

CoEmbod: A Long-Horizon Benchmark for Multi-Agent Embodied Cooperation with Diverse Inter-Agent Dependencies

Xuan Dong*, Zhe Han*, Ru Yu, Tianhao Niu, Muzhen Cai, Xinda Lv,
Qingfu Zhu[†], Weinan Zhang[†], Wanxiang Che

Harbin Institute of Technology

{xd,zhan,qfzhu,car}@ir.hit.edu.cn

*Equal contribution. [†]Corresponding author.

Abstract

Embodied planning is dedicated to the goal of building agents capable of perceiving, reasoning, and executing long-horizon tasks in physical environments. However, existing embodied benchmarks predominantly focus on single-agent execution or restricted multi-agent settings, leaving long-horizon tasks that involve complex inter-agent dependencies largely underexplored despite their prevalence in real-world embodied scenarios. To address this gap, we introduce **CoEmbod**, the first realistic 3D simulated embodied benchmark for long-horizon multi-agent embodied cooperation with diverse inter-agent dependencies. CoEmbod contains 155 expert-curated tasks across 40 scenes and 10845 objects, covering three cooperation regimes: *spatial distribution*, *parallel constraint*, and *failure recovery*. These regimes cover a broad spectrum of cooperative requirements, ranging from distributed task allocation to synchronized execution and assisted failure recovery. CoEmbod further provides a process-level evaluation framework with **Budgeted Cooperative Success** (BCS), which jointly measures goal completion, cooperation fidelity, and execution efficiency. Experiments reveal that even the leading LMM powering a well-established agentic workflow achieves only 28.6% success, highlighting the difficulty of long-horizon multi-agent cooperation with diverse inter-agent dependencies in complex embodied environments.

1 Introduction

The core vision of Embodied AI is to build agents capable of perceiving, reasoning, and acting within complex physical or simulated environments (Liu et al., 2025; Duan et al., 2022). Under this vision, embodied planning studies how agents decompose high-level goals into executable action sequences grounded in environmental states, thereby completing long-horizon tasks through interaction with

the surrounding world (Song et al., 2023). Existing single-agent benchmarks have substantially advanced this direction by evaluating instruction following in interactive 3D scenes (Shridhar et al., 2020), everyday household activities grounded in object-centric states (Srivastava et al., 2022), and vision-driven LMM agents across diverse embodied capabilities (Yang et al., 2025). These efforts have established embodied planning as a central testbed for assessing long-horizon reasoning and decision-making in embodied agents.

However, many real-world embodied tasks extend beyond individual execution. They require multiple agents to operate over long horizons while allocating roles, coordinating actions, satisfying inter-agent dependencies, and recovering from execution failures. This need has motivated embodied multi-agent benchmarks, including large-scale human-robot collaboration across diverse households (Chang et al., 2025), LLM-driven cooperative agents with explicit communication (Zhang et al., 2024), and multi-agent manipulation tasks built on compositional constraints (Qin et al., 2025). These benchmarks represent important progress toward evaluating cooperation as a core capability of embodied agents.

Despite this progress, current evaluations still leave a gap in assessing long-horizon multi-agent cooperation in realistic 3D embodied simulation. Table 1 highlights two limitations. (*Problem 1: benchmark coverage*) Existing multi-agent benchmarks have explored important forms of collaboration, including game-like coordination, human-robot household tasks, and embodied manipulation. However, abstract environments provide limited pressure on realistic 3D grounding, such as room layouts, object accessibility, occlusion, navigation distance, object states, and spatial-semantic reasoning, while many realistic embodied benchmarks emphasize learned low-level policy execution or skill-level control rather than training-free high-

Benchmark	Real 3D	Train Free	Coop. Eval.	Scale			Core Capabilities			Cooperation	
				Scenes	Objects	Seq. Len.	Intent	Plan	Nav.	Distrib.	Constr.
Single-Agent											
ALFRED (Shridhar et al., 2020)	✓	✓	✗	120	112	~12	✗	✓	✗	✗	✗
ReALFRED (Kim et al., 2025)	✓	✗	✗	120	112	~12	✗	✓	✓	✗	✗
EB-ALFRED (Yang et al., 2025)	✓	✓	✓	120	112	~12	✓	✓	✗	✗	✗
ALFWorld (Shridhar et al., 2021)	✓	✗	✗	120	66	~6	✗	✓	✗	✗	✗
VirtualHome (Puig et al., 2018)	✓	✗	✗	7	N/R	~9	✗	✓	✗	✗	✗
Multi-Agent											
Collab-Overcooked (Sun et al., 2025)	✗	✓	✓	N/R	N/R	7-34	✗	✓	✗	✓	✗
VillagerBench (Dong et al., 2024)	✗	✓	✗	N/R	N/R	N/R	✗	✓	✓	✓	✓
PARTNR (Chang et al., 2025)	✓	✗	✗	60	N/R	4.7	✗	✓	✓	✓	✗
TDW-MAT (Zhang et al., 2024)	✓	✗	✗	6	N/R	N/R	✗	✓	✓	✓	✗
CoEmbod (Ours)	✓	✓	✓	40	10845	~25	✓	✓	✓	✓	✓

Table 1: Comparison of representative embodied AI benchmarks in terms of realistic 3D environments, training-free evaluation, cooperation-aware evaluation, scale, core capabilities, and cooperation requirements. **Train Free** denotes whether the benchmark supports evaluation without task-specific policy training. **Coop. Eval.** denotes cooperation-specific evaluation. **Seq. Len.** is the average number of high-level action steps per task. **Intent**, **Plan**, and **Nav.** refer to intent understanding, long-horizon planning, and embodied navigation. **Distrib.** denotes spatially distributed coordination, while **Constr.** denotes explicit cooperative constraints such as synchronized execution or failure recovery. **N/R** means not reported.

level planning. As a result, long-horizon tasks that jointly require spatial role allocation, temporal synchronization, and adaptive failure recovery remain insufficiently covered under a unified evaluation setting. (*Problem 2: evaluation protocol*) Existing protocols are largely *outcome-centric*, relying primarily on final success rate or task reward to determine whether the goal is achieved. Such metrics provide limited evidence about how agents cooperate during execution, leaving key behaviors such as synchronization, communication timing, role allocation, and failure recovery underdiagnosed.

To address *Problem 1*, we introduce **CoEmbod**, a long-horizon benchmark for multi-agent embodied cooperation in realistic 3D scenes. CoEmbod comprises **155** expert-curated tasks across **40 scenes** and **10845 objects**, organized into three *cooperation regimes*: **spatial distribution**, **parallel constraint**, and **failure recovery**. These regimes cover a broad spectrum of inter-agent dependencies, ranging from distributed task allocation to synchronized execution and assisted failure recovery. To address *Problem 2*, we provide a **process-level** evaluation framework that goes beyond final success. In addition to tracking intermediate process scores that reflect partial task completion, we propose **Budgeted Cooperative Success (BCS)**, which jointly captures **goal completion**, **cooperation fidelity**, and **execution budgets**. This enables fine-grained diagnosis of *how* agents cooper-

ate during execution, rather than merely *whether* they reach the final goal state.

Building on CoEmbod, we evaluate representative LMMs with ReAct-style embodied agents and compare them against a human baseline. The results reveal a substantial capability gap: even the strongest model, GPT-5.4, achieves only **28.6%** success, far below human performance. Process-level evaluation further shows distinct degrees of subgoal completion, cooperative requirement satisfaction, and execution efficiency, enabling finer-grained diagnosis beyond binary success. Error analysis attributes the failures to unstable task decomposition, weak long-horizon state tracking, mistimed synchronization and communication, and limited failure recovery.

Our contributions are summarized as follows:

- We introduce **CoEmbod**, a long-horizon benchmark for multi-agent embodied cooperation in realistic 3D scenes, covering three complementary cooperation regimes that span distributed task allocation, synchronized execution, and assisted failure recovery.
- We design a process-level evaluation protocol with intermediate process scores and **Budgeted Cooperative Success (BCS)**, jointly measuring goal completion, cooperation fidelity, and execution efficiency.
- We conduct extensive experiments on repre-

sentative LMMs under single-agent and multi-agent settings, revealing a large gap from human performance and identifying key bottlenecks in long-horizon embodied cooperation.

2 Related Work

2.1 Single-Agent Embodied Benchmarks

Embodied planning benchmarks evaluate agents that understand instructions, reason over long-horizon goals, and interact with 3D environments. Early benchmarks such as ALFRED (Shridhar et al., 2020), ALFWorld (Shridhar et al., 2021), and VirtualHome (Puig et al., 2018) established household instruction following and task planning as standard testbeds, while BEHAVIOR (Srivastava et al., 2022) further expanded embodied tasks with richer object states and everyday activities. Recent efforts improve realism and evaluation breadth: Re-ALFRED (Kim et al., 2025) extends ALFRED to photo-realistic 3D captured scenes, and Embodied-Bench (Yang et al., 2025) provides a comprehensive evaluation suite for vision-driven multimodal embodied agents. However, these benchmarks primarily focus on single-agent execution, leaving real-world tasks that require multiple agents to divide labor, coordinate actions, and satisfy inter-agent dependencies underexplored.

2.2 Multi-Agent Benchmarks

Multi-agent benchmarks have increasingly moved beyond individual execution toward collaborative task solving. General multi-agent benchmarks such as Collab-Overcooked (Sun et al., 2025) and VillagerBench (Dong et al., 2024) evaluate collaboration in structured game-like environments, covering capabilities such as collaboration initiation, workload distribution, dynamic adaptation, and synchronized execution. In embodied settings, prior benchmarks often emphasize trained policies or low-level action execution: RoboFactory (Qin et al., 2025) studies multi-agent manipulation with compositional constraints, while VIKI-Bench (Kang et al., 2026) evaluates visually grounded cooperation with agent activation, task planning, and trajectory perception. Other works such as Watch-And-Help (Puig et al., 2021), CoELA (Zhang et al., 2024), and PARTNR (Chang et al., 2025) further explore human-AI or human-robot collaboration in realistic household environments. However, existing benchmarks either rely on abstract environments, focus on learned pol-

icy execution, or cover limited forms of high-level cooperation, leaving training-free long-horizon embodied planning with diverse inter-agent dependencies and process-level cooperation diagnosis insufficiently explored.

3 The CoEmbod Benchmark

3.1 Overview

CoEmbod is a long-horizon multi-agent embodied benchmark built on OmniGibson (Srivastava et al., 2022), comprising 155 carefully curated tasks across 40 scenes and 10,845 object instances. Each task is grounded in a realistic 3D scene and is designed around one of three inter-agent dependencies. Spatial-distribution tasks provide a soft cooperative advantage: a single agent may complete some final goals sequentially, whereas two agents can divide distant regions, shorten local execution horizons, and reduce search burden. Parallel-constraint and failure-recovery tasks impose hard temporal or causal dependencies that require coordinated two-agent execution. As illustrated in Figure 1, CoEmbod organizes tasks into three cooperation regimes—*spatial distribution*, *parallel constraint*, and *failure recovery*. These regimes cover a broad spectrum of inter-agent dependencies: spatial partitioning across rooms, temporally synchronized joint actions, and event-triggered peer-assisted recovery required in everyday embodied tasks and serve as the foundation for the construction pipeline and evaluation protocol.

3.2 Cooperation Regimes

CoEmbod organizes tasks into three cooperation regimes that instantiate complementary forms of inter-agent dependency, covering spatial division, temporal synchronization, and assisted recovery.

Spatial Distribution Tasks. Spatial distribution tasks require agents to divide a realistic 3D scene into spatially separated regions and execute sub-goals in parallel. As shown in the top-left of Figure 1, one agent retrieves a cup from the kitchen while the other retrieves a book from the bedroom, and both objects must eventually be placed on the shared goal location. The key dependency is not a hard physical constraint, but the need to coordinate region assignment, parallel retrieval, status synchronization, and goal convergence across distant rooms. These tasks therefore evaluate whether agents can allocate roles efficiently, maintain a

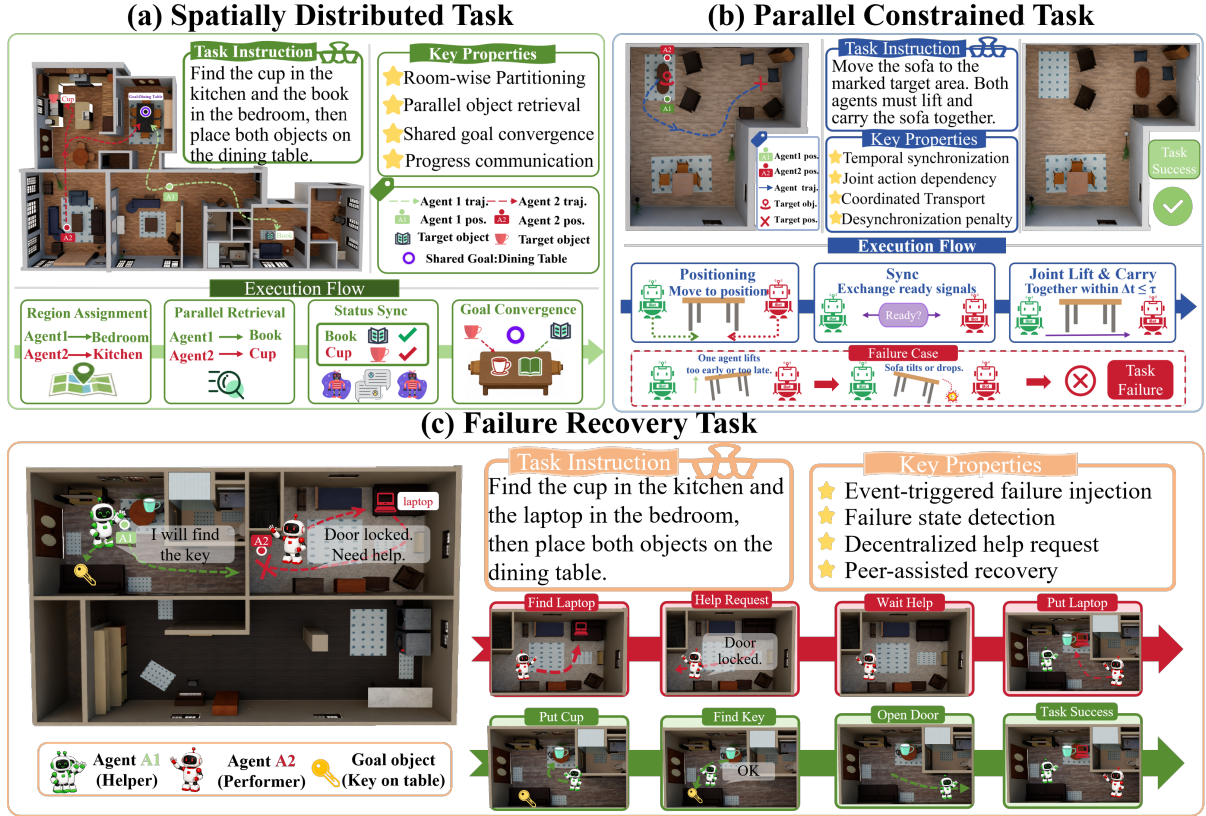


Figure 1: Overview of the three cooperation regimes in CoEmbod. **Top-left:** a *spatial distribution* task where two agents partition the scene to retrieve different target objects in parallel and converge on a shared goal. **Top-right:** a *parallel constraint* task where two agents must temporally synchronize their actions to jointly transport a sofa; desynchronization triggers task failure. **Bottom:** a *failure recovery* task where the performer agent is blocked by a locked door, requests help, and the helper agent retrieves a key to enable peer-assisted recovery.

shared task state, and coordinate progress under distributed execution.

Parallel Constraint Tasks. Parallel constraint tasks introduce explicit temporal coupling between agents, requiring critical actions to be completed within a bounded synchronization window. In the top-right example of Figure 1, two agents must move to the target area, exchange readiness signals, and jointly lift and carry a sofa. If one agent acts too early or too late, the sofa cannot be transported correctly and the task fails. These tasks directly test temporal synchronization, joint-action dependency, and coordinated object transport, where progress depends on both agents acting as a coupled team rather than as independent executors.

Failure Recovery Tasks. Failure recovery tasks inject execution-time failures that require agents to detect unexpected state changes and reorganize their cooperation strategy. The bottom example in Figure 1 shows a performer agent blocked by a locked door while fetching a laptop. The per-

former must request help, and the helper must locate the key, open the door, and enable the task to continue. This regime evaluates event-triggered cooperation, including failure state detection, decentralized help requests, and peer-assisted recovery. It complements the first two regimes by testing whether agents can adapt their roles when the original execution path becomes infeasible.

3.3 Benchmark Construction

To balance task quality, diversity, and scale, CoEmbod is constructed through a three-stage pipeline that combines human expertise with multi-model generation, as illustrated in Figure 2.

Stage 1: Example Construction. Domain experts manually author **20 seed examples** for each cooperation regime, yielding **60 seeds** in total. Each seed is grounded in a concrete OmniGibson scene and specifies the task goal, initial agent and object configuration, and scoring conditions for intermediate cooperative milestones. These seeds serve as in-context exemplars that define the ex-

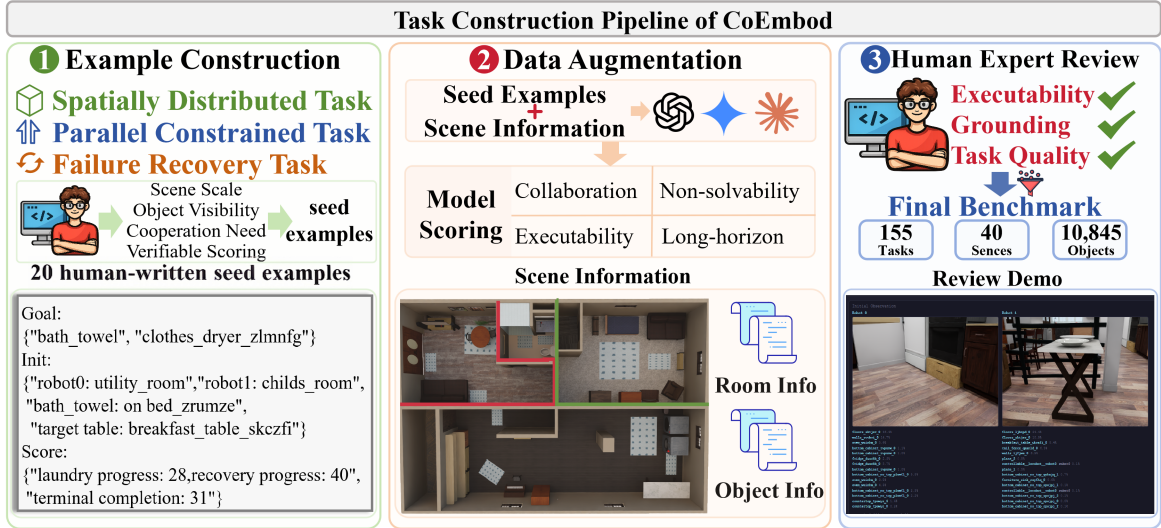


Figure 2: Overview of the CoEmbod construction pipeline. **Stage 1 (Example Construction)**: domain experts manually write 20 high-quality seed examples for each of the three cooperation regimes, each specifying the task goal, initial configuration, and structured scoring conditions. **Stage 2 (Data Augmentation)**: seed examples and extracted scene information (room and object info) are fed to three large models (GPT-5.4, Gemini-3.1-Pro, and Claude-Opus-4.6), which generate and score candidate tasks along collaboration, non-solvability, executability, and long-horizon dimensions. **Stage 3 (Human Expert Review)**: experts filter and refine the candidates by executability, grounding, and task quality, producing the final benchmark of 155 tasks, 40 scenes, and 10,845 objects.

pected cooperation patterns and structured output format.

Stage 2: Data Augmentation. We scale up the benchmark by prompting large models with the seed examples and scene information extracted from OmniGibson, including room layouts and grounded object instances. To improve diversity and reduce single-model bias, we query **GPT-5.4** (OpenAI, 2026), **Gemini-3.1-Pro** (Google DeepMind, 2026), and **Claude-Opus-4.6** (Anthropic, 2026) in turn, producing **594 candidate tasks**. Each candidate is scored along four dimensions: *collaboration*, *non-solvability*, *executability*, and *long-horizon* structure, providing an initial quality signal for human review.

Stage 3: Human Expert Review. Human experts then filter and refine all candidates according to **executability**, environmental **grounding**, and overall **task quality**. They verify room and object references, remove tasks with weak or bypassable cooperation requirements, correct hallucinated entities, complete scoring conditions, and discard tasks whose success or cooperation states cannot be reliably verified. This process yields a final benchmark of **155 tasks** across **40 scenes** and **10,845 objects**, where each task is grounded, executable, and designed to require multi-agent cooperation. Additional scene statistics, filtering details, and scoring

checkpoint examples are provided in Appendix A.

3.4 Benchmark Analysis

We analyze CoEmbod from the perspectives of task composition, sub-task structure, and cooperation difficulty, as summarized in Figure 3. The benchmark contains **155 tasks** and **558 sub-tasks** in total, with an average of **3.60 sub-tasks per task**, reflecting the long-horizon nature of the tasks. The three cooperation regimes are kept roughly balanced in scale, so that no single regime dominates the overall evaluation. The average sub-task count further varies across regimes: *spatial distribution* tasks are the most compact (2.58), as their cooperation is realized mainly through parallel spatial coverage rather than sequential stages; *parallel constraint* tasks have the most sub-tasks (4.20), since they must explicitly separate positioning, synchronization, joint execution, and finalization to verify temporal cooperation; and *failure recovery* tasks lie in between (3.90), with conditional sub-tasks such as failure detection and help request that are triggered only when execution is blocked.

Beyond task composition, CoEmbod also spans two levels of cooperation difficulty. As shown on the right of Figure 3, each cooperation regime contains both *simple-cooperation* tasks, which require only coarse role assignment or short cooperation chains, and *complex-cooperation* tasks, which in-

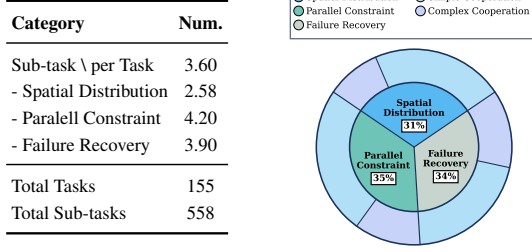


Figure 3: Statistics of CoEmbod. **Left:** the average number of sub-tasks per task, both overall and within each cooperation regime, together with the total number of tasks and sub-tasks. **Right:** the distribution of tasks across the three cooperation regimes (spatial distribution, parallel constraint, and failure recovery), where each regime is further divided into simple-cooperation and complex-cooperation tasks.

volve tighter temporal synchronization, longer state maintenance, or multi-step recovery. This two-level design ensures that the benchmark remains discriminative across a range of cooperation intensities rather than collapsing to a single difficulty level. Together, these statistics show that CoEmbod provides a balanced, long-horizon, and difficulty-diverse testbed for evaluating multi-agent embodied cooperation.

4 Evaluation Framework and Metrics

4.1 Single-Agent and Multi-Agent ReAct Frameworks

Single-Agent Framework. As a baseline, we adopt a standard ReAct framework (Yao et al., 2022) in which a single agent completes the entire task on its own. At each step t , the agent takes the instruction I , its current observation o_t , and history h_t , and a multimodal model generates a reasoning trace r_t and an action a_t :

$$r_t, a_t = \text{VLM}(I, o_t, h_t) \quad (1)$$

This baseline contains no task decomposition, communication, or synchronization, and thus quantifies how strongly each task depends on multi-agent cooperation. A single agent may solve some spatial distribution tasks by traversing regions sequentially, but fails to satisfy the cooperation conditions of parallel constraint and failure recovery tasks.

Multi-Agent Framework. For the cooperative setting, we adopt a centralized multi-agent ReAct framework with one central agent and two executor agents. The central agent performs global task understanding, decomposition, sub-task assignment,

and dynamic replanning, while the executors interact with the environment in parallel and communicate in an event-triggered manner. At the start of an episode, the central agent maps the instruction I and the initial observations o_0^1, o_0^2 to a global plan and two sub-tasks:

$$P_0, \tau_0^1, \tau_0^2 = \text{Central}(I, o_0^1, o_0^2). \quad (2)$$

Each executor e_i then follows the ReAct paradigm to produce reasoning and actions from its sub-task, observation, and history. The two actions jointly update the environment:

$$r_t^i, a_t^i = \text{Executor}_i(\tau_t^i, o_t^i, h_t^i) \quad (3)$$

Instead of communicating every step, an executor sends a message only when a triggering condition $\phi(\cdot)$ holds, such as completing a sub-task, failing to locate an object, or requiring synchronization:

$$m_t^i = \begin{cases} \text{Msg}_i(o_t^i, h_t^i, \tau_t^i), & \text{if } \phi_t^i = 1, \\ \emptyset, & \text{otherwise.} \end{cases} \quad (4)$$

Here $\phi_t^i = \phi(o_t^i, h_t^i, \tau_t^i)$ denotes the triggering condition. Upon receiving messages, the central agent replans to enable dynamic role adjustment:

$$P_t, \tau_t^1, \tau_t^2 = \text{Central}(I, P_{t-1}, m_t^1, m_t^2), \quad (5)$$

This design couples centralized coordination with decentralized execution, supporting analysis of task decomposition, parallel execution, communication, and failure recovery. The complete executor action space is provided in Appendix Table 13.

4.2 Evaluation Metrics

We evaluate models along four metrics covering both task outcome and cooperation quality.

Success Rate (SR). SR is the percentage of tasks whose terminal goal conditions are all satisfied within the step budget. It reflects whether a system can complete the task.

Average Step (Step). Step is the mean number of environment steps over all episodes. Lower values indicate more efficient execution under the same budget, while redundant exploration or repeated failed attempts increase the count.

Process Score (Score). Since SR only reflects final completion, we additionally report a process score based on task-specific scoring checkpoints. For each task, we define a set of checkpoints $\mathcal{C}_e$,

Table 2: Comprehensive evaluation of four representative LMMs under the single-agent and multi-agent frameworks on CoEmbod, together with the human baseline. Results are reported for the three cooperation regimes (Spat.: spatial distribution, Par.: parallel constraint, Fail.: failure recovery) under four metrics: success rate (SR, %), process score (Score), average step (Step), and budgeted cooperative success (BCS). The best result in each column under the multi-agent framework is in bold.

Base LMM	SR (%) $\uparrow$				Score $\uparrow$				Step $\downarrow$				BCS $\uparrow$			
	Spat.	Par.	Fail.	Avg.	Spat.	Par.	Fail.	Avg.	Spat.	Par.	Fail.	Avg.	Spat.	Par.	Fail.	Avg.
<i>Single-Agent Framework</i>																
GPT-5.4	2.3	0.0	0.0	0.7	25.0	0.0	3.4	8.9	69.2	70.0	70.0	69.8	–	–	–	–
Qwen3.5-Plus	0.0	0.0	0.0	0.0	13.9	0.0	3.2	5.4	70.0	70.0	70.0	70.0	–	–	–	–
Kimi-K2.6	4.5	0.0	0.0	1.4	21.5	0.0	3.08	7.7	69.2	70.0	70.0	69.8	–	–	–	–
Qwen3.5-397B-A17B	4.5	0.0	0.0	1.4	13.3	0.0	6.5	6.3	68.1	70.0	70.0	69.4	–	–	–	–
<i>Multi-Agent Framework</i>																
GPT-5.4	11.4	59.3	12.0	28.6	41.4	63.8	40.9	49.2	59.3	18.7	63.3	46.3	0.32	0.59	0.25	0.39
Qwen3.5-Plus	12.5	50.0	11.5	25.5	46.9	69.2	29.1	48.8	64.8	28.6	66.2	52.4	0.32	0.52	0.15	0.34
Kimi-K2.6	4.5	43.6	2.0	17.5	32.5	70.6	17.7	41.1	65.9	30.7	69.6	54.7	0.23	0.49	0.11	0.28
Qwen3.5-397B-A17B	4.5	56.6	0.0	21.5	25.7	69.5	11.8	36.6	67.6	21.5	66.8	51.0	0.20	0.50	0.11	0.28
<i>Human Baseline</i>																
Human	95.2	100.0	91.7	95.7	97.3	100.0	94.8	97.4	23.32	12.0	32.4	22.3	0.89	0.93	0.82	0.88

where each checkpoint c has a weight w_c and $\sum_{c \in \mathcal{C}_e} w_c = 100$. The score is computed as

$$\text{Score}(e) = \sum_{c \in \mathcal{C}_e} w_c \cdot \mathbb{I}[c \text{ completed}] \quad (6)$$

This metric reflects how many task checkpoints are completed during execution, providing a denser measure of progress than final success alone.

Budgeted Cooperative Success (BCS). SR and Score do not verify whether the explicit cooperation conditions are met under a bounded budget. We therefore define BCS, which jointly accounts for goal completion $G(e)$, cooperation fidelity $K(e)$, and execution cost:

$$\text{BCS}(e) = G(e) \cdot K(e) \cdot \max\left(0, 1 - \lambda \frac{T(e)}{T_{\max}(e)}\right), \quad (7)$$

Here, $G(e)$ is the fraction of satisfied final-goal predicates, while $T(e)$ and $T_{\max}(e)$ denote the actual and maximum numbers of environment steps. We use $\lambda = 0.3$ in the main experiments. The cooperation term is defined according to the task regime: $K(e) = 1$ for spatial distribution, $K(e)$ is the fraction of satisfied joint checkpoints for parallel constraint, and $K(e)$ is the fraction of triggered failures successfully recovered for failure recovery. Checkpoint weights affect only the Process Score and do not enter BCS.

BCS lies in $[0, 1]$ and multiplicatively combines goal completion, cooperation fidelity, and execution efficiency. All reported Avg. BCS values are task-level averages over the 155 tasks, rather than unweighted averages over the three regime-level means. Detailed computation rules and sensitivity results are provided in Appendices A.3 and C.3.

5 Experiments

5.1 Experimental Setup

We evaluate two agent frameworks, the single-agent ReAct (Yao et al., 2022) and our centralized multi-agent ReAct framework on the full CoEmbod benchmark. Under each framework, we benchmark four representative large multimodal models, including two closed-source models, GPT-5.4 (OpenAI, 2026) and Qwen3.5-Plus (Qwen Team, 2026), and two open-source models, Kimi-K2.6 (Moonshot AI, 2026) and Qwen3.5-397B-A17B (Qwen Team, 2026). To establish a reference upper bound, we additionally collect a human baseline in which three master-level human participants complete the same tasks under identical settings. For multi-agent model and human trajectories, we report all four metrics introduced in Section 4.2: success rate (SR), average step (Step), process score (Score), and budgeted cooperative success (BCS), which together capture task completion, execution efficiency, partial progress, and cooperation quality. For the single-agent baseline, BCS is not applica-

Setting	Spatial SR / BCS	Parallel SR / BCS	Recovery SR / BCS
GPT-5.4			
Full	11.40 / 0.32	59.30 / 0.59	12.00 / 0.25
No comm./replan	10.42 / 0.30	1.88 / 0.16	0.00 / 0.00
Decentralized	4.76 / 0.15	7.14 / 0.12	9.52 / 0.22
Qwen3.5-Plus			
Full	12.50 / 0.32	50.00 / 0.52	11.50 / 0.15
No comm./replan	8.70 / 0.29	1.89 / 0.24	0.00 / 0.01
Decentralized	2.38 / 0.25	2.38 / 0.01	7.14 / 0.15

Table 3: Controlled two-executor ablations. Each entry reports SR (%) / BCS. No comm./replan also removes dynamic replanning; Decentralized removes the central coordinator while retaining the shared task board.

ble because the cooperative execution protocol is absent; we therefore report SR, Score, and Step only.

5.2 Main Results

Table 2 shows that single-agent execution is largely insufficient for CoEmbod. Across all evaluated LMMs, the single-agent framework achieves near-zero average success, with the best Avg. SR reaching only 1.4%. Its limited progress mainly appears in spatial distribution tasks, while parallel constraint and failure recovery tasks are almost entirely unsolved, reflecting their dependence on coordinated multi-agent behavior. Although single-agent systems obtain non-zero process scores in some tasks, their steps remain close to the episode budget, indicating inefficient exploration and limited task completion.

The multi-agent framework substantially improves performance, especially on parallel-constraint tasks, where GPT-5.4 reaches 59.3% SR and Qwen3.5-397B-A17B reaches 56.6% SR. However, the gap to humans remains large: the strongest evaluated model achieves only 28.6% task-weighted Avg. SR and 49.2 Avg. Score, far below the reference human baseline of 95.7% SR and 97.4 Score. GPT-5.4 also obtains the highest model Avg. BCS of 0.393, compared with 0.88 for humans, indicating that current LMM agents still struggle with robust coordination, state tracking, and efficient long-horizon cooperation.

5.3 Controlled Ablations.

Removing online communication and dynamic replanning preserves much of the spatial-distribution BCS, but nearly eliminates terminal success on parallel-constraint and failure-recovery tasks. The

remaining non-zero parallel BCS reflects partial goal progress rather than successful synchronization. Decentralized execution sharply reduces parallel-constraint BCS and lowers SR across all three regimes, although its failure-recovery BCS remains relatively close to the full framework, indicating that the shared task board can preserve partial recovery progress without reliably reaching terminal success. Since each ablation removes a bundle of coupled components, these results are interpreted at the component-bundle level rather than as single-factor causal effects. Complete metrics and robustness analyses are reported in Appendices C.2 and C.3.

5.4 Error Analysis

We further analyze failed trajectories by categorizing the dominant failure source into four error types: *task decomposition*, *spatial grounding*, *state tracking*, and *coordination/recovery*. Task decomposition errors occur when the agent fails to assign appropriate subtasks or execution order; spatial grounding errors refer to failures in locating rooms, objects, or target regions; state tracking errors indicate incorrect memory of object states, agent positions, or completed subtasks; and coordination/recovery errors include mistimed joint actions, ineffective communication, and unsuccessful recovery from injected failures. Detailed definitions of these error types are provided in Appendix C.1.

Figure 4 shows that *coordination/recovery* dominates failures in parallel constraint and failure recovery tasks, reaching about half or more of the errors for most models. This suggests that current agents struggle most with synchronized execution, help requests, and recovery behaviors. In spatial distribution tasks, the error pattern is more mixed: *spatial grounding* becomes prominent for Kimi-K2.6 and Qwen3.5-397B-A17B, while GPT-5.4 still shows many coordination/recovery errors. Overall, failures arise from a combination of planning, grounding, memory, and coordination issues, with the dominant bottleneck varying across cooperation regimes.

6 Conclusion

We present **CoEmbod**, a realistic 3D simulated benchmark for long-horizon multi-agent embodied cooperation. CoEmbod contains 155 expert-curated tasks across three cooperation regimes, grounding diverse inter-agent dependencies in re-

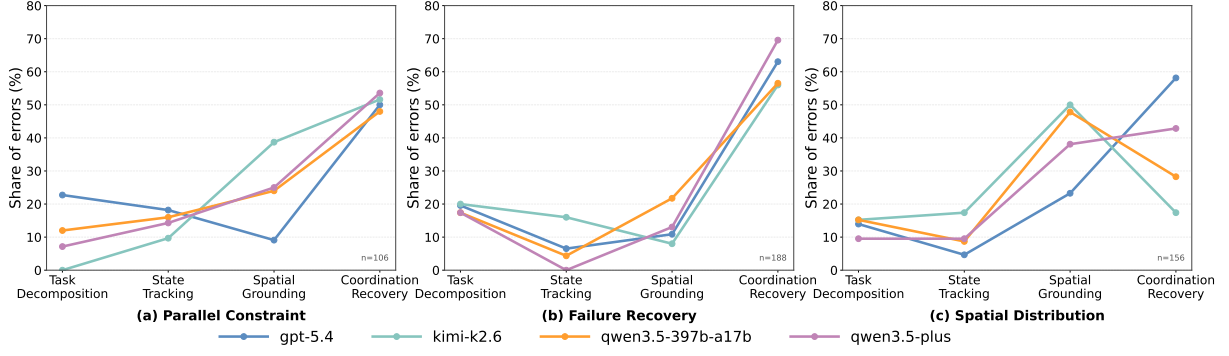


Figure 4: Error distribution across cooperation regimes. We report the share of four dominant error types, including task decomposition, spatial grounding, state tracking, and coordination/recovery, for each evaluated model under parallel constraint, failure recovery, and spatial distribution tasks.

alistic household scenes. Beyond final success, we introduce a process-level evaluation protocol with intermediate process scores and **Budgeted Cooperative Success** (BCS), enabling analysis of task progress, cooperation fidelity, and execution efficiency. Experiments show that current LMM agents remain far from human performance: single-agent execution almost completely fails on cooperation-dependent tasks, while multi-agent prompting improves performance but still struggles with coordination, state tracking, synchronization, and recovery. These findings suggest that long-horizon embodied cooperation remains challenging, and CoEmbod provides a structured testbed for diagnosing future embodied multimodal agents.

Limitations

CoEmbod still has several limitations. Although our independent re-annotation study shows high agreement, the manually specified task goals, scoring checkpoints, and cooperation conditions may still introduce some benchmark-dependent subjectivity. In addition, CoEmbod is evaluated in simulation with a controlled high-level action interface. This design allows us to focus on cooperative planning, but it abstracts away low-level navigation, manipulation, physical control, and sim-to-real challenges. Our experiments also mainly evaluate training-free ReAct-style LMM agents. More specialized systems with learned communication, persistent memory, reinforcement learning, or embodiment-specific policies may achieve different performance after benchmark-specific adaptation. Finally, our controlled ablations remove multiple coupled components at the same time. They therefore help distinguish the effects of parallel execution and coordination, but should not

be interpreted as single-factor causal estimates of individual components.

Acknowledgments

We gratefully acknowledge the support of the National Natural Science Foundation of China (NSFC) via grant 62236004 and 62476073.

References

- Anthropic. 2026. Introducing claude opus 4.6. <https://www.anthropic.com/news/claude-opus-4-6>.
- Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Motlaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M Turner, Eric Underlander, and Tsung-Yen Yang. 2025. **PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks**. In *The Thirteenth International Conference on Learning Representations*.
- Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. **VillagerAgent: A graph-based multi-agent framework for coordinating complex task dependencies in Minecraft**. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16290–16314, Bangkok, Thailand. Association for Computational Linguistics.
- Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. 2022. **A survey of embodied ai: From simulators to research tasks**. *Preprint*, arXiv:2103.04918.
- Google DeepMind. 2026. **Gemini 3.1 Pro Model Card**. Released February 19, 2026.
- Li Kang, Xiufeng Song, Heng Zhou, Yiran Qin, Jie Yang, Xiaohong Liu, Philip Torr, LEI BAI, and Zhenfei Yin. 2026. **VIKI-r: Coordinating embodied multi-agent cooperation via reinforcement learning**. In *The*

- Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track.*
- TaeWoong Kim, Cheolhong Min, Byeonghwi Kim, Jinyeon Kim, Wonje Jeung, and Jonghyun Choi. 2025. Realfred: An embodied instruction following benchmark in photo-realistic environments. In *Computer Vision – ECCV 2024*, pages 346–364, Cham. Springer Nature Switzerland.
- Yang Liu, Weixing Chen, Yongjie Bai, Xiaodan Liang, Guanbin Li, Wen Gao, and Liang Lin. 2025. [Aligning cyber space with physical world: A comprehensive survey on embodied ai](#). *Preprint*, arXiv:2407.06886.
- Moonshot AI. 2026. Kimi K2.6: Advancing open-source coding. <https://www.kimi.com/blog/kimi-k2-6>. Official technical blog.
- OpenAI. 2026. Introducing GPT-5.4. <https://openai.com/index/introducing-gpt-5-4/>. Accessed 31 August 2026.
- Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B. Tenenbaum, Sanja Fidler, and Antonio Torralba. 2021. [Watch-and-help: A challenge for social perception and human-ai collaboration](#). In *International Conference on Learning Representations*.
- Yiran Qin, Li Kang, Xiufeng Song, Zhenfei Yin, Xiaohong Liu, Xihui Liu, Ruimao Zhang, and Lei Bai. 2025. Robofactory: Exploring embodied agent collaboration with compositional constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10075–10085.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. [{ALFW}orld: Aligning text and embodied environments for interactive learning](#). In *International Conference on Learning Representations*.
- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. [Llm-planner: Few-shot grounded planning for embodied agents with large language models](#). *Preprint*, arXiv:2212.04088.
- Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. 2022. [Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments](#). In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 477–490. PMLR.
- 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.
- Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. 2025. [Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents](#). In *Forty-second International Conference on Machine Learning*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024. [Building cooperative embodied agents modularly with large language models](#). In *The Twelfth International Conference on Learning Representations*.

A Benchmark Details

A.1 Scene and Object Statistics

CoEmbod is constructed from a curated subset of OmniGibson scenes. The current benchmark contains 155 tasks across 40 unique scenes, covering 250 object categories and 10,845 object instances. Here, an object instance refers to a concrete object instantiated in the simulator state of the selected scenes. This count includes both interactable objects, which can be directly manipulated or whose states can be changed by agents, and non-interactable objects, which serve as spatial context, obstacles, containers, landmarks, or background scene structure. We count each object instance

Cooperation Regime	Tasks	Simple	Complex	Objects
Failure Recovery	52	32	20	3,689
Parallel Constraint	55	38	17	8,652
Spatial Distribution	48	34	14	4,893
Unique Total	155	104	51	10,845

Table 4: Task statistics by cooperation regime. Object counts at the regime level are computed over the scenes used by each regime and can overlap across regimes; therefore, they should not be summed. The benchmark-level total contains 155 tasks, 40 unique scenes, 250 object categories, and 10,845 unique object instances.

once at the scene level, regardless of how many tasks refer to the same scene.

Table 6 reports the distribution of tasks over different scene categories. Residential scenes account for the largest portion of tasks, while the benchmark also covers retail, hospitality, dining, workplace, and public education environments. Since the same scene can be reused across different cooperation regimes, the scene counts in this table indicate regime-level scene usage rather than unique scene totals and therefore should not be summed to recover the benchmark-level number of unique scenes.

Table 4 reports the distribution of tasks across the three cooperation regimes. The benchmark is roughly balanced across failure recovery, parallel constraint, and spatial distribution tasks, while also separating simple and complex cooperation cases within each regime. Table 5 further summarizes family-level object statistics. Since the same scene can support tasks from multiple cooperation regimes, family-level object counts are not mutually exclusive and should not be summed to obtain the benchmark-level total. For example, a kitchen scene may support both a spatial-distribution task that requires distributed object retrieval and a failure-recovery task that requires one agent to unlock access for another agent. Therefore, the benchmark-level count of 10,845 refers to unique object instances across the 40 selected scenes, whereas the family-level counts reflect the objects available within the scenes used by each regime.

We further annotate each task with a coarse cooperation difficulty label. Simple tasks involve relatively direct cooperation patterns, such as assigning two agents to different regions or completing a short cooperative chain. Complex tasks require more demanding long-horizon coordination, such

Cooperation Regime	Objects	Inter.	Non-inter.
Failure Recovery	3,689	1,737	1,952
Parallel Constraint	8,652	4,337	4,315
Spatial Distribution	4,893	2,323	2,570

Table 5: Family-level object statistics. **Objects** denotes total object instances within the scenes used by each cooperation regime; **Inter.** and **Non-inter.** denote interactable and non-interactable instances, respectively. Since scenes can be reused across regimes, these counts overlap and are not additive.

Scene Category	Scene Uses	Tasks
Residential	39	90
Retail	7	7
Hospitality	7	14
Dining	8	14
Workplace	10	20
Public/Edu.	6	10
Total	77	155

Table 6: Scene-category distribution in CoEmbod. **Scene Uses** denotes regime-level scene usage, where the same unique scene can appear in multiple cooperation regimes. Therefore, the total number of scene uses is not equal to the benchmark-level count of 40 unique scenes.

as searching over larger scenes, maintaining state over more rooms, manipulating multiple target objects, or satisfying tighter synchronization and recovery constraints. The label is assigned based on both scene-level and task-level factors, including the number and size of rooms, the number of target objects, the specificity of object-location hints in the instruction, and the perceptual salience of target objects. For instance, a task that asks agents to find a large teddy bear in a specified room is usually easier than a task that requires locating a small key without an explicit room hint.

Table 6 gives a preliminary summary of the scene families used by each task family. The key point is that CoEmbod is not limited to a single household layout: it covers multiple realistic indoor scene families and reuses scenes across different cooperation regimes when the same environment can instantiate different inter-agent dependencies.

A.2 Task Construction and Filtering

GPT-5.4, Gemini-3.1-Pro, and Claude-Opus-4.6 contributed approximately equal shares of 594 candidate tasks. Three master-level annotators with experience in embodied AI or agent evaluation then

reviewed and refined the candidate pool. The review examined environmental grounding, execution feasibility, regime-specific cooperation requirements, long-horizon structure, and the observability of scoring conditions.

For spatial-distribution tasks, annotators required a meaningful cooperative advantage through spatial separation, parallel search, or shorter per-agent execution chains; these tasks were not required to be strictly impossible for a single agent to complete sequentially. For parallel-constraint and failure-recovery tasks, the required temporal or causal dependency had to require coordinated two-agent execution.

After filtering and refinement, 155 tasks were retained, corresponding to an acceptance rate of 26.1%. The principal rejection categories are reported in Table 7. When disagreements arose during task refinement, checkpoint verification, or difficulty labeling, the annotators discussed the case until reaching consensus. Candidates that remained ambiguous, inconsistent, or difficult to verify were discarded.

Annotation Agreement. To evaluate the annotation components that directly affect the reported results, two master-level annotators independently re-annotated stratified subsets covering all three cooperation regimes and both simple and complex settings. Agreement was measured before consensus discussion. The first study evaluates whether checkpoints and cooperation conditions are valid, observable, and consistently applicable; the second evaluates the primary error assigned to failed trajectories.

The high agreement on checkpoint and cooperation-condition validity indicates that these definitions can be applied consistently across annotators. Error categorization is more interpretive because a failed trajectory may contain multiple interacting failure sources, but the annotators still achieve substantial agreement under the primary-error taxonomy.

A.3 Scoring Checkpoints and Cooperation Conditions

For each task, we define task-specific scoring checkpoints that decompose the full objective into verifiable intermediate states. The checkpoint weights are normalized so that every task has a maximum process score of 100. In addition to these progress checkpoints, we define cooperation

conditions for BCS to verify whether the task is completed through the intended cooperative behavior rather than through inefficient or unintended execution.

Regime-Specific BCS Computation. Checkpoint weights are normalized to 100 within each task and affect only the Process Score; they do not enter BCS. The cooperation term is defined deterministically for each regime:

$$K(e) = \begin{cases} 1, & \text{spatial distribution,} \\ \frac{N_{\text{joint}}^{\text{sat}}(e)}{N_{\text{joint}}(e)}, & \text{parallel constraint,} \\ \frac{N_{\text{rec}}(e)}{N_{\text{fail}}(e)}, & \text{failure recovery.} \end{cases} \quad (8)$$

For spatial-distribution tasks, the intended cooperative structure is encoded by the distributed goals and the execution-efficiency term, rather than by an additional event-level condition. Spatial BCS should therefore be interpreted as budgeted goal completion; division of labor is analyzed separately through trajectories, execution efficiency, and the component ablations.

For parallel-constraint tasks, $N_{\text{joint}}^{\text{sat}}(e)$ is the number of joint checkpoints completed within the synchronization window, and $N_{\text{joint}}(e)$ is the total number of required joint checkpoints. The synchronization window is **five environment steps** in the current benchmark.

For failure-recovery tasks, $N_{\text{fail}}(e)$ is the number of designated failures triggered during execution, while $N_{\text{rec}}(e)$ is the number successfully resolved through assistance from the other agent. We use $\lambda = 0.3$ for the execution-efficiency term in the main experiments. Table 9 gives representative examples for the three cooperation regimes.

B Agent and Evaluation Details

B.1 Action Space and Observation Interface

Each executor interacts with the environment through a discrete high-level action interface. At each step, the executor receives a first-person RGB observation from the robot camera in OmniGibson, together with its current status, visible objects, holding state, action history, and the assigned sub-task. The executor then outputs exactly one action, which is executed by the simulator before the next observation is returned. We set the episode budget to $T_{\text{max}} = 70$ high-level environment steps; an

Rejection Category		Review Criterion	Representative Example
Hallucinated Scene Element		The task refers to a room, object, or spatial relation absent from the selected OmniGibson scene.	The task requires retrieving a key from a garage, but the scene contains neither a garage nor a valid key instance.
Infeasible Execution		The task cannot be completed reliably under the simulator state, object accessibility, or supported action interface.	The target object is unreachable or cannot be manipulated using the available high-level actions.
Weak or Bypassable Cooperation		The task lacks a meaningful cooperative advantage in spatial distribution or the required temporal or causal dependency in the other two regimes.	The agents retrieve nearby independent objects without meaningful spatial division, synchronization, or recovery dependence.
Unverifiable Scoring Condition		The success state, intermediate checkpoint, or cooperation condition cannot be detected reliably.	The task requires the agents to “cooperate smoothly” without an observable event or simulator-state predicate.
Insufficient Structure	Long-Horizon	The task is too short or lacks meaningful intermediate dependencies.	The task can be completed in one or two high-level actions without sustained planning or state tracking.

Table 7: Principal rejection categories used during expert filtering of the model-generated candidate tasks.

Annotation Item	Evaluation Scope	Raw Agreement	Cohen’s κ
Checkpoint and Cooperation-Condition Validity	60 tasks covering 210 conditions	96.7%	0.93
Error Categorization	100 failed trajectories	88.0%	0.79

Table 8: Inter-annotator agreement measured before consensus discussion.

episode terminates when the task succeeds, fails, or reaches this step budget.

B.2 Single-Agent and Multi-Agent ReAct Prompts

We use ReAct-style prompting for both the single-agent and multi-agent settings. In the single-agent setting, the agent directly receives the task goal, first-person observation, current robot state, visible objects, holding state, and recent action history, and then outputs one executable action at each step. The full system prompt for the single-agent baseline is shown in Figure 5.

In the multi-agent setting, we decompose the workflow into a central coordinator and two executor agents. The coordinator receives the global task goal, the current progress of both executors, their communication messages, and recent execution results, and is responsible for task decomposition, role assignment, synchronization planning, and replanning. Its system prompt is shown in Figure 6. Each executor receives its assigned sub-task, first-person observation, robot state, visible objects, holding state, and local action history, and outputs exactly one primitive action or a communication message at each step. The executor prompt is shown in Figure 7. Together, these prompts define a training-free multi-agent ReAct workflow

in which planning, communication, and embodied execution are evaluated under the same action and observation interface as the single-agent baseline.

B.3 Model Configuration

All evaluated LMMs are accessed through the same unified API interface, with only the model name changed across runs. We use identical prompting templates, action formats, decoding settings, and retry rules for all models to ensure a controlled comparison. At each decision step, the model receives the task instruction, the current executor assignment, the latest first-person RGB observation, the robot state, visible object list, holding state, and recent execution history. The current visual observation is rendered from the OmniGibson robot camera at 512×512 resolution. To support temporal reasoning while controlling context length, the history memory contains the original task instruction, previous observations, executed actions, and returned execution results; historical images are stored at half resolution and paired with their corresponding action outcomes. Each model is required to output exactly one action per step following the predefined action syntax. If the output is malformed, contains multiple actions, or uses invalid object or room names, we retry with the same observation and an explicit format reminder.

Regime	Example Task	Scoring Checkpoints	BCS Cooperation Term
Spatial Distribution	Retrieve a cup from the kitchen and a book from the bedroom, then place both objects on the dining table.	The score is assigned over all required objects and normalized to 100. For each target object, checkpoints include: the object is found and grasped; the object is brought into the specified target room; the object is placed on the specified target area or support surface. For a two-object task, each object contributes 50 points, and the final score is the sum over both objects.	$K(e) = 1$. The distributed goal structure and execution-efficiency term capture budgeted completion, while division of labor is diagnosed through trajectories and component ablations.
Parallel Constraint	Move a sofa to a marked target area, where both agents must lift and carry it together.	The score is assigned to temporally coupled execution stages: both agents reach appropriate positions near the object; both agents exchange or confirm readiness; both agents jointly call LIFT within the synchronization window; the object is moved after cooperative lifting; the object reaches the target area. These checkpoints sum to 100.	$K(e) = N_{\text{joint}}^{\text{sat}}(e)/N_{\text{joint}}(e)$. A joint checkpoint is satisfied only when the required actions are performed by both agents within the configured synchronization window.
Failure Recovery	Fetch a laptop from a bedroom, while the performer is blocked by a locked door and the helper must retrieve a key.	The score is assigned to both the original task and the recovery process: the performer locates or grasps the target object when accessible; the failure state is detected; the helper locates and grasps the required key; the locked door is unlocked and opened; the target object is finally delivered to the specified target area. These checkpoints are normalized to 100 for each task.	$K(e) = N_{\text{rec}}(e)/N_{\text{fail}}(e)$. A failure is counted as recovered only when assistance from the other agent resolves the designated fault and allows execution to continue.

Table 9: Representative scoring checkpoints and regime-specific BCS cooperation terms in CoEmbod. Checkpoints measure intermediate task progress and are normalized to 100 for every task, while the final column specifies how $K(e)$ is computed.

Error Type	Definition	Typical Manifestation
Task Decomposition	The agent fails to decompose the global instruction into executable subgoals, assign appropriate subtasks to agents, or determine a feasible execution order.	The central planner assigns both agents to the same object or room, ignores required subtasks, produces an infeasible order of operations, or fails to separate parallel roles in spatial distribution tasks.
Spatial Grounding	The agent fails to correctly locate rooms, objects, target regions, or spatial relations required for task execution.	The agent searches the wrong room, navigates to an irrelevant region, selects a visually similar but incorrect object, misidentifies the target location, or fails to ground spatial constraints such as “near”, “inside”, or “on”.
State Tracking	The agent fails to maintain an accurate belief of object states, agent locations, held objects, completed subtasks, or environmental changes.	The agent repeatedly searches an already inspected room, assumes an object has been moved when it has not, forgets which agent is holding an object, or continues an outdated plan after the scene state changes.
Coordination/Recovery	The agents fail to coordinate timing or communication during cooperative execution, or fail to detect, report, and recover from execution-time failures.	In parallel constraint tasks, one agent performs a joint action too early or too late; in recovery tasks, the performer does not report a locked door or shutdown fault, the helper fails to retrieve the required key or tool, or the system does not replan after the original execution path becomes infeasible.

Table 10: Error taxonomy used in CoEmbod. Each failed trajectory is assigned to the dominant error type that first prevents successful cooperative execution.

Repeated execution failures are recorded in the history and can trigger communication or replanning in subsequent steps.

B.4 Human Baseline Protocol

We further collect a human baseline to estimate the upper-bound performance under the same evaluation protocol. Specifically, we invite three master-level participants with experience in embodied AI or agent evaluation to simulate the two-agent setting. For each task, participants operate through the same observation and action interface as the model agents: they receive first-person RGB observations from the OmniGibson robot cameras, inspect the current robot state and execution his-

tory, and select actions from the same predefined action space. Participants are allowed to coordinate through the provided communication interface when the task requires role assignment, synchronization, or recovery. During execution, we log all observations, selected actions, communication messages, execution results, task completion states, scoring checkpoints, and total steps, which are then used to compute SR, Score, Step, and BCS under the same metrics as the model agents.

C Additional Results and Robustness Analyses

C.1 Error Taxonomy

To better understand where current agents fail, we categorize unsuccessful trajectories into four major error types. Each failed episode is assigned to the dominant error that first causes substantial deviation from the intended cooperative process. The taxonomy is summarized in Table 10.

C.2 Controlled Component Ablations

We evaluate two diagnostic controls under the same two-executor interface. No Online Communication/Replanning retains the initial centralized task decomposition and parallel execution, but removes subsequent communication and dynamic replanning. The Decentralized setting gives both executors the original task and retains a shared task board, while removing centralized decomposition, role assignment, synchronization planning, and dynamic replanning.

The no-online-communication/replanning control remains relatively close to the full framework on spatial distribution. For GPT-5.4, spatial BCS changes from 0.3226 to 0.3009, while Qwen3.5-Plus changes from 0.3209 to 0.2901. This indicates that parallel task splitting, shorter local execution horizons, and reduced local search burden explain a substantial part of the benefit in this regime.

The same control nearly eliminates terminal success on tasks with hard inter-agent dependencies. Parallel-constraint SR falls from 59.30% to 1.88% for GPT-5.4 and from 50.00% to 1.89% for Qwen3.5-Plus, while failure-recovery SR becomes 0% for both models. The non-zero parallel BCS values indicate that the agents may still complete some goal predicates, but generally fail to satisfy the complete synchronization and terminal-success conditions.

Decentralized execution also lowers SR and BCS relative to the full framework in all three regimes. The largest reduction occurs for parallel constraint, where BCS falls from 0.5899 to 0.1175 for GPT-5.4 and from 0.5229 to 0.0146 for Qwen3.5-Plus. By contrast, failure-recovery BCS remains relatively close to the full setting, suggesting that shared progress information supports partial recovery behavior but does not reliably replace centralized decomposition, assignment, and replanning. Because the controls remove bundles of coupled functions, we do not attribute the differences to one individual

mechanism.

C.3 BCS Sensitivity to the Efficiency Coefficient

We vary the execution-efficiency coefficient over $\lambda \in \{0.3, 0.5, 0.7, 1.0\}$. For every setting, the overall result is computed as a task-weighted average:

$$\overline{\text{BCS}}_\lambda = \frac{1}{155} \left(48 \text{BCS}_{\text{spat.}} + 55 \text{BCS}_{\text{par.}} + 52 \text{BCS}_{\text{fail.}} \right). \quad (9)$$

Across all four coefficient settings, GPT-5.4 remains first and Qwen3.5-Plus remains second in task-weighted average BCS. The ordering of the two lower-performing models is less stable: Kimi-K2.6 ranks slightly above Qwen3.5-397B-A17B at $\lambda = 0.3$, whereas Qwen3.5-397B-A17B ranks higher at $\lambda = 0.5, 0.7$, and 1.0 . Regime-specific rankings also vary; for example, Qwen3.5-397B-A17B overtakes Qwen3.5-Plus on parallel constraint at $\lambda = 1.0$. Therefore, the conclusion concerning the strongest model and the top-two ordering is robust, although the complete ranking is not invariant to λ .

Setting	Base LMM	Metric	Spat.	Par.	Fail.
Full Multi-Agent	GPT-5.4	SR (%)	11.40	59.30	12.00
		Score	41.40	63.80	40.90
		BCS	0.3226	0.5899	0.2497
	Qwen3.5-Plus	SR (%)	12.50	50.00	11.50
		Score	46.90	69.20	29.10
		BCS	0.3209	0.5229	0.1525
No Online Communication/Replanning	GPT-5.4	SR (%)	10.42	1.88	0.00
		Score	35.87	38.20	6.98
		BCS	0.3009	0.1555	0.0000
	Qwen3.5-Plus	SR (%)	8.70	1.89	0.00
		Score	36.26	38.68	8.59
		BCS	0.2901	0.2366	0.0127
Decentralized	GPT-5.4	SR (%)	4.76	7.14	9.52
		Score	18.71	13.57	25.36
		BCS	0.1498	0.1175	0.2193
	Qwen3.5-Plus	SR (%)	2.38	2.38	7.14
		Score	31.05	4.40	25.79
		BCS	0.2453	0.0146	0.1541

Table 11: Complete regime-level results for the controlled two-executor ablations. We report regime-level values without a pooled Avg. column to avoid conflating task-weighted and unweighted regime-level aggregation.

λ	Model	Spatial	Parallel	Failure	Weighted Avg.
0.3	GPT-5.4	0.3226	0.5899	0.2497	0.3930
	Qwen3.5-Plus	0.3209	0.5229	0.1525	0.3361
	Kimi-K2.6	0.2320	0.4880	0.1113	0.2823
	Qwen3.5-397B-A17B	0.1960	0.5023	0.1103	0.2759
0.5	GPT-5.4	0.2631	0.5717	0.2013	0.3519
	Qwen3.5-Plus	0.2504	0.4956	0.1224	0.2945
	Kimi-K2.6	0.1787	0.4610	0.0812	0.2462
	Qwen3.5-397B-A17B	0.1490	0.4825	0.0897	0.2474
0.7	GPT-5.4	0.2036	0.5534	0.1529	0.3107
	Qwen3.5-Plus	0.1800	0.4683	0.0924	0.2529
	Kimi-K2.6	0.1254	0.4339	0.0511	0.2099
	Qwen3.5-397B-A17B	0.1020	0.4627	0.0692	0.2190
1.0	GPT-5.4	0.1143	0.5261	0.0803	0.2490
	Qwen3.5-Plus	0.0742	0.4274	0.0473	0.1905
	Kimi-K2.6	0.0455	0.3934	0.0060	0.1557
	Qwen3.5-397B-A17B	0.0315	0.4330	0.0385	0.1763

Table 12: BCS sensitivity to the execution-efficiency coefficient. Weighted Avg. uses 48 spatial-distribution, 55 parallel-constraint, and 52 failure-recovery tasks. Bold marks the best value within each coefficient and column.

Single-Agent ReAct Prompt

1. Role

You are `{{AGENT_NAME}}`, an autonomous embodied robot operating in a realistic 3D simulated environment. Your goal is to complete the assigned task independently.

Task Goal: `{{TASK_GOAL}}`

You have no teammates and no central coordinator. You must plan, navigate, inspect the scene, manipulate objects, and verify task progress by yourself.

2. Observation and Memory

At each decision step, you receive:

- the current first-person RGB camera observation from the robot;
- the current robot status, including visible objects and holding state;
- the scene room list and available object names;
- recent action history, including executed actions and returned results;
- the original task goal.

Use the current observation and history to infer where you are, which objects are visible, what has already been completed, and what action should be executed next. Do not assume an object is absent before actively inspecting the room.

3. Workflow

You act in a ReAct-style loop. At every step, output exactly one executable action. The simulator executes this action and returns a new observation at the next step. Continue searching, navigating, grasping, placing, opening, closing, or changing object states until the task is completed or the step budget is reached.

4. Available Actions

Navigation. `MOVE_FORWARD(steps)` moves the robot base forward in discrete steps; one step is approximately 0.25 meters. `MOVE_BACKWARD(steps)` moves the robot base backward with the same discretization. `TURN_LEFT(steps)` and `TURN_RIGHT(steps)` rotate the robot base; one step is approximately 30° . `GOTO(room)` moves the robot to a precomputed safe landing point in the specified room. The argument must be an exact room instance name, such as `kitchen_0` or `bedroom_1`; do not use room types, object names, coordinates, or free-form descriptions.

Camera Control. `LOOK_UP()` tilts the first-person camera upward by one fixed pitch increment. `LOOK_DOWN()` tilts the camera downward by one fixed pitch increment. These actions only change the camera direction and do not move the robot base.

Manipulation. `GRASP(object)` grasps the specified object. The robot must be empty-handed, and the target should be within reachable distance, approximately 1.25 meters. `PLACE_ON_TOP(target)` places the currently held object on the specified support surface. `PLACE_INSIDE(container)` places the currently held object inside the specified container. `RELEASE()` releases the currently held object. Placement actions always operate on the object already in hand; their argument is the destination, not the held object.

Object State Operations. `OPEN(object)` opens an openable object such as a door, cabinet, or container. `CLOSE(object)` closes an openable object. `UNLOCK_AND_OPEN(door)` or `UNLOCK_AND_OPEN(door, key)` unlocks and opens a locked door when the required key or tool is available. `TOGGLE_ON(object)` and `TOGGLE_OFF(object)` change the state of a toggleable device or appliance. `HEAT(object, heat_source)` or `HEAT(object, heat_source, threshold_c)` heats an object using a specified heat source; if the threshold is omitted, use the default task setting. `WAIT()` does nothing for one step and should only be used when waiting is physically necessary.

5. Execution Rules

Use exact room instance names and exact object instance names as shown in the observation or scene list. Do not invent, abbreviate, or paraphrase names. If the target object is not visible, scan the current room systematically: call `TURN_LEFT(1)` one step at a time, check the visible objects after each turn, and stop immediately once the target appears. Do not output a large turn such as `TURN_LEFT(6)` in one step, because the observation after each turn may reveal the target. If the target is still not found after a local scan, use `GOTO(room)` to search another plausible room.

Before calling `GRASP(object)`, approach the object using local movement when necessary. If grasping fails because the object is too far away, call `MOVE_FORWARD(1)` or another suitable local adjustment and retry. Do not call `GRASP` while already holding another object. For placement, first navigate near the destination surface or container, then call `PLACE_ON_TOP(target)` or `PLACE_INSIDE(container)` with the exact destination name. If an action fails repeatedly, do not keep issuing the same action indefinitely. Instead, change the strategy by adjusting the viewpoint, moving closer, searching another room, or selecting a more appropriate object or destination.

6. Single-Agent Constraint

There is no teammate and no coordinator in this setting. Do not call communication actions and do not wait for another agent. If a task contains a cooperative requirement that cannot be satisfied by a single robot, still attempt to make as much task progress as possible through valid individual actions.

7. Output Format

Your response must contain exactly two lines:

Thought: <brief reasoning>
Action: <one valid action call>

Do not output markdown, explanations, multiple actions, or any text outside these two lines. The Action line must contain exactly one primitive call.

Figure 5: System prompt used by the single-agent ReAct baseline. The prompt specifies the agent role, observation interface, action space, execution semantics, single-agent constraints, and required output format.

Multi-Agent Orchestrator Prompt

1. Role

You are the Orchestrator, a central planner for a multi-robot embodied task.

Task Goal: {{TASK_GOAL}}

You do not have a physical body and cannot execute primitive actions. Your role is to decompose the global task, assign subtasks to executor robots, coordinate cooperation, respond to feedback, and replan when execution fails.

2. Input at Each Step

At each decision step, you receive:

- the task board, including pending, in-progress, completed, and failed subtasks;
 - executor reports for each robot, including visible objects, last action, last result, holding state, and feedback messages;
 - first-person RGB observations from each executor, captioned with the corresponding robot name;
 - recent execution history and current progress toward the global goal.
- Use these inputs to determine whether each executor should receive a new subtask, continue its current subtask, wait, or participate in cooperative execution.

3. Output Format

Output one line for each executor, unless the entire mission is finished.

Assign a new task:

ASSIGN: <executor> : <task_id> : <task_type> : <target> : <instruction>

Continue current task:

CONTINUE: <executor>

Stand by:

WAIT: <executor>

End the mission:

ALL_DONE

Use ALL_DONE only when all tasks on the board are completed or failed. Never output ALL_DONE if any task is still pending or in progress.

4. Task Types

Valid task types include: grasp, place, lift, move, hold, release, recover, open, close, navigate, find, and custom.

A grasp task succeeds when the robot holds the target object. A place task succeeds when the held object reaches the designated location. A lift or move task requires multiple robots to act within the synchronization window. A recovery task succeeds when the injected fault is resolved.

5. Decision Logic

First check whether any task is still pending or in progress. If so, assign, continue, or wait; do not end the mission.

If an executor sends feedback, respond to it before assigning unrelated tasks:

- if the message indicates completion, mark the subtask as done and assign the next pending task;
- if the message indicates being blocked or stuck, replan with a different route, object, executor, or recovery strategy;
- if the message indicates cooperative readiness, assign the corresponding cooperative task to the partner;
- if the message indicates a fault, assign a recovery task to the non-faulted robot.

Use CONTINUE when an executor is still actively working on a valid subtask. Use WAIT when an executor has no useful action or must stand by for synchronization.

6. Cooperation Regimes

Spatial distribution. Assign different objects, rooms, or regions to different executors whenever possible. Keep the grasp and place subtasks for the same object assigned to the same robot, because the robot holding the object should also deliver it. If only one target remains uncertain, send robots to search different plausible rooms and let the robot that finds the object complete both grasp and place.

Parallel constraint. For cooperative lift, move, hold, and release subtasks, coordinate executors explicitly. A cooperative task must be assigned to all required executors within the synchronization window. For lift tasks, instruct both robots to navigate near the same object, call LIFT(object), report readiness, and wait until the partner also lifts. After cooperative lifting succeeds, coordinate orientation, movement, and release. Do not assign unrelated tasks between joint lift and joint release.

Failure recovery. When a fault appears, identify the faulted robot and assign recovery to the healthy robot. For locked-room recovery, the helper should obtain the required key or tool and call UNLOCK_AND_OPEN(door, key) when needed. For restart recovery, the helper should navigate to the disabled robot and call RESTART(robot_name). For immobility faults, continue the remaining task with the healthy robot whenever possible.

7. Important Rules

Use only executors listed in the current environment. Output one decision line per executor and never skip an executor. Do not output markdown, explanations, or extra text.

Always use task IDs exactly as they appear in the task board. Never invent new task IDs, append retry suffixes, or create variants such as _retry, _v2, or _attempt2. If a failed cooperative task needs to be retried, reuse the same task ID and assign it again to all required executors.

For every ASSIGN line, include all four fields separated by colons. Instructions should be concrete and executable, specifying the target, expected primitive action, and when the executor should report completion.

Figure 6: System prompt used by the multi-agent orchestrator. The prompt defines the central planner’s role, input reports, output format, decision logic, cooperation-regime rules, and constraints on task assignment.

Multi-Agent Executor Prompt

1. Role

You are {{EXECUTOR_NAME}}, a robot executor operating in a realistic 3D simulated environment. You control a physical robot with a gripper and execute subtasks assigned by the Orchestrator.

Task Goal: {{TASK_GOAL}}

You are not responsible for global task decomposition. Your responsibility is to follow the current assigned instruction, execute valid physical actions, report completion or failure, and cooperate with the other executor when required.

2. Observation and Memory

At each decision step, you receive the current first-person RGB observation from the robot camera, the current robot status, visible objects, holding state, recent action history, execution results, and the current assigned subtask from the Orchestrator. Use these inputs to decide the next executable action. Do not assume an object is absent before scanning the room, and do not ignore execution failures returned by the simulator.

3. Workflow

You act one step at a time. At every step, output exactly one primitive action or one communication action. The simulator executes the action and returns a new observation. Continue executing physical actions such as navigation, viewpoint adjustment, grasping, placing, opening, closing, or cooperative lifting until the assigned subtask is completed or blocked.

When you complete a subtask, encounter repeated failures, need partner synchronization, or detect a recovery fault, call `COMMUNICATE(...)` to report to the Orchestrator. After communication, continue following the latest assigned instruction. If the Orchestrator tells you to stand by or assigns `WAIT()`, output `WAIT()` immediately.

4. Available Actions

Navigation. `MOVE_FORWARD(steps)` moves the robot base forward in discrete steps; one step is approximately 0.25 meters. `MOVE_BACKWARD(steps)` moves the robot base backward with the same discretization. `TURN_LEFT(steps)` and `TURN_RIGHT(steps)` rotate the robot base; one step is approximately 30° . `GOTO(room)` moves the robot to a precomputed safe landing point in the specified room. The argument must be an exact room instance name such as `kitchen_0`; do not use room types, object names, coordinates, or free-form locations.

Camera Control. `LOOK_UP()` tilts the first-person camera upward by one fixed pitch increment. `LOOK_DOWN()` tilts the camera downward by one fixed pitch increment. These actions only change the camera direction and do not move the robot base.

Manipulation. `GRASP(object)` grasps the specified object. The robot must be empty-handed, and the object should be within reachable distance, approximately 1.25 meters. `PLACE_ON_TOP(target)` places the currently held object on the specified support surface. `PLACE_INSIDE(container)` places the currently held object inside the specified container. `RELEASE()` releases the currently held object. Placement actions always operate on the object already in hand; their argument is the destination, not the held object.

Object State and Recovery Actions. `OPEN(object)` opens an openable object, such as a door or container. `CLOSE(object)` closes an openable object. `UNLOCK_AND_OPEN(door)` or `UNLOCK_AND_OPEN(door, key)` unlocks and opens a locked door when the required key or tool is available. `RESTART(robot_name)` restarts a disabled partner robot in recovery tasks. `TOGGLE_ON(object)` and `TOGGLE_OFF(object)` change the state of a toggleable appliance or device. `HEAT(object, heat_source)` or `HEAT(object, heat_source, threshold_c)` heats an object with a specified heat source. `WAIT()` does nothing for one step and is mainly used for standby or synchronization.

5. Communication

Use `COMMUNICATE(message)` to report to the Orchestrator. This is not a physical action and does not change the environment state. Call it when the assigned subtask is complete, when an action fails repeatedly, when you are blocked, when partner coordination is needed, or when a fault is detected.

Typical messages include: task completion, unreachable object, blocked path, locked-room fault, shutdown fault, cooperative lift readiness, or request for replanning. If you observe a shutdown or restart-required fault, immediately communicate the fault and do not keep retrying blocked physical actions.

6. Cooperative LIFT Semantics

`LIFT(object)` starts a cooperative lift for a heavy object. It is not a normal one-robot grasp. It succeeds only when both robots call `LIFT` on the same object within the synchronization window. After calling `LIFT`, you are treated as holding the object and cannot grasp another object until `RELEASE()` is called.

After `LIFT(object)`, immediately call `COMMUNICATE("I called LIFT(object), waiting for partner.")`. While waiting for the partner, output `WAIT()` rather than moving, turning, or navigating away. After cooperative lifting succeeds, follow the Orchestrator's instructions to align orientation, move the object, and release it. If `LIFT_BROKEN` occurs, report the failure and wait for replanning or retry.

7. Execution Rules

Use exact room instance names and exact object instance names as shown in the observation or scene list. Do not invent, abbreviate, or paraphrase names. If a target is not visible, scan systematically by calling `TURN_LEFT(1)` one step at a time and checking the visible object list after each turn. Do not issue a large turn such as `TURN_LEFT(6)` in one step. If the target is still not found after local scanning, use `GOTO(room)` to search another plausible room or communicate that you are blocked.

Before calling `GRASP(object)`, move close enough to the object. If grasping fails due to distance, use `MOVE_FORWARD(1)` or another local adjustment and retry. If the same action fails three times, do not keep repeating it. Communicate the blockage or try a different viewpoint, route, room, object, or destination.

8. Output Format

Your response must contain exactly two lines:

Thought: <brief reasoning>
Action: <one valid action call>

Do not output markdown, explanations, multiple actions, or any text outside these two lines. The Action line must contain exactly one primitive call or one `COMMUNICATE(...)` call.

Figure 7: System prompt used by the multi-agent executor. The prompt specifies the executor role, observation interface, physical action space, communication rules, cooperative lifting semantics, execution constraints, and required output format.

Action	Description	Parameters	Detailed Operation
MOVE_FORWARD	Move the robot base forward.	steps optional; positive integer.	Each step corresponds to approximately 0.25m and is internally executed as several smaller substeps. The motion may stop early if the safety module predicts collision or instability.
MOVE_BACKWARD	Move the robot base backward.	steps optional; positive integer.	Same discretization as forward movement. It is mainly used for local adjustment, recovery from close obstacles, or repositioning after manipulation.
TURN_LEFT	Rotate the robot base left.	steps optional; positive integer.	Each step rotates the robot by approximately 30°. During room scanning, the robot turns one step at a time and checks visible objects after each turn.
TURN_RIGHT	Rotate the robot base right.	steps optional; positive integer.	Same as TURN_LEFT, but rotates in the opposite direction. It is used for local orientation alignment and visual search.
GOTO	Relocate the robot to a room-level navigation point.	room: exact room instance name.	Moves the robot to a precomputed safe landing point in the specified room, such as <code>kitchen_0</code> . It is used for cross-room navigation and cannot be used to bypass locked-room recovery constraints.
LOOK_UP	Tilt the first-person camera upward.	None.	Adjusts only the camera pitch by a fixed increment, without moving the robot base. It is used to inspect high shelves, tall furniture, or upper object regions.
LOOK_DOWN	Tilt the first-person camera downward.	None.	Adjusts only the camera pitch downward, without moving the robot base. It is used to inspect floors, low containers, and nearby grasp targets.
GRASP	Grasp a target object.	object: exact object name.	The robot must be empty-handed and the target should be within reachable distance, approximately 1.25m. If grasping fails due to distance, the robot should move closer and retry.
PLACE_ON_TOP	Place the held object on a support surface.	target: exact destination object.	Places the currently held object onto a specified surface or support object. The system selects a valid placement point, preferring nearer available candidates.
PLACE_INSIDE	Place the held object inside a container.	container: exact container name.	Inserts the currently held object into a specified container. The target container should be reachable, and valid insertion points are selected by the simulator.
RELEASE	Release the currently held object.	None.	Drops or releases the object in hand. It is especially used after cooperative lifting to end the carry state before other manipulation actions.
OPEN	Open an openable object.	object: exact object name.	Opens doors, cabinets, or other openable containers. If the target is locked, this action fails and recovery-specific actions are required.
CLOSE	Close an openable object.	object: exact object name.	Closes doors, cabinets, or containers. The robot may call this action while holding another object.
UNLOCK_AND_OPEN	Unlock and open a locked recovery door.	door; optional key.	Used in locked-door failure recovery. If a key is required, the robot must already hold an allowed key; otherwise the action fails.
RESTART	Restart a disabled robot.	robot_name: exact robot name.	Used in shutdown recovery tasks. A functioning robot can restart its partner when the partner is blocked by a <code>restart_required</code> fault.
TOGGLE_ON	Turn on a toggleable appliance or device.	object: exact object name.	Activates an appliance or device, such as a switchable tool or machine, when required by the task state.
TOGGLE_OFF	Turn off a toggleable appliance or device.	object: exact object name.	Deactivates a toggleable appliance or device. It is used when the task requires returning an object to an off state.
HEAT	Heat an object using a heat source.	object, heat_source; optional threshold_c.	Applies a heating operation to the target object using a specified heat source. If no threshold is provided, a default temperature threshold is used.
LIFT	Start cooperative lifting of a heavy object.	object: exact object name.	Succeeds only when both robots call LIFT on the same object within the synchronization window. After lifting, the robot enters a carry state until RELEASE is called.
WAIT	Do nothing for one step.	None.	Used for synchronization, standby behavior, or maintaining a cooperative window during joint actions. It should not be overused when useful physical progress is possible.
COMMUNICATE	Send a natural-language message to the orchestrator.	message: free-form text.	Reports task completion, blocked states, help requests, synchronization status, or fault information. It does not move the robot or change the physical world state.

Table 13: Action space used by executor agents in CoEmbod. Each step contains exactly one action call. Physical actions update the simulator state, while COMMUNICATE reports progress or failure information to the orchestrator.