Utility-Aware Task Decomposition and Exchange across LLM Agents

Shunta Kimura[1,2], **Takuya Hiraoka**[1,3], **Ryota Higa**[1,3],
Yoshimasa Tsuruoka[1,2], **Katsuhide Fujita**[1,4]

[1]National Institute of Advanced Industrial Science and Technology [2]The University of Tokyo
[3]NEC Corporation [4]Tokyo University of Agriculture and Technology
{shunta-kimura, yoshimasa-tsuruoka}@g.ecc.u-tokyo.ac.jp,
{takuya-h1, r-higaryouta}@nec.com,
katfuji@cc.tuat.ac.jp

Abstract

In recent years, large language model (LLM)-based multi-agent environments have attracted growing attention. Most existing research on LLM agents assumes the presence of either a central coordinator or a shared utility function. In real-world settings, however, agents often operate in environments that are not fully collaborative, requiring them to estimate the utility functions of others and negotiate task assignments. In this work, we introduce a framework that enables LLM agents to perform task decomposition and exchange in a utility-aware manner, and we assess its capabilities under such circumstances. The framework incorporates mechanisms for utility estimation and per-turn proposal validation. Experimental results show that these mechanisms improve agreement stability and negotiation efficiency when applied to LLMs with strong reasoning capabilities.

1 Introduction

Recent years have witnessed a surge of interest in large language model (LLM)-based multi-agent systems in both academia and industry because such systems have proven effective in accomplishing complex tasks (e.g., software development) that single-agent systems would struggle with (Chen et al. 2024). Software infrastructure–such as the Agent2Agent protocol (A2A Project Contributors 2025)–that facilitates collaboration between multiple agents owned by different individuals or organizations is also being actively developed. This trend makes research on designing effective mechanisms for agent collaboration under diverse conditions increasingly important.

The majority of existing research on LLM-based multi-agent systems focuses on scenarios where either *a central coordinator* or a *shared utility function is present*, leaving situations without these conditions relatively underexplored (see Section 2). In real-world contexts, however, differences in authority or governance often make it difficult to establish a central coordinator–particularly when agents are owned by different organizations or individuals. Conflicts of interest can also arise, which can discourage agents from disclosing their utility functions for strategic reasons. In such cases, agents must negotiate task allocation. During negotiation, agents estimate other agents' utility functions, decompose tasks into subtasks that counterparts are likely to accept, and then delegate or exchange those subtasks. Understanding the extent to which current LLM agents can handle such scenarios is critical for designing effective methods of collaboration and coordination.

In this paper, we present a novel framework that equips LLMs to perform task decomposition and exchange in a utility-aware manner, and evaluate the capability of current LLM agents under such settings (Section 3). Within this framework, agents coordinate under our *utility-aware strategy* comprising (i) prompts designed to encourage reasoning about the utility functions of others and (ii) per-turn proposal validation with feedback. We then use this framework to conduct experiments examining how LLM agents behave in two types of negotiation environments: competitive and cooperative (Section 4 and 5).

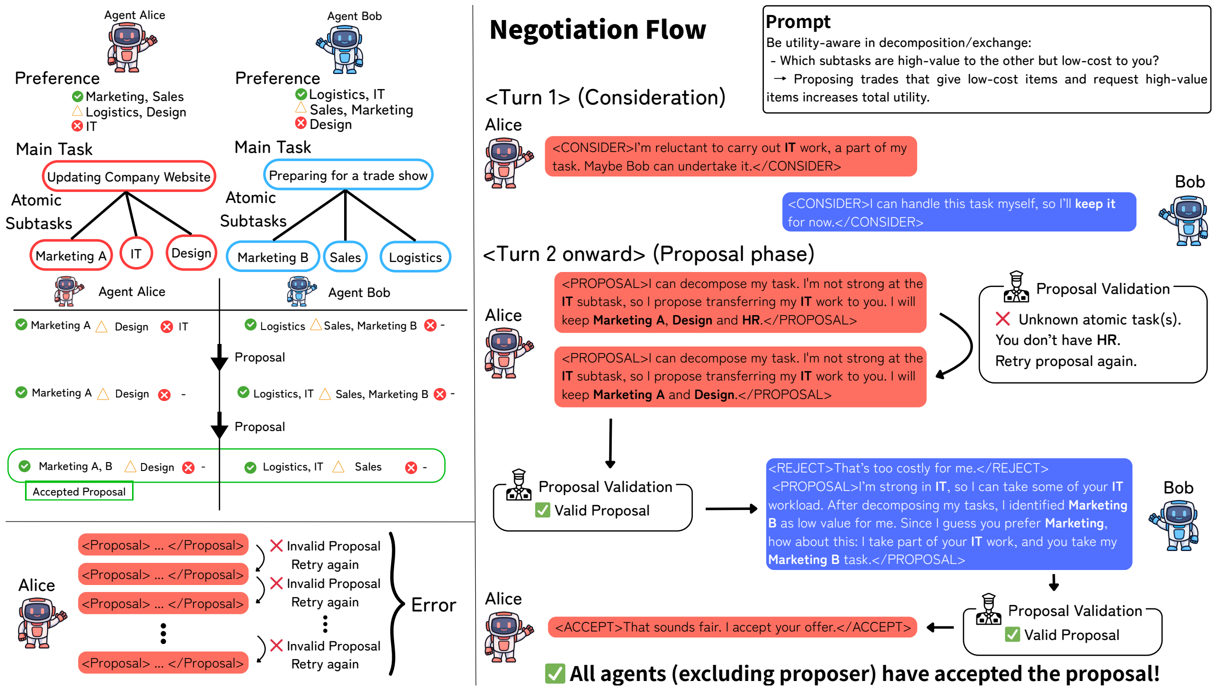
2 Related Work

Our work is related to prior research on single LLM agents, multi-LLM agents, and negotiation agents in the following respects.

Single LLM Agents. LLM agents are autonomous systems that use LLMs for reasoning and interact with external tools or environments to complete tasks (Luo et al. 2025). A common feature is task decomposition, where a complex task is broken down into smaller, clearer subtasks that can be sequentially solved and aligned with appropriate tools (Yao et al. 2023; Qin et al. 2023; Wang et al. 2023; Long 2023; Shen et al. 2024). This stepwise reasoning makes otherwise intractable tasks manageable, but existing studies consider only a single agent and thus do not address coordination in multi-agent settings.

Multi-LLM Agents. Multi-agent approaches have been applied to a variety of complex tasks, such as software development, mathematical reasoning, and embodied environments (Li et al. 2023; Chen et al. 2024; Hong et al. 2024; Huang et al. 2025). These studies typically assume shared objectives or the presence of a central coordinator. In contrast, our work targets decentralized settings where agents have different utilities and no central coordinator.

Negotiation LLM Agents. Another line of work explores negotiation scenarios, showing that LLM agents can adopt



Prompt
Be utility-aware in decomposition/exchange:
- Which subtasks are high-value to the other but low-cost to you?
→ Proposing trades that give low-cost items and request high-value items increases total utility.

<CONSIDER>I'm reluctant to carry out **IT** work, a part of my task. Maybe Bob can undertake it.</CONSIDER>

<CONSIDER>I can handle this task myself, so I'll **keep it** for now.</CONSIDER>

<PROPOSAL>I can decompose my task. I'm not strong at the **IT** subtask, so I propose transferring my **IT** work to you. I will keep **Marketing A**, **Design** and **HR**.</PROPOSAL>

✗ Unknown atomic task(s). You don't have **HR**. Retry proposal again.

<PROPOSAL>I can decompose my task. I'm not strong at the **IT** subtask, so I propose transferring my **IT** work to you. I will keep **Marketing A** and **Design**.</PROPOSAL>

<REJECT>That's too costly for me.</REJECT>
<PROPOSAL>I'm strong in **IT**, so I can take some of your **IT** workload. After decomposing my tasks, I identified **Marketing B** as low value for me. Since I guess you prefer **Marketing**, how about this: I take part of your **IT** work, and you take my **Marketing B** task.</PROPOSAL>

<ACCEPT>That sounds fair. I accept your offer.</ACCEPT>

Figure 1: Negotiation example with utility-aware task decomposition and exchange. **Top-left:** each agent's main task (3 indivisible subtasks) with skill icons—green check (strong), yellow triangle (neutral), red cross (weak). **Left-middle:** task ownership across proposals under utility-aware decomposition/exchange. **Bottom-left:** termination by *Error*. **Right:** flow—consideration (Turn 1), proposals (Turn 2+); each proposal is validated before delivery to the counterpart.

diverse strategies and compete or cooperate with humans in economic or strategic games (Gandhi, Sadigh, and Goodman 2023; Duan et al. 2024; Bianchi et al. 2024; Hua et al. 2024; Shapira et al. 2024; Zhou et al. 2024; Abdelnabi et al. 2024). These studies demonstrate negotiation and strategic reasoning skills but do not consider explicit task delegation or utility-aware task decomposition, which are central to our work.

3 Proposed Framework

In this section, we introduce a novel framework for assessing the capabilities of LLM agents. The framework evaluates LLM agents using a *utility-aware strategy*, defined as the integration of two components (right panel of Figure 1): (i) a prompt that asks agents to reason about *both* parties' utilities; and (ii) per-turn *proposal validation* with feedback. As shown in the top-left panel of Figure 1, each agent starts with one *main task* (three atomic subtasks) and a natural-language preference profile.

As illustrated in Figure 1, agents seek a mutually acceptable task decomposition and exchange through negotiation. A negotiation proceeds in three steps:

(1) Strategic consideration: In Turn 1, each agent privately plans to maximize its utility, compares options, and selects an initial proposal and overall plan; no proposal is sent in this turn.

(2) Proposal phase: From Turn 2 onward, agents alternate proposals: either decomposing and exchanging tasks or keeping them unchanged; accept/reject signals reveal the counterpart's valuations and inform the next proposal. Invalid proposals are detected before being delivered to the counterpart; when flagged, the proposer receives targeted feedback and may retry locally without consuming the turn. These mechanisms elicit proposals responsive to revealed preferences and drive efficient convergence.

(3) Termination and outcomes: The negotiation ends when (i) the counterpart accepts (`Agreed`), (ii) the turn limit is reached without agreement (`Disagreed`), or (iii) an agent exceeds the per-turn limit on invalid proposals (`Error`) (bottom-left panel of Figure 1). In the `Error` case, no final allocation is recorded and no utilities are computed.

Utility-aware Decomposition and Exchange. During negotiation, agents may decompose their current composite task into smaller composites while explicitly considering the counterpart's utility. For example, in the right panel of Figure 1, Bob infers from Alice's prior proposal that Alice favors *Marketing* and incorporates that signal into the next offer. Because each rejection and counterproposal reveals what the other side values or disfavors, agents use this feedback to refine their decompositions and adjust subsequent proposals so they are easier to accept, helping the dialogue converge more quickly to a mutually satisfactory exchange.

4 Experimental Setup

Using our utility-aware framework, we compare the framework to a non-utility-aware baseline in both cooperative and

Table 1: Comparison across models (10 runs per setting). All numerical values are reported as mean $\pm$ standard deviation (SD) unless otherwise noted. Columns: **Agreement** = rate (% of valid runs, i.e., excluding `Error`); **Error** = rate (% of all runs); **Turns** = number of turns per valid run; **Alice/Bob** = mean normalized utility of the final allocation (range $[0, 1]$); **SW (social welfare)** = Alice+Bob (range $[0, 2]$); L_2 **to Pareto** = Euclidean distance to the Pareto front. **Proposed** = utility-aware (explicit utility reasoning + per-turn proposal validation with feedback). Dashes indicate no valid runs.

(a) Cooperative

Model	Setting	Agreement	Error	Turns	Alice	Bob	SW	L_2 to Pareto
`o4-mini`	Proposed	$100 \pm 0\%$	$0 \pm 0\%$	3.0 ± 1.4	0.78 ± 0.07	0.74 ± 0.15	1.52 ± 0.20	0.27 ± 0.15
	Baseline	$83.3 \pm 15.2\%$	$40 \pm 15\%$	4.8 ± 4.0	0.81 ± 0.10	0.79 ± 0.12	1.60 ± 0.22	0.21 ± 0.15
`GPT-4.1-mini`	Proposed	$100 \pm 0\%$	$80 \pm 13\%$	2.0 ± 0.0	0.75 ± 0.00	0.70 ± 0.00	1.45 ± 0.00	0.32 ± 0.0
	Baseline	$28.6 \pm 17.1\%$	$30 \pm 14\%$	7.7 ± 3.9	0.75 ± 0.00	0.70 ± 0.00	1.45 ± 0.00	0.32 ± 0.0
`GPT-4.1-nano`	Proposed	$0 \pm 0\%$	$100 \pm 0\%$	—	—	—	—	—
	Baseline	$0 \pm 0\%$	$50 \pm 16\%$	10 ± 0.0	0.75 ± 0.00	0.70 ± 0.00	1.45 ± 0.00	0.32 ± 0.0

(b) Competitive

Model	Setting	Agreement	Error	Turns	Alice	Bob	SW	L_2 to Pareto
`o4-mini`	Proposed	$100 \pm 0\%$	$10 \pm 9\%$	3.2 ± 2.0	0.62 ± 0.14	0.26 ± 0.06	0.88 ± 0.13	0.022 ± 0.013
	Baseline	$60.0 \pm 21.9\%$	$50 \pm 16\%$	6.8 ± 4.0	0.58 ± 0.00	0.25 ± 0.00	0.83 ± 0.00	0.028 ± 0.0
`GPT-4.1-mini`	Proposed	$0 \pm 0\%$	$100 \pm 0\%$	—	—	—	—	—
	Baseline	$0 \pm 0\%$	$0 \pm 0\%$	10 ± 0.0	0.58 ± 0.00	0.25 ± 0.00	0.83 ± 0.00	0.028 ± 0.0
`GPT-4.1-nano`	Proposed	$0 \pm 0\%$	$100 \pm 0\%$	—	—	—	—	—
	Baseline	$0 \pm 0\%$	$40 \pm 15\%$	10 ± 0.0	0.58 ± 0.00	0.25 ± 0.00	0.83 ± 0.00	0.028 ± 0.0

(c) Initial utilities

Scenario	Alice	Bob	SW	L_2 to Pareto
Cooperative	0.75	0.70	1.45	0.32
Competitive	0.58	0.25	0.83	0.028

competitive scenarios.

Tasks. We adopt two tasks from Shen et al. (2024) as the *main tasks*, each decomposed into three indivisible *atomic tasks*; each agent initially holds one of these main tasks. To enumerate all combinations of the atomic tasks, we prompt an LLM to generate coherent natural-language task descriptions. We refer to each synthesized description as a *composite task*. See Appendix A.1 for examples of main, atomic, and composite tasks.

Utility Generation. We construct the agent–task utility table using an LLM. For each composite task c and agent $a \in \{\text{Alice}, \text{Bob}\}$, we supply (i) the description of c and (ii) the agent's natural-language preferences, and ask the model to return a utility for c. Repeating this for all c yields a utility vector that is aligned one-to-one with the set of composite task descriptions. See Appendix A.2 for examples of these natural-language preferences and the corresponding generated utilities.

Scenario. We implement two scenario types: *competitive* and *cooperative*. We control the degree of competition by manipulating the degree of overlap in the agents' natural-language preferences. See Appendix A.3 for examples of utility allocation under each scenario.

Proposal Validation. At each turn, the proposal undergoes validation before being delivered to the other agent; if it is invalid, we record the invalid proposal pattern and ask the proposer to retry with targeted feedback; these retries do not advance the turn counter. We cap per-turn retries at $K = 5$. If an agent exceeds this limit in a single turn, we terminate the negotiation and record the outcome as `Error`. See Appendix B.2 for examples of validation rules and invalid proposal patterns.

Negotiation Strategies. We compare our *utility-aware strategy* against a baseline with two modifications:

- **Remove utility-aware guidance from the prompt.** We delete prompt text that encourages or instructs agents to adopt a utility-aware strategy.
- **No per-turn validation.** We omit proposal-validity checks during intermediate turns, while retaining the final-turn check.

See Appendix B for the full prompt templates of these strategies.

5 Results and Analysis

Table 1 shows the results of negotiation experiments using three LLMs: o4-mini-2025-04-16, GPT-4.1-mini-2025-04-14, and GPT-4.1-nano-2025-04-14. In each run, both agents use the same model. Across models, the impact of the utility-aware strategy is model-dependent. With `o4-mini`, the strategy delivers *robust and fast* convergence in both scenarios: agreement: 100±0.0%; errors: 0±0.0% in cooperative, 100±0.0% and 10±9% in competitive, and it shortens negotiations relative to the baseline (3.0±1.4 vs. 4.8±4.0 turns in cooperative; 3.2±2.0 vs. 6.8±4.0 in competitive). For non-reasoning models, the strategy is fragile: `GPT-4.1-mini` reaches agreements only in the cooperative setting and with higher error rates, while `GPT-4.1-nano` reaches no agreements under either setting.

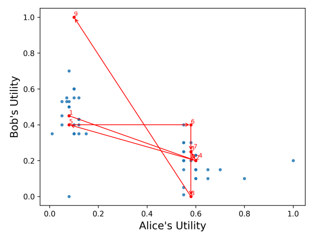
Cooperative Scenario. The cooperative scenario contains many "high–high" allocations, so agents often become satisfied *before* fully pushing toward the Pareto front. This manifests as shorter turns than in the competitive case (utility-aware `o4-mini`: 3.0±1.4 vs. 3.2±2.0; baseline: 4.8±4.0 vs. 6.8±4.0) and outcomes that, while reliable and fast, sit slightly below the baseline in utility (SW 1.52±0.20 vs. 1.60±0.22) and remain farther from the front on average (L_2=0.27±0.15 vs. 0.21±0.15). In this case, the abundance of attractive options encourages earlier acceptance, trading marginal utility gains for quicker and more dependable settlements.

Competitive Scenario. In competitive settings, genuinely "both-high" points are scarce. The baseline struggles (lower agreement, longer runs), whereas the utility-aware strategy with `o4-mini` remains reliable (100±0% agreement on valid runs), converges faster (3.2±2.0 vs. 6.8±4.0 turns), achieves higher social welfare (0.88 vs. 0.83), and moves closer to the Pareto front on average (L_2=0.022 ± 0.013 vs. 0.028 ± 0.0). Here, targeted guidance toward mutually beneficial trades pays off more clearly: the search space is tighter, so structured decomposition and validation help uncover exchanges that improve both efficiency and final allocations.

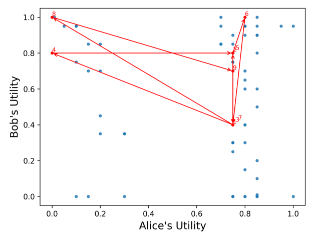
Utility transition across negotiation strategies. Under the baseline setting, in both competitive and cooperative scenarios, agents often pursue unilateral gains with minimal compromise. As shown in Figure 2 (top row), several runs trace corner-to-corner paths in the utility plane, jumping between extremes. With the utility-aware strategy (bottom row), this behavior is largely absent: trajectories settle earlier toward interior allocations, and negotiations terminate in fewer turns (Table 1).

6 Conclusion

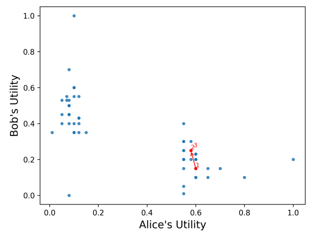
In this paper, we introduce a framework that equips LLMs with negotiation capabilities and evaluate it through utility-aware task decomposition and exchange under partial information, incorporating per-turn validation. Empirically, the strategy helps stronger models (e.g., `o4-mini`) converge faster and more reliably—especially in competitive settings with improved Pareto proximity—while gains are modest in cooperative cases and largely absent for non-reasoning models. As future work, we will continue the empirical analysis under a wider range of conditions.



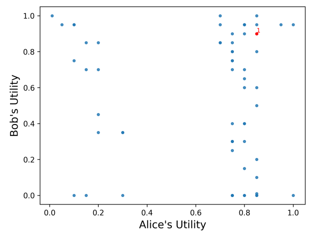
(a) Competitive (baseline)



(b) Cooperative (baseline)



(c) Competitive (utility-aware strategy)



(d) Cooperative (utility-aware strategy)

Figure 2: Proposal–utility trajectories for `o4-mini` across scenarios and settings. Arrows indicate transitions between consecutive proposals; numbers near points denote turn indices. Top row: baseline. Bottom row: utility-aware strategy.

References

A2A Project Contributors. 2025. A2A: Agent-to-Agent Protocol. https://github.com/a2aproject/A2A. Accessed: 2025-08-17.

Abdelnabi, S.; Gomaa, A.; Sivaprasad, S.; Schönherr, L.; and Fritz, M. 2024. Cooperation, Competition, and Maliciousness: LLM-Stakeholders Interactive Negotiation. In *Proc. NeurIPS*.

Bianchi, F.; Chia, P. J.; Yuksekgonul, M.; Tagliabue, J.; Jurafsky, D.; and Zou, J. 2024. How Well Can LLMs Negotiate? NegotiationArena Platform and Analysis. In *Proc. ICML*.

Chen, W.; Su, Y.; Zuo, J.; Yang, C.; Yuan, C.; Chan, C.-M.; Yu, H.; Lu, Y.; Hung, Y.-H.; Qian, C.; et al. 2024. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors. In *Proc. ICLR*.

Duan, J.; Zhang, R.; Diffenderfer, J.; Kailkhura, B.; Sun, L.; Stengel-Eskin, E.; Bansal, M.; Chen, T.; and Xu, K. 2024. GTBench: Uncovering the Strategic Reasoning Limitations of LLMs via Game-Theoretic Evaluations. *arXiv preprint arXiv:2402.12348*.

Gandhi, K.; Sadigh, D.; and Goodman, N. D. 2023. Strategic reasoning with language models. *arXiv preprint arXiv:2305.19165*.

Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S. K. S.; Lin, Z.; et al. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *Proc. ICLR*.

Hua, W.; Liu, O.; Li, L.; Amayuelas, A.; Chen, J.; Jiang, L.; Jin, M.; Fan, L.; Sun, F.; Wang, W.; et al. 2024. Game-theoretic LLM: Agent workflow for negotiation games. *arXiv preprint arXiv:2411.05990*.

Huang, J.-t.; Zhou, J.; Jin, T.; Zhou, X.; Chen, Z.; Wang, W.; Yuan, Y.; Lyu, M. R.; and Sap, M. 2025. On the Resilience of LLM-Based Multi-Agent Collaboration with Faulty Agents. *arXiv preprint arXiv:2408.00989*.

Li, G.; Hammoud, H. A. A. K.; Itani, H.; Khizbullin, D.; and Ghanem, B. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Proc. NeurIPS*.

Long, J. 2023. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*.

Luo, J.; Zhang, W.; Yuan, Y.; Zhao, Y.; Yang, J.; Gu, Y.; Wu, B.; Chen, B.; Qiao, Z.; Long, Q.; et al. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Shapira, E.; Madmon, O.; Reinman, I.; Amouyal, S. J.; Reichart, R.; and Tennenholtz, M. 2024. GLEE: A Unified Framework and Benchmark for Language-based Economic Environments. *arXiv preprint arXiv:2410.05254*.

Shen, Y.; Song, K.; Tan, X.; Zhang, W.; Ren, K.; Yuan, S.; Lu, W.; Li, D.; and Zhuang, Y. 2024. Taskbench: Benchmarking large language models for task automation. In *Proc. NeurIPS*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. React: Synergizing reasoning and acting in language models. In *Proc. ICLR*.

Zhou, X.; Zhu, H.; Mathur, L.; Zhang, R.; Yu, H.; Qi, Z.; Morency, L.-P.; Bisk, Y.; Fried, D.; Neubig, G.; and Sap, M. 2024. SOTOPIA: Interactive Evaluation for Social Intelligence in Language Agents. In *Proc. ICLR*.

A Detailed Experimental Setup

A.1 Tasks

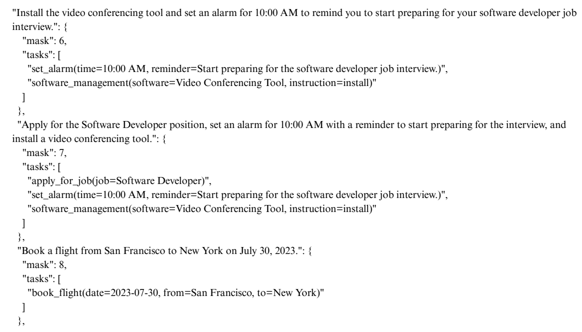
We take two tasks from Shen et al. (2024) and named *main tasks*, each decomposed into three indivisible *atomic tasks*; each agent initially holds one of these *main tasks*. Let $n = 6$ be the total number of atomic tasks across agents. We enumerate all 2^n subsets using an n-bit mask $m \in \{0,1\}^n$ (1=included, 0=omitted) and, for each m, prompt an LLM to generate a coherent natural-language task description. We refer to each synthesized description as a *composite task*.

Since each task description is encoded as an n-bit mask over the universe of atomic tasks $U = \{a_1, \ldots, a_n\}$, the complementary task description mask is obtained by flipping the bits within this n-bit universe. Let $z(d) \in \{0,1\}^n$ be the indicator vector for description$d$, let $M = \mathbf{1}^n$ be the all-ones mask. Then,

$$\begin{aligned} S(d) &= \{a_i | z(d)_i = 1\}, \\ z_{comp}(d) &= z(d) \oplus M, \\ S_{comp}(d) &= \{a_i | z_{comp}(d)_i = 1\} \end{aligned}$$

where $\oplus$ denotes bitwise XOR between two vectors. $S(d)$ and $S_{\text{comp}}(d)$ represent the sets of atomic tasks constituting the composite task described by d and its complementary composite task, respectively. In this way, complementary atomic tasks are computed with bit operations.

Figure 3 shows a subset of the *composite tasks*. Each composite task is associated with a bit mask (shown in decimal) and comprises a set of atomic tasks. For example, "Apply for the Software Developer position, set an alarm for 10:00 AM with a reminder to start preparing for the interview, and install a video conferencing tool." has bit mask 7 (bitwise: `000111`) and consists of three atomic tasks—"apply for job", "set alarm", and "software_management". Figure 4 illustrates each main task alongside its three constituent atomic tasks. Each main task comprises three numbered atomic tasks, and the numbering is consistent across panels; for example, the atomic tasks labeled (i) make up main task (i).

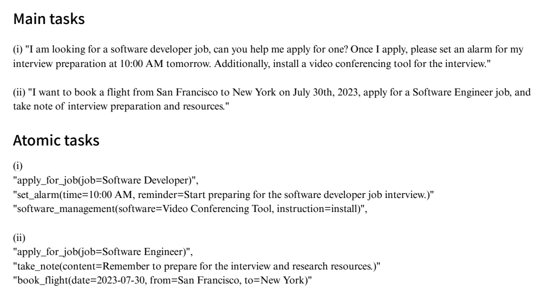


```
"Install the video conferencing tool and set an alarm for 10:00 AM to remind you to start preparing for your software developer job
interview.": {
   "mask": 6,
   "tasks": [
     "set_alarm(time=10:00 AM, reminder=Start preparing for the software developer job interview.)",
     "software_management(software=Video Conferencing Tool, instruction=install)"
   ]
  },
  "Apply for the Software Developer position, set an alarm for 10:00 AM with a reminder to start preparing for the interview, and
install a video conferencing tool.": {
   "mask": 7,
   "tasks": [
     "apply_for_job(job=Software Developer)",
     "set_alarm(time=10:00 AM, reminder=Start preparing for the software developer job interview.)",
     "software_management(software=Video Conferencing Tool, instruction=install)"
   ]
  },
  "Book a flight from San Francisco to New York on July 30, 2023.": {
   "mask": 8,
   "tasks": [
     "book_flight(date=2023-07-30, from=San Francisco, to=New York)"
   ]
  },
```

Figure 3: A part of *composite tasks*

A.2 Utility Generation

We construct the agent–task utility table with an LLM. For each *composite task* c and agent $a \in \{\text{Alice}, \text{Bob}\}$, we provide (i) the composite task description and (ii) the agent's


Main tasks

(i) "I am looking for a software developer job, can you help me apply for one? Once I apply, please set an alarm for my interview preparation at 10:00 AM tomorrow. Additionally, install a video conferencing tool for the interview."

(ii) "I want to book a flight from San Francisco to New York on July 30th, 2023, apply for a Software Engineer job, and take note of interview preparation and resources."

Atomic tasks

(i)
"apply_for_job(job=Software Developer)",
"set_alarm(time=10:00 AM, reminder=Start preparing for the software developer job interview.)"
"software_management(software=Video Conferencing Tool, instruction=install)",

(ii)
"apply_for_job(job=Software Engineer)",
"take_note(content=Remember to prepare for the interview and research resources.)"
"book_flight(date=2023-07-30, from=San Francisco, to=New York)"

Figure 4: Example of *main tasks* and their *atomic* components. Each main task consists of three numbered atomic tasks, and the numbering is consistent across panels; for instance, the set of atomic tasks labeled (i) composes main task (i).

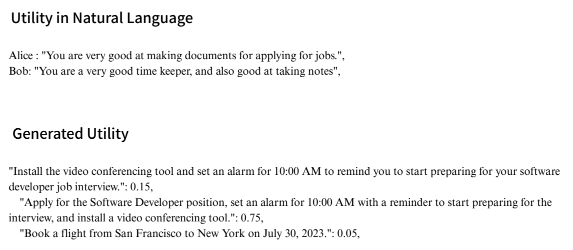
preference written in natural-language, and request a raw score $r_{a,c} \in [0, 100]$. Agent preferences are supplied as a lightweight JSON object, e.g.,

```
{
    "Alice": "You are good at ...",
    "Bob": "You are a marketing
            professional ..."
}
```

For each agent a, we normalize by the agent's maximum score across composites,

$$m_a = \max_c r_{a,c}, \qquad U(a,c) = \begin{cases} r_{a,c}/m_a & \text{if } m_a > 0, \\ 0 & \text{otherwise,} \end{cases}$$

so that each agent's top-scoring composite has utility 1.0 and all others lie in $[0, 1]$. This normalization places both agents' preferences on a common scale during negotiation.


Utility in Natural Language

Alice : "You are very good at making documents for applying for jobs.",
Bob: "You are a very good time keeper, and also good at taking notes",

Generated Utility

"Install the video conferencing tool and set an alarm for 10:00 AM to remind you to start preparing for your software developer job interview.": 0.15,
"Apply for the Software Developer position, set an alarm for 10:00 AM with a reminder to start preparing for the interview, and install a video conferencing tool.": 0.75,
"Book a flight from San Francisco to New York on July 30, 2023.": 0.05,

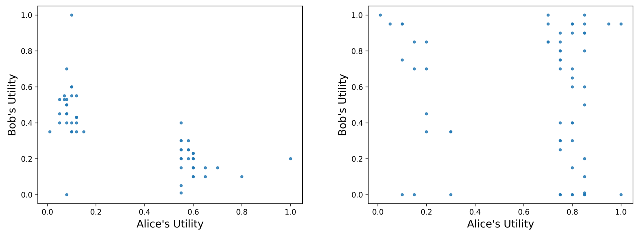
Figure 5: Example of natural-language preferences and LLM-generated utilities. The preferences (used in the *cooperative* scenario) are converted into utilities over a subset of *composite tasks*; see Figure 3.

Figure 5 illustrates how natural-language preferences are specified and how the corresponding utilities are generated for a subset of the *composite tasks* (cf. Figure 3). The natural-language preferences shown here are those used in the *cooperative* scenario.

A.3 Scenario

We implement two scenario types: *competitive* and *cooperative*. We control the degree of competition by manipulating the degree of overlap in the agents' natural-language preferences. When both agents have similar preferences (i.e., they value the same tasks and avoid the same tasks), the interaction tends to be competitive: one agent's high utility tends to preclude the other's utility. Conversely, when preferences are nearly complementary, the setting becomes cooperative and both agents can achieve high utility.

Figure 6 contrasts the resulting utility allocations. In the competitive scenario (Figure 6a), high utility for one agent coincides with low utility for the counterpart due to aligned preferences. In the cooperative scenario (Figure 6b), both agents can reach high utility because their preferences are diverse.



(a) Competitive scenario (b) Cooperative scenario

Figure 6: Utility allocations under *competitive* (left) and *cooperative* (right) scenarios.

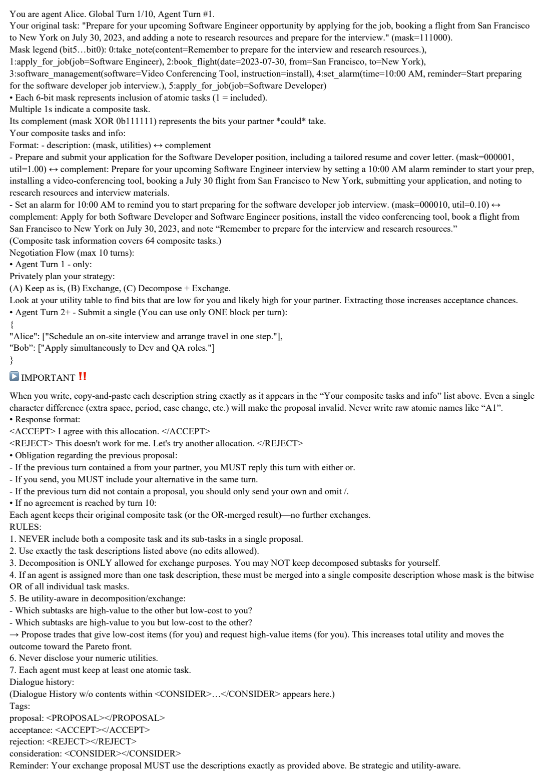
A.4 Messages

We constrain agents to an XML-like messaging format as used in Bianchi et al. (2024) to improve mutual understanding. Agents use the tags <PROPOSAL>...</PROPOSAL>, <CONSIDER>...</CONSIDER>, <ACCEPT>...</ACCEPT>, and <REJECT>...</REJECT>. Contents inside tags except for <CONSIDER> tags are passed to the next agent as dialogue history. Agreement is reached when all non-proposers react with <ACCEPT> for the most recent valid proposal.

Each turn, we validate the current proposal; if it is invalid, we record the invalid proposal pattern and ask the proposer to retry with targeted feedback.

A.5 Prompt

At each turn, each agent receives a prompt to make a proposal. The prompt includes: (i) the set of composite tasks and, for each, its complementary task; (ii) the agents' utility tables over these tasks; (iii) the negotiation rules; and (iv) the dialogue history (i.e., the negotiation history excluding content inside <CONSIDER>...</CONSIDER> tags). Because negotiation often involves rejection, agents are instructed to remain utility-aware of the counterpart's preferences. The disagreement fallback (reverting to the initial main tasks) is also specified in the prompt. Figure 7 shows an example prompt provided to Alice in Turn 1. In total there are 64 (2^6) composite tasks, each with a complementary task; for readability, Figure 7 shows only two and omits the rest.



```
You are agent Alice. Global Turn 1/10, Agent Turn #1.
Your original task: "Prepare for your upcoming Software Engineer opportunity by applying for the job, booking a flight from San Francisco
to New York on July 30, 2023, and adding a note to research resources and prepare for the interview." (mask=111000).
Mask legend (bit5…bit0): 0:take_note(content=Remember to prepare for the interview and research resources.),
1:apply_for_job(job=Software Engineer), 2:book_flight(date=2023-07-30, from=San Francisco, to=New York),
3:software_management(software=Video Conferencing Tool, instruction=install), 4:set_alarm(time=10:00 AM, reminder=Start preparing
for the software developer job interview.), 5:apply_for_job(job=Software Developer)
• Each 6-bit mask represents inclusion of atomic tasks (1 = included).
Multiple 1s indicate a composite task.
Its complement (mask XOR 0b111111) represents the bits your partner *could* take.
Your composite tasks and info:
Format: - description: (mask, utilities) ↔ complement
- Prepare and submit your application for the Software Developer position, including a tailored resume and cover letter. (mask=000001,
util=1.00) ↔ complement: Prepare for your upcoming Software Engineer interview by setting a 10:00 AM alarm reminder to start your prep,
installing a video-conferencing tool, booking a July 30 flight from San Francisco to New York, submitting your application, and noting to
research resources and interview materials.
- Set an alarm for 10:00 AM to remind you to start preparing for the software developer job interview. (mask=000010, util=0.10) ↔
complement: Apply for both Software Developer and Software Engineer positions, install the video conferencing tool, book a flight from
San Francisco to New York on July 30, 2023, and note "Remember to prepare for the interview and research resources."
(Composite task information covers 64 composite tasks.)
Negotiation Flow (max 10 turns):
• Agent Turn 1 - only:
Privately plan your strategy:
(A) Keep as is, (B) Exchange, (C) Decompose + Exchange.
Look at your utility table to find bits that are low for you and likely high for your partner. Extracting those increases acceptance chances.
• Agent Turn 2+ - Submit a single (You can use only ONE block per turn):
{
"Alice": ["Schedule an on-site interview and arrange travel in one step."],
"Bob": ["Apply simultaneously to Dev and QA roles."]
}
▶️ IMPORTANT ‼️
When you write, copy-and-paste each description string exactly as it appears in the "Your composite tasks and info" list above. Even a single
character difference (extra space, period, case change, etc.) will make the proposal invalid. Never write raw atomic names like "A1".
• Response format:
<ACCEPT> I agree with this allocation. </ACCEPT>
<REJECT> This doesn't work for me. Let's try another allocation. </REJECT>
• Obligation regarding the previous proposal:
- If the previous turn contained a from your partner, you MUST reply this turn with either or.
- If you send, you MUST include your alternative in the same turn.
- If the previous turn did not contain a proposal, you should only send your own and omit /.
• If no agreement is reached by turn 10:
Each agent keeps their original composite task (or the OR-merged result)—no further exchanges.
RULES:
1. NEVER include both a composite task and its sub-tasks in a single proposal.
2. Use exactly the task descriptions listed above (no edits allowed).
3. Decomposition is ONLY allowed for exchange purposes. You may NOT keep decomposed subtasks for yourself.
4. If an agent is assigned more than one task description, these must be merged into a single composite description whose mask is the bitwise
OR of all individual task masks.
5. Be utility-aware in decomposition/exchange:
- Which subtasks are high-value to the other but low-cost to you?
- Which subtasks are high-value to you but low-cost to the other?
→ Propose trades that give low-cost items (for you) and request high-value items (for you). This increases total utility and moves the
outcome toward the Pareto front.
6. Never disclose your numeric utilities.
7. Each agent must keep at least one atomic task.
Dialogue history:
(Dialogue History w/o contents within <CONSIDER>…</CONSIDER> appears here.)
Tags:
proposal: <PROPOSAL></PROPOSAL>
acceptance: <ACCEPT></ACCEPT>
rejection: <REJECT></REJECT>
consideration: <CONSIDER></CONSIDER>
Reminder: Your exchange proposal MUST use the descriptions exactly as provided above. Be strategic and utility-aware.
```

Figure 7: Example prompt provided to the agent each turn under the utility-aware strategy.

B Negotiation Strategies

We prepare two negotiation strategies—*utility-aware strategy* and a *baseline* strategy. We compare our utility-aware strategy against a baseline with two modifications:

1. **Remove prompt about the utility-aware strategy**: delete prompt text that encourages or instructs agents to adopt a utility-aware strategy.
2. **No per-turn validation**: omit proposal-validity checks during intermediate turns; retain the final-turn check.

B.1 Prompt difference between the two strategies

In the utility-aware strategy, agents are prompted via Rule 5 in Figure 7 to act in a utility-aware manner as follows:

```
Be utility-aware in decomposition/
exchange:
- Which subtasks are high-value to the
  other but low-cost to you?
- Which subtasks are high-value to you
  but low-cost to the other?
→ Propose trades that give low-cost
  items (for you) and request high-value
  items (for you). This increases total
  utility and moves the outcome
  toward the Pareto front.
```

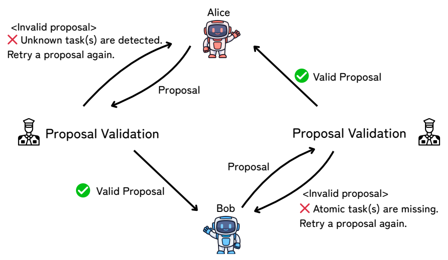


Figure 8: Per–turn proposal validation pipeline.

In the baseline setting, we remove the above guidance from the prompt in Figure 7. This is the only prompt-level difference between the two strategies.

B.2 Proposal Validation

Under the utility-aware strategy, every outbound proposal is validated *before* being delivered to the counterpart (Figure 8). Only validated proposals are forwarded. If a proposal is judged invalid, it is returned to the proposer with targeted feedback; the proposer may revise and resubmit within the same turn without advancing the turn counter.

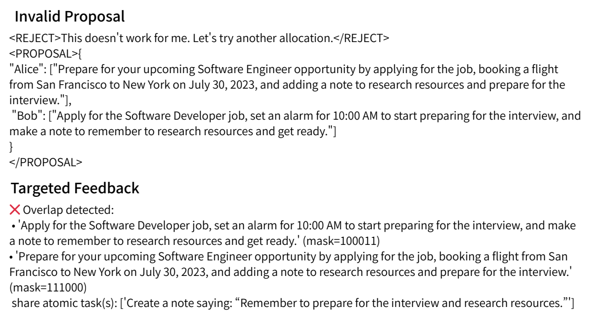
We categorize invalid proposals into six patterns:

- **Missing `<PROPOSAL>` tags**: the JSON-like proposal is not enclosed in `<PROPOSAL>... </PROPOSAL>`.
- **Unknown tasks**: the proposal references composite tasks that are not in the registry.
- **Multiple descriptions per agent**: more than one composite description is assigned to the same agent in a single turn.
- **Overlap**: the same composite task is assigned to multiple agents simultaneously.
- **Missing atomic tasks**: some atomic tasks are left unassigned (i.e., the full set is not covered).
- **Empty agent**: one agent receives all tasks while the other receives none.

When any of these patterns is detected, the feedback specifies the reason and, when applicable, includes the relevant bit masks for the *composite tasks* and the overlapping atomic task(s). Figure 9 shows a typical **Overlap** error and the corresponding feedback. We cap per–turn retries at K (typically $K{=}5$). If an agent exceeds this limit within a single turn, the negotiation terminates with status `Error`. The retry counter resets when control passes to the other agent.

C More Analysis

Since the utility-aware strategy exhibits its clearest effects with `o4-mini`, we restrict our analysis to this model.



Invalid Proposal

```
<REJECT>This doesn't work for me. Let's try another allocation.</REJECT>
<PROPOSAL>{
"Alice": ["Prepare for your upcoming Software Engineer opportunity by applying for the job, booking a flight
from San Francisco to New York on July 30, 2023, and adding a note to research resources and prepare for the
interview."],
 "Bob": ["Apply for the Software Developer job, set an alarm for 10:00 AM to start preparing for the interview, and
make a note to remember to research resources and get ready."]
}
</PROPOSAL>
```

Targeted Feedback

```
❌ Overlap detected:
 • 'Apply for the Software Developer job, set an alarm for 10:00 AM to start preparing for the interview, and make
a note to remember to research resources and get ready.' (mask=100011)
• 'Prepare for your upcoming Software Engineer opportunity by applying for the job, booking a flight from San
Francisco to New York on July 30, 2023, and adding a note to research resources and prepare for the interview.'
(mask=111000)
 share atomic task(s): ['Create a note saying: "Remember to prepare for the interview and research resources."']
```

Figure 9: Example of an invalid proposal (**Overlap**) and the targeted feedback returned to the proposer.

C.1 Invalid proposal patterns

As summarized in Table 2, we report invalid–proposal counts across models and settings. For `o4-mini` under the utility-aware setting in the *cooperative* scenario, four categories appear in equal numbers: *Missing <PROPOSAL> tag* (4), *Missing atomic task(s)* (4), *Unknown task(s)* (4), and *Overlap detected* (4). In the *competitive* scenario, *Missing atomic task(s)* dominates (12), followed by *Unknown task(s)* (5), *Missing <PROPOSAL> tag* (3), *Overlap detected* (3), and *Missing </PROPOSAL> tag* (1). (Counts are totals across the 10 runs per condition.) Because the utility-aware setting validates every turn whereas the baseline performs only a final check, direct pattern-level comparison between the two strategies is not very informative; we therefore concentrate on the utility-aware setting.

A similar shift holds for `GPT-4.1-mini`: the share of *Missing atomic task(s)* increases markedly from cooperative to competitive, while other categories remain comparatively stable. Concretely, the *Missing atomic task(s)* share rises from $4/42 = 9.52\%$ to $12/52 = 23.08\%$ for `o4-mini`, and from $4/81 = 4.94\%$ to $17/106 = 16.04\%$ for `GPT-4.1-mini`; other categories (e.g., *Unknown task(s)*, *Overlap*, *Missing <PROPOSAL> tag*) change little by comparison.

C.2 Pareto proximity

We report the mean distance from the Pareto front over valid runs. Closer final utility vectors indicate more *Pareto-efficient* outcomes. In the *cooperative* setting, the baseline is closer on average (0.21) than the utility-aware strategy (0.27), though both improve on the initial distance (0.32). In the *competitive* setting, the utility-aware strategy is closer (0.022) than the baseline (0.028), which matches the initial distance (0.028). Taken together, the utility-aware approach improves Pareto proximity in the competitive setting but not in the cooperative setting.

C.3 Turns to termination

Considering only valid runs, with `o4-mini` the utility-aware strategy resolves negotiations in fewer turns than the baseline in both scenarios: 3.00 vs. 4.83 in the cooperative case and 3.22 vs. 6.80 in the competitive case. Under the utility-aware strategy, agents compromise more readily, and negotiations converge faster.

Table 2: Invalid–proposal patterns only. Columns use abbreviations: **UNK** = Unknown task(s), **MAT** = Missing atomic task(s), **OVL** = Overlap detected, **MPT** = Missing `<PROPOSAL>` tag, **MET** = Missing `</PROPOSAL>` tag, **MDA** = Multiple descriptions per agent, **EMP** = Empty agent. Each numeric cell shows *count* and, in parentheses, the *percentage of proposal attempts* for that condition. "Attempts" is the total number of proposal attempts (turns + invalid proposals). A dash (—) indicates attempts were not logged, so percentages are omitted.

(a) Utility-aware strategy (UAS)

Model	**Scen.**	**Attempts**	**UNK**	**MAT**	**OVL**	**MPT**	**MET**	**MDA**	**EMP**
`o4-mini`	Coop.	42	4 (9.52%)	4 (9.52%)	4 (9.52%)	4 (9.52%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	Comp.	52	5 (9.62%)	12 (23.08%)	3 (5.77%)	3 (5.77%)	1 (1.92%)	0 (0.00%)	0 (0.00%)
`GPT-4.1-mini`	Coop.	81	29 (35.80%)	4 (4.94%)	23 (28.40%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	Comp.	106	38 (35.85%)	17 (16.04%)	23 (21.70%)	3 (2.83%)	0 (0.00%)	2 (1.89%)	0 (0.00%)
`GPT-4.1-nano`	Coop.	54	43 (79.63%)	1 (1.85%)	0 (0.00%)	6 (11.11%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	Comp.	42	29 (69.05%)	1 (2.38%)	2 (4.76%)	11 (26.19%)	7 (16.67%)	0 (0.00%)	0 (0.00%)

(b) Baseline

Model	**Scen.**	**Attempts**	**UNK**	**MAT**	**OVL**	**MPT**	**MET**	**MDA**	**EMP**
`o4-mini`	Coop.	49	0 (0.00%)	3 (6.12%)	1 (2.04%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	Comp.	—	2 (—)	3 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)
`GPT-4.1-mini`	Coop.	—	3 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)
	Comp.	—	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)
`GPT-4.1-nano`	Coop.	—	5 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)
	Comp.	—	4 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)	0 (—)