Closed-Loop Vision-Language Planning for Multi-Agent Coordination

Zhiyuan Li
Department of Electrical Engineering and Automation, Aalto University
Finland
zhiyuan.li@aalto.fi

Wenshuai Zhao
Department of Computer Science, Aalto University
Finland
wenshuai.zhao@aalto.fi

Joni Pajarinen
Department of Electrical Engineering and Automation, Aalto University
Finland
joni.pajarinen@aalto.fi

ABSTRACT

Cooperative multi-agent reinforcement learning (MARL) struggles with sample efficiency, interpretability, and generalization. While Large Language Models (LLMs) offer powerful planning capabilities, their application has been hampered by a reliance on text-only inputs and a failure to handle the non-Markovian, partially observable nature of multi-agent tasks. We introduce COMPASS, a multi-agent framework that overcomes these limitations by integrating Vision-Language Models (VLMs) for decentralized, closed-loop decision-making. COMPASS dynamically generates and refines interpretable, code-based strategies stored in a skill library that is bootstrapped from expert demonstrations. To ensure robust coordination, it propagates entity information through a structured multi-hop communication protocol, allowing teams to build a coherent understanding from partial observations. Evaluated on the challenging SMACv2 benchmark, COMPASS significantly outperforms state-of-the-art MARL baselines. Notably, in the symmetric Protoss 5v5 task, COMPASS achieved a 57% win rate, a 30 percentage point advantage over QMIX (27%). Project page can be found at https://stellar-entremet-1720bb.netlify.app/.

KEYWORDS

Legends, Myths, Folktales

ACM Reference Format:
Zhiyuan Li, Wenshuai Zhao, and Joni Pajarinen. 2026. Closed-Loop Vision-Language Planning for Multi-Agent Coordination. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 26 pages.

1 INTRODUCTION

A major long-term goal for the field of cooperative multi-agent systems (MAS), e.g. multi-robot control [7, 10], power management [32], and multi-agent games [17, 39], is to build a protocol of collaboration among the agents. Multi-agent reinforcement learning (MARL), proven as an advanced paradigm of distributed artificial intelligence (AI), holds promise for discovering collective behavior from interactions. One line of works follows centralized training decentralized execution (CTDE) paradigm [21, 23, 25, 38, 53, 59]. CTDE assumes a central controller that exploits global information, while the individual policies are designed to allow for decentralized execution. However, in many real-world scenarios, the central controller becomes unfeasible due to the communication overhead that exponentially scales with the number of agents, thereby compromising the scalability of MAS. In contrast, the decentralized training decentralized execution paradigm (DTDE) discards this assumption and is scalable to large-scale systems [26, 42, 57]. However, DTDE requires complicated learning and planning under uncertainty, as partial observability magnifies the discrepancy between each agent's local observation and global information. Although much progress has been made, MARL suffers from compromised sample efficiency, interpretability, and transferability.

The emergence of Large Language Models (LLMs) has revitalized this field. LLM-based multi-agents have been proposed to leverage their remarkable capacity to perform task-oriented collective behaviors [11, 29, 34, 54, 56]. The LLMs are used for high-level planning to generate centralized [4, 11, 34] or decentralized plans [29, 56], often adopting a hierarchical decision-making structure in conjunction with a pre-defined low-level controller. While these methods have succeeded in a set of multi-agent problems including Overcooked-AI [3], SMAC [39], and VirtualHome [37], heavy reliance on text-based observation prevents them from learning from multi-modal information. Moreover, they ignore the non-Markovian nature of MAS, where learning and planning necessitate a decentralized closed-loop solution.

In this paper, we mitigate the existing research limitations and advance general decision-making for cooperative multi-agent systems. At a high level, COMPASS combines a vision language models (VLMs)-based planner with a dynamic skill library for storing and retrieving complex behaviors, along with a structured communication protocol. A diagram of COMPASS is provided in Figure 1. Inspired by Cradle [43], the VLM-based planner perceives the visual and textual observation and suggests the most suitable executable code from the skill library. We adopt the code-as-policy paradigm [49] instead of task-specific primitive actions, as it constrains generalizability and fails to fully leverage foundation models' extensive world knowledge and sophisticated reasoning capabilities.

Traditional open-loop methods struggle to produce effective plans that adapt to dynamics in stochastic, partially observable environments. To address this challenge, the VLM-based planer attempts to solve challenging and ambiguous final tasks, such as "Defeat all enemy units in the StarCraft multi-agent combat scenario while coordinating with allied units", by progressively proposing a sequence of clear, manageable sub-tasks while incorporating environmental feedback and task progress. COMPASS generates Python scripts through LLMs as semantic-level skills to accomplish sub-tasks, incrementally building a skill library throughout the task progress. Each skill is indexed through its documentation embeddings, enabling retrieval based on task-skill relevance. However, developing the skill library from scratch requires extensive

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

exploration to discover viable strategies. In contrast, we pre-collect demonstration videos and introduce a "warm start" by initializing the skill library with strategies derived from the expert-level dataset.

Moreover, building autonomous agents to cooperate in completing tasks under partial observation requires an efficient communication protocol. However, naive communication leads to the risks of hallucination caused by meaningless chatter between agents [19]. Inspired by entity-based MARL [5, 14], we present a structured communication protocol to formulate the communication among agents along with a global memory that allows all agents to retrieve. The protocol incorporates a multi-hop propagation mechanism, enabling agents to infer information about entities beyond their field of view through information shared by teammates. Similar to previous approaches, each agent maintains a local memory to preserve current and historical experiences.

Empirically, COMPASS demonstrates effective adaptation and skill synthesis in cooperative multi-agent scenarios. Through its dynamic skill library, it creates reusable and interpretable code-based behaviors that evolve during task execution. We evaluate COMPASS systematically in the improved StarCraft Multi-Agent Challenge (SMACv2) using both open-source (Qwen2-VL-72B [48]) and closed-source (GPT-4o-mini[1], Claude-3-Haiku[2]) VLMs. COMPASS achieves strong results in Protoss scenarios with a 57% win rate, substantially outperforming state-of-the-art MARL algorithms, including QMIX [38], MAPPO [53], HAPPO [16], and HASAC [23]. COMPASS maintains moderate performance in Terran scenarios and handles asymmetric settings effectively, though showing limited success in Zerg task. We evaluate the contribution of individual COMPASS components to its overall performance. We further demonstrate COMPASS's ability to bootstrap effective strategies from expert demonstrations.

2 RELATED WORK

2.1 Agents in the StarCraft Multi-Agent Challenge

SMAC [39], a predominant cooperative MARL benchmark based on the StarCraft II real-time strategy game [44], focuses on decentralized micromanagement scenarios where each unit operates under decentralized execution with partial observability to defeat enemy units controlled by Starcraft II's built-in AI opponent. Previous research in SMAC resort to MARL which can be divided into two categories: 1) Online MARL: One line of representative research is value decomposition (VD) [38, 41, 47], which decomposes the centralized action-value function into individual utility functions. On the other hand, multi-agent policy gradient (MAPG) methods [13, 16, 18, 23, 33, 51, 53] extend single-agent policy gradient algorithm to multi-agent with coordination modeling. Researches such as MAPPO [53], HAPPO [16], and HASAC [23] combine trust region and maximum entropy with MARL in a non-trivial way respectively. To encourage coordination, communication methods [13, 24], sequential modeling methods [18, 51], and cooperative exploration methods [28, 33] have been proposed. 2) Offline MARL: Recent efforts such as MADT [31], ODIS [55], and MADiff [59] leverage data-driven training via pre-collected offline datasets to enhance policy training efficiency. However, the near-optimal performance of these existing approaches on SMAC highlights the benchmark's limited stochasticity and partial observability. To address these limitations, SMACv2 [6] introduces more complexity to necessitate decentralized closed-loop control policies. There have been some recent attempts [8, 20, 21, 30] to evaluate MARL algorithms on SMACv2, and the results confirm the complexity. However, current learning-based multi-agent methods are computationally inefficient and non-interpretable. In the quest to find methods that are sample-efficient and interpretable, LLM-SMAC [4] leverage LLMs to generate centralized decision tree code under global information in an open-loop framework. Unlike prior works, COMPASS integrates a Vision-Language Model (VLM) with each agent in a decentralized closed-loop manner under partial observability, improving both real-world applicability and scalability.

2.2 LLM-based Multi-Agent System

Based on the inspiring capabilities of LLMs, such as zero-shot planning and complex reasoning [1, 15, 50, 58], embodied single-agent researches have demonstrated the effectiveness of LLMs in solving complex long-horizon tasks [2, 27, 40, 43, 46, 49, 52]. Despite significant advances in single-agent applications, developing real-world multi-agent systems with foundation models remains challenging, primarily due to the nature of decentralized control under partial observability in multi-agent settings [36]. Most prior efforts [11, 29, 34, 54, 56] leverage a hierarchical framework with components like perception, communication, planning, execution, and memory to build multi-agent systems with collective behaviors. These approaches can be roughly classified into two groups. 1) Centralized plan: MindAgent [11] adopts a centralized planning scheme with a pre-defined oracle in a fully observable multi-agent game. LLaMAR [34] employs LLMs to manage long-horizon tasks in partially observable environments without assumptions about access to perfect low-level policies. 2) Decentralized plan: ProAgent [54] introduces Theory of Mind (ToM), enabling agents to reason about others' mental states. RoCo [29] and CoELA [56] assign separate LLMs to each embodied agent for collaboration with communication. However, RoCo and CoELA assume a skill library with a low-level heuristic controller, which is impractical in real-world applications. Moreover, RoCo's open-loop plan-and-execute paradigm fails to incorporate environmental feedback during decision-making. In contrast, our work does not assume any pre-defined low-level controller and generates code-based action through VLMs in a closed-loop manner.

3 PRELIMINARIES

We model a fully cooperative multi-agent game with N agents as a *decentralized partially observable Markov decision process* (Dec-POMDP) [35], which is formally defined as a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{S}, \mathcal{O}, \mathbb{O}, \mathcal{B}, \mathcal{A}, \mathcal{T}, \Omega, R, \gamma, \rho_0)$. $\mathcal{N} = \{1, \ldots, N\}$ is a set of agents, $s \in \mathcal{S}$ denotes the state of the environment and ρ_0 is the distribution of the initial state. $\mathcal{A} = \prod_{i=1}^{N} A^i$ is the joint action space, $\mathbb{O} = \prod_{i=1}^{N} O^i$ is the set of joint observations. At time step t, each agent i receives an individual partial observation $o_t^i \in O^i$ given by the observation function $O : (a_t, s_{t+1}) \mapsto P(o_{t+1}|a_t, s_{t+1})$ where a_t, s_{t+1} and

[1] https://platform.openai.com/docs/models#gpt-4o-mini
[2] https://www.anthropic.com/news/claude-3-haiku

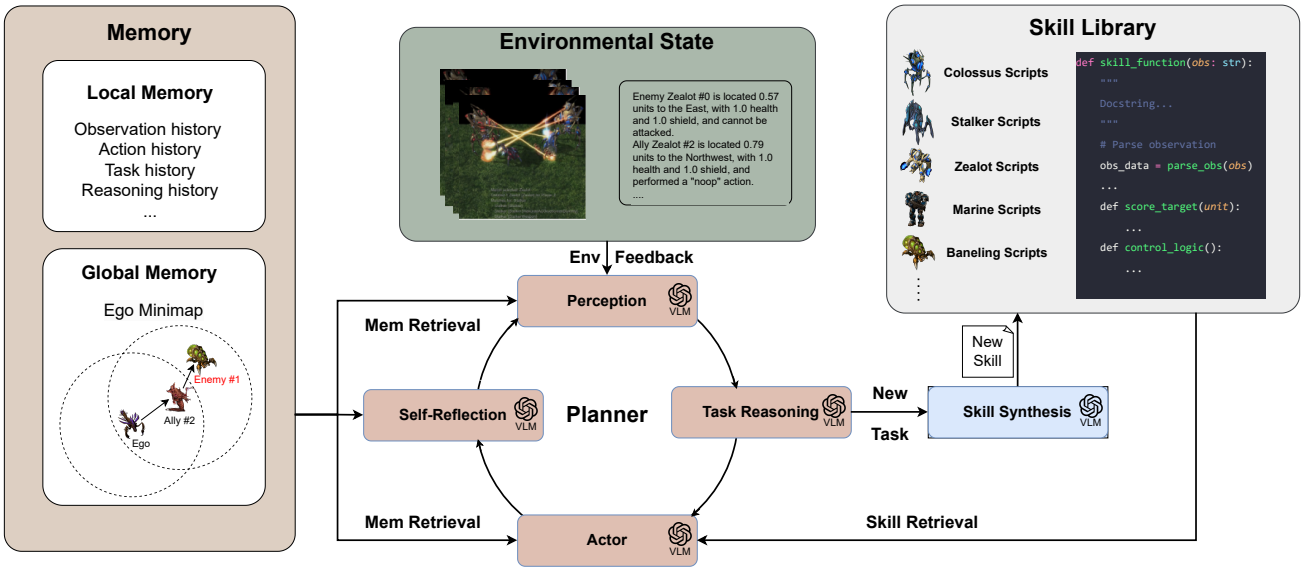


Figure 1: Overview of the COMPASS architecture, a novel framework that advances cooperative multi-agent decision-making through three synergistic components: (1) A VLM-based closed-loop planner that enables decentralized control by continuously processing multi-modal feedback and adapting strategies, addressing the non-Markovian challenge of multi-agent systems; (2) A dynamic skill synthesis mechanism that combines demonstration bootstrapping with incremental skill generation, improving sample efficiency and interpretability; and (3) A structured communication protocol that facilitates efficient information sharing through entity-based multi-hop propagation, enhancing cooperative perception under partial observability.

o_{t+1} are the joint actions, states and joint observations respectively. Each agent i uses a stochastic policy $\pi^i(a_t^i|h_t^i, \omega_t^i)$ conditioned on its action-observation history $h_t^i = (o_0^i, a_0^i, \ldots, o_{t-1}^i, a_{t-1}^i)$ and a random seed $\omega_t^i \in \Omega_t$ to choose an action $a_t^i \in A^i$. A belief state b_t is a probability distribution over states at time t, where $b_t \in \mathcal{B}$, and $\mathcal{B}$ is the space of all probability distributions over the state space. Actions a_t drawn from joint policy $\pi(a_t|s_t, \omega_t)$ conditioned on state s_t and joint random seed $\omega_t = (\omega_t^1, \ldots, \omega_t^N)$ change the state according to transition function $\mathcal{T} : (s_t, a_t^1, \ldots, a_t^N) \mapsto P(s_{t+1}|s_t, a_t^1, \ldots, a_t^N)$. All agents share the same reward $r_t = R(s_t, a_t^1, \ldots, a_t^N)$ based on s_t and a_t. γ is the discount factor for future rewards. Agents try to maximize the expected total reward, $\mathcal{J}(\pi) = \mathbb{E}_{s_0,a_0,\ldots}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t, \omega_t)$.

4 METHODS

COMPASS, illustrated in Figure 1, is a decentralized closed-loop framework for cooperative multi-agent systems that continuously incorporate environmental feedback for strategy refinement. The architecture comprises three core components: 1) a VLM-based closed-loop planner that iteratively perceives, reasons, reflects and acts to adaptively complete tasks (Sec. 4.1); 2) an adaptive skill synthesis mechanism for generating executable codes tailored to proposed sub-tasks (Sec. 4.2); and 3) a structured communication protocol that enables agents to share visible entity information under partial observability (Sec. 4.3). The pseudo-code of COMPASS is shown in Appendix.

4.1 VLM-based Closed-Loop Planner

Inspired by recent advances in cognitive architectures for autonomous systems [43], COMPASS implements a sophisticated modular planning framework that emulates key aspects of cognitive decision-making. The planner adopts a modular formulation, utilizing four specialized models: Perception, Task Reasoning, Self-Reflection, and Actor. Each model fulfills a distinct yet interconnected role in the decision-making process. The Perception model processes multi-modal inputs, integrating both visual and textual information to build comprehensive environmental understanding. The Task Reasoning model analyzes the perceived information to decompose complex objectives into manageable sub-tasks, ensuring systematic progress toward the final goal. The Self-Reflection model continuously evaluates task execution and outcome quality, enabling adaptive behavior refinement. The Actor model translates plans into actions by selecting and executing the most appropriate skills from the skill library. We next discuss the various components in detail:

Perception forms the foundation of COMPASS's decision-making capabilities by enabling robust multi-modal understanding of complex environments. Solving complex real-world tasks often involves data of multiple modalities [45], each contributing unique and complementary information for decision-making. We leverage the VLMs' ability to fuse and analyze a broader spectrum of data, including text- and image-based environment feedback, to enable agents to sense the surrounding environment. The system's perception mechanism operates at two levels: direct observation processing

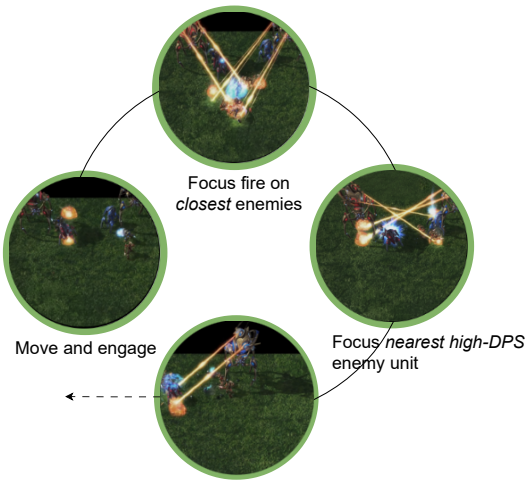


Figure 2: Visualization of COMPASS's dynamic task reasoning process in the StarCraft Multi-Agent Challenge (SMACv2) environment. The figure demonstrates how the VLM-based planner decomposes a complex final goal ("defeat all enemy units") into a sequence of concrete, executable sub-tasks that adapt to the changing battlefield conditions. This closed-loop task decomposition enables efficient coordination among multiple agents under partial observability, as each sub-task provides clear, actionable objectives that agents can execute while maintaining overall mission alignment.

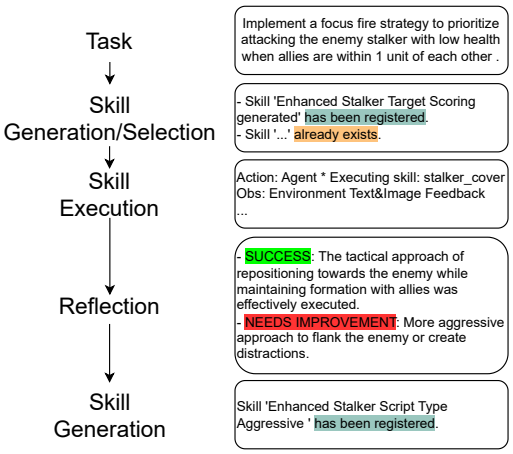


Figure 3: Dynamic skill evolution through self-reflection. COMPASS continuously refines its tactical capabilities by analyzing the performance of executed skills. Here, feedback on the 'stalker_cover' skill reveals a need for more aggression. This insight prompts the immediate generation and registration of a new, specialized skill ('Enhanced Stalker Script Type Aggressive'), expanding the agent's behavioral repertoire.

and collaborative information synthesis. At the direct level, VLMs process raw inputs to extract meaningful features and relationships from both visual and textual data. At the collaborative level, COMPASS addresses the inherent challenge of partial observability in multi-agent systems through an innovative multi-hop communication protocol (detailed in the Structured Communication Protocol section) that enables agents to construct a more holistic understanding of their environment by sharing and aggregating observations. This dual-level perception architecture ensures that each agent maintains both detailed local awareness and broader contextual understanding, essential for effective decision-making in complex cooperative tasks.

Task Reasoning enables COMPASS to systematically approach complex cooperative challenges through collective task decomposition. Given a simple general final task in the cooperative multi-agent setting, e.g., "defeat all enemy units", in order to complete the task more efficiently, agents are required to decompose it into multiple sub-tasks and figure out the right one to focus on, while considering alignment among others (See Figure 2). COMPASS harnesses the power of VLMs to analyze high-level task instructions in conjunction with environmental feedback and team member objectives to generate tractable sub-tasks that collectively advance the overall mission. As agents act under stochastic, partially observable environments, the task reasoning model continuously adapts its plans, proposing and refining sub-tasks based on emerging situations and progress assessment. This dynamic approach enables COMPASS to maintain strategic coherence while adjusting tactical decisions in response to changing circumstances.

Actor serves as the critical bridge between high-level reasoning and concrete action execution. Building upon recent advances in code-writing language models for embodied control [22, 49], the Actor leverages the skill library by first identifying relevant skills for the proposed sub-task, then synthesizes perception and self-reflection inputs to select the optimal skill for execution. This streamlined approach ensures efficient skill selection while maintaining task alignment.

Self-Reflection enables COMPASS to continuously evaluate and refine its decision-making processes through systematic performance analysis (See Figure 3). COMPASS instantiates the Self-Reflection model as a VLM which takes a sequence of visual results from the last skill execution with corresponding descriptions as input to assess the quality of the decision produced by the Actor and whether the task was completed. Additionally, we also request the VLM to generate verbal self-reflections to provide valuable feedback on the completion of the task.

4.2 Adaptive Skill Synthesis

COMPASS employs a dynamic skill library that maintains and evolves a collection of executable behaviors. Each skill is represented as an executable Python function with comprehensive documentation describing its functionality and corresponding embedding that enables semantic retrieval. This skill library undergoes continuous refinement through two complementary mechanisms (Figure 4): incremental synthesis, where new skills are generated and existing ones are refined during task execution, and

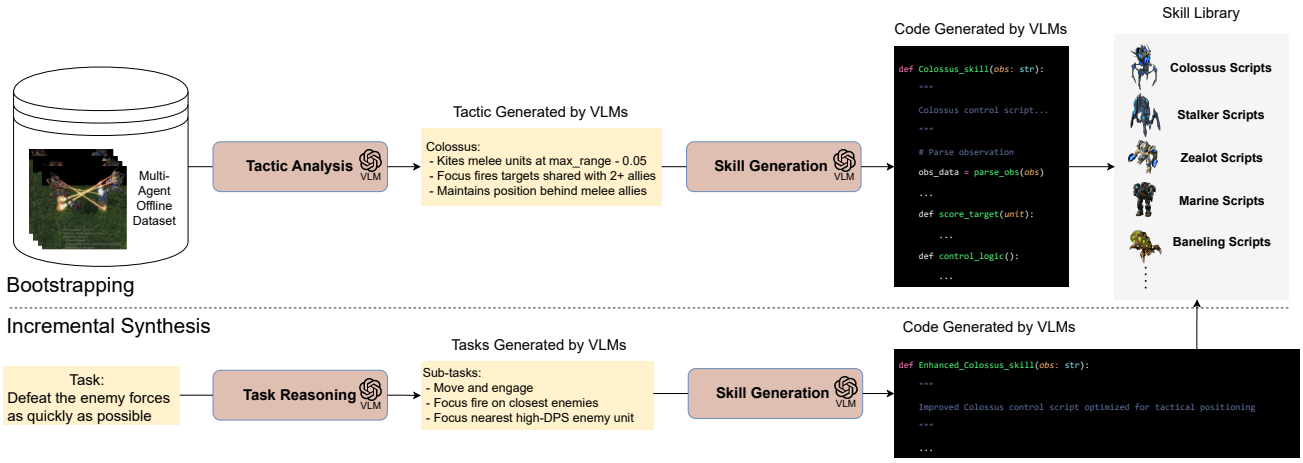


Figure 4: Overview of Adaptive Skill Synthesis. VLMs perform (Top) Bootstrapping by analyzing offline data for initial Tactic Analysis and Skill Generation into a Skill Library. (Bottom) Incremental Synthesis uses Task Reasoning to dynamically generate or enhance code-based skills, evolving the library for new tasks. The skills follow a structured decision-making pipeline with two core components: *score_target(unit)* for dynamic target prioritization and *control_logic()* for coordinating behavior. Textual observations are parsed into structured data (obs_data), mapping raw text to attributes, e.g., "Can move North: yes" to can_move='north': True.

demonstration-based bootstrapping, which initializes the library with behaviors extracted from expert demonstrations.

Incremental Synthesis With the Task Reasoning component consistently proposing sub-tasks, COMPASS first attempts to retrieve relevant skills from the library using semantic similarity between the sub-task description and skill documentation embeddings. If no suitable skill exists, or if existing skills prove inadequate, the VLM generates a new Python script specifically tailored to the sub-task.

Bootstrapping However, developing the skill library from scratch requires extensive interactions with environments, which potentially leads to inefficient learning in the early stages. Inspired by offline MARL approaches [31, 55, 59], which leverage pre-collected datasets to enhance sample efficiency, we leverage MAPPO as the behavior policy to collect experiences, which are recorded as video sequences. The VLMs then analyze these demonstrations through a multi-stage process: first identifying key strategic patterns and behavioral primitives, then translating these patterns into executable Python functions with appropriate documentation. This initialization methodology establishes a foundational set of validated skills, substantially reducing the exploration overhead typically required for discovering effective behaviors. The resulting baseline skill library enables efficient task execution from the onset while maintaining the flexibility to evolve through incremental synthesis.

Skill Analysis. We now analyze COMPASS's capability to synthesize and execute diverse tactical behaviors. COMPASS develops four key tactical patterns: (1) An exponentially-scaled focus fire implementation that coordinates multiple units' target selection based on allied attacker density (Figure 6), (2) A position-aware kiting

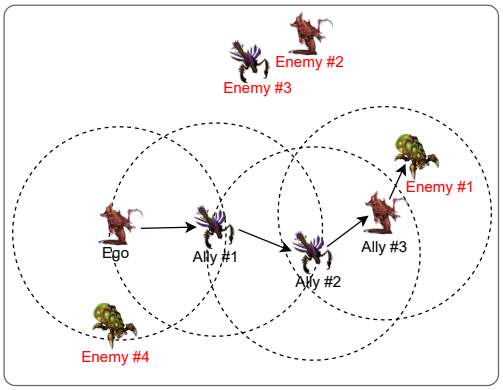


Figure 5: Illustration of COMPASS's structured multi-hop communication protocol that enables efficient information sharing under partial observability. The figure demonstrates how information about Enemy #1 propagates to the Ego agent through a chain of allied units (Ally #1, #2, #3), despite Enemy #1 being outside Ego's sight range. Each dashed circle represents an agent's local observation field, while arrows indicate the flow of entity-based information sharing. This mechanism enables agents to build a more holistic understanding of the environment by propagating critical information (e.g., enemy positions, status) through intermediate allies, effectively addressing the partial observability challenge in decentralized multi-agent systems.

mechanism that maintains optimal engagement ranges while managing unit positioning relative to threats (Figure 7), (3) A formation-based isolation tactic that enables systematic target elimination through coordinated unit movements (Figure 8), and (4) An area-of-effect (AOE) tactic that maximizes splash damage through cluster density calculation (Figure 9). These synthesized skills exhibit clear strategic intent while maintaining interpretability.

4.3 Structured Communication Protocol

To facilitate effective collaboration under partial observability, recent LLM-based multi-agent work [19, 56] employs conversational framework with unconstrained communication protocol. However, while natural language offers flexibility, unrestricted communication can lead to potential hallucinations caused by ambiguous or irrelevant messages between agents. Drawing from advances in structured communication frameworks [12] and entity-based MARL [5, 14], COMPASS implements a hierarchical communication protocol that focuses on efficient entity-based information sharing and multi-hop propagation (Figure 5). Each agent maintains an observation buffer containing information about entities in its field of view. At each timestep, agents share their local observations, which are then aggregated into a global entity memory accessible to all. COMPASS employs a multi-hop communication mechanism to propagate information about distant entities, enabling agents to build a more holistic observation of the environment by leveraging the collective knowledge of the team.

5 EXPERIMENTS

We conducted a comprehensive experimental evaluation of COMPASS to assess its performance and capabilities in complex multi-agent scenarios. Our evaluation focused on the improved StarCraft Multi-Agent Challenge (SMACv2) [6], which provides an ideal testbed for examining cooperative behavior under partial observability and stochasticity. Through systematic experimentation, we investigated two fundamental questions: (1) How does COMPASS perform compared to state-of-the-art MARL methods? (2) What are the individual contributions of each component in COMPASS? Experiments utilize both open-source (Qwen2-VL-72B) and closed-source VLMs (GPT-4o-mini, Claude-3-Haiku), with Jina AI embeddings for skill retrieval. All results are averaged over 5 seeds to account for environmental stochasticity. Token usage is approximately 0.4 million per episode.

5.1 Experimental Setup

Scenarios Our evaluation scenarios span three distinct race matchups (Protoss, Terran, and Zerg) and two categories (symmetric and asymmetric), as detailed in Appendix. The symmetric scenarios (5v5) test coordination in balanced engagements, while asymmetric scenarios (5v6) evaluate adaptation to numerical disadvantages. Each race combination presents unique tactical challenges due to different unit abilities and constraints. We followed the setting p=0 in the SMACv2 original paper (i.e., prob_obs_enemy: 0.0 in the .yaml file), meaning that only the first agent to initially spot a specific enemy unit can continue observing it, introducing the *Extended Partial Observability Challenge*, which baselines struggled with.

Table 1: Evaluation scenarios spanning three SMACv2 race matchups under symmetric equal-force and asymmetric outnumbered configurations.

Task	Scenarios	Categories
Protoss	protoss 5 vs 5	symmetric
	protoss 5 vs 6	asymmetric
Terran	terran 5 vs 5	symmetric
	terran 5 vs 6	asymmetric
Zerg	zerg 5 vs 5	symmetric
	zerg 5 vs 6	asymmetric

Baselines We compared COMPASS against the state-of-the-art MARL algorithms representing both value-based and policy-gradient approaches:

- Value-Based Methods: QMIX [38] uses a mixing network architecture to decompose joint action-values while maintaining monotonicity constraints.
- Policy Gradient Methods: MAPPO [53] extends PPO to multi-agent settings with the CTDE paradigm. HAPPO [16] performs sequential policy updates by utilizing other agents' newest policy under the CTDE framework and provably obtains the monotonic policy improvement guarantee. HASAC [23] combines the maximum entropy framework with trust region optimization to enhance exploration and coordination. MAT[51] models the multi-agent decision process as a sequence-to-sequence generation problem with a powerful transformer architecture.
- Communication-based Methods: CommFormer [13] is a method for learning optimal communication graphs in multi-agent systems using attention.
- Offline Methods: Oryx [9] is a state-of-the-art offline MARL algorithm, addressing extrapolation error and miscoordination.
- LLM-based Methods: LLM-SMAC [4] generates decision tree code using Large Language Models (GPT-4o-mini) to solve the SMAC task. We modified its codebase for SMACv2, but the resulting win rates were 0% across all settings. Given this performance, we ignore this baseline in our final comparison.

Datasets To enable effective bootstrapping of the skill library, we constructed a comprehensive demonstration dataset capturing diverse multi-agent strategies and interactions. We employed MAPPO with original hyper-parameters as our behavior policy for data collection, leveraging its strong performance in cooperative multi-agent tasks. Our final dataset comprises over 300 complete game episodes, each recorded as a video sequence capturing the full state-action trajectory. These demonstrations span all symmetric scenario types described in Appendix.

5.2 Main Results

Performance. Table 2 reveals substantial performance gains for COMPASS in SMACv2, with the clearest advantages emerging in Protoss scenarios. Using GPT-4o-mini, COMPASS achieves a 57% win rate in symmetric Protoss engagements, exceeding QMIX by 30 percentage points, MAPPO by 25 points, and HAPPO by 23 points.

```
score = base_priority
# Improved Focus Fire Logic with enhanced commitment
num_attackers = sum(1 for ally in obs_data.allies if ally.last_action >= 6 and ally.last_action - 6 == unit.id)

if num_attackers > 0:
    focus_bonus = 1.2 ** num_attackers  # Enhanced focus fire emphasis
    score *= focus_bonus
```

(a) Focus Fire Logic

(b) Protoss (c) Terran (d) Zerg

Figure 6: Focus Fire Logic Implementation. (a) VLM-generated Python code snippet implementing dynamic focus fire logic. The code prioritizes enemy units based on the number of allied attackers, scaling the attack bonus exponentially. (b–d) Visualizations of focus fire execution across Protoss, Terran, and Zerg.

Table 2: Comparative performance of COMPASS (with three VLM variants: G-4o=GPT-4o-mini, C-Hk=Claude-3-Haiku, Q2-VL=Qwen2-VL-72B) and state-of-the-art baselines on SMACv2. Median win rates (%) and standard deviations (subscripts) are reported across Protoss, Terran, and Zerg scenarios in symmetric (5v5) and asymmetric (5v6) categories. Results are averaged over 5 seeds. Bold values denote the best performance in each scenario. N/A* indicates no datasets available in these settings.

		Protoss		Terran		Zerg	
Method	**Type**	5v5	5v6	5v5	5v6	5v5	5v6
Online MARL							
QMIX	Online	$0.27_{0.03}$	$0.01_{0.01}$	$0.38_{0.04}$	$0.06_{0.02}$	$0.21_{0.01}$	$\mathbf{0.18_{0.03}}$
MAPPO	Online	$0.32_{0.07}$	$0.04_{0.04}$	$0.36_{0.10}$	$0.07_{0.06}$	$0.27_{0.04}$	$0.13_{0.09}$
HAPPO	Online	$0.34_{0.07}$	$0.02_{0.03}$	$0.35_{0.10}$	$0.01_{0.03}$	$0.20_{0.11}$	$0.09_{0.02}$
HASAC	Online	$0.20_{0.08}$	$0.01_{0.02}$	$0.29_{0.01}$	$0.05_{0.02}$	$0.24_{0.07}$	$0.08_{0.05}$
MAT	Online	$0.39_{0.03}$	$0.04_{0.04}$	$0.36_{0.11}$	$0.05_{0.01}$	$0.32_{0.06}$	$0.11_{0.08}$
CommFormer	Communication	$0.39_{0.16}$	$0.02_{0.01}$	$0.30_{0.09}$	$0.03_{0.01}$	$\mathbf{0.39_{0.10}}$	$0.16_{0.01}$
Offline MARL							
Oryx	Offline	N/A*	N/A*	$0.18_{0.04}$	N/A*	$0.10_{0.06}$	N/A*
VLM-based (Ours)							
COMPASS (G-4o)	VLM	$\mathbf{0.57_{0.08}}$	$\mathbf{0.08_{0.04}}$	$\mathbf{0.39_{0.01}}$	$\mathbf{0.10_{0.03}}$	$0.16_{0.07}$	$0.03_{0.01}$
COMPASS (C-Hk)	VLM	$0.49_{0.06}$	$0.06_{0.05}$	$0.38_{0.05}$	$\mathbf{0.10_{0.01}}$	$0.18_{0.02}$	$0.04_{0.01}$
COMPASS (Q2-VL)	VLM	$0.45_{0.04}$	$0.06_{0.03}$	$0.31_{0.02}$	$0.06_{0.03}$	$0.14_{0.03}$	$0.02_{0.01}$

The margin over MAT and CommFormer remains significant at 18 percentage points despite their stronger baseline performance.

Performance stratifies sharply across race matchups. Terran scenarios yield a 39% win rate, positioning COMPASS marginally above all baselines. Zerg scenarios expose a critical limitation: COMPASS attains only 16% win rate, falling below CommFormer at 39%. This disparity stems directly from Zerg unit mechanics. Banelings and Zerglings require continuous micromanagement at sub-second timescales due to their melee attack ranges and swarm coordination demands, while COMPASS queries the VLM every 10 to 20 timesteps. The mismatch between decision frequency and tactical requirements undermines combat effectiveness.

Asymmetric 5v6 scenarios test adaptation under numerical disadvantage. COMPASS maintains leads over MARL baselines in Protoss and Terran configurations, reaching 8% and 10% win rates respectively where most baselines achieve 1% to 7%. This suggests the skill library and structured communication enable tactical compensation for unit deficits. VLM choice affects outcomes moderately: GPT-4o-mini consistently outperforms Claude-3-Haiku and Qwen2-VL-72B by 2 to 8 percentage points across most scenarios, though all three variants follow the same performance ranking across races.

The Zerg 5v6 asymmetric case merits attention. COMPASS achieves only 2% to 4% win rate compared to QMIX at 18% and CommFormer at 16%. QMIX benefits from dense value function updates

Table 3: Win rates achieved by the bootstrapped skill library alone, without incremental synthesis during deployment. Skills were extracted from MAPPO demonstration trajectories and evaluated directly on SMACv2 test episodes.

	PROTOSS	TERRAN	ZERG
5V5	$0.35_{0.06}$	$0.24_{0.04}$	$0.06_{0.01}$
5V6	$0.04_{0.05}$	$0.06_{0.02}$	$0.02_{0.03}$

that capture rapid state changes, while COMPASS relies on periodic high-level replanning. When both numerical disadvantage and high-frequency control demands coincide, the VLM-based approach degrades substantially.

5.3 Ablation Studies

Skill Initialization To evaluate the impact of our skill initialization, we analyze the performance of COMPASS using only the initialized skill library derived from expert demonstrations. The results in Table 3 demonstrate that skill initialization alone achieves non-trivial performance across different scenarios, particularly in symmetric matchups. Moreover, the gap between initialized skills and COMPASS underscores the necessity of incremental skill synthesis. A script example for skill initialization is in Appendix.

Communication To demonstrate the critical role of communication, we evaluated COMPASS on Protoss 5v5 under the *Extended Partial Observability* setting, using only local information without multi-hop propagation. The resulting win rate with GPT-4o-mini decreased to $0.06_{0.04}$, a significant drop from 0.57 with full communication. This degradation occurs because the Extended Partial Observability setting restricts direct enemy visibility to the first agent that initially spots it. The VLMs generated control logic heavily relies on the presence of enemies in the local observation to determine engagement and targeting. Without communication relaying enemy positions, agents other than the discoverer cannot 'see' enemies known to teammates, even if within attack range. Consequently, their control logic frequently defaults to 'no enemy' behaviors, such as moving towards allies or executing random default actions, preventing effective target engagement and coordinated attacks, thus drastically reducing combat effectiveness and the overall win rate. We study the effect of a communication fault. As shown in the Table 4, the system degrades under moderate packet loss and suffers a drastic drop at >50% loss.

VLM call frequency We study the impact of VLM call frequency. As shown in the Table 5, at higher call frequencies, the performance remains similar but incurs greater cost, while lower frequencies fail to keep up with the dynamics of the battlefield.

Self Reflection In order to show the effectiveness of self-reflection, we evaluate the performance of COMPASS w/o self-reflection on protoss 5 vs 5. Removing the module leads to a drop of 10% ($0.47_{0.04}$).

Visual information We tested visual information contribution by omitting image inputs. This forces agents to rely solely on textual information, eliminating visual grounding for spatial understanding. Performance drops 10% without visual input. Analysis of VLM outputs reveals that absent visual information, the VLM must infer map boundaries from textual cues such as 'West (unavailable movement)' that encode action constraints rather than explicit spatial geometry. Visual input enables direct perception of these spatial details from the image. The resulting degradation in spatial awareness produces suboptimal tactical decisions for movement and positioning.

Table 4: Communication protocol resilience on Protoss 5v5 scenarios. Top: win rates increase with propagation depth as agents access information beyond direct observation range. Bottom: performance degrades gracefully under moderate packet loss but collapses beyond 50% loss when multi-hop propagation fails.

Setting	Specification	Win Rate
Hop Count	1-hop	$0.46_{0.19}$
	2-hop	$0.54_{0.06}$
	3-hop	$\mathbf{0.57_{0.08}}$
Packet Loss	20%	$\mathbf{0.32_{0.03}}$
	50%	$0.12_{0.02}$
	80%	$0.07_{0.05}$
	100%	$0.06_{0.04}$

Table 5: VLM query frequency ablation on Protoss 5v5 scenarios with average episode length of 60 steps. Performance remains stable between 10-step and 20-step intervals but degrades at 40-step intervals when replanning cannot track battlefield dynamics.

Frequency	Win Rate
Every 10 steps	$0.56_{0.05}$
Every 20 steps	$\mathbf{0.57_{0.08}}$
Every 40 steps	$0.40_{0.08}$

6 CONCLUSION

COMPASS demonstrates that vision-language models can generate interpretable tactical behaviors for cooperative multi-agent control through skill synthesis and structured communication under partial observability. The framework achieves a 57% win rate on symmetric Protoss scenarios in SMACv2, exceeding the QMIX baseline by 30 percentage points and outperforming all tested MARL methods. However, performance collapses on Zerg scenarios to 16% win rate, falling below several baselines including CommFormer at 39%. This failure exposes a fundamental constraint: VLM query intervals of 10 to 20 timesteps cannot provide the continuous low-level control required for melee swarm units with sub-second tactical windows. The results indicate that VLM-based planning offers advantages for scenarios where strategic coordination dominates and where actions retain validity across multiple timesteps, but the approach remains unsuitable for domains requiring rapid reactive control. Future work must either reduce VLM inference latency by orders of magnitude or develop hybrid architectures that delegate high-frequency decisions to learned reactive policies while reserving VLMs for strategic supervision.

REFERENCES

[1] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 16 (Mar. 2024), 17682–17690. https://doi.org/10.1609/aaai.v38i16.29720

[2] brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *6th Annual Conference on Robot Learning*. https://openreview.net/forum?id=bdHkMjBJG_w

[3] Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. 2019. *On the utility of learning about humans for human-AI coordination.* Curran Associates Inc., Red Hook, NY, USA.

[4] Yue Deng, Weiyu Ma, Yuxin Fan, Yin Zhang, Haifeng Zhang, and Jian Zhao. 2024. A New Approach to Solving SMAC Task: Generating Decision Tree Code from Large Language Models. arXiv:2410.16024 [cs.AI] https://arxiv.org/abs/2410.16024

[5] Ziluo Ding, Wanpeng Zhang, Junpeng Yue, Xiangjun Wang, Tiejun Huang, and Zongqing Lu. 2023. Entity Divider with Language Grounding in Multi-Agent Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 8103–8119. https://proceedings.mlr.press/v202/ding23d.html

[6] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. 2024. SMACv2: an improved benchmark for cooperative multi-agent reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1634, 27 pages.

[7] Yuming Feng, Chuye Hong, Yaru Niu, Shiqi Liu, Yuxiang Yang, Wenhao Yu, Tingnan Zhang, Jie Tan, and Ding Zhao. 2024. Learning Multi-Agent Loco-Manipulation for Long-Horizon Quadrupedal Pushing. arXiv:2411.07104 [cs.RO] https://arxiv.org/abs/2411.07104

[8] Claude Formanek, Asad Jeewa, Jonathan Shock, and Arnu Pretorius. 2023. Off-the-Grid MARL: Datasets and Baselines for Offline Multi-Agent Reinforcement Learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems* (London, United Kingdom) *(AAMAS '23)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2442–2444.

[9] Juan Claude Formanek, Omayma Mahjoub, Louay Ben Nessir, Sasha Abramowitz, Ruan John de Kock, Wiem Khlifi, Daniel Rajaonarivonivelomanantsoa, Simon Verster Du Toit, Arnol Manuel Fokam, Siddarth Singh, et al. [n.d.]. Oryx: a Scalable Sequence Model for Many-Agent Coordination in Offline MARL. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[10] Jiawei Gao, Ziqin Wang, Zeqi Xiao, Jingbo Wang, Tai Wang, Jinkun Cao, Xiaolin Hu, Si Liu, Jifeng Dai, and Jiangmiao Pang. 2024. CooHOI: Learning Cooperative Human-Object Interaction with Manipulated Object Dynamics. arXiv:2406.14558 [cs.RO] https://arxiv.org/abs/2406.14558

[11] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Yusuke Noda, Zane Durante, Zilong Zheng, Demetri Terzopoulos, Li Fei-Fei, Jianfeng Gao, and Hoi Vo. 2024. MindAgent: Emergent Gaming Interaction. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 3154–3183. https://doi.org/10.18653/v1/2024.findings-naacl.200

[12] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VtmBAGCN7o

[13] Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. 2024. Learning Multi-Agent Communication from Graph Modeling Perspective. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=Qox9rO0kN0

[14] Shariq Iqbal, Christian A Schroeder De Witt, Bei Peng, Wendelin Boehmer, Shimon Whiteson, and Fei Sha. 2021. Randomized Entity-wise Factorization for Multi-Agent Reinforcement Learning. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 4596–4606. https://proceedings.mlr.press/v139/iqbal21a.html

[15] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2024. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1613, 15 pages.

[16] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. 2022. Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning. arXiv:2109.11251 [cs.AI] https://arxiv.org/abs/2109.11251

[17] Karol Kurach, Anton Raichuk, Piotr Stanczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. 2020. Google Research Football: A Novel Reinforcement Learning Environment. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (Apr. 2020), 4501–4510. https://doi.org/10.1609/aaai.v34i04.5878

[18] Chuming Li, Jie Liu, Yinmin Zhang, Yuhong Wei, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. 2023. ACE: Cooperative Multi-Agent Q-learning with Bidirectional Action-Dependency. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 7 (Jun. 2023), 8536–8544. https://doi.org/10.1609/aaai.v37i7.26028

[19] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2024. CAMEL: communicative agents for "mind" exploration of large language model society. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2264, 18 pages.

[20] Zhiyuan Li, Wenshuai Zhao, Lijun Wu, and Joni Pajarinen. 2024. AgentMixer: Multi-Agent Correlated Policy Factorization. arXiv:2401.08728 [cs.MA] https://arxiv.org/abs/2401.08728

[21] Zhiyuan Li, Wenshuai Zhao, Lijun Wu, and Joni Pajarinen. 2024. Backpropagation Through Agents. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 12 (Mar. 2024), 13718–13726. https://doi.org/10.1609/aaai.v38i12.29277

[22] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as Policies: Language Model Programs for Embodied Control. arXiv:2209.07753 [cs.RO] https://arxiv.org/abs/2209.07753

[23] Jiarong Liu, Yifan Zhong, Siyi Hu, Haobo Fu, QIANG FU, Xiaojun Chang, and Yaodong Yang. 2024. Maximum Entropy Heterogeneous-Agent Reinforcement Learning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=tmqOhBC4a5

[24] Yat Long Lo, Biswa Sengupta, Jakob Nicolaus Foerster, and Michael Noukhovitch. 2024. Learning Multi-Agent Communication with Contrastive Learning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=vZZ4hhniJU

[25] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6382–6393.

[26] Chengdong Ma, Aming Li, Yali Du, Hao Dong, and Yaodong Yang. 2024. Efficient and scalable reinforcement learning for large-scale network control. *Nature Machine Intelligence* 6, 9 (2024), 1006–1020.

[27] Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. 2024. Large Language Models Play StarCraft II: Benchmarks and A Chain of Summarization Approach. arXiv:2312.11865 [cs.AI] https://arxiv.org/abs/2312.11865

[28] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. 2019. *MAVEN: multi-agent variational exploration.* Curran Associates Inc., Red Hook, NY, USA.

[29] Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 286–299. https://doi.org/10.1109/ICRA57147.2024.10610855

[30] Joshua McClellan, Naveed Haghani, John Winder, Furong Huang, and Pratap Tokekar. 2024. Boosting Sample Efficiency and Generalization in Multi-agent Reinforcement Learning via Equivariance. arXiv:2410.02581 [cs.LG] https://arxiv.org/abs/2410.02581

[31] Linghui Meng, Muning Wen, Chenyang Le, Xiyun Li, Dengpeng Xing, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, Yaodong Yang, et al. 2023. Offline pre-trained multi-agent decision transformer. *Machine Intelligence Research* 20, 2 (2023), 233–248.

[32] Claire Bizon Monroc, Ana Busic, Donatien Dubuc, and Jiamin Zhu. 2024. WFCRL: A Multi-Agent Reinforcement Learning Benchmark for Wind Farm Control. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=ZRMAhpZ3ED

[33] Hyungho Na, Yunkyeong Seo, and Il chul Moon. 2024. Efficient Episodic Memory Utilization of Cooperative Multi-Agent Reinforcement Learning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=LjivA1SLZ6

[34] Siddharth Nayak, Adelmo Morrison Orozco, Marina Ten Have, Vittal Thirumalai, Jackson Zhang, Darren Chen, Aditya Kapoor, Eric Robinson, Karthik Gopalakrishnan, James Harrison, Brian Ichter, Anuj Mahajan, and Hamsa Balakrishnan. 2025. LLaMAR: Long-Horizon Planning for Multi-Agent Robots in Partially Observable Environments. arXiv:2407.10031 [cs.RO] https://arxiv.org/abs/2407.10031

[35] Frans A. Oliehoek and Christopher Amato. 2016. *A Concise Introduction to Decentralized POMDPs* (1st ed.). Springer Publishing Company, Incorporated.

[36] Joni Pajarinen and Jaakko Peltonen. 2011. Efficient planning for factored infinite-horizon DEC-POMDPs. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One* (Barcelona, Catalonia, Spain) *(IJCAI'11)*. AAAI Press, 325–331.

[37] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. VirtualHome: Simulating Household Activities via Programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[38] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Journal of Machine Learning Research* 21, 178 (2020), 1–51. http://jmlr.org/papers/v21/20-081.html

[39] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (Montreal QC, Canada) *(AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2186–2188.

[40] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 8634–8652. https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf

[41] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 5887–5896. https://proceedings.mlr.press/v97/son19a.html

[42] Kefan Su and Zongqing Lu. 2024. A Fully Decentralized Surrogate for Multi-Agent Policy Optimization. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=MppUW90uU2

[43] Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. 2024. Cradle: Empowering Foundation Agents Towards General Computer Control. arXiv:2403.03186 [cs.AI] https://arxiv.org/abs/2403.03186

[44] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. 2017. StarCraft II: A New Challenge for Reinforcement Learning. arXiv:1708.04782 [cs.LG] https://arxiv.org/abs/1708.04782

[45] Chao Wang, Stephan Hasler, Daniel Tanneberg, Felix Ocker, Frank Joublin, Antonello Ceravola, Joerg Deigmoeller, and Michael Gienger. 2024. LaMI: Large Language Models for Multi-Modal Human-Robot Interaction. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 218, 10 pages. https://doi.org/10.1145/3613905.3651029

[46] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=ehfRiF0R3a

[47] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. 2021. {QPLEX}: Duplex Dueling Multi-Agent Q-Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Rcmk0xxIQV

[48] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. arXiv:2409.12191 [cs.CV] https://arxiv.org/abs/2409.12191

[49] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable Code Actions Elicit Better LLM Agents. arXiv:2402.01030 [cs.CL] https://arxiv.org/abs/2402.01030

[50] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[51] Muning Wen, Jakub Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. 2022. Multi-Agent Reinforcement Learning is a Sequence Modeling Problem. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 16509–16521. https://proceedings.neurips.cc/paper_files/paper/2022/file/69413f87e5a34897cd010ca698097d0a-Paper-Conference.pdf

[52] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

[53] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and YI WU. 2022. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24611–24624. https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf

[54] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. 2025. ProAgent: building proactive cooperative agents with large language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'24/IAAI'24/EAAI'24)*. AAAI Press, Article 1962, 9 pages. https://doi.org/10.1609/aaai.v38i16.29710

[55] Fuxiang Zhang, Chengxing Jia, Yi-Chen Li, Lei Yuan, Yang Yu, and Zongzhang Zhang. 2022. Discovering generalizable multi-agent coordination skills from multi-task offline data. In *The Eleventh International Conference on Learning Representations*.

[56] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024. Building Cooperative Embodied Agents Modularly with Large Language Models. arXiv:2307.02485 [cs.AI] https://arxiv.org/abs/2307.02485

[57] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. 2018. Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 5872–5881. https://proceedings.mlr.press/v80/zhang18n.html

[58] Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1387, 21 pages.

[59] Zhengbang Zhu, Minghuan Liu, Liyuan Mao, Bingyi Kang, Minkai Xu, Yong Yu, Stefano Ermon, and Weinan Zhang. 2025. MADiff: Offline Multi-agent Learning with Diffusion Models. arXiv:2305.17330 [cs.AI] https://arxiv.org/abs/2305.17330

A PSEUDOCODE

The pseudo-code of the COMPASS algorithm is shown in Pseudocode 1.

Algorithm 1 COMPASS Agent Decision-Making Loop

```
Initialize:
skill_manager.bootstrap(demonstration_data)
agent_state ← environment.reset(agent_id)
local_memory ← initialize_local_memory()
global_memory ← initialize_global_memory()
previous_action_result ← None
while True do
  {1. Communication Phase}
  local_observations ← agent_state.get_observations()
  communication_protocol.share_local_observations(agent_id, local_observations, global_memory)
  global_entity_info ← communication_protocol.get_global_memory.update(global_memory)
  {2. Perception Phase}
  processed_state ← vlm_perception.process(raw_observation=agent_state,
      communication_data=global_entity_info, local_memory=local_memory)
  local_memory.update(processed_state)
  {3. Self-Reflection Phase}
  if previous_action_result ≠ None then
    reflection_feedback ← vlm_self_reflection.reflect(previous_action_result)
  end if
  {4. Task Reasoning Phase}
  sub_task ← vlm_task_reasoning.props_subtask(processed_state, overall_goal, reflection_feedback)
  {5. Skill Generation Phase}
  new_skill_code ← vlm_skill_generator.generate_skill(sub_task, processed_state)
  skill_manager.add_skill(new_skill_code, sub_task)
  {6. Actor Phase}
  relevant_skills ← skill_manager.retrieve_skills(sub_task)
  chosen_skill_code ← vlm_actor.select_skill(sub_task, relevant_skills, processed_state)
  {7. Execution Phase}
  (next_agent_state, reward, done, info) ← environment.step(agent_id, chosen_skill_code)
  agent_state ← next_agent_state
  previous_action_result ← (chosen_skill_code, info, reward, done)
  local_memory.add_action(chosen_skill_code)
  if done then
    break
  end if
end while
```

B IMPLEMENTATION DETAILS

COMPASS integrates VLMs to process multi-modal inputs and generate executable skills in two stages. Each skill follows a standardized interface:

Listing 1: Interface of Generated skills.

```
def skill_template(obs: str):
    obs_data = parse_obs(obs)
    def score_target(unit):
        ...
        return score
    def control_logic():
        ...
        return atomic_action
```

The skills follow a structured decision-making pipeline with two core components: score_target(unit) and control_logic():

- score_target(unit): Dynamically calculates a threat/priority score by evaluating unit type, health, distance, formations, and matchups to guide optimal attack/heal targeting decisions.
- control_logic(): Dynamically coordinates unit behavior by integrating observations, target priorities, and pathfinding to execute role-optimized strategies (e.g., stalkers attack colossus at max range while moving away from zealots).

COMPASS evolves skills through iterative refinement and task-guided synthesis:

- iterative refinement: When errors occur during skill execution, VLMs analyze the error messages and attempt to fix the bugs.
- task-guided synthesis: When a new task is proposed, VLMs first determine whether a new skill needs to be generated to align with the task. If necessary, VLMs generate new score_target or control_logic components to fulfill the task requirements and integrate them with the existing code to construct a new skill.

For example, if the task is: "*Implement an aggressive advance movement pattern for the colossus unit when enemy stalkers are within sight range and allies are positioned to provide covering fire*," and the current skills in the library are not aggressive enough, the VLMs will refine the control_logic to implement a more aggressive behavior pattern.

COMPASS maintains a global entity memory, where agents act as nodes connected if within sight range. Each node stores information about visible allies and enemies. When Agent A observes entities, it propagates updates through adjacent nodes recursively, up to max_hops (default = 3). This allows agents to infer off-screen threats via ally intermediaries. For implementation details, see ./common/memory/global_memory.py.

C ENVIRONMENT SETTINGS AND MORE RESULTS

Table 6: We report quantitative results on SMACv2 under sparse reward settings, excluding COMPASS due to its inherent insensitivity to reward sparsity.

	$QMIX_s$	$MAPPO_s$	$HAPPO_s$	$HASAC_s$
PROTOSS				
5V5	0	0	0	0
5V6	0	0	0	0
TERRAN				
5V5	0	0	0	0
5V6	0	0	0	0
ZERG				
5V5	$0.02_{0.01}$	0	0	0
5V6	0	0	0	0

D MORE SKILL ANALYSIS

E TOKEN USAGE ANALYSIS

We acknowledge the computational cost. We provide a breakdown of token usage per step per agent (call frequency: 20).

- Average Episode Total Tokens: 80,000
- Average Tokens per Step: 1,333 (Perception: 2k, Actor: 2k, Self-Reflection: 2k, Task Reasoning: 10k, Skill Synthesis: 10k)

The bottlenecks are Task Reasoning and Skill Synthesis, which require historical context and code history. Skill synthesis often requires multiple rounds of self-correction to generate executable code. Future work will focus on reducing this cost by exploiting structural symmetries among agents. We may also reduce token usage by appropriately lowering the frequency of queries.

F BASELINE TRAINING RESULTS

For QMIX, we utilized Epymarl[3], and for others, we used HARL[4]. Task difficulty was set to 5v5 for SYMMETRIC tasks and 5v6 for ASYMMETRIC tasks; the 5v6 scenario introduces a significant force disadvantage (20% outnumbered vs 10% in 10v11).

[3]https://github.com/uoe-agents/epymarl
[4]https://github.com/PKU-MARL/HARL

(a)

(b)

(c)

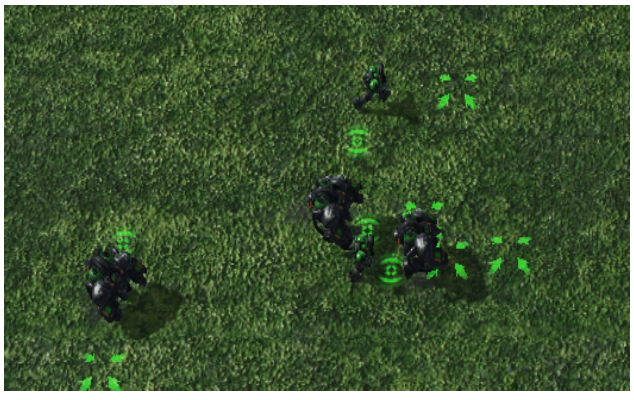
```
if closest_enemy.distance <= 4 / obs_data.own_sight_range
and obs_data.last_action >= 6:

    enemy_x = sum(e.position[0] for e in obs_data.enemies) /
len(obs_data.enemies)
    enemy_y = sum(e.position[1] for e in obs_data.enemies) /
len(obs_data.enemies)

    return find_path(obs_data, - enemy_x, - enemy_y)
```

(d) Kitting Logic

Figure 7: Illustration and implementation of Kitting logic. (a)-(c) demonstrate progressive stages of the kitting tactic where allied units strategically maintain optimal attack range while retreating from melee enemies. (d) shows the corresponding Python code snippet generated by the VLMs.



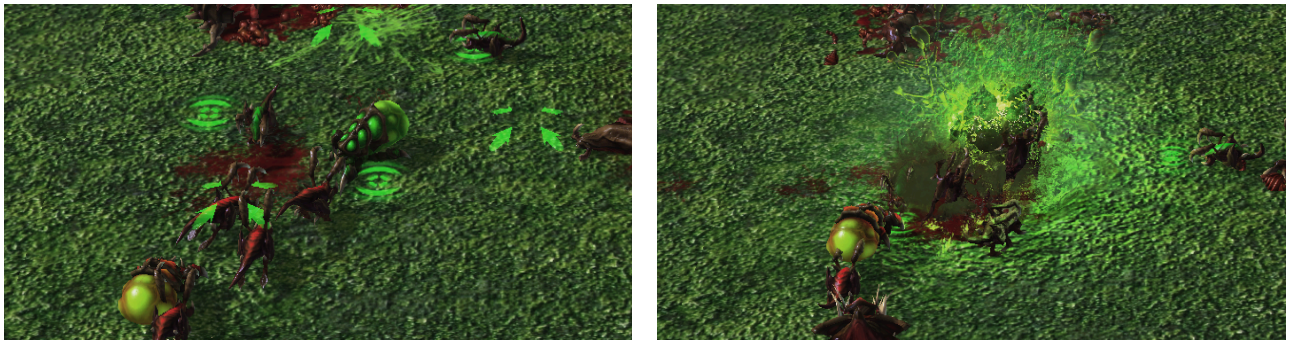
(a) Assemble

(b) Isolate

Figure 8: Illustration of Isolating logic. (a) Allied units strategically assemble into a cohesive formation. (b) The assembled units execute a rapid engagement against an isolated enemy unit, eliminating it before reinforcements can arrive, thus creating a numerical advantage.

```
enemy_clusters = {}
    for enemy in obs_data.enemies:
    nearby_enemies = []
    # Dynamic cluster radius based on unit type
    cluster_radius = 0.3 if obs_data.own_unit_type.lower() == 'baneling' else 0.2
    for other in obs_data.enemies:
    distance = ((other.position[0] - enemy.position[0])**2 + (other.position[1] - enemy.position[1])**2)**0.5
    if distance <= cluster_radius:
        nearby_enemies.append(other)
    enemy_clusters[enemy.id] = len(nearby_enemies)
```

(a) Baneling Cluster



(b) (c)

Figure 9: Demonstration of area-of-effect (AOE) optimization for Baneling units in SMACv2. (a) The VLM-generated Python code calculates optimal detonation positions by analyzing enemy cluster density and positions. (b-c) Visual sequence showing Baneling execution, where the unit identifies a dense cluster of enemy units and detonates for maximum AOE damage.

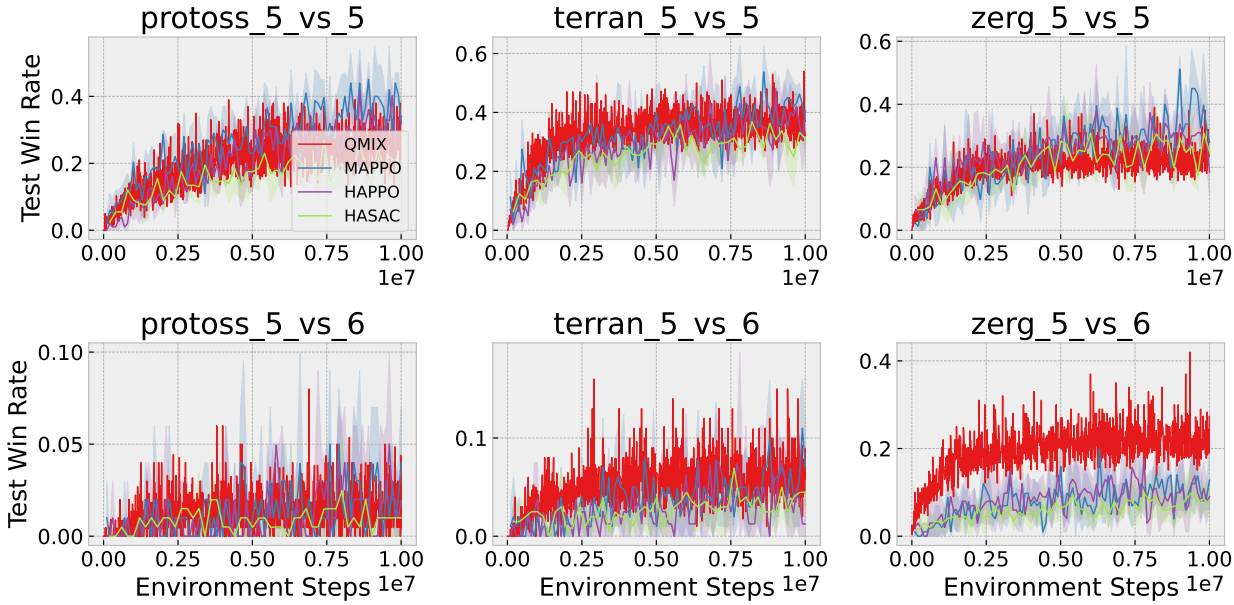


Figure 10: Baseline training results on SMACv2.

G PROMPTS USED IN COMPASS

Prompt for Perception

You are an AI assistant helping with academic research in the StarCraft II's SMAC (StarCraft Multi-Agent Challenge) environment, controlling a <unit_type> unit with ID <unit_id> in micromanagement scenarios <scenario_name> to help your team defeat the enemy forces. You operate under decentralized execution with partial observability, making decisions based only on local observations within your unit's field of view. Your advanced capabilities enable you to process and interpret gameplay screenshots and other relevant information.
I will give you the following information:
<few_shots>
Reasoning for the last episode:
<last_episode_reasoning>
Strategic situation analysis:
<info_summary>
Below is the current in-game screenshot and its description:
<image_introduction>
Minimap information:
<ego_minimap>
Current task:
<task_description>
Tactics recommendation:
<web_search>
Based on the above information, you should first analyze the current game situation by integrating the information from the in-game screenshot, its description, and other provided information.
Game situation:
You should think step by step and provide detailed reasoning to determine the current state of the game. You need to answer the following questions step by step:
1. What is your unit_id, unit type?
2. What map borders are you near? Check which cardinal directions (N/S/E/W) have unavailable movement actions.
3. What is the current health status of your unit? What is the current shield status of your unit?
4. Are there any enemy units visible, either in observation or minimap?
5. Are there any ally units visible, either observation or minimap?
6. Are you positioned at the optimal attack range from enemies, or do you need to reposition based on the enemies' locations and directions?
Region of interest:
What unit or location should be interacted with to complete the task based on the current screenshot and the current task? You should obey the following rules:
1. If your chosen region of interest is a unit, format the output as "[Enemy/Ally] #[target_id]" (e.g., "Enemy #0" for enemy unit with ID 0, "Ally #1" for ally unit with ID 1)
2. If your chosen region of interest is location, format the output as "Location: [direction]" where direction must be one of: "North", "Northeast", "East", "Southeast", "South", "Southwest", "West", "Northwest", "Center" (e.g., "Location: Northeast")
3. If there are units visible, prioritize using unit as region of interest.
4. If the target_id is required, you MUST only use enemy/ally's unit_ids that are currently visible in your shooting range.
5. If your chosen region of interest is location, you MUST verify its availability.
6. If shared minimap information reveals enemies outside your sight range, prioritize moving to those locations unless there are enemies within your current vision range.
7. Your chosen region of interest should align with the current task description and ally's intentions.
8. Your chosen region of interest should enable you to quickly engage in combat or efficiently achieve the task in cooperation with allies?
Reasoning of region of interest:
Why was this region of interest chosen?
You should only respond in the format described below with a line break after each section colon (##Section##:) and NOT output comments or other information:
##Game_situation##:
1. ...
##Region_of_interest##:
region of interest
##Reasoning_of_region_of_interest##:
1. ...

Prompt for Task Reasoning

You are an AI assistant helping with academic research in the StarCraft II's SMAC (StarCraft Multi-Agent Challenge) environment, controlling a <unit_type> unit with ID <$unit_id$> in micromanagement scenarios <scenario_name> to help your team defeat the enemy forces. You operate under decentralized execution with partial observability, making decisions based only on local observations within your unit's field of view. You will be sequentially given <event_count> screenshots and corresponding descriptions of recent events. You will also be given a summary of the history that happened before the last screenshot. By analyzing these inputs, you gain a comprehensive understanding of the current context and situation within the game. You should assist in summarizing the next immediate task to do in SMACv2. Your ultimate goal is to help your team defeat the enemy forces as quickly as possible.
I will give you the following information:
Reasoning for the last episode:
<last_episode_reasoning>
Cumulative reward for the executing skill:
<cumulative_reward>
Current task:
<task_description>
Ally's tasks:
<ally_task>
Minimap information:
<ego_minimap>
Current game situation:
<game_situation>
Tactics recommendation:
<web_search>
The following are successive screenshots:
<image_introduction>
Skill set in Python format to select the next skill:
<skill_library>
Current executing skill:
<previous_action>
Implementation of the skill:
<action_code>
Reasoning for the skill:
<previous_reasoning>
Self-reflection for the last executed skill:
<previous_self_reflection_reasoning>
Task_guidance:
Based on the comprehensive game state analysis and team context, decompose the primary objective of "defeat all enemy units" into ONE specific tactical sub-task that enhances either target prioritization (score_target) or behavior control (control_logic). This sub-task should be concrete, implementable, and aligned with team coordination. Consider the following in your task decomposition:
1. Final Objective: Defeat enemy forces while preserving allies
2. Team Context:
- Your unit's current assigned task - Ally units' assigned tasks - Progress made on previous tasks 3. Tactical Layer:
- Enemy unit compositions and strategies - Team formation and positioning The task should follow one of these formats:
For target prioritization (score_target):
"Adjust [scoring weight/multiplier/threshold] to [specific combat calculation] based on [unit composition + battle state] where [precise condition]"
For behavior control (control_logic):
"Implement [unit movement pattern/formation/targeting] when [combat state + ally positions] satisfy [precise conditions]" Task Requirements:
Specificity: Must define exact behavior modification Measurability: Must have clear success criteria Actionability: Must be achievable using available atomic actions Coordination: Must support team tactical objectives Adaptability: Must respond to changing battle conditions If current task implementation remains unsuccessful, output 'null'.
Reasoning_of_task:
Why was this new task chosen, or why is there no need to propose a new task?
Skill_guidance:
Based on the current executing skill and the proposed next task, evaluate if there is alignment between them. Output True if the current skill effectively supports the task requirements, or False if a new skill is needed.
Reasoning_of_skill:
Why was this decision chosen?
You should only respond in the format described below with a line break after each section colon (##Section##:) and NOT output comments or other information:
##Task_guidance##:

[task guidance]
##Skill_guidance##:
[True or False]

Prompt for Skill Generation

You are an AI assistant helping with academic research in the StarCraft II's SMAC (StarCraft Multi-Agent Challenge) environment, controlling a <unit_type> unit with ID <$unit_id$> in micromanagement scenarios <scenario_name> to help your team defeat the enemy forces. You operate under decentralized execution with partial observability, making decisions based only on local observations within your unit's field of view. Your task is to enhance combat effectiveness:
Reasoning for the last episode:
<last_episode_reasoning>
Cumulative reward for the executing skill:
<cumulative_reward>
Current task:
<task_description>
Ally's tasks:
<ally_task>
Minimap information:
<ego_minimap>
Current game situation:
<game_situation>
<image_introduction>
Skill set in Python format to select the next skill:
<skill_library>
Current executing skill:
<previous_action>
Implementation of the skill:
<action_code>
Reasoning for the skill:
<previous_reasoning>
Self-reflection for the last executed skill:
<previous_self_reflection_reasoning>
Combat Analysis Task:
1. Analyze the provided script's effectiveness
2. Analyze the score_target(unit) function's effectiveness and weaknesses.
3. Analyze the control_logic() function's effectiveness and weaknesses.
4. Based on the current executing skill, the existing skills in skill library, and current task, evaluate if there is alignment between them.
5. If a new skill is needed, design tactical improvements while maintaining code structure.
6. If the current skill or there is any skill in skill library effectively supports the task requirements, output 'null' to avoid unnecessary token consumption.
Identify critical function for improvement (choose ONE Prioritize score_target(unit)):
1. score_target(unit): Target priority and scoring system. (Preferred)
2. control_logic(): Unit movement and attack decision making.
Skill_generation:
If there is no enemies, only output 'null'.
If the current skill or there is any skill in skill library effectively supports the task requirements, only output 'null'.
Otherwise:
The content of the improved code should obey the following code rules:
1. Output Format: Only provide the complete improved function (score_target(unit) (Preferred) OR control_logic()).
2. If the improved function is score_target(unit), there is exactly one parameter named "unit".
3. If the improved function is control_logic(), it should take no parameters.
4. The code should be surrounded in the '"'python' and '"'' structure.

You should only respond in the format described below with a line break after each section colon (##Section##:) and NOT output comments or other information:

##Skill_generation##:
"'python
def [function_name]([parameters]):

[improved implementation] “‘

Prompt for Actor

You are an AI assistant helping with academic research in the StarCraft II's SMAC (StarCraft Multi-Agent Challenge) environment, controlling a <unit_type> unit with ID <$unit_id$> in micromanagement scenarios <scenario_name> to help your team defeat the enemy forces. You operate under decentralized execution with partial observability, making decisions based only on local observations within your unit's field of view. Utilizing this insight, you are tasked with identifying the most suitable skill to take next, given the current task. You control the game unit and can execute skills from the available skill set. Upon evaluating the provided information, your role is to articulate the precise skill you would deploy, considering the game's present circumstances, and specify any necessary parameters for implementing that skill:
<last_episode_reasoning>
Cumulative reward for the executing skill:
<cumulative_reward>
Current task:
<task_description>
Ally's tasks:
<ally_task>
Minimap information:
<ego_minimap>
Current game situation:
<game_situation>
<image_introduction>
Skill set in Python format to select the next skill:
<skill_library>
Current executing skill:
<previous_action>
Implementation of the skill:
<action_code>
Reasoning for the skill:
<previous_reasoning>
Self-reflection for the last executed skill:
<previous_self_reflection_reasoning>
Skills:
The best skill to execute next to progress in achieving the goal. Pay attention to the names of the available skills and to the previous skills already executed, if any. You should also pay more attention to the following skill rules:
1. ONLY choose skill in the provided skill set.
2. Output skills in Python code format with required keyword parameters.
3. The ONLY required keyword parameter is "obs: str" - you MUST include this parameter as "obs='current'" in every skill. The actual observation will be automatically injected at runtime.
4. If there is summarization of history, consider this information when selecting the skill.
5. If the error report indicates that the last skill was unavailable, you MUST select a different skill.
6. Consider coordination with other units and choose skills that enhance team performance and cooperation.
7. Avoid repeating the same skill as the last executed skill unless there is a compelling strategic reason.
You should only respond in the format described below with a line break after each section colon (##Section##:) and NOT output comments or other information:

##Skills##:
“‘python
skill_name(obs='current')
“‘

```python
def race_melee_ranged_medivac_navi_A_star_score_type_default_center(obs: str):
    """
    Zealot/Zergling/Baneling/Colossus/Stalker/Hydralisk/Marauder/Marine/Medivac Controls Logic:
        Medivac:
            - Heals allies below 100% HP
            - Maintains 0.75 sight range from enemies
            - Centers between allies when no healing targets

        Melee (Zealot/Zergling/Baneling):
            - Attacks highest threat target within 0.7 sight range
            - Pursues targets using A* pathfinding
```

```
            - Groups with allies at >0.7 distance threshold

        Ranged (Colossus/Stalker/Hydralisk/Marauder/Marine):
            - Kites melee units at max_range - 0.05
            - Focus fires targets shared with 2+ allies
            - Maintains position behind melee allies

        Key Implementation:
        - Pathfinding: A* pathfinding in 32x32 grid with unit collision radius
        - Target scoring: [0-10] based on type (colossus 9 > baneling 8 > zealot 7 > stalker 6 > hydralisk 5 > marauder
    ↪ 4 > marine 3 > zergling 2 > medivac 1)/health (0-0.6)/distance (0-0.3)/last attacked (0.1)
        - Default action: Move to center of map, parse region of interest, random choice

    Args:
        obs (str): Observation string containing game state
    """
    import math
    # Parse observation
    obs_data = parse_obs(obs)
    # Get set of available actions
    valid_actions = obs_data.available_actions

    if 0 in valid_actions:
        return 0

    def score_target(unit):
        """Enhanced target scoring with improved kiting and formation control"""
        if unit.health <= 0:
            return -1

        score = 0

        # Refined unit type priorities with enhanced threat scaling
        unit_priorities = {
            'colossus': 35.0,  # Further increased priority
            'stalker': 30.0,   # Enhanced anti-armor focus
            'zealot': 45.0,    # Higher melee threat recognition

            'marine': 45.0,    # Balanced damage dealer priority
            'marauder': 35.0,  # Anti-armor specialist
            'medivac': 30.0,   # Support unit priority

            'hydralisk': 30.0,  # High priority for their sustained DPS
            'zergling': 35.0,   # Medium priority as swarm units
            'baneling': 45.0,   # Critical priority due to splash damage
        }

        # Dynamic matchup priorities with improved counter weighting
        unit_counters = {
            'colossus': {'colossus': 1.2, 'stalker': 1.0, 'zealot': 1.5},
            'stalker': {'colossus': 1.2, 'stalker': 1.0, 'zealot': 1.5},
            'zealot': {'colossus': 1.2, 'stalker': 1.0, 'zealot': 1.5},

            'marine': {'marine': 1.5, 'medivac': 1.0, 'marauder': 1.2},
            'marauder': {'marine': 1.5, 'medivac': 1.0, 'marauder': 1.2},
            'medivac': {'marine': 1.5, 'medivac': 1.0, 'marauder': 1.2},

            'hydralisk': {'hydralisk': 1.0, 'zergling': 1.2, 'baneling': 1.5},
            'zergling': {'hydralisk': 1.2, 'zergling': 1.5, 'baneling': 1.0},
            'baneling': {'hydralisk': 1.2, 'zergling': 1.5, 'baneling': 1.0},

        }

        base_priority = unit_priorities.get(unit.unit_type.lower(), 5.0)
        is_ranged = unit.unit_type.lower() not in ['zealot', 'zergling', 'baneling']
        own_is_ranged = obs_data.own_unit_type.lower() not in ['zealot', 'zergling', 'baneling']

        # Enhanced threat assessment with improved melee handling
        matchup_mult = unit_counters.get(obs_data.own_unit_type.lower(), {}).get(unit.unit_type.lower(), 1.0)
        base_priority *= matchup_mult

        if hasattr(unit, 'can_attack'):  # Enemy unit
            score = base_priority

            distance_factor = max((1 - unit.distance) + 1, 0.5)
            score *= distance_factor

            if not own_is_ranged:
```

```
                range_ally = [ally for ally in obs_data.allies if ally.unit_type.lower() not in ['zealot', 'zergling',
    ↪ 'baneling']]
                if range_ally:
                    ally_x = sum(ally.position[0] for ally in range_ally) / len(range_ally)
                    ally_y = sum(ally.position[1] for ally in range_ally) / len(range_ally)
                    ally_distance = ((ally_x - unit.position[0])**2 + (ally_y - unit.position[1])**2)**0.5
                    distance_factor = max((1 - ally_distance) + 1, 0.5)
                    score *= distance_factor

            # Enhanced Position Analysis with improved spacing
            position_x, position_y = unit.position

            def calculate_combat_power(units, radius=0.5):  # Further reduced for tighter control
                total_power = 0
                ranged_count = 0
                melee_count = 0
                unit_positions = []

                for u in units:
                    dist = ((u.position[0] - position_x)**2 +
                        (u.position[1] - position_y)**2)**0.5
                    unit_positions.append(u.position)

                    if dist <= radius:
                        base_power = unit_priorities.get(u.unit_type.lower(), 5.0)

                        # Unit type specific power calculation
                        if u.unit_type.lower() in ['zealot', 'zergling', 'baneling']:
                            melee_count += 1
                            if melee_count >= 2:
                                base_power *= 1.4
                        else:
                            ranged_count += 1
                            base_power *= 1.3

                        # Health-based power scaling
                        health_factor = 1.5 if u.health > 0.7 else 1.0 if u.health > 0.4 else 0.6
                        position_factor = 1.3 - (dist/radius)

                        total_power += base_power * health_factor * position_factor

                # Enhanced formation cohesion calculation
                cohesion = 0
                if len(unit_positions) > 2:
                    center_x = sum(p[0] for p in unit_positions) / len(unit_positions)
                    center_y = sum(p[1] for p in unit_positions) / len(unit_positions)
                    avg_dist = sum(((p[0] - center_x)**2 + (p[1] - center_y)**2)**0.5
                                for p in unit_positions) / len(unit_positions)
                    max_desired_dist = 0.3  # Tighter formation control
                    cohesion = 2.0 / (1.0 + (avg_dist / max_desired_dist))

                return total_power * (1 + cohesion), melee_count, ranged_count

            ally_power, ally_swarms, ally_ranged = calculate_combat_power(obs_data.allies)
            enemy_power, enemy_swarms, enemy_ranged = calculate_combat_power(obs_data.enemies)

            # Improved Focus Fire Logic with enhanced commitment
            num_attackers = sum(1 for ally in obs_data.allies
                            if ally.last_action >= 6 and ally.last_action - 6 == unit.id)

            if unit.id == obs_data.last_action - 6:
                persistence_bonus = 2.0  # Stronger target commitment
                score *= persistence_bonus

            if num_attackers > 0:
                focus_bonus = 1.2 ** num_attackers  # Enhanced focus fire emphasis
                # if num_attackers too high, discourage prevent overcommitment
                if num_attackers >= 3 and unit.id != obs_data.last_action - 6 and obs_data.own_unit_type.lower() in
    ↪ ['zergling', 'baneling']:
                    focus_bonus = 0.5
                score *= focus_bonus

            # Improved Combat Advantage Factor
            advantage_factor = 1.0
            if ally_power > enemy_power * 1.3:
                advantage_factor = 1.2  # More aggressive advantage pursuit
                if ally_swarms >= 3:
                    advantage_factor *= 1.2
```

```
            # prioritize isolated enemies
            if (enemy_swarms + enemy_ranged) == 1:
                advantage_factor *= 2.0
            elif (ally_swarms + ally_ranged) > (enemy_swarms + enemy_ranged):
                advantage_factor *= 1.2

            score *= advantage_factor

            # Health factor
            health_factor = (1 - unit.health) + 1
            score *= health_factor

        else:  # Ally unit
            score = base_priority

            # Improved Support Priority
            health_factor = (1 - unit.health) + 1
            score *= health_factor

            distance_factor = max((1 - unit.distance) + 1, 0.5)
            score *= distance_factor

        return score

    def control_logic():
        # Medivac units control logic
        if obs_data.own_unit_type.lower() == 'medivac':
            attack_actions = [a for a in valid_actions if a >= 6]
            # If there are allies
            if obs_data.allies:
                lowest_health_ally = min(obs_data.allies, key=lambda x: x.health)
                # If there are both allies and enemies
                if obs_data.enemies:
                    enemy_in_range = [enemy for enemy in obs_data.enemies if enemy.distance < 1]
                    # Check if any melee ally, if so and last action is not attack, move to center of melee allies
                    melee_ally = [ally for ally in obs_data.allies if ally.unit_type.lower() in ['zealot', 'zergling',
 ↪ 'baneling']]
                    if melee_ally:
                        # Move to center of melee allies
                        ally_x = sum(ally.position[0] for ally in melee_ally) / len(melee_ally)
                        ally_y = sum(ally.position[1] for ally in melee_ally) / len(melee_ally)
                    else:
                        ally_x = sum(ally.position[0] for ally in obs_data.allies) / len(obs_data.allies)
                        ally_y = sum(ally.position[1] for ally in obs_data.allies) / len(obs_data.allies)
                    # Calculate retreat position
                    if len(enemy_in_range) > 0:
                        enemy_center = (sum(e.position[0] for e in enemy_in_range) / len(enemy_in_range),
                                        sum(e.position[1] for e in enemy_in_range) / len(enemy_in_range))
                    else:
                        enemy_center = (sum(e.position[0] for e in obs_data.enemies) / len(obs_data.enemies),
                                        sum(e.position[1] for e in obs_data.enemies) / len(obs_data.enemies))

                    dx = ally_x - enemy_center[0]
                    dy = ally_y - enemy_center[1]

                    distance = (dx ** 2 + dy ** 2) ** 0.5

                    safe_x = ally_x + (dx / abs(dx)) * 2/obs_data.own_sight_range if dx != 0 else ally_x
                    safe_y = ally_y + (dy / abs(dy)) * 2/obs_data.own_sight_range if dy != 0 else ally_y

                    distance = (safe_x ** 2 + safe_y ** 2) ** 0.5
                    criterion = 5/obs_data.own_sight_range
                    if obs_data.last_action >= 6 or len(attack_actions) == 0:
                        target_angle = math.atan2(enemy_center[1], enemy_center[0])
                        safe_angle = math.atan2(ally_y, ally_x)
                        angle_diff = abs(target_angle - safe_angle)
                        if distance > criterion and (math.pi/9 < angle_diff < 17*math.pi/9):
                            path_action = find_path(obs_data, safe_x, safe_y)
                            if path_action:
                                return path_action
                target_scores = {ally.id: score_target(ally) for ally in obs_data.allies}
                # Check if there are same max score targets
                max_score = max(target_scores.values())
                max_score_target_ids = [target_id for target_id, score in target_scores.items() if score == max_score]
                max_score_targets = [ally for ally in obs_data.allies if ally.id in max_score_target_ids]
                closest_ally = min(obs_data.allies, key=lambda x: x.distance)
                if len(max_score_targets) > 1:
```

```
                    # Chose the closest target
                    best_target = min(max_score_targets, key=lambda x: x.distance)
                else:
                    best_target = max_score_targets[0]
                if (best_target.id + 6) in valid_actions and 0 < best_target.health < 0.9:
                    return heal(best_target.id)
                elif (closest_ally.id + 6) in valid_actions and 0 < closest_ally.health < 0.9:
                    return heal(closest_ally.id)
                elif (lowest_health_ally.id + 6) in valid_actions and 0 < lowest_health_ally.health < 0.9:
                    return heal(lowest_health_ally.id)
                else:
                    # Move to the target
                    dx = best_target.position[0]
                    dy = best_target.position[1]
                    path_action = find_path(obs_data, dx, dy, target_type=best_target.unit_type.lower())
                    if path_action:
                        return path_action
            # If there are no allies
            else:
                # If there are only enemies
                if obs_data.enemies:
                    enemy_x = sum(e.position[0] for e in obs_data.enemies) / len(obs_data.enemies)
                    enemy_y = sum(e.position[1] for e in obs_data.enemies) / len(obs_data.enemies)
                    target_x = - enemy_x
                    target_y = - enemy_y

                    g_x = target_x * obs_data.own_sight_range + (obs_data.own_position[0] * 32)
                    g_y = target_y * obs_data.own_sight_range + (obs_data.own_position[1] * 32)
                    if not (0 <= g_x <= 32 and 0 <= g_y <= 32):
                        target_x = (0.5 - obs_data.own_position[0]) * 32 / obs_data.own_sight_range
                        target_y = (0.5 - obs_data.own_position[1]) * 32 / obs_data.own_sight_range

                    path_action = find_path(obs_data, target_x, target_y)
                    if path_action:
                        return path_action
        # Melee units control logic
        elif obs_data.own_unit_type.lower() in ['zealot', 'zergling', 'baneling']:
            # If there are enemies
            if obs_data.enemies:
                enemy_in_range = [enemy for enemy in obs_data.enemies if enemy.distance < 1]
                attack_actions = [a for a in valid_actions if a >= 6]

                if len(enemy_in_range) > 0:
                    enemy_center = (sum(e.position[0] for e in enemy_in_range) / len(enemy_in_range),
                                    sum(e.position[1] for e in enemy_in_range) / len(enemy_in_range))
                else:
                    enemy_center = (sum(e.position[0] for e in obs_data.enemies) / len(obs_data.enemies),
                                    sum(e.position[1] for e in obs_data.enemies) / len(obs_data.enemies))
                if obs_data.allies:
                    # Check if any melee ally, if so and last action is not attack, move to center of melee allies
                    melee_ally = [ally for ally in obs_data.allies if ally.unit_type.lower() in ['zealot', 'zergling',
    ↪ 'baneling']]
                    if melee_ally:
                        # Move to center of melee allies
                        ally_x = sum(ally.position[0] for ally in melee_ally) / len(melee_ally) / 2
                        ally_y = sum(ally.position[1] for ally in melee_ally) / len(melee_ally) / 2

                        safe_x = ally_x
                        safe_y = ally_y

                        distance = (safe_x ** 2 + safe_y ** 2) ** 0.5
                        criterion = 2/obs_data.own_sight_range
                        if len(attack_actions) == 0 or distance > 0.5:
                            target_angle = math.atan2(enemy_center[1], enemy_center[0])
                            safe_angle = math.atan2(ally_y, ally_x)
                            angle_diff = abs(target_angle - safe_angle)
                            if distance > criterion and (math.pi/9 < angle_diff < 17*math.pi/9 or distance > 0.5):
                                path_action = find_path(obs_data, safe_x, safe_y)
                                if path_action:
                                    return path_action

                # Enhanced cluster detection with dynamic radius
                enemy_clusters = {}
                cluster_centers = {}
                for enemy in obs_data.enemies:
                    nearby_enemies = []
                    center_x, center_y = enemy.position[0], enemy.position[1]

```

```
                    # Dynamic cluster radius based on unit type
                    cluster_radius = 0.3 if obs_data.own_unit_type.lower() == 'baneling' else 0.2

                    for other in obs_data.enemies:
                        distance = ((other.position[0] - enemy.position[0])**2 +
                                (other.position[1] - enemy.position[1])**2)**0.5
                        if distance <= cluster_radius:
                            nearby_enemies.append(other)
                            center_x += other.position[0]
                            center_y += other.position[1]

                    if nearby_enemies:
                        center_x /= len(nearby_enemies)
                        center_y /= len(nearby_enemies)

                    enemy_clusters[enemy.id] = len(nearby_enemies)
                    cluster_centers[enemy.id] = (center_x, center_y)

                # Enhanced target scoring with tactical considerations
                target_scores = {}
                for enemy in obs_data.enemies:
                    base_score = score_target(enemy)

                    # Enhanced cluster bonus for splash damage
                    if obs_data.own_unit_type.lower() == 'baneling':
                        cluster_bonus = 1.5 ** enemy_clusters[enemy.id]
                    else:
                        cluster_bonus = 1.2 ** enemy_clusters[enemy.id]
                    # Calculate final score with all factors
                    target_scores[enemy.id] = (base_score + cluster_bonus)

                # Check if there are same max score targets
                max_score = max(target_scores.values())
                max_score_targets = [enemy for enemy in obs_data.enemies
                          if target_scores[enemy.id] >= max_score]  # Allow for close scores
                if len(max_score_targets) > 1:
                    # Choose target balancing distance and cluster potential
                    best_target = min(max_score_targets,
                            key=lambda x: x.distance)
                else:
                    best_target = max_score_targets[0]

                if best_target.can_attack:
                    return attack(best_target.id)
                else:
                    # Move to the target
                    dx = best_target.position[0]
                    dy = best_target.position[1]
                    path_action = find_path(obs_data, dx, dy, target_type=best_target.unit_type.lower())
                    if path_action:
                        return path_action
                    elif attack_actions:
                        attackable_enemies = [enemy for enemy in obs_data.enemies if enemy.can_attack]
                        if obs_data.last_action in attack_actions:
                            return obs_data.last_action
                        if attackable_enemies:
                            return attack(min(attackable_enemies, key=lambda e: e.distance).id)
                        return random.choice(attack_actions)

            # If there are no enemies
            else:
                # If there are only allies
                if obs_data.allies:
                    # Improved melee group formation
                    melee_allies = [ally for ally in obs_data.allies
                                if ally.unit_type.lower() in ['zealot', 'zergling', 'baneling']]
                    if melee_allies:
                        spacing = 0.1 if obs_data.own_unit_type.lower() == 'baneling' else 0.05
                        # Dynamic group positioning
                        center_x = sum(ally.position[0] for ally in melee_allies) / len(melee_allies)
                        center_y = sum(ally.position[1] for ally in melee_allies) / len(melee_allies)

                        # Calculate spread from center
                        max_spread = max(((ally.position[0] - center_x)**2 +
                                          (ally.position[1] - center_y)**2)**0.5
                                         for ally in melee_allies)

                        own_distance = ((center_x)**2 + (center_y)**2)**0.5
```

```
                    if own_distance > spacing or max_spread > 0.1:
                        # Move toward center while maintaining minimum spacing
                        adjusted_x = center_x * 0.85  # Slight offset to prevent overcrowding
                        adjusted_y = center_y * 0.85
                        path_action = find_path(obs_data, adjusted_x, adjusted_y)
                        if path_action:
                            return path_action
                else:
                    ally_x = sum(ally.position[0] for ally in obs_data.allies) / len(obs_data.allies)
                    ally_y = sum(ally.position[1] for ally in obs_data.allies) / len(obs_data.allies)
                    distance = (ally_x ** 2 + ally_y ** 2) ** 0.5
                    if distance > 0.05:
                        dx = ally_x
                        dy = ally_y
                        path_action = find_path(obs_data, dx, dy)
                        if path_action:
                            return path_action
    # Ranged units control logic
    else:
        attack_actions = [a for a in valid_actions if a >= 6]
        # If there are enemies
        if obs_data.enemies:
            # If there are both allies and enemies
            # Calculate retreat position
            enemy_in_range = [enemy for enemy in obs_data.enemies if enemy.distance < 1]
            if len(enemy_in_range) > 0:
                enemy_center = (sum(e.position[0] for e in enemy_in_range) / len(enemy_in_range),
                                sum(e.position[1] for e in enemy_in_range) / len(enemy_in_range))
            else:
                enemy_center = (sum(e.position[0] for e in obs_data.enemies) / len(obs_data.enemies),
                                sum(e.position[1] for e in obs_data.enemies) / len(obs_data.enemies))
            if obs_data.allies:
                # Check if any melee ally, if so and last action is not attack, move to center of melee allies
                melee_ally = [ally for ally in obs_data.allies if ally.unit_type.lower() in ['zealot', 'zergling',
↪ 'baneling']]
                melee_enemy = [enemy for enemy in enemy_in_range if enemy.unit_type.lower() in ['zealot',
↪ 'zergling', 'baneling']]
                if melee_ally:
                    # Move to center of melee allies
                    ally_x = sum(ally.position[0] for ally in melee_ally) / len(melee_ally)
                    ally_y = sum(ally.position[1] for ally in melee_ally) / len(melee_ally)

                else:
                    ally_x = sum(ally.position[0] for ally in obs_data.allies) / len(obs_data.allies) / 2
                    ally_y = sum(ally.position[1] for ally in obs_data.allies) / len(obs_data.allies) / 2

                dx = ally_x - enemy_center[0]
                dy = ally_y - enemy_center[1]
                safe_x = ally_x
                safe_y = ally_y
                if melee_ally:
                    safe_x = safe_x + (dx / abs(dx)) * 2/obs_data.own_sight_range if dx != 0 else safe_x
                    safe_y = safe_y + (dy / abs(dy)) * 2/obs_data.own_sight_range if dy != 0 else safe_y
                melee_threaten = False
                if melee_enemy:
                    closest_melee_enemy = min(melee_enemy, key=lambda x: x.distance)
                    if closest_melee_enemy.distance <= 4/obs_data.own_sight_range:
                        melee_threaten = True
                        dx = safe_x - closest_melee_enemy.position[0]
                        dy = safe_y - closest_melee_enemy.position[1]
                        safe_x = safe_x + (dx / abs(dx)) * 1/obs_data.own_sight_range if dx != 0 else safe_x
                        safe_y = safe_y + (dy / abs(dy)) * 1/obs_data.own_sight_range if dy != 0 else safe_y

                distance = (safe_x ** 2 + safe_y ** 2) ** 0.5
                criterion = 4/obs_data.own_sight_range
                if obs_data.last_action >= 6 or len(attack_actions) == 0 or distance > 0.9:
                    target_angle = math.atan2(enemy_center[1], enemy_center[0])
                    safe_angle = math.atan2(ally_y, ally_x)
                    angle_diff = abs(target_angle - safe_angle)
                    if distance > criterion and ((math.pi/9 < angle_diff < 17*math.pi/9) or melee_threaten or
↪ distance > 0.9):
                        path_action = find_path(obs_data, safe_x, safe_y)
                        if path_action:
                            return path_action
                # Focus fire logic
                # Count how many allies are attacking each enemy
                target_counts = {}
```

```
                    for ally in obs_data.allies:
                        if ally.last_action >= 6:
                            target_id = ally.last_action - 6
                            target_counts[target_id] = target_counts.get(target_id, 0) + 1
                    # Distance to safe point of each enemy affect target choosing
                    enemy_safe_distance = {enemy.id: ((enemy.position[0] - safe_x) ** 2 + (enemy.position[1] - safe_y)
    ↪ ** 2) ** 0.5 for enemy in obs_data.enemies}
                    # Find best target combining focus fire and threat scoring
                    target_scores = {enemy.id: score_target(enemy) for enemy in obs_data.enemies}
                    for target_id, count in target_counts.items():
                        if target_id in target_scores:
                            target_scores[target_id] += count * 0.5
                    for target_id, scores in target_scores.items():
                        target_scores[target_id] = scores * (1 - enemy_safe_distance[target_id] * 0.3)
                    best_target_id = max(target_scores.items(), key=lambda x: x[1])[0]
                    best_target = next(enemy for enemy in obs_data.enemies if enemy.id == best_target_id)
                    if best_target.can_attack:
                        return attack(best_target_id)
                    else:
                        # Best target is not in shoot range, move to target
                        dx = best_target.position[0]
                        dy = best_target.position[1]

                        # Only move to target if its direction is not conflicting with the safe point
                        # Check if target direction aligns with safe point direction
                        target_angle = math.atan2(dy, dx)
                        safe_angle = math.atan2(ally_y, ally_x)
                        angle_diff = abs(target_angle - safe_angle)
                        # Only move if angle difference is less than 90 degrees
                        if angle_diff < math.pi/9 or angle_diff > 17*math.pi/9 or not melee_ally:
                            if best_target.distance > obs_data.own_shoot_range / obs_data.own_sight_range:
                                path_action = find_path(obs_data, dx, dy, target_type=best_target.unit_type.lower())
                                if path_action:
                                    return path_action
                        if attack_actions:
                            if obs_data.last_action in attack_actions:
                                return obs_data.last_action
                            attackable_enemies = [enemy for enemy in obs_data.enemies if enemy.can_attack]
                            closest_enemy = min(
                                [enemy for enemy in attackable_enemies],
                                key=lambda enemy: enemy.distance,
                                default=None,
                            )
                            if closest_enemy and closest_enemy.can_attack:
                                return attack(closest_enemy.id)
                            return random.choice(attack_actions)
                        else:
                            if distance > criterion:
                                path_action = find_path(obs_data, safe_x, safe_y)
                                if path_action:
                                    return path_action

                # If there are only enemies
                else:
                    # Closest enemy as target
                    closest_enemy = min(
                        [enemy for enemy in obs_data.enemies],
                        key=lambda enemy: enemy.distance,
                        default=None,
                    )
                    # No allies, kitting melee enemies
                    if closest_enemy.unit_type.lower() in ['zealot', 'zergling', 'baneling']:
                        if closest_enemy.distance <= 4 / obs_data.own_sight_range and obs_data.last_action >= 6:
                            enemy_x = sum(e.position[0] for e in obs_data.enemies) / len(obs_data.enemies)
                            enemy_y = sum(e.position[1] for e in obs_data.enemies) / len(obs_data.enemies)
                            target_x = - enemy_x
                            target_y = - enemy_y

                            g_x = target_x * obs_data.own_sight_range + (obs_data.own_position[0] * 32)
                            g_y = target_y * obs_data.own_sight_range + (obs_data.own_position[1] * 32)
                            if not (0 <= g_x <= 32 and 0 <= g_y <= 32):
                                target_x = (0.5 - obs_data.own_position[0]) * 32 / obs_data.own_sight_range
                                target_y = (0.5 - obs_data.own_position[1]) * 32 / obs_data.own_sight_range

                            path_action = find_path(obs_data, target_x, target_y)
                            if path_action:
                                return path_action
                        if closest_enemy.can_attack:
```

```
                            return attack(closest_enemy.id)
                    else:
                        # No melee enemies, highest priority enemy as target
                        target_scores = {enemy.id: score_target(enemy) for enemy in obs_data.enemies}
                        # Check if there are same max score targets
                        max_score = max(target_scores.values())
                        max_score_target_ids = [target_id for target_id, score in target_scores.items() if score ==
    ↪ max_score]
                        max_score_targets = [enemy for enemy in obs_data.enemies if enemy.id in max_score_target_ids]
                        if len(max_score_targets) > 1:
                            # Chose the closest target
                            best_target = min(max_score_targets, key=lambda x: x.distance)
                        else:
                            best_target = max_score_targets[0]
                        if best_target.can_attack:
                            return attack(best_target.id)
                        else:
                            # Best target is not in shoot range, move to target
                            dx = best_target.position[0]
                            dy = best_target.position[1]
                            if best_target.distance > obs_data.own_shoot_range / obs_data.own_sight_range:
                                path_action = find_path(obs_data, dx, dy, target_type=best_target.unit_type.lower())
                                if path_action:
                                    return path_action
                                elif attack_actions:
                                    if obs_data.last_action in attack_actions:
                                        return obs_data.last_action
                                    attackable_enemies = [enemy for enemy in obs_data.enemies if enemy.can_attack]
                                    closest_enemy = min(
                                        [enemy for enemy in attackable_enemies],
                                        key=lambda enemy: enemy.distance,
                                        default=None,
                                    )
                                    if closest_enemy and closest_enemy.can_attack:
                                        return attack(closest_enemy.id)
                                    return random.choice(attack_actions)
            # If there are no enemies
            else:
                # If there are only allies
                if obs_data.allies:
                    # Check if any melee ally
                    melee_ally = [ally for ally in obs_data.allies if ally.unit_type.lower() in ['zealot', 'zergling',
    ↪ 'baneling']]
                    if melee_ally:
                        # Move to center of melee allies
                        melee_ally_x = sum(ally.position[0] for ally in melee_ally) / len(melee_ally)
                        melee_ally_y = sum(ally.position[1] for ally in melee_ally) / len(melee_ally)
                        dx = melee_ally_x
                        dy = melee_ally_y
                        distance = (dx ** 2 + dy ** 2) ** 0.5
                        if distance > 0.05:
                            path_action = find_path(obs_data, dx, dy)
                            if path_action:
                                return path_action
                    else:
                        # No melee allies, move to target ally
                        ally_x = sum(ally.position[0] for ally in obs_data.allies) / len(obs_data.allies)
                        ally_y = sum(ally.position[1] for ally in obs_data.allies) / len(obs_data.allies)
                        distance = (ally_x ** 2 + ally_y ** 2) ** 0.5
                        if distance > 0.05:
                            dx = ally_x
                            dy = ally_y
                            path_action = find_path(obs_data, dx, dy)
                            if path_action:
                                return path_action
        return default_action(obs)

    return control_logic()
```

Listing 2: Example Script of Skill Initialzation