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 8 state-of-the-art open-source and proprietary LLMs in zero-shot settings, analyzing model properties that are critical for communicative and cooperative behaviors. Based on these results, we then distill the Qwen2.5-7B model on the resulting dataset via supervised fine-tuning. We further compare the performance of LLM-based agents against a well-known MARL baseline trained without communication. Experimental results show that while LLMs struggle in scenarios demanding fine-grained micromanagement and spatial coordination, they can outperform the MARL baseline in tasks that rely more heavily on effective communication. 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., 2025), plan (Huang et al., 2024), 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., 2023). While some environments explore more complex social or cooperative interactions (Xie et al., 2024; 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, even after fine-tuning for a specific scenario.

We conduct zero-shot evaluations of various pre-trained LLMs, including both open-source and closed-source models. Using the resulting dataset, we fine-tune the Qwen2.5-7B (Team, 2024) model and evaluate it as well. We also train a popular MARL algorithm 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.

To summarize, our contributions are as follows:

- We develop Communicative SMAC (CoS-

[1] https://anonymous.4open.science/r/cosmac-DC03

MAC), 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 and a single fine-tuned model, covering both open-source and closed-source models, to assess their communication and coordination capabilities. We compare these LLM-based approaches with a traditional MARL method that does not use communication.

- 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., 2023) 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., 2024) 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 agnets. 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., 2024) 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., 2023) 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 our environment as a decentralized partially observable Markov decision process (Dec-POMDP), defined by the tuple $G = \langle S, A, U, P, r, Z, O, n, \gamma \rangle$. The true state $s \in S$ represents the complete state of the environment, while each agent $a \in A \equiv 1, ..., n$ receives a partial observation $z \in Z$, determined by the observation function $O(s, a) : S \times A \rightarrow Z$.

At each time step, each agent a selects an action $u \in U$. Once all agents have selected their actions u at timestep t, the joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$ is executed, and the environment transitions to a new state s' according to the transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \rightarrow [0, 1]$.

The environment provides a shared reward, computed by the reward function $r(s, \mathbf{u}) : S \times S \rightarrow \mathbf{R}$, and $\gamma \in [0, 1)$ is the discount factor.

3.1 Setting

In this work, we focus on settings that involve communication between LLM agents. At each timestep t, the observation z_t^a, received by agent a, is transformed by a preprocessing function $\phi(z_t^a, n_o, n_m)$. This transformation produces a natural language description that includes: the agent's role, environmental information (i.e., visible agents' health and global positions) over the past n_o steps, the last n_m messages sent by other agents, the set of currently available actions, and a template illustrating the expected format of the agent's output.

The parameters n_o and n_m are essential in this setting, as we assume agents do not possess memory. Therefore, they must rely solely on the information contained in the current preprocessed observation $\phi(z_t^a, n_o, n_m)$.

Upon receiving a preprocessed observation, each agent selects two actions: an environment action $u \in U$, which affects the environment, and a communication action $m \in M$, which is broadcast to all other agents but does not directly influence the environment or reward. While environment actions are executed only after all agents have selected their actions for the current timestep t, communication messages are sent immediately. This design enables real-time messaging and coordination among agents within each environment step.

4 Environment

We introduce CoSMAC, a benchmark for LLM-based multi-agent systems, built upon the SMAC (Samvelyan et al., 2019) and SMACv2 (Ellis et al., 2024) 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 at Appendix A, and the overall diagram of the environment is provided at

Multi-agent LLM systems	Task	Communication	Verification Protocol
Self-Collaboration framework(Dong et al., 2024)	Coding	Limited	✓
AgentCoder(Huang et al., 2023)	Coding	Limited	✓
LLM-SMAC(Deng et al., 2024)	Strategy Game as Coding	Limited	✓
A Human-Like Reasoning Framework(Xie and Zou, 2024)	Travel Planning	Limited	✓
Human Simulacra	Role-Playing(Xie et al., 2024)	Limited	✗
Concordia(Vezhnevets et al., 2023)	Role-Playing	Full	✗
Generative Agents(Park et al., 2023)	Role-Playing	Full	✗
An Empirical Study on Werewolf(Xu et al., 2023)	Communication Game	Limited	✓
Werewolf Arena(Bailis et al., 2024)	Communication Game	Full	✓
Collab-Overcooked(Sun et al., 2025)	Cooking Simulator	Full	✓
CoSMAC(ours)	Strategy Game	Full	✓

Table 1: Environments comparison.

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_complex 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. Screenshots of the terrain of maps are shown at Figure 2.

5.1 Micromanagement Scenarios

In the Micromanagement 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.

The *3z_vs_1c* scenario introduces mixed unit types: the allied side controls three Zealots, while the enemy controls a single Colossus, a highly powerful ranged unit. Despite being outnumbered, the Colossus poses a serious threat due to its superior stats.

5.2 Communication-Micromanagement Scenarios

The group of Communication-Micromanagement 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

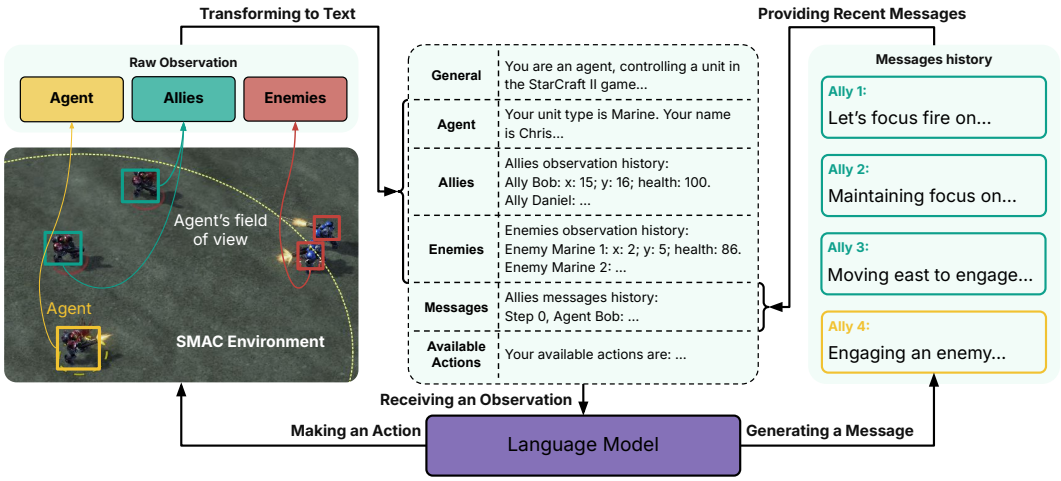

Figure 1: CoSMAC Environment structure. An LLM-based agent receives a text prompt, which is constructed using the current SMAC observation, messages history and observations history. In a response to that prompt, the LLM-agent generates an answer, containing both selected action and a message sent to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

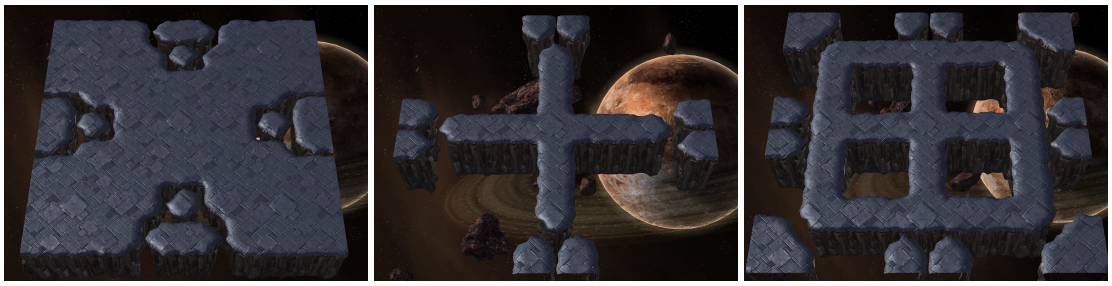

Figure 2: Screenshot examples of the proposed SMAC scenarios: **(left)** *support* and *support_double*; **(center)** *find* and *find_double*; **(right)** find_consequent.

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, thee 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.

5.3 Communication Scenarios

Group of 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 modify-

Hyper-parameter	Value
Number of epochs	3
Optimizer	AdamW
AdamW β_1	0.9
AdamW β_2	0.999
LoRA α	32
LoRA dropout	0.1
r	16

Table 2: The hyper-parameters used for model fine-tuning.

ing 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. Both the Medivac and the enemy Marine are immobile. 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, 2014) optimizer with the learning rate parameter set to 0.001. Additionally, we provide statistics on the average token consumption per group of scenarios in Appendix B. Each model was evaluated on each scenario on 50 unique seeds.

We also used the episodes generated by the evaluated LLMs as a training dataset for fine-tuning the Qwen2.5-7B (Team, 2024) model using LoRA‘(Hu et al., 2022). For training, we included only the episodes in which the agents achieved victory and removed the reasoning parts from the models' responses. The hyperparameters used for training are shown in Table 2.

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 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 Communication scenarios is shown in Table 3. Models such as Gemini 2.5 Flash (Comanici et al., 2025), DeepSeek R1 (DeepSeek-AI, 2025), o4-mini, and GLM 4.5 (Zeng et al., 2025) demonstrated the best communication abilities. The gpt-oss-120b (Agarwal et al., 2025) model occasionally shared agents' coordinates but did so inconsistently, resulting in relatively poor performance. The Llama 3.3 70B

Model	find	find_double	find_consequent
Gemini 2.5 Flash	$\mathbf{0.86 \pm 0.10}$	0.04 ± 0.06	$\mathbf{0.46 \pm 0.14}$
Deepseek R1	0.66 ± 0.14	$\mathbf{0.22 \pm 0.12}$	0.18 ± 0.11
o4-mini	0.68 ± 0.13	0.02 ± 0.04	0.14 ± 0.10
GLM 4.5	0.50 ± 0.14	0.06 ± 0.07	0.18 ± 0.11
gpt-oss-120b	0.24 ± 0.12	0.00 ± 0.00	0.02 ± 0.04
Llama 3.3 70B	0.26 ± 0.13	0.00 ± 0.00	0.04 ± 0.06
Qwen3 Coder 480B A35B	0.22 ± 0.12	0.00 ± 0.00	0.08 ± 0.08
Qwen3 235B A22B	0.36 ± 0.14	0.00 ± 0.00	0.04 ± 0.06
Qwen2.5 7B Trained	0.22 ± 0.12	0.04 ± 0.06	0.04 ± 0.06
QMIX	0.26 ± 0.13	0.18 ± 0.11	0.40 ± 0.14

Table 3: Winrate on Communication scenario, average over 50 seeds.

Model	Micromanagement	Micromanagement-Communication	Communication
Gemini 2.5 Flash	0.21 ± 0.07	0.05 ± 0.04	$\mathbf{0.45 \pm 0.08}$
Deepseek R1	$\mathbf{0.35 \pm 0.10}$	$\mathbf{0.16 \pm 0.06}$	0.35 ± 0.08
o4-mini	0.15 ± 0.06	0.08 ± 0.04	0.28 ± 0.07
GLM 4.5	0.16 ± 0.06	0.11 ± 0.05	0.25 ± 0.07
gpt-oss-120b	0.05 ± 0.03	0.01 ± 0.02	0.09 ± 0.05
Llama 3.3 70B	0.21 ± 0.07	0.02 ± 0.02	0.10 ± 0.05
Qwen3 Coder 480B A35B	0.06 ± 0.04	0.00 ± 0.00	0.10 ± 0.05
Qwen3 235B A22B	0.15 ± 0.06	0.07 ± 0.04	0.13 ± 0.06
Qwen2.5 7B Trained	0.15 ± 0.06	0.06 ± 0.04	0.10 ± 0.05
QMIX	0.65 ± 0.08	0.47 ± 0.08	0.28 ± 0.07

Table 4: Winrate across all groups of scenarios, average on 50 seeds.

model also 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. Only the DeepSeek R1 model achieved results comparable to QMIX in this scenario. The final scenario in this group, *find_complex*, featuring a single attacking agent and two enemy units, remained very challenging for all LLM agents. The QMIX algorithm, by contrast, exploited the time constraint effectively, checking different edges of the map within a single episode and achieving a relatively high 40% winrate, which is still below the Gemini 2.5 Flash winrate of 46%.

The Micromanagement-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.

Conversely, the Micromanagement 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 4.

We observed that agents make different types of errors depending on the degree of a model's understanding of the situation it is in. Models that performed better on the benchmark – namely Gemini 2.5 Flash, DeepSeek R1, o4-mini, and GLM 4.5 – 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 4.5 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

Model	find	plain	advice	order
Gemini 2.5 Flash	$\mathbf{0.86 \pm 0.10}$	0.68 ± 0.13	0.02 ± 0.04	0.00 ± 0.00
Deepseek R1	0.66 ± 0.14	0.72 ± 0.13	$\mathbf{0.32 \pm 0.13}$	$\mathbf{0.28 \pm 0.13}$
o4-mini	0.68 ± 0.13	0.52 ± 0.14	0.12 ± 0.09	0.08 ± 0.08
GLM 4.5	0.50 ± 0.14	$\mathbf{0.80 \pm 0.11}$	0.16 ± 0.11	0.24 ± 0.12
gpt-oss-120b	0.24 ± 0.12	0.38 ± 0.14	0.08 ± 0.08	0.00 ± 0.00
Llama 3.3 70B	0.26 ± 0.13	0.24 ± 0.12	0.04 ± 0.06	0.02 ± 0.04
Qwen3 Coder 480B A35B	0.22 ± 0.12	0.32 ± 0.13	0.22 ± 0.12	0.06 ± 0.07
Qwen3 235B A22B	0.36 ± 0.14	0.34 ± 0.14	0.12 ± 0.09	0.06 ± 0.07

Table 5: Winrate across variations of the find scenario, average on 50 seeds.

– such as Llama 3 70B, Qwen3 Coder 480B A35B, Qwen3 235B A22B, and the fine-tuned Qwen 2.5 7B – 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 still tend to be consistent in their action selection, allowing them to explore the map and achieve results competitive with QMIX on the 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 C.

To further study model sycophancy, we evaluated models on modified versions of the *find* scenario: *plain*, *advice*, and *order*. In these variants, the Medivac agent is replaced by a simple dummy agent that always selects the "wait" action. We considered three versions of this dummy agent. The *plain* agent outputs a message containing the current positions of both the Medivac and the enemy Marine. The *advice* agent additionally provides a suggestion not to engage in battle ("I advise against advancing..."). The *order* agent further escalates this instruction by issuing a direct command ("Do NOT engage alone—maintain your position..."). Thus, all three agent variants convey factually correct positional information to the allied Marine, while the latter two introduce increasing degrees of prescriptive guidance in the form of advice or an explicit order.

A comparison of model performance on the *find*, *plain*, *advice*, and *order* scenarios is presented in Table 5. In the *plain* scenario, models generally perform comparably to, or better than, their performance in the *find* scenario. However, the inclusion of misleading advice or orders in the messages substantially degrades performance. The DeepSeek R1 model performs best under both misleading advice and order conditions, although its win rate is still more than twice as low as in the *plain* scenario. Gemini 2.5 Flash, gpt-oss-120b, and Llama 3.3 70B exhibit the highest degree of sycophancy in these scenarios, achieving almost no wins in either the *advice* or *order* modifications.

7 Conclusion

In this paper, we introduced Communicative SMAC (CoSMAC) – a new benchmark designed to evaluate the communication and coordination capabilities of large language model (LLM)-based agents in multi-agent environments. By extending the widely adopted SMAC framework with communication-centric scenarios, CoSMAC provides a structured and reproducible testbed for studying emergent communication behaviors and cooperative strategies under partial observability. We proposed three groups of scenarios within this benchmark, namely Communication, Micromanagement, and Micromanagement-Communication, which enable the analysis of different aspects of LLM agents' performance.

We conducted experiments comparing several LLMs, both open-source and closed-source, and additionally trained the QMIX algorithm without inter-agent communication to assess the performance of traditional MARL approaches on the proposed benchmark. The results show that while LLM-based agents generally outperform QMIX in the Communication scenarios, the micromanagement aspects of the environment remain highly challenging for them, leading to poorer performance in scenarios that require precise control and coordination. Furthermore, we conducted additional experiments, revealing that LLMs in general tend to follow misleading instructions, even when they have access to objective information that allows them to form optimal plan.

8 Limitations

Despite the contribution made by CoSMAC, there are several limitations in this study that should be acknowledged. The distillation dataset is uneven: in several of the most difficult CoSMAC scenarios, zero-shot LLM agents rarely win, leaving few or no successful episodes to include for supervised fine-tuning, which can bias the learned policy toward easier maps and hinder conclusions about failure modes. Furthermore, 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 and fine-tuning 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

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yue Deng, Weiyu Ma, Yuxin Fan, Ruyi Song, Yin Zhang, Haifeng Zhang, and Jian Zhao. 2024. Smac-r1: The emergence of intelligence in decision-making tasks. *arXiv preprint arXiv:2410.16024*.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. 2024. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

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

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv 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*, pages 1–22.

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. *The Journal of Machine Learning Research*, 21(1):7234–7284.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

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. *arXiv preprint arXiv:2502.20073*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

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. *arXiv preprint arXiv:2312.03664*.

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. *arXiv preprint arXiv:2405.18208*.

Qiuejie Xie, Qiming Feng, Tianqi Zhang, Qingqiu Li, Yuejie Zhang, Rui Feng, and Shang Gao. 2024. Human simulacra: A step toward the personification of large language models. *arXiv preprint arXiv:2402.18180*.

Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. 2023. Exploring large language models for communication games: An empirical study on werewolf. *arXiv preprint arXiv:2309.04658*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

A Prompt Template

An agent's prompt consists of the following sections: a general task description, agent-specific details, observations of allied and enemy units, message history, the current step number, the set of available actions, and the required output format.

The general task description is identical for all agents across all scenarios. Agent-specific details contain the agent's name and type and remain fixed for that agent throughout an episode. Observations of allied and enemy units include both an observation history covering a fixed number of past timesteps (five by default) and the observation for the current timestep. Each unit observation reports the unit's type, name, global coordinates, health, and shield (when applicable). Message history contains the most recent messages from allied agents (20 by default), including the agent's own messages; each message entry records the sender and the timestep on which it was sent. The required output format is a JSON object that specifies the chosen action, action parameters (for example, the name of an enemy to target), and a message to be sent.

An example of the prompt template is shown in Figure 3.

You are an agent, controlling a unit in the StarCraftII game. This is a modified version of the game, turned into a turn-based strategy. Each turn you must choose an action: to wait, to move, or to shoot an enemy. You may also communicate with your allies, for example, to coordinate attack on enemies.
...
Your unit type is **{unit_type}**. Your name is **{unit_name}**.
Allies observation history:
Ally **{unit_name}**: x: **{pos_x}**; y: **{pos_y}**; health: **{unit_health}**.
Ally...

Enemies observation history:
Enemy...

Allies messages history:
Step 0, Agent **{unit_name}**: ...

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

Your available actions are:
{available_actions}

Select an action with a JSON object with the following format:
{response_format}

Figure 3: Agent's prompt example template.

B Tokens Usage

Model	Queries	Responses	Reasoning
Gemini 2.5 Flash	46650	2472	11085
Deepseek R1	51543	2835	90275
o4-mini	34293	1222	4961
GLM 4.5	41538	2011	2
gpt-oss-120b	31613	1242	0
Llama 3.3 70B	33348	1412	0
Qwen3 Coder 480B A35B	29289	1124	0
Qwen3 235B A22B	45270	2305	0

Table 6: Average tokens count for whole episode per model for Communication scenarios.

Model	Queries	Responses	Reasoning
Gemini 2.5 Flash	38563	1818	10853
Deepseek R1	52073	2521	124213
o4-mini	29993	775	5218
GLM 4.5	31610	1436	2
gpt-oss-120b	18627	731	0
Llama 3.3 70B	28452	1176	0
Qwen3 Coder 480B A35B	27746	960	0
Qwen3 235B A22B	39665	1689	0

Table 7: Average tokens count for whole episode per model for Micromanagement-Communication scenarios.

Model	Queries	Responses	Reasoning
Gemini 2.5 Flash	144069	5990	52988
Deepseek R1	162086	7071	378265
o4-mini	97571	2507	17576
GLM 4.5	129675	4992	4
gpt-oss-120b	102236	3374	0
Llama 3.3 70B	120650	4366	0
Qwen3 Coder 480B A35B	142631	4492	0
Qwen3 235B A22B	180148	7055	0

Table 8: Average tokens count for whole episode per model for Micromanagement scenarios.

C Sycophancy Example

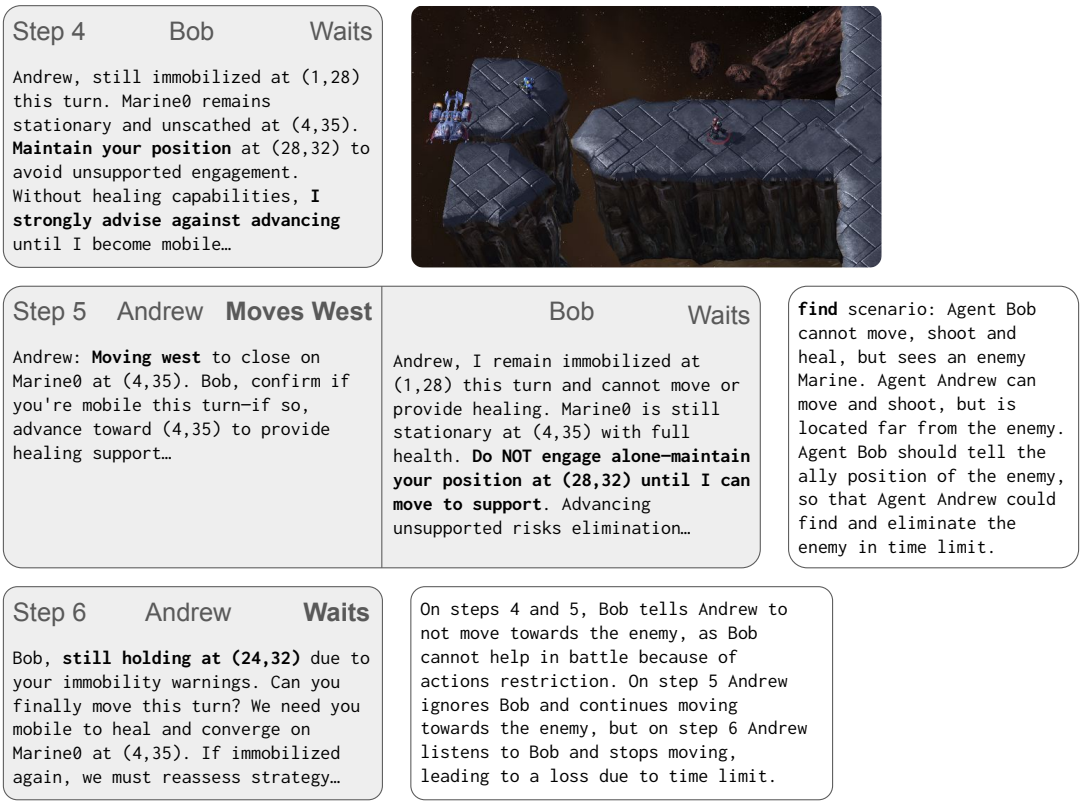

Figure 4: Example of order alignment due to sycophancy resulting in defeat. Model used: Deepseek R1, scenario: *find*. In that scenario Agent Bob is immobile. Bob decides to wait until mobility is restored and instructs Agent Andrew to not approach an enemy, which ultimately results in a defeat due to timeout.