_i = o_i^{\text{tgt}} \parallel o_i^{\text{obs}} \parallel n_i^{\text{found}} \parallel p_i \parallel v_i,$$

where $(o_i^{\text{tgt}}, o_i^{\text{obs}})$ are local nearby sensors, n_i^{found} is the target count, p_i is the position, v_i is the velocity, and $\parallel$ denotes concatenation. These are concatenated with h_i to form x_i.

A Graph Attention Network (GAT) runs over a radius-defined graph to produce team-conditioned embeddings z_i. Finally, the sentence embedding $g = E(c)$ is concatenated with $\mathbf{z}_i$ and passed through a policy MLP to output the per-agent action distribution a_i. An overview of the architecture is shown in Figure 3.2.

3.4.2 CNN Head for Feature Extraction

Each agent maintains a two-channel grid $U_i \in \mathbb{R}^{2\times G_{\text{map}}\times G_{\text{map}}}$ with visited cells and seen or visited targets. At each time step, agent i has access to a local observation u_i of its internal map in an $L'\times L'$ window. The CNN head applies a single convolution, flattens, and projects to a hidden vector:

$$W_i = \text{Conv}_{3\times 3}(u_i) \in \mathbb{R}^{N_{\text{conv}}\times L'\times L'}, \tag{3.10}$$

$$w_i = \text{Flatten}(W_i) \in \mathbb{R}^{N_{\text{conv}}L'L'}, \tag{3.11}$$

$$c_i = W_{\text{lin}} w_i + b_{\text{lin}} \in \mathbb{R}^{d_h}. \tag{3.12}$$

This produces a compact representation of the agent's internal map for downstream aggregation.

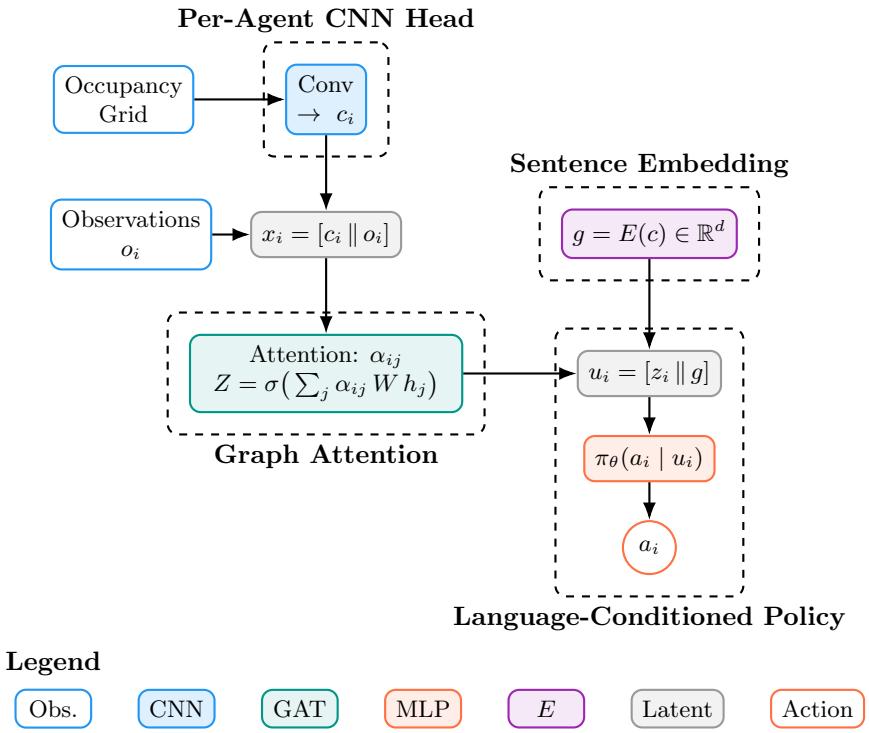


Figure 3.2: Architecture overview.

3.4.3 GAT for Communication and Team-Conditioned State Estimation

We construct a dynamic, undirected proximity graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ at each timestep k from agent positions p_i.

Each node $i \in \mathcal{V}_k$ corresponds to an agent. The node feature vector for agent i is defined as

$$x_i = [c_i \,\|\, o_i],$$

where h_i denotes the CNN embedding of the agent's perceptual input, and o_i contains additional local observations and state information.

An undirected edge $(i, j) \in \mathcal{E}_k$ exists if and only if the Euclidean distance between agents i and j is less than or equal to the interaction radius $r_{\text{edge}} > 0$:

$$(i, j) \in \mathcal{E}_k \quad \Leftrightarrow \quad \|p_i - p_j\|_2 \leq r_{\text{edge}}.$$

Thus, edges represent proximity-based interactions and change dynamically as agents move.

A Graph Attention Network (GAT) [47] aggregates information over $\mathcal{G}_k$ to produce a team-conditioned embedding z_i for each agent. The graph structure enables agent-specific weighting of neighbor information while preserving permutation invariance to agent ordering. Crucially, the attention mechanism allows each agent to assign higher weights to its own self-looped features while selectively incorporating information from neighbors. This selective aggregation lets an agent form a state representation that reflects the team's joint activity while still preserving agent-specific features.

3.4.4 Policy MLP

The policy is a multi-layer perceptron with ReLU activations. For agent i, the input is the concatenation of the GAT output and the sentence embedding $g = \phi(c) \in \mathbb{R}^d$:

$$u_i = [\, z_i \| g \,], \qquad \pi_\theta(a_i \mid u_i) = \mathrm{MLP}(u_i).$$

This yields decentralized, language-conditioned actions for all agents.

3.5 Evaluation and Discussion

In this chapter, we presented a complete framework for language-conditioned multi-agent exploration. The method integrates (i) a simulation environment with holonomic agents and local perception, (ii) a large-scale synthetic data generation pipeline for creating diverse sentence–label pairs, (iii) a reward parametrization scheme that grounds natural-language instructions in spatial and semantic cues, and (iv) a decentralized policy architecture combining CNN-based local map encoding, graph attention–based communication, and fixed sentence embeddings. Each episode is conditioned on a single pre-embedded instruction, with the corresponding label defining the reward structure for that episode. Together, these elements enable policies that can associate linguistic input with collaborative exploration strategies while retaining agent-specific decision-making.

Having established the methodological foundations, we now turn to empirical evaluation. The following experiments are designed to probe different components of our framework: first, we assess whether sentence embeddings indeed encode recoverable task semantics via supervised decoders; second, we examine how these semantics translate into improved multi-agent exploration when grounded in natural language. Together, these studies provide both quantitative and qualitative evidence of how language can guide decentralized policies toward more efficient collaborative behavior.

3.5.1 Sentence Embedding Decoders for Semantic Extraction

To verify that our synthetic language–label corpus encodes actionable semantics, we train a supervised decoder $d_\phi : \mathbb{R}^d \to \mathcal{L}$ that maps each sentence embedding $g = \mathcal{D}_\phi(c)$ to ground-truth labels $l = (l_{\mathrm{spat}}, l_{\mathrm{conf}}, l_{\mathrm{color}}, l_{\mathrm{num}})$. A decoder that generalizes beyond the training set indicates that the embeddings capture both categorical and spatial intent. Decoder performance (i) validates the data-generation pipeline, (ii) reveals architectural choices that best align with the latent space, and (iii) provides a quantitative proxy for comparing sentence encoders.

Decoder Architecture We use a multilayer perceptron (MLP) with ReLU activation and configurable depth. Depth and hidden size are varied systematically to assess their effect on performance.

The objective is for the decoder to jointly predict multiple outputs:

- **Spatial labels** l_{spat}: multi-label grid cells (binary per cell). Loss: BCE.
- **Confidence levels** $l_{\text{conf}} \in \{1, 2, 3\}$: single-label 3-class. Loss: CE.
- **Color classes** $l_{\text{color}} \in \{1, \ldots, 5\}$: single-label 5-class. Loss: CE.
- **Target counts** $l_{\text{num}} \in \{1, \ldots, 5\}$: single-label 5-class. Loss: CE.

We optimize the weighted sum:

$$\begin{aligned} L_{\text{total}} &= a_{\text{spat}} L_{\text{BCE}}(l_{\text{spat}}, \hat{y}_{\text{spat}}) \\ &\quad + b_{\text{conf}}\, L_{\text{CE}}(l_{\text{conf}}, \hat{y}_{\text{conf}}) \\ &\quad + c_{\text{color}}\, L_{\text{CE}}(l_{\text{color}}, \hat{y}_{\text{color}}) \\ &\quad + d_{\text{num}}\, L_{\text{CE}}(l_{\text{num}}, \hat{y}_{\text{num}}), \end{aligned} \tag{3.13}$$

with equal weights by default ($a_{\text{spat}} = b_{\text{conf}} = c_{\text{color}} = d_{\text{num}} = 1$). We use Adam optimization [48] (learning rate 10^{-4}), batch size 128, and train for 750 epochs on a fixed dataset of 3000 sentence–label pairs. We report validation metrics on held-out data.

We evaluate a set of relatively lightweight sentence transformer encoders, each with fewer than 500 million parameters in our setup: `GTE-Large` [49], `Instructor-Large` [50], `BGE-Large` [51], `Jina-Embeddings-v3` [52], `Mxbai-Large-v1` [53], `GIST-Large` [54], and `UAE-Large` [55]. Their relatively small size makes them practical to download locally and efficient enough to run online without specialized hardware.

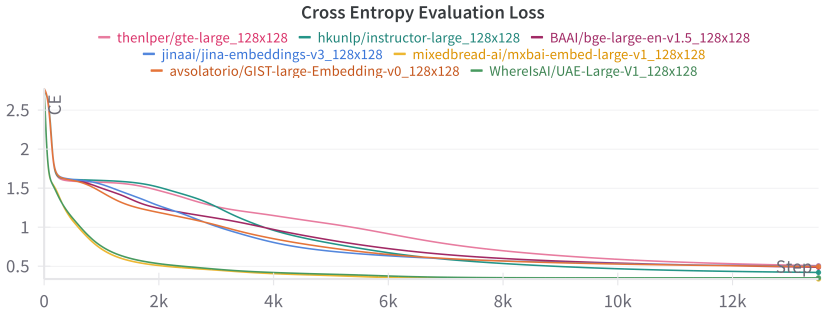


Figure 3.3: Validation loss versus epochs across embedding models. All runs use a two-layer 128-hidden MLP decoder.

Findings Some encoders converge faster, yet most reach comparable final validation loss and accuracy. This suggests the MLP can learn to read out the same latent factors across models, although the optimization landscape differs. While `Mxbai-Large-v1` and `BGE-Large` converge quickly and often achieve slightly lower early-epoch loss, we observe higher variance for larger decoders on these models (signs of overfitting). In contrast, `GTE-Large` exhibits more stable learning as decoder size grows. We therefore adopt `GTE-Large` as the default sentence encoder for subsequent experiments.

Qualitative Probes We visualize decoder outputs on hand-written instructions outside the training set. Varying location, size, color, and confidence terms produces

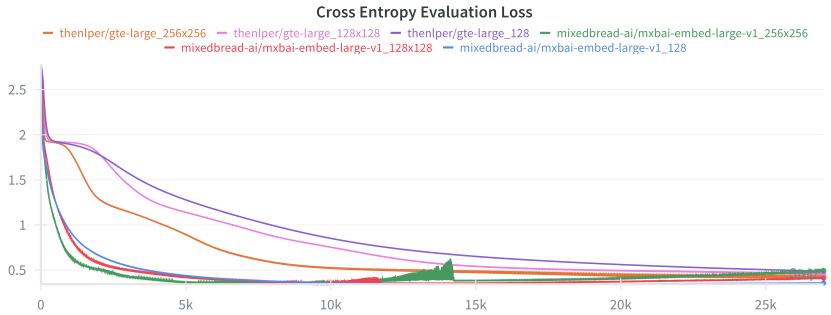
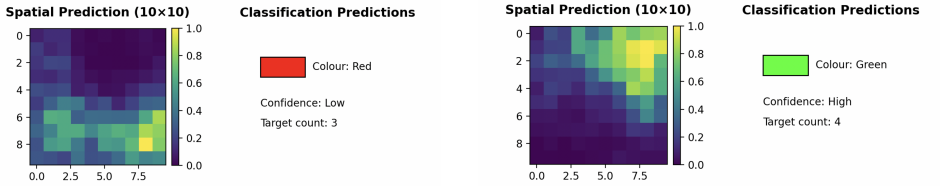


Figure 3.4: Validation loss across decoder sizes for two encoders. We compare hidden layouts [128], [128, 128], and [256, 256].



Prompt: `All right Agents! We are looking for three red targets... likely in a small, tight group in the south-east... confidence: low.`

Prompt: `Ok Team, we are searching for four green targets... most definitely agglomerated in a very large area in the north-east.`

Figure 3.5: Example decoder outputs for out-of-training prompts. Spatial grids and categorical predictions adapt coherently to changes in location, size, color, and confidence.

systematic, localized changes in predicted spatial grids and categorical heads, which indicates interpolation in embedding space rather than memorization.

The supervised decoder demonstrates that (i) our dataset generation pipeline yields embeddings with recoverable semantics, and (ii) a small MLP is sufficient to extract the task-relevant factors later used by goal-conditioned policies (Chapter 2). Although it generalizes well to unseen phrasing and domains, this remains a classification-style probe, underscoring the limitations of single-stage language-conditioned policies and motivating our multi-stage formulation with sequence modeling. We revisit these decoders later, as their properties are valuable both for evaluating exploration tasks and for training the RNN in the multi-stage setting.

3.5.2 Language for Guided Exploration

We demonstrate the application of our single-stage framework to an exploration task, where agents leverage natural language insights from a human operator to reduce exploration effort. The policy learns to recognize spatial cues in the instruction and target a specific exploration region in the environment. Our results show that higher-level human guidance can substantially reduce the exploration burden on the robot team: rather than covering the entire map, the agents can focus on a smaller region. This direct mapping from language to actions is especially valuable

in real-world settings, where the operator may lack the time or means to express commands in code or mathematical terms.

Since the base policies are exploration policies, agents will eventually discover targets given enough time. However, as shown in Figure 3.6, policies that use language guidance complete the task in fewer steps, with the advantage becoming more pronounced on larger maps.

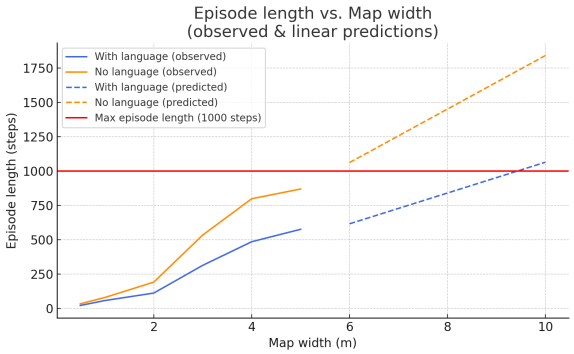


Figure 3.6: Episode length with and without language prompts to aid exploration.

We also find that the policy can decode semantic hints within the input. Thanks to our effective data collection pipeline, it generalizes to a wide range of vocabulary and can handle sentences never seen during training. Figure 3.7 illustrates exploration coverage patterns from a 5-agent team given prompts about the target location. The agents are able to interpret and act on nuanced spatial cues, focusing exploration in the specified region.

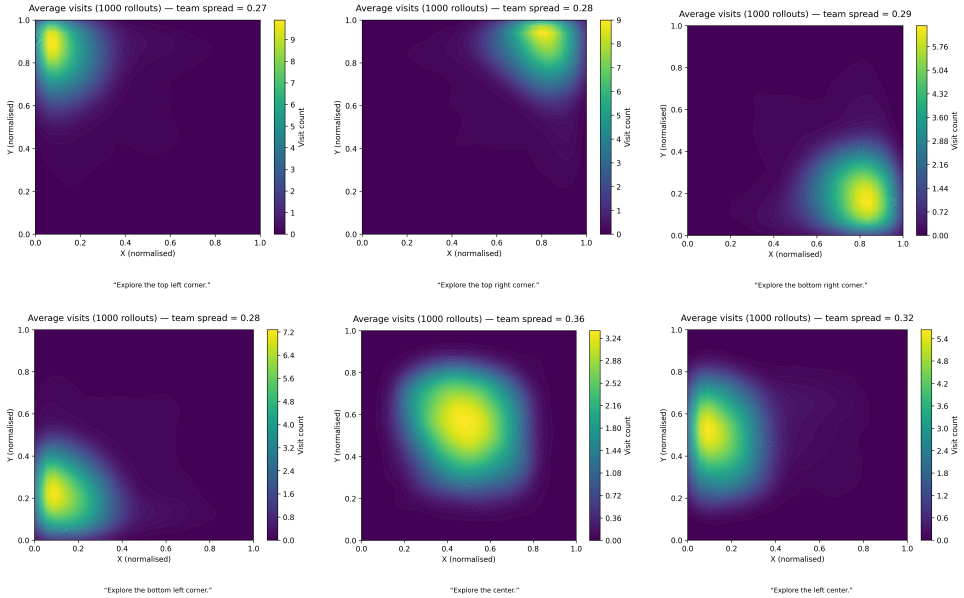


Figure 3.7: Exploration coverage patterns from a 5-agent team (averaged over 1000 runs) given prompts specifying a region of interest.

Finally, Figure 3.8 shows an example where the policy responds to area size information in the prompt. The heat maps (left, middle) depict visit frequency for

prompts containing "huge area" and "tiny area," respectively. Qualitatively, the agents adapt their exploration footprint according to the described scale, with the variance plot (right) showing faster convergence when the described area is smaller.

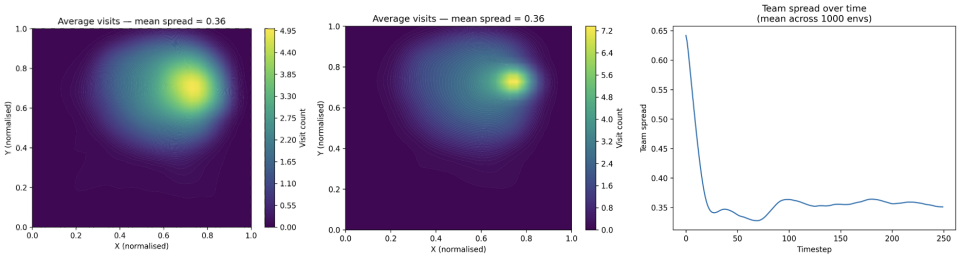


Figure 3.8: Exploration coverage patterns from a 5-agent team (averaged over 1000 runs) given the prompts "huge area on the top right" **(left)** and "tiny area on the top right" **(middle)**. The variance **(right)** shows how quickly the team converges to the specified area, demonstrating understanding of relative size.

3.6 Looking beyond single-stage tasks

Taken together, the experiments show that our single-stage framework can effectively ground natural language in decentralized multi-agent exploration. Agents learn to extract spatial and categorical cues from instructions, adjust their behavior accordingly, and reduce exploration effort through high-level human guidance. At the same time, the approach remains fundamentally limited by its reliance on direct semantic extraction: it treats each instruction as a static embedding, without accounting for compositional structure, temporal dependencies, or sequential reasoning. These limitations motivate the move toward a multi-stage formulation, where policies can integrate richer language dynamics.

Chapter 4

Reasoning through Natural Language

In this chapter, we present the centerpiece of this thesis – a complete pipeline for multi-agent reasoning and coordination through natural language. We call this framework **DeCLaRe** (*Decentralized Coordination and Language-Driven Reasoning*). In the previous chapter, we introduced our work on the single-stage setting and demonstrated that we could build on prior research [5] to condition a team of robots on instructions expressed in natural language, with far greater semantic diversity.

However, as discussed in the problem definition (Chapter 2), many real-world scenarios require robot agents to solve more complex tasks that involve reasoning and sequential decision-making. While the previous framework handles semantically rich instructions effectively, it struggles to interpret and act upon the compositional structure of multi-step commands.

To address this, we develop a hybrid framework that combines automata theory with recurrent neural networks (RNNs) to model and execute structured temporal logic in a decentralized multi-robot setting. This enables agents to decompose high-level natural language commands into ordered subgoals and coordinate their execution across the team.

An overview of the framework is provided in Figure 4.1. The initial offline phase generates a dataset of task sequences and associated language embeddings by prompting a LLM with different natural-language specifications (blue). Then, training is done in two offline stages (green): first, an RNN is trained with a supervised loss, using the generated dataset, to learn a set of semantic-aware automata; second, a GNN-based policy is trained with a reinforcement learning loss, running the RNN locally at each robot, to learn a low-level policy optimised to solve each sub-tasks encoded in the RNN. At online deployment (magenta), robots only have to translate the human command into an embedding and execute the RNN and the GNN to solve the task.

The pipeline contains three modules: the *Generation*, the *Training*, and the *Execution*. We address each in this order. Where relevant, we refer back to key aspects of the single-stage setup, as several components are directly reused or adapted in

this multi-stage formulation.

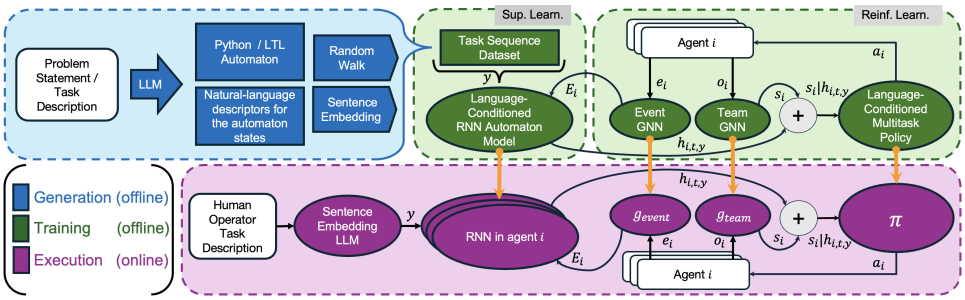


Figure 4.1: Overview of **DeCLaRe** (*Decentralized Coordination and Language-Driven Reasoning*). The framework consists of three modules: Generation, Training, and Execution. In the offline phase, natural language specifications are converted into task automata and policies; during deployment, robots ground human instructions in embeddings and execute them through the RNN and GNN.

4.1 Generation

4.1.1 From LLM Reasoning to Automaton

A core idea of the method is to leverage the reasoning capabilities of a large language model (LLM) to decompose a complex task into an automaton that a robotic system can consume. We draw on automata theory to represent the task as a finite set of states and transitions. Using carefully engineered prompts, the LLM generates a deterministic finite automaton (DFA) representation of the task.

In practice, we use the LLM to generate Python code that defines the automata used for synthetic data generation. As a reminder, in our context the DFA is defined as the tuple

$$\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, F),$$

where $\mathcal{Q}$ is the finite set of states, Σ the input alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ the transition function, $q_0 \in \mathcal{Q}$ the initial state, and $F \subseteq \mathcal{Q}$ the set of accepting states.

We assume an event dictionary

$$\mathcal{E} = \{e_1, \ldots, e_n\},$$

which assigns each event a unique bit index. The input alphabet is therefore

$$\Sigma = \{e_1, \ldots, e_n\}.$$

At runtime, the environment produces observations in the form of subsets of Σ where each observation can be encoded as a binary vector

$$v = (v_1, \ldots, v_n) \in \{0,1\}^n, \qquad v_j = \mathbf{1}\{\, e_j \in \mathcal{E} \text{ is observed at this step} \,\}.$$

Thus, a word over the alphabet is a sequence of observed event-sets. The automaton is not written directly. Instead, the LLM composes guards and combinators from a small library. We define a *guard* as a boolean predicate

$$b : \{0,1\}^n \to \{0,1\},$$

evaluated on an event vector $v \in \{0,1\}^n$. The library provides primitive guards that check for the presence or absence of specific events:

$$\texttt{has}(e_j)(v) \equiv (v[j] = 1), \qquad \texttt{not\_has}(e_j)(v) \equiv (v[j] = 0).$$

More complex guards are built compositionally using combinators:

$$\texttt{all\_of}(b_1, \ldots, b_k)(v) \equiv \bigwedge_{j=1}^{k} b_j(v), \qquad \texttt{any\_of}(b_1, \ldots, b_k)(v) \equiv \bigvee_{j=1}^{k} b_j(v).$$

Combinators turn such predicates into transitions between automaton states $q \in \mathcal{Q}$:

$$\texttt{advance\_if}(b, q^+, q^-)(v) = \begin{cases} q^+ & b(v), \\ q^- & \text{otherwise}, \end{cases}$$

$$\texttt{advance\_case}\big((b_1, q_1), \ldots, (b_m, q_m); q_{\text{def}}\big)(v) = \begin{cases} q_j & \text{for the first } j \text{ with } b_j(v), \\ q_{\text{def}} & \text{if none hold.} \end{cases}$$

Given a textual task description, the LLM is responsible for enumerating the states $\mathcal{Q}$ and assembling the transition function δ exclusively through these guards and combinators. This guarantees that the resulting specification is both executable Python code and consistent with the environment schema.

4.1.2 An Illustrative Example

Consider the environment illustrated in Figure 4.2.

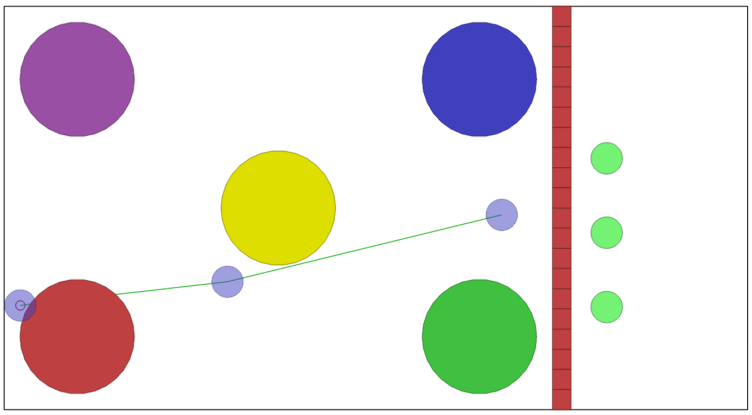


Figure 4.2: Four Flags environment with a switch (yellow), four flags (red, green, blue, purple), agents (light blue), and communication lines (green).

The environment consists of four colored flags randomly placed in the corners of the leftmost room, a yellow switch, a separating wall, and randomized agent goals in the opposite room. Agents can freely navigate to any object; however, to open the gate to the rightmost room and reach the goals, they must find a flag of a specific color and place it onto the switch. Furthermore, once the flag is collected, it may be lost, in which case the agents must navigate back to the original flag position and retrieve it.

The input to the LLM consists of:

- an environment description, specifying the available objects and constraints (e.g, the switch can only be triggered by the red flag),
- a task instruction in natural language (e.g., "find the red flag, carry it back to the switch, then navigate to the goal"),
- the alphabet

$$\Sigma = \left\{ \begin{array}{ll} R := \texttt{found\_red}, & G := \texttt{found\_green}, \\ B := \texttt{found\_blue}, & P := \texttt{found\_purple}, \\ SW := \texttt{found\_switch} & \end{array} \right\}.$$

  where each symbol encodes the detection of the corresponding object, and
- the robot capabilities, namely the ability to navigate to each object.

Given these inputs, the LLM instantiates the DFA corresponding to the task. Concretely, for the instruction above, the automaton is

$$\mathcal{Q} = \{\texttt{red},\ \texttt{switch},\ \texttt{goal}\}, \quad q_0 = \texttt{red}, \quad F = \{\texttt{goal}\},$$

and the transitions are:

$\delta(\texttt{red}, v) = \texttt{advance\_if}(\texttt{has}(R),\ \texttt{switch},\ \texttt{red})(v)$,

$\delta(\texttt{switch}, v) = \texttt{advance\_case}\big((\texttt{has}(SW),\ \texttt{goal}),\ (\texttt{not\_has}(R),\ \texttt{red});\ \texttt{switch}\big)(v)$,

$\delta(\texttt{goal}, v) = \texttt{goal}$.

This construction ties each mathematical component of $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ to concrete, pre-implemented guard functions and transition combinators, ensuring the LLM's output is both verifiable and directly executable.

In addition to the automaton, the LLM produces natural-language descriptions of each automaton state. For example, in a simple "capture the flag" scenario, we can describe the high-level task as: *"Find the flag and bring it back to base."* with two sub-tasks: "Find the flag." and "Bring the flag back to base.".

4.1.3 Dataset Construction

Given these two elements, we implement a dataset generation process: From the automaton, we generate random walks on the automaton graph, yielding example sequences of state transitions. These sequences serve as demonstrations from which we can train an RNN to replicate the automaton's transition behaviour. For each state in the sequence, we store:

- the natural-language description of the sub-task;
- its corresponding sentence embedding;
- the triggering event(s) that caused the transition.

Iterating through this process yields a large collection of task sequences, each structured as summarized in Table 4.1. Specifically, each automaton A_j is associated with a one-hot identity vector α_j, and each state $q \in \mathcal{Q}$ is likewise represented by a one-hot vector. Together, these encodings provide a canonical representation for every state occurrence. Given a random walk $S_{j,k}$ and the t^{th} step in automaton A_j, we define:

$$u^t_{j,k} = [\,\alpha_j \parallel q^t_k\,],$$

where $\parallel$ denotes concatenation of the task-level and state-level one-hot encodings. This ensures that each state in every task instance can be uniquely identified in the training data.

Table 4.1: Formal structure of the k_{th} generated task sequence $S_{j,k}$ for automaton $A_j \in \mathcal{A}$.

Symbol	Description
$W_{j,k}$	Natural-language description of the automaton A_j indexed by k
$y_{j,k}$	Sentence Embedding of the task description $W_{j,k}$
α_j	Binary Encoding of automaton A_j
$Q_{j,k} = \big(q^1_{j,k}, \ldots, q^M_{j,k}\big)$	Sequence of automaton states visited during the random walk
$w_{j,k} = \big(w^1_{j,k}, \ldots, w^M_{j,k}\big)$	Ordered sequence of M natural-language sub-task descriptions
$G_{j,k} = \big(g^1_{j,k}, \ldots, g^M_{j,k}\big)$	Corresponding sentence embeddings, where $g^m_{j,k} \in \mathbb{R}^d$
$E_{j,k} = \big(e^1_{j,k}, \ldots, e^{M-1}_{j,k}\big)$	Sequence of events triggering transitions between states

4.2 Training the RNN Sequence Model

The model is trained as a recurrent neural network f_θ that predicts the next latent state in a sequence of task-driven events. Formally, for a given generated sequence $S_{j,k}$ corresponding to automaton A_j, at each timestep i the network receives as input:

- the current event vector $e^t_{j,k} \in \mathbb{R}^p$;
- the previous hidden state $h^t_{j,k} \in \mathbb{R}^d$;
- the fixed high-level task embedding $y_{j,k} \in \mathbb{R}^d$, corresponding to the sentence embedding of the overall task description $W_{j,k}$.

Therefore, the RNN update is given by

$$h^{t+1}_{j,k} = \mathcal{F}_\theta\big(e^t_{j,k},\ h^t_{j,k} \big|\, y_{j,k}\big),$$

where $h_{j,k}^{(t+1)}$ is the updated hidden state, representing the model's internal estimate of the latent task state after processing $e_{j,k}^{t}$ under the conditioning of $y_{j,k}$. By including $y_{j,k}$ at every step, the model is explicitly conditioned on the structure and constraints of the relevant task automaton, enabling it to interpret incoming events in the correct task 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 fixed task embedding $y_{j,k}$ shifts their trajectories in the latent space so that $h_{j,k}^{t}$ for one task cannot overlap with $h_{j',k'}^{t'}$ from another.

At each timestep, the hidden state is decoded into a predicted discrete state label,

$$\hat{u}_{j,k}^{t} = \mathcal{D}_{\theta}\left(h_{j,k}^{t}\right),$$

which is trained to approximate the ground-truth label $u_{j,k}^{t}$. The decoder $\mathcal{D}_{\theta}$ is implemented as a single-hidden-layer MLP, deliberately kept shallow to ensure that it cannot itself capture sequential or semantic dynamics. Its sole role is to map the RNN's latent state into the supervised label space, while all temporal reasoning and semantic grounding remain encoded in the hidden dynamics of f_{θ}. This design prevents the decoder from absorbing knowledge that should reside in the recurrent state.

In addition, we introduce an optional auxiliary pathway based on the same pre-trained decoder $\mathcal{D}_{\phi}$ that was presented in Chapter 3 for single-stage problems. This decoder is trained separately to receive a subgoal embedding $g_k \in \mathbb{R}^d$ as input and output a vector of m task-relevant attributes $l_k \in \{0,1\}^m$. In the multi-stage setting, we get:

$$\hat{l}_{j,k}^{t} = \mathcal{D}_{\phi}(g_{j,k}^{t}).$$

The prediction $\hat{l}_{j,k}^{t}$ is supervised against the ground-truth attribute vector $l_{j,k}^{t}$, which can be obtained automatically during the random-walk data collection process. The output encodes aspects of the sub-task that are critical for downstream execution, such as spatial references, counts, or object properties. For example, in an exploration scenario with the instruction *"Explore the north-east corner for three blue targets."*, the high-importance elements are "north-east", "three", and "blue". The pre-trained decoder is designed to extract precisely such elements from the sentence embedding and, crucially, is kept frozen during training. This constraint encourages the learned latent representation h to align with the semantic space of g, ensuring that the model's internal features remain grounded in task-relevant linguistic attributes.

Another key element of the approach is grounding the hidden state in the sub-task language description. During training, the hidden state is occasionally replaced with the true sub-task embedding $g_{j,k}^{t}$. This random replacement acts as a form of grounding: it reinforces the association between the evolving RNN state and the semantic meaning of the sub-task, while allowing the RNN to still learn its own temporal dynamics. This method reduces the risk of hidden states drifting away from the intended semantic space.

The overall loss combines the main state classification objective with the optional auxiliary reconstruction loss. The classification loss on the state label ensures accurate step-by-step state predictions, while the auxiliary term –when used– serves as a regularizer that maintains semantic fidelity to the sub-task description. Training proceeds by backpropagating the combined loss through the RNN and decoder components, optimizing the model to jointly capture both temporal dynamics of

events and the semantic grounding provided by language.

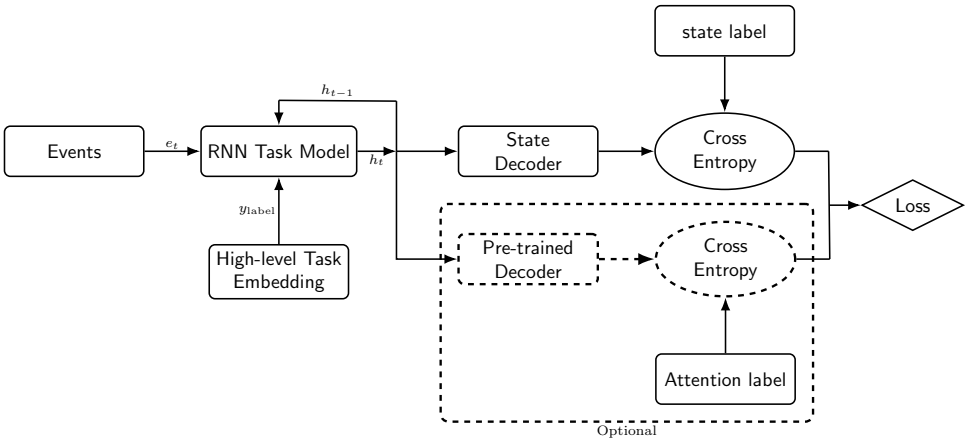


Figure 4.3: Training pipeline for the RNN Task Model.

4.3 Training the Multi-Task Policy

We now describe the training of the multi-task policy, which learns to solve each sub-task required to accomplish the overall task. Unlike a monolithic policy that operates over entire episodes, this policy is trained to act over short temporal segments corresponding to individual sub-tasks. At execution time, the appropriate sub-task policy is selected and conditioned on the hidden state of the high-level RNN, which encodes the current position within the task automaton.

Formally, for agent i at timestep t of sequence $S_{j,k}$, the policy is

$$\pi_\theta\big(a_t^t \mid z_t^t,\ h_{j,k}^t\big),$$

where a_t^t is the action, $h_{j,k}^t$ is the hidden state from the RNN encoding the current sub-task, and z_t^t is the team-conditioned observation embedding produced by the GNN from all agents' raw observations. The embedding z_t^t captures both the local observation of agent i and information received through inter-agent communication.

The multi-task policy architecture follows the same design as in the single-stage scenario described in Section 3.4. A graph neural network (GNN) processes agent-specific observations and facilitates inter-agent communication, producing a team-conditioned representation of the environment. This representation is concatenated with the RNN hidden state and passed through a multilayer perceptron (MLP) that outputs the next action for each agent.

By restricting training to individual sub-tasks, the learning process becomes more efficient and substantially reduces the computational load on the reinforcement learning algorithm. High-level decision-making, such as sequencing, switching, and coordinating sub-tasks, is delegated entirely to the RNN, allowing the multi-task policy to focus solely on low-level execution within the context of its current sub-task.

4.4 Training the Team Event GNN

The event GNN plays a critical role in enabling coordinated behaviour across agents. As illustrated in Figure 4.1, each agent maintains its own RNN sequence model f_θ, processing its individual event history. This design is important for decentralization, as it allows each agent to operate independently in the absence of communication. However, if each agent advances through the task automaton solely based on its own local events, without awareness of the others' progress, the team may fail to exploit coordination opportunities.

Consider the "hit the switch" scenario: two rooms, agents starting in room A, and a goal located in room B. The rooms are separated by a gate that opens only if at least one agent presses a switch located in room A. The task automaton for this scenario can be modelled with two states, as shown in Figure 4.4: (1) *Find the switch*, and (2) *Navigate to the goal*, with a transition triggered by the event "switch pressed".

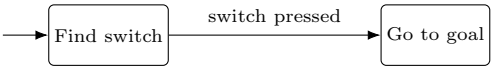


Figure 4.4: Two-state automaton for the "hit the switch" task.

If agents rely solely on their own RNN state, each may redundantly attempt to press the switch before moving to the goal, wasting time and resources. To address this, we introduce a *team-event GNN* $\mathcal{G}_\xi$ that aggregates individual agent event observations into a shared, team-level event representation:

$$z^t_{\text{team}} = \mathcal{G}_\xi\big(e^t_1, e^t_2, \ldots, e^t_n\big),$$

where $e^t_t \in \mathbb{R}^p$ is the event vector for agent i at timestep t, and $z^t_{\text{team}} \in \mathbb{R}^r$ encodes the team's joint event context. This representation is used to update the RNN state for each agent:

$$h^{t+1}_{j,k} = \mathcal{F}_{\theta,\, y_{j,k}}\big(z^t_{\text{team}},\ h^t_{j,k}\big).$$

The team-event GNN is trained via reinforcement learning in the same environment as the multi-task policy, using the same sub-task rewards (e.g., pressing the switch, reaching the goal), with the transition between sub-task rewards handled by a team-RNN only seen during training. During this phase, the parameters of the policy π_θ and the RNN f_θ are frozen; only the parameters ξ of the team-event GNN are updated. In this way, $\mathcal{G}_\xi$ learns to produce aggregated event representations that drive RNN state transitions optimally for coordinated task completion.

4.4.1 Deterministic team-event aggregation

In practice, training a separate team-event GNN is possible but not always necessary. In many multi-robot tasks, the relevant team-level signals reduce to simple logical combinations of agent-level events. For example, the event "switch pressed" can be expressed as

$$e^t_{\text{team}} = \bigvee_{i=1}^{n} e^t_{i,\text{switch}},$$

indicating that the team-event holds if at least one agent has triggered its local "switch" event. Similarly, "all reached goal" can be expressed as

$$e^t_{\text{team}} = \bigwedge_{i=1}^{n} e^t_{i,\text{goal}}.$$

In such cases, it is often more effective to replace learning with a deterministic definition of the team-event through boolean logic. This provides a lightweight, interpretable alternative that avoids additional model training while still capturing the essential coordination structure needed for the automaton to progress. Importantly, decentralization is not compromised: each agent continues to maintain its own RNN state, and the boolean aggregation can be computed locally wherever limited communication is available.

4.5 Evaluation

At initialization, a human operator provides a high-level task description in natural language. This task must belong to the set of tasks $\mathbb{T}$ on which the RNN sequence model has been trained to operate. From the operator's description of task $T_{j,k}$, we apply a Sentence Transformer to produce the fixed high-level task embedding $y_{j,k}$. This embedding conditions the RNN at every timestep and remains constant throughout task execution.

By default, the hidden state is initialized to a null vector:

$$h^0_{j,k} = \mathbf{0},$$

representing the automaton's start state. However, in Section 4.6.1 we show that the model can also be initialized at any state of the automaton for $T_{j,k}$ purely from natural language. This capability is enabled by our grounding strategy: during training, the hidden state is anchored in the semantic space of the Sentence Transformer embeddings. As a result, we can substitute the embedding $g^0_{j,k}$ of any sub-task description as the initial hidden state:

$$h^0_{j,k} = g^0_{j,k},$$

allowing execution to start from that sub-task directly.

This flexible initialization enables heterogeneous agent configurations in which different team members begin at different stages of the same task automaton. Such partial initialization can yield significant efficiency gains in multi-agent coordination, as demonstrated in our experimental results.

Finally, since the model conditions on a continuous sentence-embedding space, we hypothesize that interpolation to unseen but semantically related tasks is possible. Exploring this capability lies outside the scope of the present work, but we identify it as an important direction for future research.

4.6 Experiments and Discussion

We now evaluate the proposed architecture.

4.6.1 Learning and Executing Automata with RNNs

We demonstrate that recurrent neural networks (RNNs) can learn to represent multiple deterministic automata specified in natural language. Importantly, the learned model can be initialized at arbitrary substates through natural language commands and then rolled out consistently with the underlying automaton.

Task setup. We consider an unobstructed 2D environment in which a target (flag) is placed at a random location, and the robot base is placed at the center. We design two distinct tasks, both inspired by the capture-the-flag game. Each task is represented as a deterministic automaton whose states are described in natural language and whose transitions are triggered by environment events.

The first task, *Find the flag and return*, is illustrated in Figure 4.5. The automaton consists of the states

$$Q = \{\text{Explore}, \text{Navigate Home}, \text{Idle at Base}\}.$$

Agents explore the environment until the flag is located. Once found, they must navigate back to the base. If the flag is dropped and reset to its original location, the automaton transitions back to the *Explore* state. When the base is reached, the automaton terminates in the *Idle at Base* state.

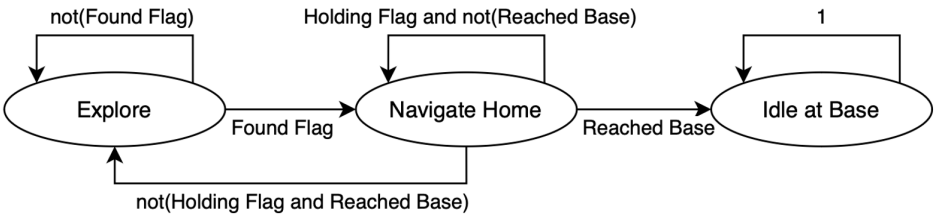


Figure 4.5: Automaton flowchart for the task of finding the flag and returning it to the base.

The second task, *Find and defend the flag*, is shown in Figure 4.6. After locating the target, agents must defend it rather than returning to base. The automaton includes two defense modes: a *wide defense*, where agents circle the flag at a large radius, and a *tight defense*, activated only if an enemy is detected. In the latter case, agents close in to form a tight circle around the flag. The task terminates at the end of the episode.

Model configuration. We implement the automaton learner as a gated recurrent unit (GRU) with hidden dimension d equal to that of the `gte-large` sentence transformer used for encoding natural language specifications. At each time step t, the model receives (i) a task embedding $y_{j,k}$, and (ii) an event vector $e^t_{j,k} \in$

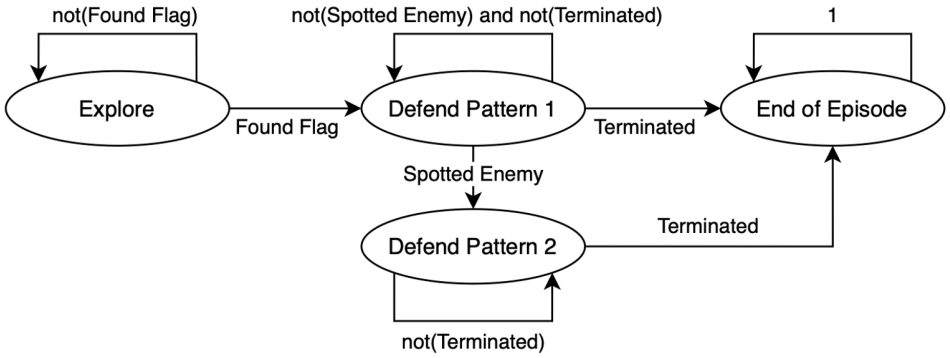


Figure 4.6: Automaton flowchart for the task of finding and defending the flag.

$\{0,1\}^m$, where m equals the maximum alphabet size across tasks. For clarity, in this implementation, the event vector retains the same dimensional interpretation even when some events are irrelevant for a given automaton (e.g., the "enemy spotted" event is inactive in Task 1).

The decoder $\mathcal{D}_\theta$ maps the hidden state $h^t_{j,k}$ to a predicted output $\hat{u}^t_{j,k}$ of dimension seven: two continuous action parameters α_j and one-hot state predictions for the union of states across both tasks. While this design slightly abuses notation by representing inactive states, it ensures a consistent representation across tasks.

Experimental aims. This experiment aims to demonstrate three key properties:

1. The RNN respects the deterministic transitions of both automata.
2. The RNN can be initialized in any state using natural language commands unseen during training, and it will converge to the correct automaton state.
3. The hidden latent representation $h^t_{j,k}$ preserves semantic information about the task specification.

Evaluation. To validate Properties (1) and (2), we construct controlled event sequences and roll out the RNN according to the automaton transitions. Figure 4.7 illustrates the information flow for a two-event sequence. We perform four experiments by varying the initial state provided through natural language and observe how the initialization influences the subsequent state trajectory. Results for the two tasks are shown in Figures 4.8 and 4.9.

Results: Find the Flag and Return In the first task, *find the flag and return*, we consider two different initializations. When initialized in the *Explore* state, the event "flag found" triggers a transition to the *Navigate Home* state, as prescribed by the automaton. In contrast, when initialized directly in the *Navigate Home* state, the same event is interpreted differently: the RNN transitions immediately to the *Idle at Base* state, correctly respecting the automaton structure. The results are visualized in Figure 4.8.

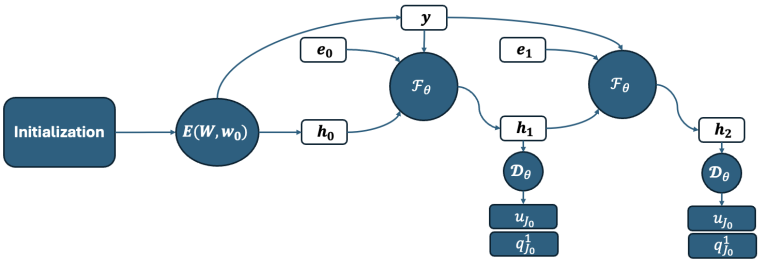


Figure 4.7: Information flow for a two-event sequence processed by the RNN $\mathcal{F}_\theta$.

Initialization Task W_1		*"Team, look for the target in the top-right corner, then navigate home"*	
Rollout K_1	**Events**	**(Found Flag, Spotted Enemy, On Base)**	
	e_0	[1 , 0, 1]	
	e_1	[0 , 0, 1]	
Output Classification Labels		(Task 1, Task 2, Explore, Navigate Home, Idle at Base, Defend Wide, Defend Tight)	
Initialization Subtask w^0_{1,K_1}		*"Start exploring the top-right corner."*	
Predicted Rollout $\hat{u}^t_{1,K_1} = \mathcal{D}_\theta(h^t_{1,K_1})$	$t = 1$	[1., 0., 0., 0.999, 0.001, 0., 0.]	Navigate back to base
	$t = 2$	[1., 0., 0., 0., 1., 0., 0.]	Stay Idle at base
Initialization Subtask w^0_{2,K_1}		*"Agents, return to the home base"*	
Predicted Rollout $\hat{u}^t_{2,K_1} = \mathcal{D}_\theta(h^t_{2,K_1})$	$t = 1$	[1., 0., 0., 0.287, 0.868, 0., 0.]	Stay Idle at base
	$t = 2$	[1., 0., 0., 0., 1., 0., 0.]	Stay Idle at base

Figure 4.8: RNN rollout results for the task of finding the flag and returning it to base. Different initializations lead to distinct but consistent trajectories.

Results: Find the Flag and Defend In the second task, *find and defend the flag*, we apply the same evaluation procedure. When initialized in the *Explore* state, the two test events do not provide evidence of the flag being found, and the RNN remains in the initial state, as expected. When initialized in the *Flag Found* state, however, the same event sequence takes on a new semantic meaning: the RNN activates the defense behaviors, leading the agents into their respective formations. This demonstrates that the RNN both adheres to the automaton transitions and correctly conditions on the natural-language-specified initialization (Figure 4.9).

Latent semantic preservation. In our third evaluation, we show that the hidden state of the RNN retains semantic meaning, a property that is necessary for downstream tasks.

The exploration tasks in our analysis are particularly illustrative. Rather than issuing exploration as a generic instruction, we aim to leverage the knowledge gained from the single-stage problem to guide exploration behavior in a more nuanced way. To make full use of the semantic structure of natural-language specifications, the internal RNN state must preserve this semantic information. Simply encoding exploration as a single binary state label would discard important details about the target's location or characteristics.

This motivates the inclusion of the frozen pre-trained decoder in our pipeline. By encouraging the learned latent representation h to align with a semantic embedding space, the decoder ensures that the model's internal features remain grounded in

Initialization Task W_2		*"Ok agents, find the flag in the bottom-right corner and defend it as best you can."*	
Rollout K_2	**Events**	**(Found Flag, Spotted Enemy, On Base)**	
	e_0	[0 , 0 , 0]	
	e_1	[0 , 1 , 0]	
Output Classification Labels		(Task 1, Task 2, Explore, Navigate Home, Idle at Base, Defend Wide, Defend Tight)	
Initialization Subtask w^0_{1,K_2}		*"Look for the target"*	
Predicted Rollout $\hat{u}^t_{1,K_2} = \mathcal{D}_\theta(h^t_{1,K_2})$	$t = 1$	[0., 1., 0.999, 0.016, 0., 0., 0.]	Explore the environment
	$t = 2$	[0., 1. , 1., 0., 0., 0., 0.]	Explore the environment
Initialization Subtask w^0_{2,K_2}		*"You found the target, defend it in a wide formation"*	
Predicted Rollout $\hat{u}^t_{2,K_2} = \mathcal{D}_\theta(h^t_{2,K_2})$	$t = 1$	[0. , 1. , 0. , 0. , 0., 1. , 0.]	Defend the flag in a wide formation
	$t = 2$	[0. , 1. , 0. , 0. , 0., 0., 1.]	Defend the flag in a tight formation

Figure 4.9: RNN rollout results for the task of finding and defending the flag. Natural-language initialization determines whether agents remain in exploration or enter defense formations.

task-relevant linguistic attributes.

Results. To validate this property, we evaluate whether the RNN latent state retains spatial information about the target location after processing a sequence of events. As shown in Figure 4.10, using the same pre-trained decoder D_ϕ from the single-stage problem, which maps latent vectors into spatial grid patterns, we decode the hidden state after three events and demonstrate that it reconstructs the correct target grid. This confirms that the RNN's latent dynamics preserve semantically meaningful information throughout execution.

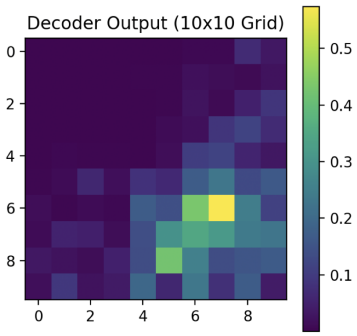
Initialization Task W_2		*"Search the bottom-right for the flag, then defend it by forming rings and arcs around it."*
Rollout K_3	**Events**	**(Found Flag, Spotted Enemy, On Base)**
	e_0	[0 , 0, 0]
	e_1	[0 , 0, 0]
	e_2	[0 , 0, 0]
Predicted Grid $l^2_{2,K_3} = \mathcal{D}_\phi(h^2_{2,K_3})$		

Figure 4.10: Decoding the hidden RNN state into spatial information using the frozen pre-trained decoder D_ϕ. Even after processing multiple null events, the latent representation preserves semantic information about the target's location.

State-classification validation. Finally, to further support our analysis, we report validation loss curves for state classification under three ablations: (i) no latent

grounding via random replacement of the hidden state with the ground-truth subtask embedding g and no pre-trained decoder, (ii) latent grounding enabled but no decoder, and (iii) latent grounding together with the pre-trained decoder. The RNN is optimized using a binary cross-entropy loss on the predicted automaton state,

$$L \;=\; L_{\mathrm{BCE}}\big(u^t_{j,k}, \hat{u}^t_{j,k}\big) \;+\; \lambda\, L_{\mathrm{BCE}}\Big(l^t_{j,k}, \hat{l}^t_{j,k}\Big)\,,$$

where $u^t_{j,k}$ denotes the ground-truth automaton state and $\hat{u}^t_{j,k}$ the predicted distribution. The optional second term compares latent representations $l^t_{j,k}$ and $\hat{l}^t_{j,k}$, and is only active when the pre-trained decoder $\mathcal{D}_\phi$ is included in the model, with $\lambda \in \{0, 1\}$. For consistency across ablations, however, the validation loss curves report only the first term, i.e. the state-classification loss.

Results. As shown in Figure 4.11, all three settings converge reliably: the validation accuracy approaches nearly 100% with a residual failure rate on the order of 10^{-5}. Since these metrics are computed on held-out sequences not seen during training, they indicate that the RNN generalizes the automaton's deterministic transitions and corrects to the appropriate state even when initialized from natural-language prompts.

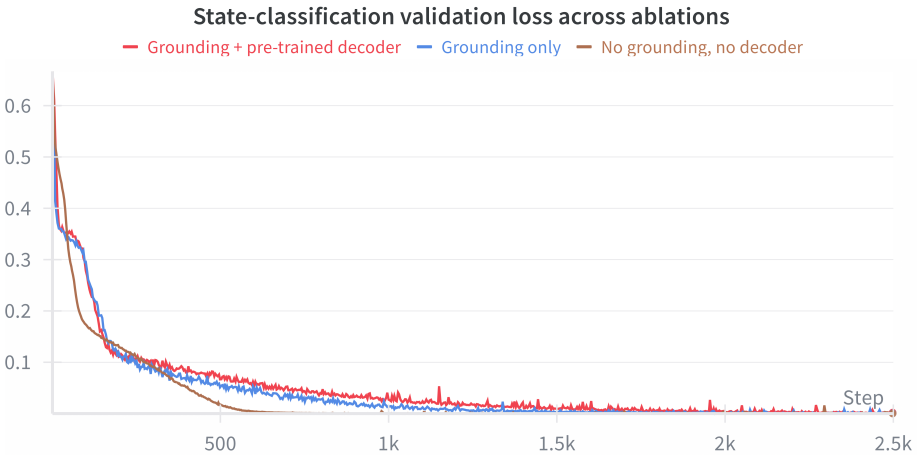


Figure 4.11: Validation loss curves for state classification under three ablations: (i) no latent grounding and no decoder, (ii) latent grounding only, and (iii) latent grounding with pre-trained decoder. All configurations converge to near-perfect accuracy with a residual failure rate around 10^{-5}.

4.6.2 Training a Team-GNN for Coordinated Events

The event GNN plays a critical role in enabling coordinated behavior across agents. We show that it is possible to train a GNN that aggregates and communicates events across the team, allowing agents to synchronize their internal states and maximize overall task performance.

We revisit the simple two-stage problem presented in Section 4.4 and illustrated in Figure 4.4. To reach their goal located in an adjacent room, agents must first

activate a switch that opens a gate. If agents rely solely on their individual RNN states, each may redundantly attempt to press the switch before proceeding to the goal, resulting in wasted time and resources. To address this, we introduce a team-event GNN that aggregates individual agent event observations into a shared, team-level event representation.

Training procedure. Figure 4.12 shows the reward curves for the two-stage training process with five agents. In the first stage, we train a multitask policy to perform the two subtasks of the problem: (i) navigating to the switch, and (ii) navigating to the goal. Owing to the simplicity of the navigation behaviors required, this policy converges rapidly, within 40 training iterations.

In the second stage, we freeze the multitask policy and introduce the team-event GNN. At this point, the agents execute full episodes of the task, but the reward functions remain identical to the first stage, allowing us to represent both training phases on the same plot. Initially, the GNN outputs random events, leading to a temporary drop in performance. Over time, however, it learns to synchronize event messages across agents, thereby improving coordination and overall reward.

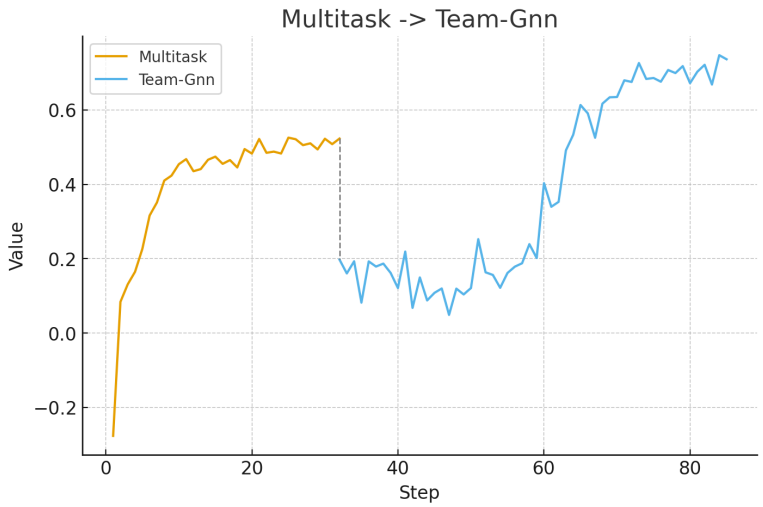


Figure 4.12: Reward curves for the two training stages in the *Hit the Switch* scenario with 5 agents. The multitask policy (orange) converges within 40 iterations. In the second stage, training the team-GNN enables further performance gains.

Results. When the switch is activated, the optimal behavior is for agents to propagate this event to their neighbors, allowing all team members to update their internal RNN states and transition to the next subtask (navigating to the goal). The team-GNN successfully learns this behavior, effectively broadcasting the relevant event and enabling coordinated task execution. We validate this outcome in simulation, as shown in Figure 4.13.

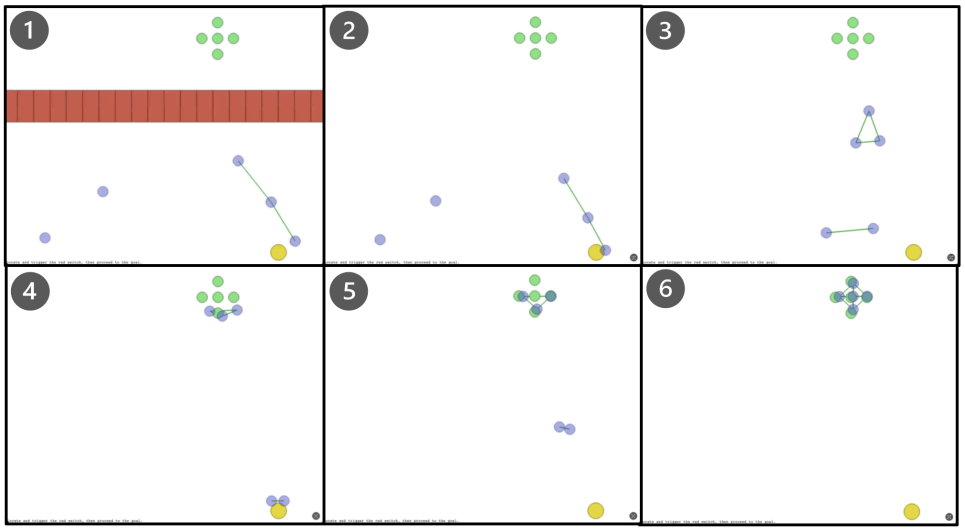


Figure 4.13: Coordinated execution of the *Hit the Switch* task with 5 agents. Agents are grouped into two proximity-based teams. When one agent in the left team activates the switch, the event is propagated through the team via the GNN, enabling all members to transition directly to the goal. The right team, lacking this information, continues navigating toward the switch until one of its agents activates it. Only then does the event propagate through the second team, allowing them to also proceed to the goal.

4.7 Scaling to Complex Multi-Stage Tasks: The Four-Flags Scenario

For our final analysis, we revisit the environment introduced in Section 4.1.2. The setup consists of four colored flags randomly placed in the corners of the leftmost room, a yellow switch, a separating wall, and randomized agent goals in the opposite room. Agents may freely navigate to any object; however, to open the gate and reach their goals, they must first locate the correct flag and place it on the switch.

This scenario extends the simple two-stage *Hit the Switch* problem by adding significant complexity. Agents must complete at least three sequential subgoals to succeed, but since flags can be dropped and reset, the task can in principle be of unbounded length.

Scaling multitask policies. We have already shown that RNNs can robustly capture automaton dynamics. Here, we test whether our multitask policy can scale to more complex problems. With four possible flag colors (red, green, blue, purple), each introducing a variation of the problem, and each variation consisting of three states, the multitask policy must cover 12 total state–task combinations. Despite this increased complexity, we find that the policy scales effectively and achieves strong performance, far outperforming single-stage RL baselines. Crucially, the benefits are not only in agent behavior but also in the reduced design and training effort compared to traditional RL approaches.

Implementations compared. We evaluate three language-conditioned RL implementations:

1. **Vanilla single-stage RL.** Agents are rewarded only for reaching their goals. They must implicitly infer that the flag and switch must be used to achieve this, without explicit task progression signals. Observations include events, global task descriptions, and agent kinematics.

2. **Hard-coded RL with shaped rewards.** The reward is manually aligned with task progression: finding the flag triggers a transition to rewarding switch interactions, and so on. While this improves learning, it requires significant reward engineering. Observations remain the same as above.

3. **DeCLaRe.** Our method, which trains the RNN via supervised learning on random-walk sequences, while the multitask policy is trained on individual subtasks. Training is balanced by sampling subtasks uniformly and retrieving matching RNN rollouts. Coordination across agents is handled by simple Boolean logic, eliminating the need for a team-GNN.

Evaluation. We roll out each policy 500 times in simulation and record subtask success rates: whether an agent finds the correct flag, whether it successfully places the flag on the switch, and whether the full team reaches their goals. Results are shown in Figure 4.14.

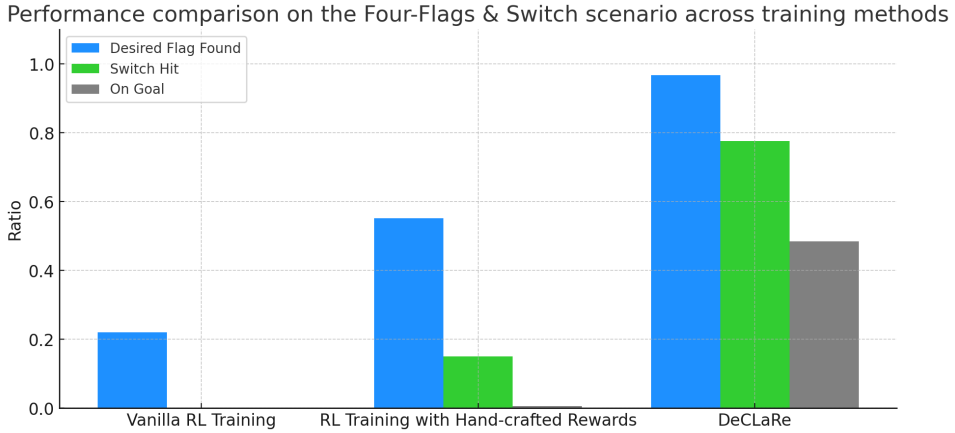


Figure 4.14: Subtask success rates for the three implementations. DeCLaRe achieves substantially higher success across all stages compared to vanilla and hard-coded single-stage RL baselines.

Results. DeCLaRe achieves markedly higher success rates than both baseline implementations. While the final goal success rate is approximately 50%, we attribute this primarily to agent kinematics (difficulty staying idle on the goal) and collision-avoidance constraints, rather than failures in task sequencing. While further reward shaping could improve the hard-coded RL baseline, such improvements come at the cost of extensive engineering effort. In contrast, DeCLaRe requires minimal manual design: agents only need to master individual subtasks, while the RNN handles sequence composition. This demonstrates the scalability and efficiency of our framework for complex, multi-stage coordination problems.

4.7.1 Deploying the Four-Flags Scenario

Finally, we demonstrate a one-shot deployment of the trained policy from simulation to the real world. Without additional fine-tuning, the multitask policy and RNN are executed directly on a team of three RoboMaster [56] platforms in a physical arena. The task mirrors the simulation setup: agents must retrieve the designated flag, place it on the switch to open the gate, and then navigate to their assigned goal locations. This experiment validates that the learned representations and coordination strategies transfer effectively from simulation to hardware. This is, to our knowledge, the first demonstration of a language-conditioned multitask policy with automata deployed with sim-to-real transfer.

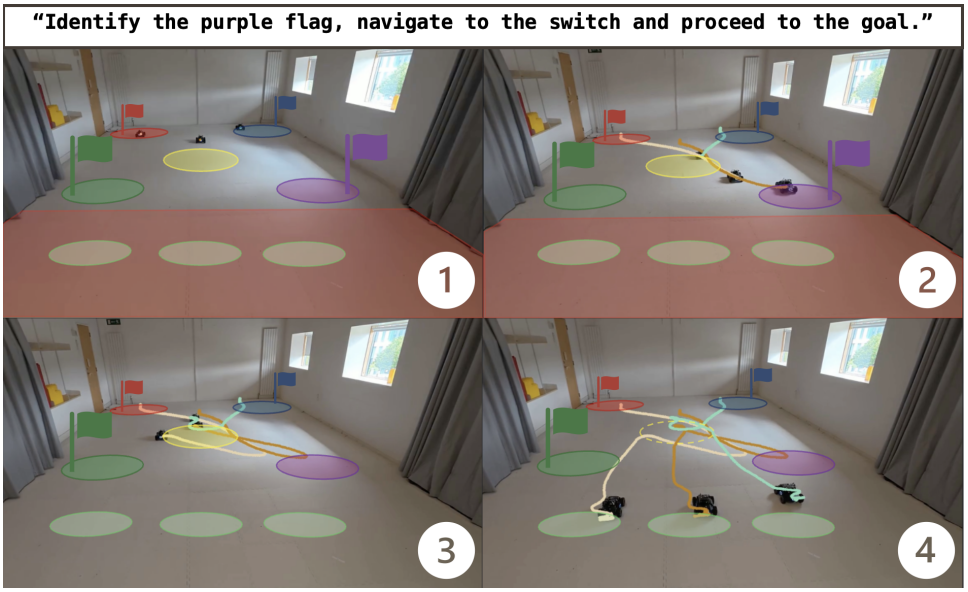


Figure 4.15: Real-world deployment of the Four-Flags scenario using three RoboMaster platforms. The robots are tasked with retrieving the purple flag and placing it on the switch. Once the switch is activated, the gate opens and the robots proceed to their final goal locations.

Chapter 5

Conclusion

5.1 Summary

This thesis set out to investigate how natural language can guide decentralized multi-robot systems in a principled and scalable way. The motivation was to connect the intuitive expressiveness of human communication with the robustness and efficiency required for distributed control. To this end, we developed a framework that translates language into compact semantic representations and integrates them directly into decentralized policies.

In the single-stage setting, we showed that language-conditioned policies improve exploration efficiency by allowing high-level intent to be embedded into each agent's decision-making, without requiring explicit decomposition of the task. This established that lightweight sentence embeddings are sufficient to bias decentralized policies towards meaningful team behaviors.

Building on this, we introduced a hybrid architecture combining deterministic finite automata with recurrent neural networks to extend the approach to multi-stage tasks. The automata were generated automatically using large language models and embedded into the RNN hidden state, enabling agents to follow temporally structured instructions in a decentralized manner. Evaluation confirmed the viability of this approach: 1) the RNN was able to learn embeddings of multiple automata tied to natural language specifications, robustly rolling out from arbitrary subtasks, while retaining semantically relevant information; 2) we trained team-level graph neural networks (team-GNNs) to enable coordinated event propagation in a multi-agent "hit-the-switch" scenario, showing that agents can share and synchronize task-relevant events through local communication, thereby improving efficiency and avoiding redundant behaviors; 3) in a complex "find the flag, bring it to the switch, and navigate to the goal" task, our method achieved significantly higher task and subtask success rates than baselines trained directly from reward signals, including those with additional reward shaping; 4) and finally, we demonstrated real-world deployment of the approach on the RoboMaster robotic platform.

Taken together, these results establish that natural language can be grounded in structured automata to bridge free-form task descriptions and executable decentralized policies, even in complex, temporally extended domains.

5.2 Outlook

This work suggests several directions for future research. Large language models could play a more direct role in generating the boolean conditions required for event aggregation, further automating the connection between natural language and execution logic. Exploring richer automaton encoders such as transformers could capture a broader task space and reduce reliance on manual structure. Building on this, we could further test how such interpretability scales when applied to larger teams and increasingly complex environments.

Another promising avenue is to decode the hidden state of the RNN into natural language, enabling agents or human operators to obtain interpretable, real-time feedback on subgoals during execution.

Finally, we assumed a homogeneous policy shared across all agents. Introducing heterogeneity, where agents specialize in complementary roles while coordinating through shared language grounding, offers an exciting opportunity to expand the expressiveness and efficiency of decentralized multi-robot systems.

Bibliography

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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," Nov. 2024, arXiv:2409.17348.

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

[12] 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," Jun. 2025, arXiv:2506.17486.

[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," Jul. 2025, arXiv:2503.08090.

[15] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil, "Universal sentence encoder for english," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 169–174.

[16] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *EMNLP*, 2019.

[17] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020, pp. 6769–6781.

[18] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Dubrovnik, Croatia: Association for Computational Linguistics, 2023, pp. 2014–2037.

[19] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 670–680.

[20] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021.

[21] B. ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do as i can, not as i say: Grounding language in robotic affordances," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, Dec 14–18 2023, pp. 287–318.

[22] 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.

[23] M. MacMahon, B. Stankiewicz, and B. Kuipers, "Walk the talk: connecting language, knowledge, and action in route instructions," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'06. AAAI Press, 2006, p. 1475–1482.

[24] 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, Mar. 2022.

[25] 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.

[26] 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.

[27] 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," Mar. 2025, arXiv:2410.03035.

[28] 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.

[29] 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.

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

[31] 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," Apr. 2024, arXiv:2309.04077.

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

[33] 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," Oct. 2024, arXiv:2410.02511.

[34] 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.

[35] 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," May 2025, arXiv:2505.09108.

[36] V. L. N. Venkatesh and B.-C. Min, "ZeroCAP: Zero-Shot Multi-Robot Context Aware Pattern Formation via Large Language Models," Mar. 2025, arXiv:2404.02318 version: 3.

[37] M. Lin, S. Shi, Y. Guo, V. Tadiparthi, B. Chalaki, E. M. Pari, S. Stepputtis, W. Kim, J. Campbell, and K. Sycara, "Speaking the Language of Teamwork: LLM-Guided Credit Assignment in Multi-Agent Reinforcement Learning," Mar. 2025, arXiv:2502.03723.

[38] V. Mahadevan, S. Zhang, and R. Chandra, "GameChat: Multi-LLM Dialogue for Safe, Agile, and Socially Optimal Multi-Agent Navigation in Constrained Environments," Mar. 2025, arXiv:2503.12333.

[39] K. Nagpal, D. Dong, J.-B. Bouvier, and N. Mehr, "Leveraging Large Language Models for Effective and Explainable Multi-Agent Credit Assignment," *arXiv preprint arXiv:2502.16863*, 2025.

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

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

[42] 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.

[43] 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 *2024 IEEE International Conference on Robotics and Automation (ICRA)*, May 2024, pp. 14 062–14 069.

[44] 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 *2024 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2024, pp. 6695–6702.

[45] 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," Jul. 2025, arXiv:2506.07454.

[46] A. R. Kashyap, T.-T. Nguyen, V. Schlegel, S. Winkler, S. K. Ng, and S. Poria, "A comprehensive survey of sentence representations: From the bert epoch to the chatgpt era and beyond," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 1738–1751.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[49] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang, "Towards general text embeddings with multi-stage contrastive learning," 2023.

[50] H. Su, W. Shi, J. Kasai, Y. Wang, Y. Hu, M. Ostendorf, W. tau Yih, N. A. Smith, L. Zettlemoyer, and T. Yu, "One embedder, any task: Instruction-finetuned text embeddings," 2023.

[51] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, "C-pack: Packaged resources to advance general chinese embedding," 2023.

[52] S. Sturua, I. Mohr, M. K. Akram, M. Günther, B. Wang, M. Krimmel, F. Wang, G. Mastrapas, A. Koukounas, A. Koukounas, N. Wang, and H. Xiao, "jina-embeddings-v3: Multilingual embeddings with task lora," 2024.

[53] X. Li and J. Li, "Angle-optimized text embeddings," *arXiv preprint arXiv:2309.12871*, 2023.

[54] A. V. Solatorio, "Gistembed: Guided in-sample selection of training negatives for text embedding fine-tuning," *arXiv preprint arXiv:2402.16829*, 2024.

[55] X. Li and J. Li, "AoE: Angle-optimized embeddings for semantic textual similarity," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024, pp. 1825–1839.

[56] J. Blumenkamp, A. Shankar, M. Bettini, J. Bird, and A. Prorok, "The cambridge robomaster: An agile multi-robot research platform," 2024.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. **In consultation with the supervisor**, one of the following two options must be selected:

☐ I hereby declare that I authored the work in question independently, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

☒ I hereby declare that I authored the work in question independently. In doing so I only used the authorised aids, which included suggestions from the supervisor regarding language and content and generative artificial intelligence technologies. The use of the latter and the respective source declarations proceeded in consultation with the supervisor.

Title of paper or thesis:

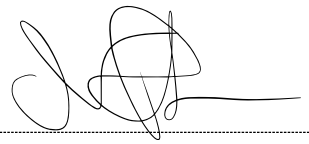
Multi-Robot Coordination and Reasoning through Natural Language

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Pfitzer	Nicolas

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guidelines.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Cambridge, UK 30/08/2025	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] For further information please consult the ETH Zurich websites, e.g. https://ethz.ch/en/the-eth-zurich/education/ai-in-education.html and https://library.ethz.ch/en/researching-and-publishing/scientific-writing-at-eth-zurich.html (subject to change).