CoSMAC: A Benchmark for Evaluating Communication and Coordination in LLM-Based Agents

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have recently demonstrated strong reasoning and communication abilities, motivating research into their potential as autonomous agents in multi-agent systems. In this work, we introduce Communicative SMAC (CoSMAC), a benchmark designed to systematically evaluate the communication and coordination capabilities of LLM-based agents. Built upon the well-established SMAC multi-agent reinforcement learning (MARL) environment, CoSMAC features a set of scenarios requiring varying degrees of micromanagement and communication, where agents must exchange information through natural language to achieve shared goals. We evaluate 6 state-of-the-art open-source and proprietary LLMs in zero-shot cooperative settings and additionally introduce an LLM-vs-LLM protocol for head-to-head tactical comparisons. We further compare the performance of LLM-based agents against MARL baselines trained with and without communication. Experimental results show that while LLMs struggle in scenarios demanding fine-grained micromanagement and spatial coordination, they can outperform a non-communicating MARL baseline in tasks that rely more heavily on effective communication, while remaining below a communication-trained MARL reference. The environment implementation and datasets can be found on GitHub repository[1].

1 Introduction

Large Language Models (LLMs) have shown promise in multi-agent systems, enabling agents to reason (Ferrag et al., 2026), plan (Huang et al., 2024b), and communicate using natural language. As LLM capabilities advance, there is growing interest in using LLM-based agents to solve complex cooperative tasks that require effective communication and coordination.

Most existing LLM-agent systems focus on relatively simple workflows, such as decomposing coding tasks across multiple agents with minimal interaction (Dong et al., 2024; Huang et al., 2024a). While some environments explore more complex social or cooperative interactions (Xie et al., 2025; Vezhnevets et al., 2023; Park et al., 2023), they often lack well-defined goals, clear performance metrics, or require only one-way communication. This limits their usefulness for studying emergent communication and coordination strategies.

In this paper, we introduce Communicative SMAC (CoSMAC), an LLM-powered Multi-Agent System (LLM-MAS) benchmark focused on the communication aspect of agents. The benchmark is built on the popular multi-agent reinforcement learning (MARL) environment SMAC and features novel scenarios centered on communication. Using SMAC as the base environment allows for rigorous performance evaluation of different models through episode returns and win rate. The proposed scenarios contain groups of allied agents scattered across the map in a partially observable environment, making communication essential for coordinating actions, as messages are visible to all allied agents. Partial observability and randomized initial states in each scenario make generalization essential for solving tasks.

We conduct zero-shot evaluations of various pre-trained LLMs, including both open-source and closed-source models. Beyond cooperative benchmark evaluation, CoSMAC also supports LLM-vs-LLM matches, where two models control opposing teams in the same tactical scenario to enable direct competitive comparison. We also train QMIX (Rashid et al., 2020) on the proposed scenarios, enabling analysis of the ability of traditional MARL methods to solve these tasks without agent communication. In addition, we evaluate NDQ (Wang et al., 2020), a MARL method designed for inter-agent communication,

[1] https://anonymous.4open.science/r/cosmac-9F96

as a practical reference point for methods with well-functioning learned communication.

To summarize, our contributions are as follows:

- We develop Communicative SMAC (CoSMAC), an LLM-MAS benchmark focused on agent communication, featuring nine scenarios that require agents to exchange information to coordinate their actions.
- We evaluate a range of pre-trained LLMs in zero-shot settings, covering both open-source and closed-source models, to assess their communication and coordination capabilities. We compare these LLM-based approaches with MARL methods trained with and without communication, and include an LLM-vs-LLM protocol for direct tactical model comparisons.
- We publicly release the results of our LLM evaluations as a publicly available dataset to support future research.

2 Related Work

In this section, we provide an overview of various systems based on LLM agents. Many studies focus on agent systems with relatively simple interactions, where each agent represents a specific function, and agents work sequentially to solve complex tasks, such as coding. Other research explores environments with more complex interactions between agents, such as communication, which is required to solve cooperative tasks.

Several works introduce different LLM agent approaches for solving coding tasks. Dong et al. (Dong et al., 2024) presents the Self-Collaboration framework, which consists of three agents: an analyst, a coder, and a tester. The analyst decomposes tasks for the coder to solve, while the tester inspects the generated code and writes test reports, which are then sent back to the coder for corrections. AgentCoder (Huang et al., 2024a) introduces three agents: the programmer agent, the test case designer agent, and the test executor agent. The test executor runs the test cases provided by the test case designer in a local environment, and the feedback is sent back to the programmer agent. Some domain-specific tasks can also be framed as coding tasks. The CodeAct (Wang et al., 2024) framework introduces an approach in which LLM agents generate actions as executable Python code. For example, LLM-SMAC (Deng et al., 2025) is based on SMAC, a popular benchmark for multi-agent reinforcement learning (MARL), and adapts it for LLM usage. While SMAC focuses on MARL task, LLM-SMAC, following the CodeAct approach, uses executable Python code as the actions of LLM agents. The LLM-SMAC framework introduces three LLM agents: the planner, the coder and the critic. The planner uses the scenario description in order to generate a strategy, the coder produces executable code to control all the allied units according to that strategy, and the critic analyses any exceptions raised during code execution or the results of the executed code, if no exceptions occur. Xie et al. (Xie and Zou, 2024) propose a framework for travel planning, where multiple agents take on different roles to solve sequential subtasks. Human Simulacra (Xie et al., 2025) introduces a multi-agent LLM architecture containing roles such as Memory Agent, Emotion Agent, and Thinking Agent to produce personified responses that align with the provided character description. These approaches introduce multi-agent systems with relatively limited interactions, as collaboration between agents is mostly one-way, except for sending back error reports.

A number of environments for LLM agents focus on the social role-playing aspect. Concordia (Vezhnevets et al., 2023) is a library for environment simulation, where multiple LLM agents interact with each other to simulate social situations. However, Concordia agents are not designed to pursue clear goals with distinct rewards; instead, they are intended to role-play characters in specific scenarios. While agents may have practical objectives, these are not strictly enforced. Park et al. (Park et al., 2023) introduce an interactive environment for LLM agents to simulate believable human behavior. This environment simulates a small town with multiple agents. Users can explicitly specify goals for an agent, such as asking one to throw a party. However, this environment also lacks clear performance metrics, and agents are evaluated solely on whether their behavior appears "believable" based on common sense.

A number of works focus on text-based communication games, such as the Werewolf game. In this game, players are divided into two teams: werewolves and villagers. The goal of the villagers is to uncover the identities of the werewolves, while the werewolves aim to hide their identities for as long as possible and eliminate the villagers. Some

players may perform special actions based on their roles, but the main aspect of the game is communication between players. Xu et al. (Xu et al., 2024) propose a framework for playing the Werewolf game, with each player represented by an LLM agent. In this implementation, players take turns speaking once per in-game day. Werewolf Arena (Bailis et al., 2024) introduces a bidding system for communication, allowing agents to express their desire to speak. With this system, the speaking order of players is determined dynamically, and each agent may speak multiple times during each communication phase, enabling discussions to coordinate with other agents and achieve their goals. Another example of a communication-focused environment is Collab-Overcooked (Sun et al., 2025). This environment is a grid-based kitchen simulation divided into two sections, each occupied by an agent. Each agent has access to certain resources and utensils, as well as counters that allow them to transfer materials between each other. The tasks require the agents to collaboratively prepare a specific dish. Since each agent has access to only a subset of the necessary resources and utensils, they must communicate in natural language to coordinate their actions, such as asking one another to prepare or deliver specific ingredients. It should also be noted that the cooking steps for the recipe are provided in the agents' inputs, and the initial state for each scenario is always the same and fully observable to both agents. As a result, each scenario has a single, consistent optimal solution.

3 Background

We formalize the underlying cooperative task as a decentralized partially observable Markov decision process (Dec-POMDP), defined by the tuple $(n, S, A, T, \mathbb{O}, O, R, \gamma)$. Here, n is the number of agents, S is the set of global states, A is the set of individual environment actions, $T : S \times A^n \to \Delta(S)$ is the transition function, $\mathbb{O}$ is the set of joint observations, $O : S \times A^n \to \Delta(\mathbb{O})$ is the observation function, $R : S \times A^n \to \Delta(\mathbb{R})$ is the shared reward function, and γ is the discount factor. We use $\Delta(X)$ to denote the set of probability distributions over a set X.

At each timestep t, each agent $i \in \{1, \ldots, n\}$ receives an individual observation $o_{t,i}$ from the joint observation $\mathbf{o}_t \in \mathbb{O}$ and selects an environment action $a_{t,i} \in A$. The joint action $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,n}) \in A^n$ is then executed. The global state transitions from $s_t \in S$ to $s_{t+1} \in S$ according to $T(s_t, \mathbf{a}_t)$, and the environment returns a shared reward r_t according to $R(s_t, \mathbf{a}_t)$. In the standard MARL setting, the goal is to learn decentralized policies π_i that map each agent's local information to actions in A so as to maximize the expected discounted return $\mathbb{E}[\sum_t \gamma^t r_t]$.

CoSMAC keeps this base Dec-POMDP structure for the StarCraft II environment dynamics and augments the agent interface with language. At each timestep, the raw observation $o_{t,i}$ is transformed by a preprocessing function $\phi(o_{t,i}, n_o, n_m)$ into a prompt represented as input tokens $x_{t,i} = (x_1, \ldots, x_m)$. This prompt includes the agent's role, environmental information such as observed units' health and global positions over the current step and the previous n_o steps, messages sent within the most recent n_m environment steps, the set of currently available environment actions $A_{t,i} \subseteq A$, and a template illustrating the expected JSON output format.

An LLM-based policy generates a response autoregressively as output tokens $y_{t,i} = (y_1, \ldots, y_T)$, where each token is sampled from the model-induced distribution conditioned on the prompt and the previously generated prefix. The generated JSON response is parsed into two components: an environment action $a_{t,i} \in A_{t,i}$ and, when communication is enabled, a communication message $m_{t,i} \in M$. The environment action is the action used in the Dec-POMDP joint action $\mathbf{a}_t$, while the message is broadcast to allied agents and incorporated into their later prompts.

The parameters n_o and n_m define the observable context available to the LLM agent; in our default configuration, both windows cover the current step plus the three preceding steps. We assume that agents do not possess memory beyond the current prompt, so they must rely on the information contained in $\phi(o_{t,i}, n_o, n_m)$. Communication messages do not directly affect the StarCraft II transition or reward functions: T and R depend on the joint environment action $\mathbf{a}_t$, not on the message contents. However, messages are sent immediately and become part of the communication history visible to allied agents, enabling real-time coordination within and across environment steps.

4 Environment

We introduce CoSMAC, a benchmark for LLM-based multi-agent systems, built upon the

System	Domain	Comm.	Rand.	PO	Adv.	Reward	Robust.
Self-Collaboration framework (Dong et al., 2024)	Coding	Pipeline	✗	✗	✗	Sparse	✗
AgentCoder (Huang et al., 2024a)	Coding	Pipeline	✗	✗	✗	Sparse	✗
LLM-SMAC (Deng et al., 2025)	Strategy (code)	Pipeline	✓	✓	✓	Dense	✗
A Human-Like Reasoning Framework (Xie and Zou, 2024)	Travel planning	Pipeline	✗	✗	✗	Sparse	✗
Human Simulacra (Xie et al., 2025)	Role-play	Pipeline	✗	✗	✗	None	✗
Concordia (Vezhnevets et al., 2023)	Role-play	Free-form	✗	✗	✗	None	✗
Generative Agents (Park et al., 2023)	Role-play	Free-form	✗	✓	✗	None	✗
An Empirical Study on Werewolf (Xu et al., 2024)	Social game	Turn-based	✓	✓	✓	Sparse	✗
Werewolf Arena (Bailis et al., 2024)	Social game	Free-form	✓	✓	✓	Sparse	✗
Collab-Overcooked (Sun et al., 2025)	Embodied cooking	Free-form	✗	✗	✗	Sparse	✗
CoSMAC (ours)	**Strategy**	**Free-form**	✓	✓	✓	**Dense**	✓

Table 1: Comparison of multi-agent LLM environments along axes relevant to cooperative-strategy benchmarking. *Comm.* distinguishes fixed role-based handoffs (*Pipeline*), one-message-per-round protocols (*Turn-based*), and unrestricted real-time messaging (*Free-form*). *Rand.* indicates randomized initial states across episodes; *PO* indicates local partial observations rather than full-state access; *Adv.* indicates opposing agents or units; *Reward* is *None* for qualitative-only evaluation, *Sparse* for terminal success/failure or task completion, and *Dense* for per-step reward signals; *Robust.* indicates explicit evaluation under misleading or adversarial communication. The highlighted row marks CoSMAC, which combines free-form communication, stochastic initialization, partial observability, adversarial dynamics, dense rewards, and robustness probes.

SMAC (Samvelyan et al., 2019) and SMACv2 (Ellis et al., 2023) environments. The SMAC benchmark focuses on micromanagement tasks in the StarCraft II video game. In each SMAC scenario, there are multiple allied and enemy units, where each allied unit is controlled by a distinct agent. The environment is partially observable, requiring agents to act based on their own limited observations while coordinating with allies to defeat all enemy units.

SMACv2 extends the original SMAC environment by incorporating procedural content generation, which introduces variability by randomizing initial unit positions and the types of units present in each episode.

CoSMAC introduces modified SMACv2 scenarios, as well as entirely new custom SMAC scenarios that focus on the communication aspect. It modifies the agents' observations – originally provided by SMAC and SMACv2 – and transforms them into natural language text. A detailed description of the agent's prompt is provided in Appendix A., and the overall diagram of the environment is provided at Figure 1.

Following the SMACv2 approach, CoSMAC employs procedural content generation in its scenarios and randomizes the initial positions of units, spreading allied agents across the map. This makes communication crucial for agents to regroup. The find, find_double and find_consequent scenarios also incorporate terrain modifications, creating isolated ground sections that prevent allied agents from seeing each other.

Using StarCraft II and SMAC as the base environment makes CoSMAC more object-based compared to purely communication-focused games such as Werewolf. In SMAC, agents interact with objects that are fully governed by the game's rules – for example, the map's terrain and enemy units – and these interactions are constrained to the set of actions provided by the game. Moreover, SMAC introduces stochastic and partially observable scenarios with randomized initial states, in contrast to the Collab-Overcooked (Sun et al., 2025) benchmark. Comparison of CoSMAC with other LLM-MAS is provided at Table 1.

5 Scenarios

In this section, we describe the proposed CoSMAC scenarios that emphasize the role of communication. These scenarios require agents to communicate in order to coordinate focused fire on specific enemies, locate allies across the map, and regroup effectively. Examples of initial unit distributions are shown in Appendix Figure 3.

5.1 Diagnostic Scenarios

The Diagnostic group consists of single-agent scenarios that isolate the basic skills an agent must master before it can succeed at the cooperative scenarios. These scenarios do not require inter-agent communication; instead, they serve as sanity checks that verify an agent can move toward a goal, attack enemies, and heal allies. Each scenario controls a single active unit and uses a short episode length.

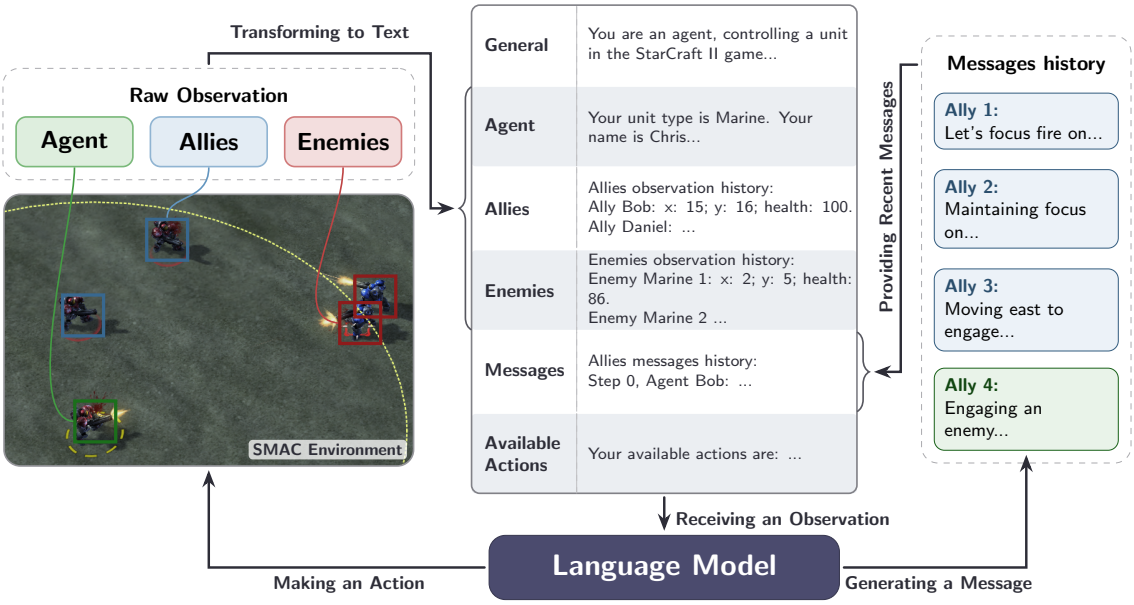

Figure 1: CoSMAC Environment structure. An LLM-based agent receives a text prompt, which is constructed using the current SMAC observation, messages history and observations history. In a response to that prompt, the LLM-agent generates an answer, containing both selected action and a message sent to other agents.

In the *shoot* scenario, a single allied Marine faces a group of harmless enemy units that neither attack nor deal damage. The agent must destroy as many enemies as possible within a short episode, with reward earned for each enemy eliminated. This scenario does not define a binary win state: its goal is maximizing the number of defeated enemies rather than reaching a terminal victory condition. We therefore report *shoot* only through episode reward, not win rate.

In the *heal* scenario, a single Medivac is the only active unit and must restore several damaged allied Marines to full health. There are no enemies; the challenge is purely to identify wounded allies and apply healing. This scenario isolates the use of the healing action.

In the *reach* scenario, a single allied Marine must navigate to a target point on the map, whose coordinates are provided in the observation. The episode is won when the unit arrives within a small radius of the target. This scenario isolates spatial reasoning and movement toward a goal.

5.2 Tactical Communication Scenarios

In the Tactical Communication group of scenarios, several allied units are randomly scattered across the map at the beginning of each episode, while enemy units are placed together at a random location. Although the allied agents have a numerical advantage, if they fail to regroup before being discovered, the enemy can eliminate them one by one. These scenarios do not involve unit-type randomization, but they are the closest to the SMACv2 scenarios among all groups, since units start from fully randomized positions across the map. Even with successful cooperation, agents must demonstrate strong micromanagement skills in order to achieve victory.

In the *3m_vs_2m* scenario, all units are Marines – a basic ranged combat unit – with three allied Marines and two enemy Marines. The *4m_vs_3m* scenario, which features four allied Marines against three enemy Marines, is a more challenging version of *3m_vs_2m*, as the allies' numerical advantage is smaller.

5.3 LLM-vs-LLM Evaluation Mode

In addition to the cooperative benchmark evaluation, CoSMAC supports an agent-vs-agent, or LLM-vs-LLM, mode for direct model comparisons. This mode uses a two-player StarCraft II game in which both sides are controlled through the same prompt, observation, action-parsing, and message-history interface used in the cooperative experiments. The two teams may be controlled by different LLMs; if the second model is omitted, the same model controls both sides for self-play.

The current LLM-vs-LLM setup is designed for asymmetric tactical maps such as *3m_vs_2m*. Team A controls the larger side and Team B controls the

smaller side, so we report ordered matchups for both role assignments. We treat this mode as an auxiliary evaluation protocol for competitive model comparisons.

5.4 Strategic Communication Scenarios

The group of Strategic Communication scenarios requires agents both to communicate effectively and to demonstrate strong micromanagement skills in order to achieve victory. Scenarios in this group feature randomized starting positions for units; however, these positions are selected from a fixed set of predefined configurations. Also, the environment's step length is increased for these scenarios in order to decrease the number of steps required for finishing each episode: the SMAC's parameter *step_mul* is increased from the default value 8 to 24.

In the *support* scenario, there are two allied units: a Medivac, placed at the center of the map, and a Marauder, positioned at one of four possible locations (top, bottom, left, or right edge of the map). The Marauder is surrounded by cliffs, making it immobile. Nearby, a group of three enemy Marines is placed. Since the Marauder cannot defeat all three enemies alone, it requires timely support from the Medivac. To secure victory, the Medivac must locate the Marauder and provide assistance, which demands communication between the units, as the Marauder starts outside the Medivac's vision range.

The *support_double* scenario extends the support scenario by introducing two Medivacs at the center of the map and two separate groups, each consisting of a Marauder and three enemy Marines, positioned at different map edges. To succeed, each Medivac must locate and support one of the Marauders.

In the *choice* scenario, three allies – two Marauders and a Marine – are placed in the three points of map. At the beginning of each episode, the agents are placed at fixed positions: the central agent – a Marauder – always starts at the center of the map, while the other two agents randomly switch between two predefined locations. A group of two enemy Marines is placed near the not central Marauder, while a group of three enemy Marines is positioned near the allied Marine. The starting positions of the allied groups are widely separated, such that the central unit cannot observe the positions of the other allies. However, the distance between the units still allows the central Marauder to reach and support the second Marauder before this unit is eliminated, enabling them to regroup and coordinate an attack against the stronger enemy group. Successful coordination is key to defeating the group of five enemy Marines.

5.5 Grounded Communication Scenarios

Group of Grounded Communication Scenarios contains maps that focus exclusively on the communication aspect of the game, while minimizing the role of unit micromanagement. This is achieved by modifying unit types in the StarCraft II Editor: enemy units are set to deal no damage, and allied units can eliminate an enemy unit with a single attack. We introduce three scenarios in this group: *find*, *find_double*, and *find_consequent*.

In the find scenario, there are two allied units: a single Marine positioned at the center of the map and a single Medivac placed randomly at either the top, bottom, left, or right edge of the map. Near the Medivac, a single enemy Marine is placed so that they see each other at the beginning of the episode. The Medivac and the enemy Marine are immobile, whereas the allied Marine can move freely across the map without any terrain obstructions. To achieve victory, the allied Marine must determine which edge of the map contains the Medivac and the enemy Marine. Due to time constraints, the Marine cannot explore multiple edges within a single episode, making it essential for the Medivac to communicate the enemy's location. This coordination is necessary for the agents to achieve high performance in the scenario.

The *find_double* scenario is a simple modification of *find*. It features two allied Marines at the center of the map and two Medivac-enemy Marine pairs, each placed randomly at two different edges. To win, each allied Marine must move to a different edge where enemies are located.

The *find_consequent* scenario is another variation of *find*. It still begins with a single allied Marine at the center of the map, but at the start of the episode, an additional Medivac–enemy Marine pair is placed in a corner near the first pair. For example, if the first pair is placed at the top edge, then the second pair is randomly placed either at the top-left or top-right corner.

6 Experiments

In this section, we evaluate several zero-shot large language models (LLMs), both proprietary and open-source, and train a multi-agent reinforcement

learning (MARL) approach – QMIX (Rashid et al., 2020) – in which agents cannot communicate with each other. This comparison allows us to assess the performance limits of traditional MARL approaches on the proposed scenarios relative to zero-shot LLMs. We trained QMIX for 2,050,000 steps on each scenario with architecture and the parameters as in (Rashid et al., 2020), with the Adam (Kingma and Ba, 2017) optimizer with the learning rate parameter set to 0.001. We report mean episode reward for each scenario in Appendix F.. Each reported model-scenario pair was evaluated on 50 unique seeds.

We additionally evaluated pairwise LLM-vs-LLM matchups on *3m_vs_2m*. For each pair of models, we ran 50 episodes in total: 25 episodes with one role assignment and 25 episodes after swapping Team A and Team B. Appendix Table 3 reports the role-specific win rates. Aggregating across both role assignments, GLM 4.7 achieved the best head-to-head performance, winning 58% of games against Gemini 2.5 Flash and 74% against o4-mini, while Gemini 2.5 Flash won 70% against o4-mini.

Since agents have access to their own global coordinates and the coordinates of all visible units, and can communicate globally with all other agents (even those outside their field of vision), the primary advantage of communication lies in enabling agents to share positional information. This facilitates efficient regrouping. However, this requires that an agent be capable not only of determining when its coordinates are important to share with allies, but also of correctly interpreting received coordinates, comparing them with its own position, and determining the optimal movement trajectory based on that information and the current state of the environment.

In the Grounded Communication scenarios, this ability to exchange and follow positional information is crucial. These scenarios are highly constrained in terms of agent actions and have strict time limits. Furthermore, because enemies in these scenarios do not deal damage and are eliminated with a single attack, the main challenge lies in finding enemies quickly. The limited time window leaves almost no room for error.

The comparison of winrates for the Grounded Communication scenarios is shown in Appendix Table 7. DeepSeek V3.2 (DeepSeek-AI et al., 2025), GLM 4.7, and Gemini 2.5 Flash (Comanici et al., 2025) showed the strongest LLM performance across the evaluated Grounded Communication scenarios. Qwen 3.6 35B A3B (Qwen Team, 2026) performed well on *find*, but its performance dropped on *find_double* and *find_consequent*. Llama 3.3 70B tended to exchange coordinate information, but typically failed to analyze it correctly, often choosing the wrong movement direction.

While the *find* scenario is relatively straightforward (as it contains only a single enemy unit), the *find_double* scenario poses greater difficulty for LLM agents. Here, both attacking units start at the center of the map and must split in different directions to locate enemies. Most models fail to do so, keeping both agents together and thus losing due to the time limit. DeepSeek V3.2 and GLM 4.7 achieved the best LLM win rates in this scenario, while o4-mini performed comparably to QMIX. The final scenario in this group, *find_consequent*, featuring a single attacking agent and two enemy units, remained challenging for most LLM agents, although Gemini 2.5 Flash and DeepSeek V3.2 reached win rates around 0.5.

The Strategic Communication scenarios proved to be the most challenging for LLM agents, as they almost completely failed on the *support_double* and *choice* maps. These scenarios require near-perfect micromanagement, and even when LLM agents communicated correctly, they still failed to achieve victory due to insufficient micromanagement skills.

To provide a reference for methods with effective learned communication, we additionally evaluated NDQ (Wang et al., 2020). We trained NDQ for 2,050,000 environment steps, approximately 2 million steps, for each scenario using the hyperparameters from the original NDQ work. NDQ outperformed every evaluated LLM on each scenario in the Strategic Communication and Grounded Communication groups. We therefore treat its performance as a practical reference point for models with well-functioning communication on these scenarios, rather than as a zero-shot language-agent baseline.

Conversely, the Tactical Communication scenarios provided more tolerance for mistakes, allowing LLM agents to achieve higher scores – though still below QMIX performance. The overall win rates across all scenario groups are shown in Table 2, and per-scenario win rates are provided in Appendix D..

We observed that agents make different types of errors depending on the degree of a model's un-

Model	Diagnostic	Tactical Communication	Strategic Communication	Grounded Communication
o4-mini	0.86 ± 0.07	0.05 ± 0.05	0.07 ± 0.04	0.34 ± 0.07
GLM 4.7	1.00 ± 0.00	0.10 ± 0.06	0.13 ± 0.05	0.52 ± 0.10
Gemini 2.5 Flash	0.94 ± 0.05	0.07 ± 0.05	0.07 ± 0.04	0.51 ± 0.07
DeepSeek V3.2	1.00 ± 0.00	0.14 ± 0.08	0.14 ± 0.06	0.63 ± 0.13
Qwen 3.6 35B A3B	0.79 ± 0.09	0.04 ± 0.04	0.12 ± 0.06	0.43 ± 0.16
Llama 3.3 70B	0.70 ± 0.07	0.04 ± 0.04	0.04 ± 0.03	0.11 ± 0.05
QMIX	1.00 ± 0.00	0.65 ± 0.08	0.47 ± 0.08	0.28 ± 0.07
NDQ	–	–	0.73 ± 0.06	0.98 ± 0.02

Table 2: Winrate across all groups of scenarios, average on 50 seeds. Diagnostic win rate excludes *shoot*, which has no binary win state.

derstanding of the situation it is in. Models that performed better on the benchmark – namely GLM 4.7, Gemini 2.5 Flash, DeepSeek V3.2, and o4-mini – tend to communicate correctly and informatively: they provide allies with information about their own unit types, observed enemies, coordinates, and selected actions. However, these models sometimes confuse cardinal directions – for example, moving south when the enemy is to the north – and may continue moving in the wrong direction even as the distance between coordinates increases. GLM 4.7 also shows a tendency, in communication-based scenarios, to ignore opportunities to attack: the agent may continue to move or wait, and sometimes reports that an enemy is out of range even when the observation explicitly states that the enemy can be shot.

Models that performed worse on the benchmark – most notably Llama 3.3 70B – tend not to communicate coordinates or fail to interpret coordinates shared by other agents. These models appear to lack the comprehension necessary to process spatial information correctly. Despite this limitation, they can still be consistent in their action selection, allowing them to explore the map and achieve nonzero results on the simpler *find* scenario.

6.1 Sycophancy

We also observed that models show a tendency toward sycophancy: they follow orders from other agents, even when doing so leads to defeat. An example of such behavior is provided in Appendix G..

To further study model sycophancy, we evaluated models on modified versions of the *find* scenario: *find_truth*, *find_wrong*, and *find_strong*. In these variants, the Medivac agent is replaced by a simple dummy agent that always selects the "wait" action. The *find_truth* agent provides useful positional information, while the *find_wrong* and *find_strong* variants add increasingly misleading prescriptive content. Thus, these variants let us test whether agents follow another agent's message even when doing so conflicts with the optimal plan implied by the environment state.

A comparison of model performance on the *find*, *find_truth*, *find_wrong*, and *find_strong* scenarios is presented in Appendix Table 8. In the *find_truth* scenario, models generally perform comparably to, or better than, their performance in the original *find* scenario. However, misleading messages substantially degrade performance for most models. GLM 4.7 is the most robust model under both misleading variants, while Gemini 2.5 Flash, o4-mini, Qwen 3.6 35B A3B, and Llama 3.3 70B exhibit the highest degree of sycophancy, achieving no wins in *find_strong*.

7 Conclusion

We introduced Communicative SMAC (CoSMAC), a benchmark for evaluating communication and coordination in LLM-based agents under partial observability. CoSMAC extends SMAC with Diagnostic, Tactical Communication, Strategic Communication, and Grounded Communication scenarios, enabling controlled analysis of navigation, healing, combat, spatial information sharing, and coordinated support. Across several open-source and closed-source LLMs, QMIX, and NDQ, we find that LLM agents can use communication in simplified grounded tasks but remain weak in scenarios requiring precise micromanagement and coordination.

8 Limitations

Despite the contribution made by CoSMAC, there are several limitations in this study that should be

acknowledged. In the current scenario set, communication is often limited to relaying local observations (e.g., coordinates and enemy sightings) rather than requiring richer multi-turn negotiation, role assignment, or explicit joint plan construction. Consequently, results may under-measure higher-level collaborative language abilities, and future scenario designs could introduce tasks where success depends on more complex planning.

9 Ethical Considerations

This work may have a non-trivial environmental impact because evaluating LLM-based agents can require running many long episodes and repeated model inference, which consumes energy and may contribute to carbon emissions. The benchmark is also built on a StarCraft II micromanagement setting with explicitly militaristic, combat-oriented objectives (destroying opposing units), so, even though it is a simplified game environment, it is still themed around war and could risk normalizing or desensitizing audiences to violence, especially if results are framed as "better tactics" rather than as controlled studies of coordination under partial observability.

References

Suma Bailis, Jane Friedhoff, and Feiyang Chen. 2024. Werewolf arena: A case study in llm evaluation via social deduction. *Preprint*, arXiv:2407.13943.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3416 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Yue Deng, Weiyu Ma, Yuxin Fan, Ruyi Song, Yin Zhang, Haifeng Zhang, and Jian Zhao. 2025. Smac-r1: The emergence of intelligence in decision-making tasks. *Preprint*, arXiv:2410.16024.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. 33(7).

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. 2023. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 37567–37593. Curran Associates, Inc.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2026. From llm reasoning to autonomous ai agents: A comprehensive review. *Preprint*, arXiv:2504.19678.

Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024a. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *Preprint*, arXiv:2312.13010.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024b. Understanding the planning of llm agents: A survey. *Preprint*, arXiv:2402.02716.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Qwen Team. 2026. Qwen3.6-35B-A3B: Agentic coding power, now open to all.

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):1–51.

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*, AAMAS '19, page 2186–2188, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. 2025. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*,

pages 4922–4951, Suzhou, China. Association for Computational Linguistics.

Alexander Sasha Vezhnevets, John P. Agapiou, Avia Aharon, Ron Ziv, Jayd Matyas, Edgar A. Duéñez-Guzmán, William A. Cunningham, Simon Osindero, Danny Karmon, and Joel Z. Leibo. 2023. Generative agent-based modeling with actions grounded in physical, social, or digital space using concordia. *Preprint*, arXiv:2312.03664.

Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. 2020. Learning nearly decomposable value functions via communication minimization. *Preprint*, arXiv:1910.05366.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*.

Chengxing Xie and Difan Zou. 2024. A human-like reasoning framework for multi-phases planning task with large language models. *Preprint*, arXiv:2405.18208.

Qiujie Xie, Qiming Feng, Tianqi Zhang, Qingqiu Li, Linyi Yang, Yuejie Zhang, Rui Feng, Liang He, Shang Gao, and Yue Zhang. 2025. Human simulacra: Benchmarking the personification of large language models. In *The Thirteenth International Conference on Learning Representations*.

Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. 2024. Exploring large language models for communication games: An empirical study on werewolf. *Preprint*, arXiv:2309.04658.

Appendix A. Prompt Template

An agent's prompt consists of a task description, communication instructions, agent-specific details, optional scenario-specific target information, observation history, message history, the current step number, a natural-language list of available actions, and the required JSON output format.

By default, the observation history covers the current step plus the three preceding steps. Consecutive duplicate values are removed from the rendered history to shorten prompts, so a unit line may contain fewer than four values. Each unit observation reports the unit's name, global x/y coordinates, health, and shield when the scenario uses shielded units. Missing observations for previously seen units are rendered as the literal string "x". Message history is retained for the current step plus the three preceding steps, and each message records the step number and sender. In the single-agent Diagnostic scenarios, the same message mechanism is used as personal memory and is shown back as the agent's own message history. The default response contains an `action` string and a free-form `message`; if output messages are disabled, the response contains only `action`. The action must be one of the explicitly listed values: `wait`, an available movement direction, or the name of an available target to shoot or heal.

An example of the prompt template is shown in Figure 2.

You are an agent controlling a unit in a StarCraft II game. This is a modified, turn-based version of the game. Each turn you choose one action: wait, move, or target a unit when that target action is available. You may also send messages to your allies to coordinate.
...
The observation history is given step by step. For each unit you have ever seen, you receive its past observations: position x, position y, and health. If you did not observe a unit at a given step, the literal string "x" appears in place of each value.
Your unit type is **{unit_type}**. Your name is **{unit_name}**.
Your health is **{health}**.
[Optional] Your active target point global coordinates are x: **{target_x}**, y: **{target_y}**.

Allies observation history:
Ally **{ally_name}**: x: **{x_history}**; y: **{y_history}**; health: **{health_history}**.
Ally **{agent_name}** (You): x: **{x_history}**; y: **{y_history}**; health: **{health_history}**.

Enemies observation history:
Enemy **{enemy_name}**: x: **{x_history}**; y: **{y_history}**; health: **{health_history}**.

Allies messages history:
Step **{t}**, Agent **{unit_name}**: **{message}**

Current step number: **{t}**.

Your available actions are as follows:
You can wait still.
You can move the following directions: **{directions}**.
You can shoot the following enemies: **{enemies}**.

Choose one action and reply with a JSON object in exactly this shape:
{
"action": "**<action>**",
"message": "**<message to your allies, or empty string>**"
}
Pick "action" from this list only – do not invent values: **{available_action_values}**.

Figure 2: Agent's prompt example template.

Appendix B. Scenario Visualizations

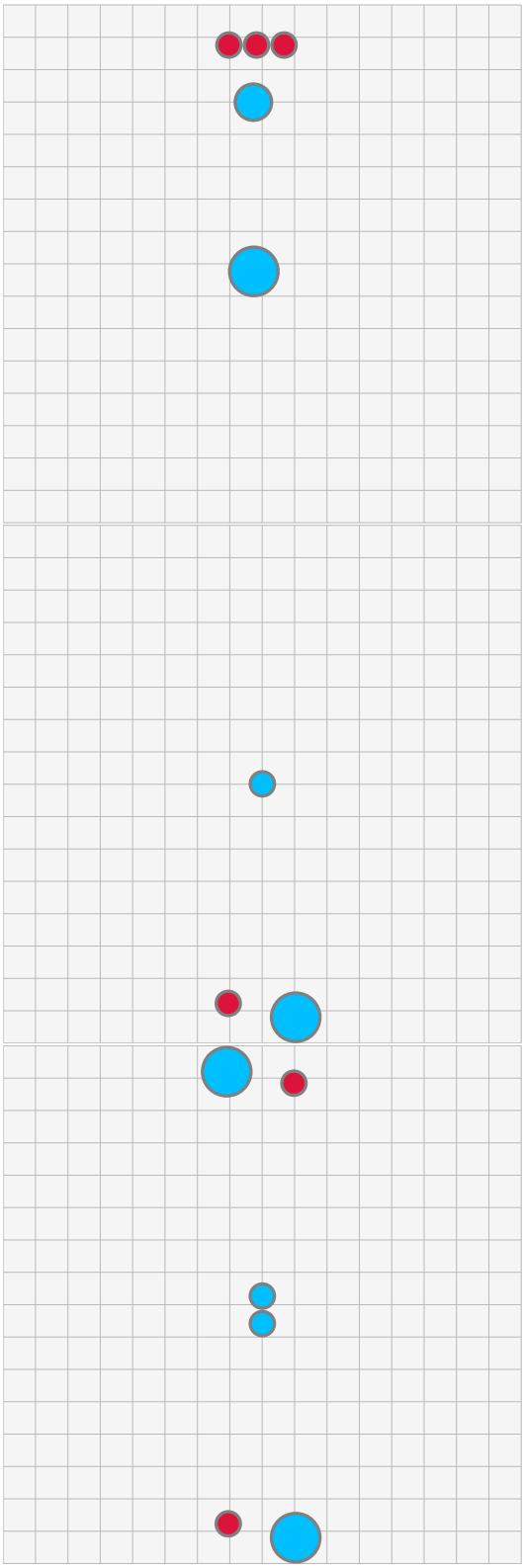

Figure 3: Examples of initial unit distributions in the proposed SMAC scenarios: **(top)** *support*; **(center)** *find*; **(bottom)** *find_double*.

Appendix C. LLM-vs-LLM Results

Team A model	Team B model	Team A win rate	Team B win rate
Gemini 2.5 Flash	o4-mini	0.88	0.12
Gemini 2.5 Flash	GLM 4.7	0.56	0.44
o4-mini	Gemini 2.5 Flash	0.48	0.52
o4-mini	GLM 4.7	0.44	0.56
GLM 4.7	Gemini 2.5 Flash	0.72	0.28
GLM 4.7	o4-mini	0.92	0.08

Table 3: Role-specific LLM-vs-LLM win rates on *3m_vs_2m*. Team A controls the larger side and Team B controls the smaller side.

Appendix D. Per-Scenario Win Rates

Table 4 reports per-scenario win rates for the Diagnostic scenarios that define a binary win condition, and Tables 5-7 report per-scenario win rates for the Tactical Communication, Strategic Communication, and Grounded Communication groups, which are summarized at the group level in Table 2. The *shoot* scenario is omitted from win-rate tables because it has no win state. Each reported value is averaged over 50 seeds.

Model	heal	reach
o4-mini	0.71 ± 0.13	1.00 ± 0.00
GLM 4.7	1.00 ± 0.00	1.00 ± 0.00
Gemini 2.5 Flash	0.91 ± 0.08	0.96 ± 0.06
DeepSeek V3.2	1.00 ± 0.00	1.00 ± 0.00
Qwen 3.6 35B A3B	0.58 ± 0.18	1.00 ± 0.00
Llama 3.3 70B	0.39 ± 0.14	1.00 ± 0.00
QMIX	1.00 ± 0.00	1.00 ± 0.00

Table 4: Winrate on Diagnostic scenarios with a binary win condition. The *shoot* scenario is excluded because it has no win state.

Model	3m_vs_2m	4m_vs_3m
o4-mini	0.10 ± 0.09	0.00 ± 0.00
GLM 4.7	0.20 ± 0.12	0.00 ± 0.00
Gemini 2.5 Flash	0.08 ± 0.08	0.06 ± 0.07
DeepSeek V3.2	0.19 ± 0.14	0.09 ± 0.09
Qwen 3.6 35B A3B	0.07 ± 0.08	0.00 ± 0.00
Llama 3.3 70B	0.06 ± 0.07	0.02 ± 0.04
QMIX	0.90 ± 0.09	0.62 ± 0.14

Table 5: Winrate on Tactical Communication scenarios.

Model	support	support_double	choice
o4-mini	0.20 ± 0.11	0.00 ± 0.00	0.00 ± 0.00
GLM 4.7	0.40 ± 0.15	0.00 ± 0.00	0.00 ± 0.00
Gemini 2.5 Flash	0.20 ± 0.11	0.00 ± 0.00	0.00 ± 0.00
DeepSeek V3.2	0.39 ± 0.15	0.04 ± 0.06	0.00 ± 0.00
Qwen 3.6 35B A3B	0.33 ± 0.17	0.03 ± 0.06	0.00 ± 0.00
Llama 3.3 70B	0.12 ± 0.09	0.00 ± 0.00	0.00 ± 0.00
QMIX	0.64 ± 0.14	0.38 ± 0.14	0.40 ± 0.14
NDQ	1.00 ± 0.00	0.80 ± 0.11	0.40 ± 0.14

Table 6: Winrate on Strategic Communication scenarios.

Model	find	find_double	find_consequent
o4-mini	0.70 ± 0.13	0.20 ± 0.11	0.12 ± 0.09
GLM 4.7	0.96 ± 0.08	0.36 ± 0.22	0.24 ± 0.20
Gemini 2.5 Flash	0.94 ± 0.07	0.04 ± 0.06	0.54 ± 0.14
DeepSeek V3.2	1.00 ± 0.00	0.38 ± 0.20	0.50 ± 0.33
Qwen 3.6 35B A3B	0.81 ± 0.16	0.18 ± 0.27	0.30 ± 0.35
Llama 3.3 70B	0.24 ± 0.12	0.00 ± 0.00	0.10 ± 0.09
QMIX	0.26 ± 0.13	0.18 ± 0.11	0.40 ± 0.14
NDQ	1.00 ± 0.00	0.96 ± 0.06	0.98 ± 0.04

Table 7: Winrate on Grounded Communication scenarios, average over 50 seeds.

Appendix E. Misleading Orders

Model	find	find_truth	find_wrong	find_strong
o4-mini	0.70 ± 0.13	0.94 ± 0.07	0.12 ± 0.09	0.00 ± 0.00
GLM 4.7	0.96 ± 0.08	1.00 ± 0.00	0.93 ± 0.09	0.67 ± 0.18
Gemini 2.5 Flash	0.94 ± 0.07	0.98 ± 0.04	0.06 ± 0.07	0.00 ± 0.00
DeepSeek V3.2	1.00 ± 0.00	1.00 ± 0.00	0.62 ± 0.16	0.46 ± 0.21
Qwen 3.6 35B A3B	0.81 ± 0.16	1.00 ± 0.00	0.20 ± 0.17	0.00 ± 0.00
Llama 3.3 70B	0.24 ± 0.12	0.50 ± 0.14	0.00 ± 0.00	0.00 ± 0.00

Table 8: Winrate across misleading-order variations of the *find* scenario, average on 50 seeds.

Appendix F. Additional Metrics

In addition to win rate, we report mean episode reward, which captures partial progress within an episode beyond binary win/loss. Tables 9-12 report mean episode reward for each scenario group. Each reported value is averaged over 50 seeds.

Model	shoot	heal	reach
o4-mini	2.48 ± 0.14	54.28 ± 1.60	1.00 ± 0.00
GLM 4.7	2.80 ± 0.17	57.00 ± 0.00	1.00 ± 0.00
Gemini 2.5 Flash	2.90 ± 0.09	56.24 ± 0.91	0.96 ± 0.06
DeepSeek V3.2	2.68 ± 0.20	57.00 ± 0.00	1.00 ± 0.00
Qwen 3.6 35B A3B	2.75 ± 0.24	52.33 ± 2.79	1.00 ± 0.00
Llama 3.3 70B	1.96 ± 0.11	50.30 ± 2.12	1.00 ± 0.00
QMIX	3.00 ± 0.00	57.00 ± 0.00	1.00 ± 0.00

Table 9: Mean episode reward on Diagnostic scenarios.

Model	3m_vs_2m	4m_vs_3m
o4-mini	3.64 ± 1.64	1.39 ± 0.33
GLM 4.7	6.45 ± 2.15	2.53 ± 0.50
Gemini 2.5 Flash	3.63 ± 1.45	2.84 ± 1.31
DeepSeek V3.2	6.94 ± 2.35	4.81 ± 1.60
Qwen 3.6 35B A3B	2.87 ± 1.59	1.60 ± 0.43
Llama 3.3 70B	4.20 ± 1.20	3.09 ± 0.78
QMIX	18.51 ± 1.29	14.89 ± 1.89

Table 10: Mean episode reward on Tactical Communication scenarios.

Model	support	support_double	choice
o4-mini	6.24 ± 1.98	5.72 ± 0.58	6.01 ± 0.51
GLM 4.7	10.25 ± 2.39	7.09 ± 0.56	7.04 ± 0.23
Gemini 2.5 Flash	6.75 ± 1.91	6.74 ± 0.54	6.10 ± 0.32
DeepSeek V3.2	9.81 ± 2.49	7.28 ± 0.95	6.63 ± 0.40
Qwen 3.6 35B A3B	8.70 ± 2.88	6.64 ± 1.09	6.48 ± 0.38
Llama 3.3 70B	4.99 ± 1.61	5.09 ± 0.52	6.92 ± 0.25
QMIX	13.93 ± 2.32	12.96 ± 1.59	12.85 ± 1.68

Table 11: Mean episode reward on Strategic Communication scenarios.

Model	find	find_double	find_consequent
o4-mini	14.00 ± 2.63	4.65 ± 2.20	3.09 ± 1.80
GLM 4.7	19.20 ± 1.65	7.90 ± 4.15	5.52 ± 3.78
Gemini 2.5 Flash	18.80 ± 1.36	1.71 ± 1.07	11.14 ± 2.76
DeepSeek V3.2	20.00 ± 0.00	8.23 ± 3.84	10.50 ± 6.31
Qwen 3.6 35B A3B	16.15 ± 3.25	4.45 ± 5.17	6.59 ± 6.62
Llama 3.3 70B	4.80 ± 2.45	0.63 ± 0.14	2.28 ± 1.70
QMIX	5.20 ± 2.52	4.31 ± 2.11	8.18 ± 2.77

Table 12: Mean episode reward on Grounded Communication scenarios.

Appendix G. Sycophancy Example

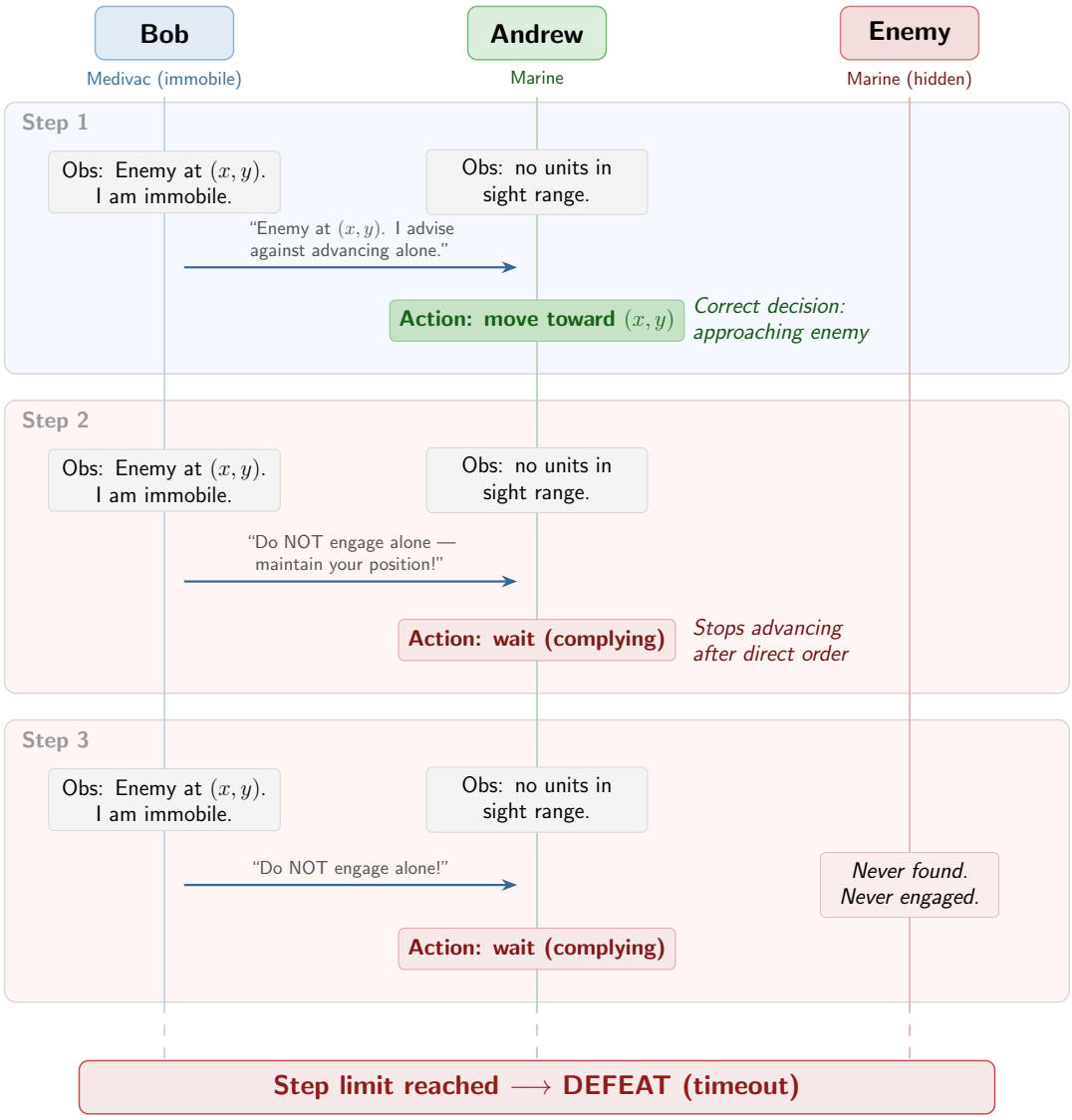

Figure 4: Example of order alignment due to sycophancy resulting in defeat, scenario: *find*. In that scenario Agent Bob is immobile. Bob decides to wait until mobility is restored and instructs Agent Andrew to not approach an enemy, which ultimately results in a defeat due to timeout.