TimeArena: Shaping Efficient Multitasking Language Agents in a Time-Aware Simulation

Yikai Zhang♠, **Siyu Yuan**♢, **Caiyu Hu**♠,
Kyle Richardson♡, **Yanghua Xiao**♠*, **Jiangjie Chen**♠*
♠Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
♢School of Data Science, Fudan University
♡Allen Institute for AI
{ykzhang22,syyuan21,cyhu24}@m.fudan.edu.cn,
kyler@allenai.org, {shawyh,jjchen19}@fudan.edu.cn

Abstract

Despite remarkable advancements in emulating human-like behavior through Large Language Models (LLMs), current textual simulations do not adequately address the notion of time. To this end, we introduce TimeArena, a novel textual simulated environment that incorporates complex temporal dynamics and constraints that better reflect real-life planning scenarios. In TimeArena, agents are asked to complete multiple tasks as soon as possible, allowing for parallel processing to save time. We implement the dependency between actions, the time duration for each action, and the occupancy of the agent and the objects in the environment. TimeArena grounds to 30 real-world tasks in cooking, household activity, and laboratory work. We conduct extensive experiments with various LLMs using TimeArena. Our findings reveal that even the most powerful models, *e.g.*, GPT-4, still lag behind humans in effective multitasking, underscoring the need for enhanced temporal awareness in the development of language agents. [1]

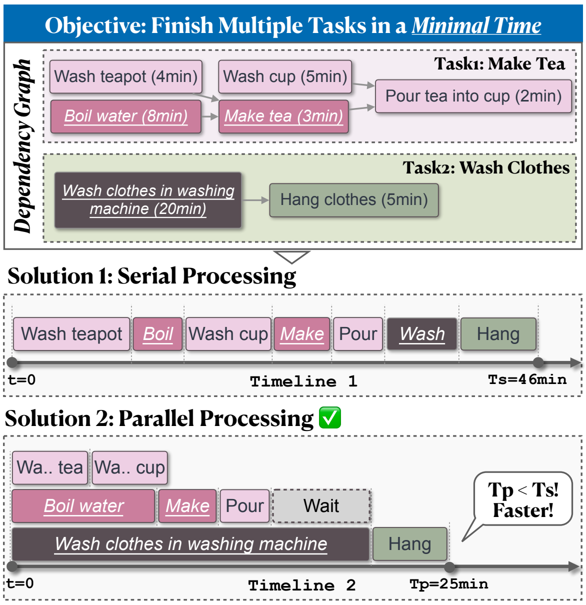


Figure 1: An example illustrating multitasking with temporal constraints in TimeArena. The completion of tasks requires actions in a predetermined dependency and order. Underlined actions do not occupy the agent, allowing other actions to be processed by the agent simultaneously. The Wait action skips the current time step, meaning the agent is idle.

1 Introduction

Large language models (LLMs) (OpenAI, 2022, 2023; Team and Google, 2023) have enabled the development of language agents (*a.k.a.* LLM-based agents), which aim to simulate human behaviors in real-world scenarios through their planning capabilities (Liu et al., 2023; Gong et al., 2023; Akata et al., 2023). However, planning in the real world involves temporal and resource constraints (Russell and Norvig, 2010), which are rarely implemented in most textual simulations for LLMs and language agents (Wang et al., 2022; Park et al., 2023).

The integration of time in simulated environments challenges agents to navigate and align with human-like efficient multitasking skills. Such a simulation requires the agent to consider the following three factors: 1) **Time Duration and Dependency:** Actions will have durations upon dependencies, requiring agents to strategize and prioritize based on time constraints and task completion progress. 2) **Agent Occupancy:** Agents will be occupied by certain actions; thus, they might be unable to perform other actions at the same time. 3) **Object Occupancy:** Some objects might be occupied for some time, and agents must use available objects in the environment for the tasks. These factors are common in real life but are seldom addressed by current textual simulations.

To help illustrate, Figure 1 shows an example of completing *make tea* (Task 1) and *wash clothes* (Task 2). The actions of each task might depend on

*Corresponding authors.
[1]Project page: https://time-arena.github.io.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3894–3916
August 11-16, 2024 ©2024 Association for Computational Linguistics

previous actions, *e.g.*, agents must `boil water` before `make tea`, and each action takes a duration in time, *e.g.*, `wash cup` takes 5 minutes. In particular, some actions let agents be idle, allowing agents to carry out other actions. For example, `wash clothes in washing machine` allows agents to perform other actions at the same time. Moreover, actions temporarily occupy objects, making them unavailable for other actions and hindering parallel processing. For example, `boil water` occupies the `pot`, delaying other actions like `cook soup` until it is available. When no action is currently available for the agent, the only option is to wait. For example, in Solution 2, the agent must `wait` for the completion of `wash clothes in washing machine`, before `hang clothes`.

In this work, we introduce TIMEARENA, a textual simulated environment featuring 30 real-world tasks involving cooking, household activity, and laboratory work. TIMEARENA is the first textual simulation to evaluate language agents on multitasking efficiency. Specifically, we incorporate the time duration of each action and set two types of actions based on agent occupancy. One type occupies agents (*e.g.*, `wash cup`) and another lets agents be idle (*e.g.*, `boil water`). Additionally, we simulate resource competition by implementing object occupancy, i.e., an object used for one task cannot be simultaneously used for another, which is common in parallel processing. Therefore, agents must focus on parallel processing, taking into account the occupancy of agents and objects, to minimize time consumption. We design four metrics in TIMEARENA to evaluate the average progress score, completion speed, task completion rate and average completion time. These metrics help to assess and analyze the efficient multitasking capabilities of language agents. Our comprehensive evaluation of seven LLMs on TIMEARENA shows that current language agents struggle in efficient multitasking. Even the most powerful LLM, GPT-4, still faces challenges in parallel processing.

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first to explore the notion of time of language agents in a textual environment, which is important for more realistic simulation.
- We create TIMEARENA, a novel text-based simulated environment consisting of 30 tasks, where LLMs can complete multiple tasks in parallel.
- Using TIMEARENA, we conduct rich experiments to evaluate the efficient multitasking capabilities of language agents. Our results demonstrate that efficient multitasking in TIMEARENA poses a significant challenge for current language agents.

2 Related Work

Simulation-based Evaluation for Language Agents With the great success of LLMs (OpenAI, 2022, 2023; Team and Google, 2023), recent works have shifted the focus from traditional NLP tasks to explore language agents in simulated environments that mimic real-world scenarios (Wu et al., 2023; Liu et al., 2023; Gong et al., 2023; Akata et al., 2023). These simulated environments can be divided into two categories: 1) Social Simulations (Park et al., 2023; Mukobi et al., 2023; Zhou et al., 2023), which aim to evaluate the behaviors of language agents in some social scenarios; 2) Problem-solving simulations, which are created based on games (Chen et al., 2023a,b; Zhang et al., 2023; Agashe et al., 2023) and scientific scenarios (Wang et al., 2022). In this paper, we focus on problem-solving simulations to investigate the efficient multitasking capabilities of language agents.

Language Planning Language planning aims to decompose a complex task into steps (Schank and Abelson, 1975, 2013). Early studies mainly focus on imbuing language models with planning capabilities by training them on specific planning datasets (Peng et al., 2018; Hua et al., 2019; Kong et al., 2021), which exhibits poor generalization. Recent studies have identified that LLMs can effectively decompose tasks into procedural steps (Wang et al., 2023c; Yuan et al., 2023; Shen et al., 2023). However, multitasking planning with parallel processing in dynamic environments still remains under-studied.

Temporal Reasoning Temporal reasoning involves comprehending, structuring, and interpreting events, actions, and states through the lens of time (Allen, 1991; Vila, 1994; Stock, 1998). Previous studies in temporal reasoning focus on temporal relation extraction (Vashishtha et al., 2019; Mathur et al., 2021; Wang et al., 2023b), event temporal reasoning (Mathur et al., 2022; Yang et al., 2023; Wang and Zhao, 2023) and explore the temporal reasoning capability of LLMs with several contemporary time-sensitive QA datasets (Zhang

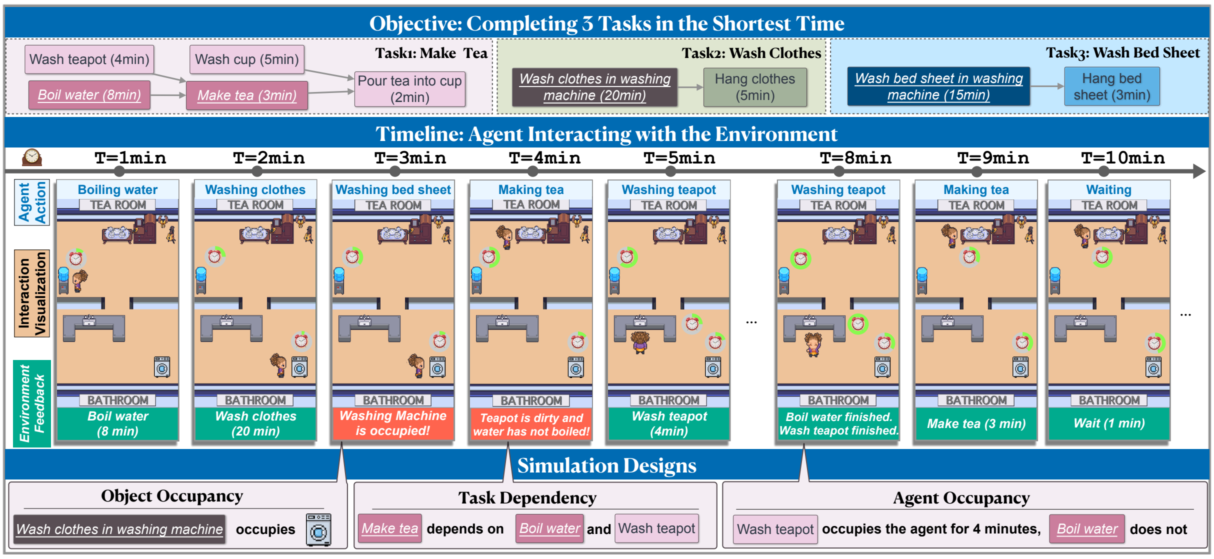


Figure 2: An overview of TIMEARENA, with a multitasking example that shows our designs of the simulation. TIMEARENA first sets an objective for the agent, and then the agent interacts with TIMEARENA over time, with the design of action dependency, object occupancy, and agent occupancy.

and Choi, 2021; Shang et al., 2022; Tan et al., 2023). Distinguished from other benchmarks (Chu et al., 2023), our TIMEARENA creates a dynamic and interactive simulated environment.

3 TIMEARENA

We create TIMEARENA, a textual simulated environment to evaluate the efficient multitasking capabilities of language agents. To help illustrate, we first show an overview and an example run of how an agent interacts with the TIMEARENA environment (§ 3.1), and then describe the design of the simulation environment in more detail (§ 3.2 and § 3.3).

3.1 Overview of TIMEARENA

TIMEARENA challenges agents to complete multiple tasks strategically in the shortest possible time. This simulation emphasizes the importance of understanding, performing, and optimizing actions within a constrained timeframe, mirroring practical scenarios involving time management.

Central to TIMEARENA are **Tasks**, **Objects**, and **Actions**. **Tasks** define the objectives for the agents, **Objects** represent elements in the environment that agents will encounter and interact with, and **Actions** are the means to accomplish these tasks. Real-time feedback and scoring mechanisms are integral to the environment, assessing agent performance and adding to the simulation's complexity and realism. Unique features like the duration and occupancy of actions and strategic resource utilization distinguish TIMEARENA from other environments.

An Example Run As in Figure 2, consider an agent tasked with *make tea* (Task 1), *wash clothes* (Task 2) and *wash bed sheet* (Task 3). The agent starts by decomposing the task into actions like `boil water`. In TIMEARENA, all actions have a duration (*e.g.*, `Boil water` needs 8 minutes) and dependencies. (*e.g.*, At T=4min, `make tea` violates the dependency because `wash teapot` and `boil water` are not completed yet.) The agent then interacts with objects (*e.g.*, `wash clothes in washing machine`), which become **occupied** during the process. The agent can engage in non-occupied actions simultaneously (*e.g.*, `wash teapot`) while others (*e.g.*, `boil water`) are in progress. Environmental feedback guides the agent, indicating the legitimacy of actions and the completion of tasks. For example, if the `washing machine` is occupied, the agent adjusts its strategy. The agent's goal is to complete all the tasks efficiently, with performance evaluated based on progress and completion time.

This dynamic interaction in TIMEARENA fosters an environment where strategic planning, resource management, and adaptability are key to an agent's success.

3.2 Components of TIMEARENA

Tasks In TIMEARENA, we design tasks within three distinct *scenarios* or simulated settings, namely, *household activity*, *cooking*, and *labora-*

tory work. Each scenario represents a specific context or environment where multitasking is an integral part of the activities involved.[2] For example, one can do `sweep floor` while doing `boil water`. Each scenario contains 10 tasks, and some actions and objects are shared across multiple tasks of a scenario. Each task requires multiple actions to be executed, which manipulates the objects in the environment for task completion. In the beginning, TIMEARENA gives a list of tasks to the agent, with a comprehensive task instruction consisting of a *task description*, an *action space*, and an *object set*:

- **Task Description**: Introduces task objectives, *e.g.*, *Make a dish of beef fried rice, which consists of cooked rice and fried beef.*
- **Action Space**: Lists the valid actions for the tasks (*e.g.*, `chop`, `wash`).
- **Object Set**: Lists the available objects in the environment for the tasks (*e.g.*, `pot`, `beaker`).

At every timestep t, the agent needs to generate valid actions on the objects and receive feedback from the environment.

Objects Objects are integral to completing tasks and situating within the environment. In TIMEARENA, there are 71 different objects for all the tasks. Every task involves a list of objects, which might overlap with other tasks of the same scenario. To mimic the resource limitation in real-world parallel processing, we introduce:

- **Object Occupancy**: the state of the object involved in an action is set to be **occupied**, *e.g.*, `wash cup` will cause the object `cup` to be occupied. This object cannot be processed until the involved action is completed (after some time). Then, this object is reset as non-occupied and waits for another action.

Actions We design a total of 46 actions for all 30 tasks. Each action consists of a detailed description (*e.g.*, `chop OBJ`, `chop the whole item into sliced pieces.`), showing a change of states the action will cause to an object.[3] Different from existing text-based simulations (Wang et al., 2022; Gong et al., 2023; Shridhar et al., 2020), in our case, an action has a duration of time and may occupy the agent from performing other actions, to the passage of time. In detail:

- **Action Dependency**: An action within the same task might depend on completing other actions within the same task. In Figure 2, `make tea` is dependent on `wash teapot`.
- **Duration of Time**: Each action holds a timeframe in the timeline, ranging from 1 to 10 minutes. In practice, agents only have an educated guess of the time duration of each action until actually interacting with TIMEARENA.
- **Agent Occupancy**: One key to parallel processing is agent occupancy, which prevents agents from performing other tasks. Therefore, we consider two types of actions based on agent occupancy: Type 1 action occupies the agent until completion (*e.g.*, `wash teapot`), and Type 2 action lets agents be idle, allowing them to perform other actions at the same time (*e.g.*, `boil water`).

3.3 Interaction between Agent and Environment

Environmental Feedback The feedback from a textual environment is important to simulate and implement the constraints in TIMEARENA using only textual messages. We define feedback as the response from the environment following an action by an agent. A feedback message could be of multiple types, including:

- **Invalid Action**: An action attempt that does not match the required format, *e.g.*, `clean teapot` *is invalid*.
- **Action on Non-existing Object**: An action attempt that visits objects that are not in the object set, *e.g.*, `pan` *is non-existent*.
- **Wrong Action Input**: An action attempt that the prerequisite action has not been completed (*e.g.*, *Cannot perform action* `add to` *on object* `shrimp`. *Because* `shrimp` *is* `raw`.) or has been completed (*e.g.*, `wash beaker` *has been completed*).
- **Action on Mismatched Object**: An action attempt that does not match the object, *e.g.*, *You cannot perform* `read` *on* `potato`.
- **Action on Occupied Object**: An action attempt on occupied objects, *e.g.*, *Object* `pot` *is being occupied by another action.*

Correspondingly, valid actions will trigger environmental feedback of the following types:

- **Action Start**: Avoiding previous errors, valid actions will receive a feedback message containing the specific performing time, marking

[2] Details of tasks are in Appendix A.1.
[3] All the actions are listed in Appendix A.2.

Scenario	# Actions	# Objects	Time (min)
Cooking	5.6	5.5	18.9
Household Activity	4.1	3.5	12.8
Laboratory Work	5.3	2.7	16.1

Table 1: Average number of actions and objects per task in each scenario, and the average shortest completion time for these tasks.

the start of the action, *e.g.*, *You are doing* `wash cup`*, it will take 9 minutes*.

- **Action Completion**: When an action is completed, the environment will send a message, *e.g.*, `cup is clean`, and reset the occupancy state of the object (`cup`).

Progress Score The progress score, denoted as a percentage, reflects the agent's completion rate of required actions within the environment, where the total duration for all actions is considered as 100%. Each action's contribution to the progress score is proportionate to its duration. Specifically, if an action's duration is t_i minutes, its contribution to the progress score is calculated as $s_i = \left(\frac{t_i}{\sum_{j=1}^{n} t_j}\right) \times 100\%$, with n representing the total number of actions. For instance, an action lasting 5 minutes in a total action duration of 20 minutes contributes 25% to the progress score.

4 Experiments

4.1 Experiment Settings

Task Set Construction In our experiments, we design three categories of task combinations based on the number of tasks: **# Task=1**, **# Task=2** and **# Task=3** scenarios. In # Task=1 scenario, agents focus on completing one task (*e.g.*, `make tea`). For the other two scenarios, we combine either two or three tasks from 10 single tasks (*e.g.*, `make tea` and `wash clothes`). Then, we randomly select 10 combined tasks for each scenario.[4]

Interaction Initially, the environment provides a comprehensive task instruction that details the task, action space, and object set. Subsequently, the agent produces an action based on this instruction, adhering to a prescribed format specified in the action space; any deviation is considered invalid. To facilitate action recognition by the environment, regular expressions are employed to parse actions from responses (*e.g.*, extracting `wash clothes` from *I will wash clothes*). For each action execution, the agent must incorporate task instructions, previous actions, and feedback from the environment into LLMs as context.[5]

Maximum Time Each combined task is allocated a maximum completion time. We set the time limit for completing a single task at 40 minutes, which exceeds the total time required for all actions in any given task. For tasks that are combined, the time limit is proportionally increased by the number of tasks involved.

Oracle Performance As shown in Table 2, *Oracle* represents the optimal performance, including the shortest completion time and the fastest completion rate, which are manually calculated. Specifically, we calculate oracle performance using a greedy-like strategy: always start the non-occupied actions as early as possible and avoid idleness when there are actions to perform.[6]

Finishing The interaction finishes under any of the following conditions: 1) Agents have completed all the actions that solve the tasks (*i.e.*, the progress score reaches 100%); 2) Time has run out; 3) Agents who have performed incorrect actions 5 times in a row are considered to fail the task.

Model Choice We employ a diverse set of language models for the agent, including Mistral-7B by MistralAI (Jiang et al., 2023), OpenChat-3.5 fine-tuned from Mistral's 7B model (Wang et al., 2023a), Vicuna-13B fine-tuned from LLaMA's 13B model with instructions (Chiang et al., 2023), Mixtral-8x7B, a Mixture-of-Expert version of Mistral (Mistral AI team, 2023), Google's Gemini Pro (Team et al., 2023), OpenAI's GPT-3.5 (`gpt-3.5-turbo-1106`) (OpenAI, 2022), and GPT-4 (`gpt-4-1106-preview`) (OpenAI, 2023). We employ greedy decoding for all the models with the temperature set to 0.

4.2 Evaluation Metrics

To comprehensively evaluate the efficient multi-tasking ability of agents, we consider both time and score and design the following four metrics:

- **Average Progress Score (score, AS)**: The average highest progress score achievable by

[4] Appendix B.1 shows examples of single and combined tasks.

[5] Appendix B.2 gives an example of interaction between the agent and the environment.

[6] Appendix A.4 shows our algorithm for calculating the oracle performance.

	Model	# Task=1				# Task=2				# Task=3			
		AS ↑	CS ↑	CR ↑	CT ↓	AS ↑	CS ↑	CR ↑	CT ↓	AS ↑	CS ↑	CR ↑	CT ↓
Cooking	Mistral-7B	63.70	3.59	30.00	25.67	42.20	1.49	0	-	39.40	1.06	0	-
	OpenChat-3.5	76.30	3.89	30.00	20.33	37.10	1.80	0	-	41.00	1.17	0	-
	Vicuna-13B	84.60	**4.10**	60.00	21.83	48.80	1.76	0	-	26.00	1.03	0	-
	Mixtral-8x7B	50.80	3.81	10.00	**19.00**	40.10	1.99	0	-	27.60	1.17	0	-
	Gemini Pro	78.30	3.57	50.00	24.60	31.00	1.75	0	-	18.50	1.26	0	-
	GPT-3.5	77.70	3.61	30.00	24.33	52.30	1.87	0	-	33.10	1.23	0	-
	GPT-4	**98.70**	3.48	**90.00**	28.22	**93.50**	1.83	**70.00**	52.57	**82.50**	1.21	**40.00**	**76.25**
	+ ReAct	95.00	4.01	**90.00**	24.33	81.90	**2.17**	60.00	45.33	51.60	**1.50**	0	-
	+ Reflxion	79.50	3.02	30.00	28.00	52.70	1.26	0	-	36.30	0.87	0	-
	+ Self-plan	89.00	3.83	60.00	26.50	64.90	2.05	10.00	**37.00**	26.20	1.15	0	-
	Oracle	100	5.31	100	18.90	100	2.85	100	35.00	100	1.94	100	52.50
Household Activity	Mistral-7B	64.80	6.00	20.00	15.50	45.30	2.46	0	-	49.90	1.78	0	-
	OpenChat-3.5	70.50	5.34	30.00	15.67	68.20	2.73	0	-	44.30	1.83	0	-
	Vicuna-13B	69.50	5.94	40.00	**14.25**	45.90	2.34	0	-	24.90	1.69	0	-
	Mixtral-8x7B	68.80	6.08	40.00	15.00	51.60	2.85	10.0	31.00	60.20	1.83	10.00	58.00
	Gemini Pro	68.10	5.92	40.00	16.50	60.50	3.02	10.00	**25.00**	40.30	1.93	0	-
	GPT-3.5	87.40	5.98	70.00	16.71	63.80	2.57	10.00	36.00	45.30	1.82	0	-
	GPT-4	**100**	5.81	**100**	17.20	**100**	2.89	**100**	34.50	**98.40**	1.82	**90.00**	54.78
	+ ReAct	**100**	**6.45**	100	15.50	99.10	**3.48**	90.00	28.56	**98.40**	**2.21**	80.00	**46.50**
	+ Reflxion	90.00	5.52	60.00	18.17	87.20	1.95	50.00	52.20	78.50	1.41	0	-
	+ Self-plan	87.20	6.01	80.00	16.37	84.50	2.80	50.00	35.20	95.30	1.93	60.00	50.16
	Oracle	100	7.81	100	12.80	100	4.23	100	23.60	100	2.82	100	35.40
Laboratory Work	Mistral-7B	70.80	4.39	30.00	21.67	47.10	2.27	0	-	38.40	1.37	0	-
	OpenChat-3.5	65.50	5.07	30.00	**13.33**	45.80	2.10	0	-	27.50	1.30	0	-
	Vicuna-13B	59.60	3.94	20.00	26.00	20.80	1.87	0	-	22.90	1.40	0	-
	Mixtral-8x7B	64.10	4.57	40.00	24.25	41.80	2.43	0	-	32.40	1.58	0	-
	Gemini Pro	88.00	5.17	70.00	19.57	57.50	2.64	20.00	**35.50**	25.70	1.61	0	-
	GPT-3.5	71.50	4.52	30.00	22.00	47.60	2.17	0	-	37.90	1.52	0	-
	GPT-4	**97.50**	5.32	**90.00**	18.67	85.30	2.61	50.00	39.20	83.10	1.71	**60.00**	60.33
	+ ReAct	**97.50**	**5.51**	**90.00**	18.00	**91.80**	2.68	**70.00**	38.71	**93.50**	**1.96**	40.00	**49.00**
	+ Reflxion	94.70	4.86	70.00	18.86	75.90	2.11	40.00	51.50	88.40	1.30	40.00	86.00
	+ Self-plan	95.30	5.09	80.00	20.12	83.00	**2.79**	50.00	36.40	70.00	1.87	**60.00**	54.66
	Oracle	100	6.21	100	16.10	100	4.14	100	24.60	100	2.84	100	35.50

Table 2: Model performance under different task combination settings in TIMEARENA. We report Average Progress Score (**AS**), Completion Speed (**CS**), Task Completion Rate (**CR**), and Average Completion Time (**CT**). **#Task=n** represents that agents are required to do n tasks altogether. We also list the Oracle result for comparison. The best results are **bolded**, and the second best ones are underlined.

an agent, calculated as: $AS = \left(\frac{\sum_{i \in N} P_i}{N}\right)$, where P_i denotes the maximum progress score of i-th task that agents can reach, and N denotes the number of all the tasks.

- **Completion Speed (score per minute, CS)**: The average of the highest score divided by the time taken to achieve it, calculated as: $CS = \left(\frac{\sum_{i \in N} P_i}{\sum_{i \in N} T_i}\right)$, where T_i denotes the time required to reach P_i of i-th task.

- **Task Completion Rate (%, CR)**: The rate of successfully completed tasks, calculated as: $CR = \left(\frac{S}{N}\right)$, where S denotes the number of tasks completed successfully. Notably, when combining tasks, a combined task counts as one task.

- **Average Completion Time (minutes, CT)**: The average time taken for completing tasks successfully: $CT = \left(\frac{\sum_{i \in S} T_i}{S}\right)$.

4.3 Main Results

As shown in Table 2, GPT-4 achieves the best performance across different task combinations. Moreover, the combined tasks are more challenging than single tasks despite the longer time given. Apart from GPT-4, most models fail to complete 2 or 3 tasks, showing their limited multitasking abilities and the challenging nature of our environment.

For open-source models, OpenChat-3.5 and Vicuna-13B are even better than GPT-3.5, demon-

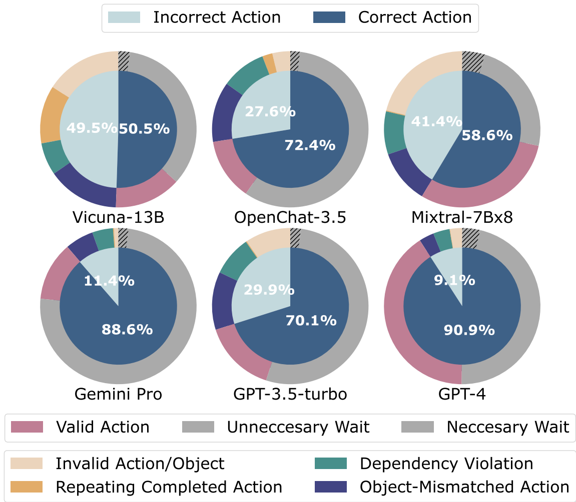


Figure 3: The proportions of correct and incorrect actions for each language agent.

Figure 4: Comparison of the performance of GPT-4 with and without resource constraints. We impose constraints by limiting to a single instance each of `pot`, `fryer`, and `oven`.

strating the potential of open-sourced models to develop multitasking capabilities. However, a lower task completion rate and higher completion speed indicate that these models quickly complete simple actions initially but then encounter difficulties. They either get caught in repetitive actions or fail to properly segment subsequent tasks, which significantly impacts task performance. For example, initially, `potato` is unpicked, so the agent first performs `pick potato`. Subsequently, the agent mistakenly opts for `cook potato in pot` rather than the correct `chop potato`, because it incorrectly decomposes the task.

To further examine whether some prompting methods benefit agents' performance, we adopt ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023) to GPT-4.[7] As shown in Table 2, the performance of ReAct is similar to vanilla GPT-4. We find that the model still struggles to decide when `wait` is unnecessary (*i.e.*, there are other available actions), which would allow for parallel processing. This leads to less efficient execution compared to oracle performance. As the number of tasks grows, the Reflexion prompting method degrades model performance. With more complex tasks, the lessons learned from history become less accurate due to complex action interdependencies, leading to incorrect actions. We also explore the potential of heuristic algorithms in improving model performance. Inspired by Khot et al. (2022), we introduce *self-plan prompting* to GPT-4 by letting it decompose tasks following a heuristic algorithm, as illustrated in Appendix B.5. Under this method, the model initially discovers the dependencies among actions, task descriptions, and objects and estimates the duration of each action. It then adopts a greedy-like strategy similar to **Oracle Performance**, favoring selecting the longest-duration actions that do not require continuous engagement from the agent in the task model to formulate a plan. Then, the agent executes this plan through interactions with the environment. However, the results indicate that *self-plan prompting* is outperformed by vanilla GPT-4. There are three possible reasons for such performance: 1) The difficulty in accurately parsing actions and identifying their dependencies; 2) The reliance on estimating action durations might introduce cascading errors, leading to inaccurate results of the greedy-like strategy; 3) The rigid adherence to flawed plans, without adapting to the dynamic nature of TIMEARENA, leads to its failure.

4.4 Analysis

Can language agents master multitasking? We conduct a detailed analysis to investigate the actions and define six fine-grained types of actions: 1) **Correct Actions**: Valid Action, Wait (necessary and unnecessary); 2) **Incorrect Actions**: Invalid Action/Object, Dependency Violation, Repeating Completed Action and Object-Mismatched Action.[8]

We calculate the frequency of these actions of each agent throughout their interactions from three combinations (#Task=1, #Task=2, and #Task=3)

[7] Details are in Appendix B.3 and B.4.

[8] Detailed description of different types of actions can be found in Appendix A.3.

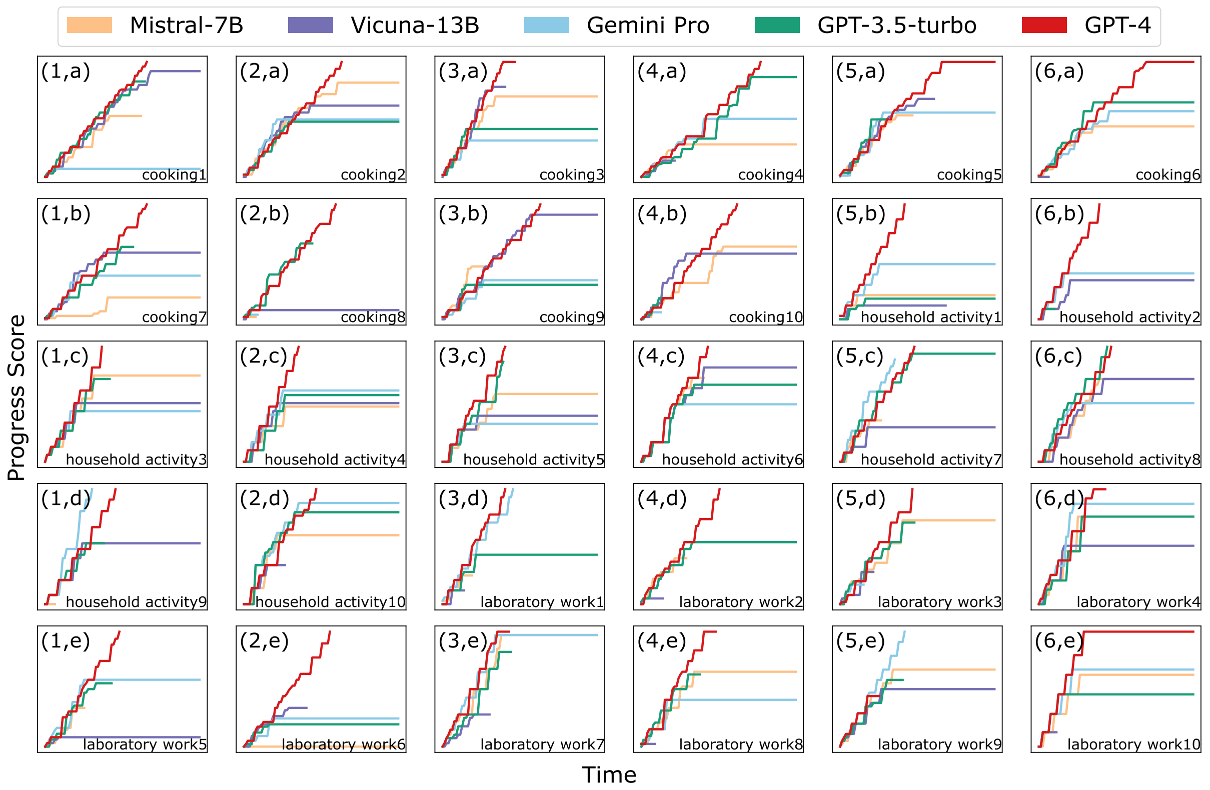


Figure 5: Task progress score curves of language agents on two task combinations in TIMEARENA. The names at the bottom-right indicate the scenario and task number. For example, `cooking1` represents the first combination of tasks in the cooking scenario.

across all the scenarios. The results in Figure 3 show that a significant proportion of invalid actions are due to dependency violations and mismatches with objects. Multitasking involves performing several tasks simultaneously. As the number of tasks increases, the complexity of objects and actions escalates, leading to intricate dependencies between actions. Thus, the high proportion of actions that violate dependencies and mismatch objects suggests that language agents face challenges in managing complex action interdependencies during multitasking, indicating a limitation in their multitasking capabilities.

Are language agents aware of parallel processing? Parallel processing can significantly reduce the time required for efficient multitasking. If an agent is capable of parallel processing, it can engage in additional actions instead of unnecessary waiting for the current action. To answer this question, we decompose `wait` action into two types: `necessary wait` and `unnecessary wait`. The former represents that no actions can currently be performed, requiring waiting for other actions to be completed. In particular, we report the maximum number of `necessary wait`. `Unnecessary wait` indicates that there are other action options available. Figure 3 shows that `wait` constitutes over half of the valid actions performed by different LLMs, and `necessary wait` only accounts for a small part of it. This indicates a tendency for agents to engage in unnecessary waiting, showing their ignorance of parallel processing and inability to complete tasks in minimal time (Table 2).

Do resource constraints affect efficient multitasking of language agents? Resource constraints refer to limitations in the availability of resources (*e.g.*, the number of objects) necessary for task completion, which is rather common in real life. To design resource constraints, we first select three objects: `pot`, `fryer` and `oven` in the cooking scenario, and choose # Task=2 setting in Table 2. Then, we set that there is only one instance of each of the three objects, simulating the limitation of resources in the environment. Figure 4 compares GPT-4's performance before and after applying these constraints. We find that the constraints do not affect the task completion time or completion speed, revealing that GPT-4 rarely attempts to process tasks in parallel. However, a noticeable decline in both completion rate and progress score indicates that the constraints prevent the models from better comprehending and decomposing multiple tasks.

Language agents are trapped in an infinite loop. To delve into why language agents struggle with multiple tasks, we analyze the progress score changes over time. As illustrated in Figure 5, Vicuna, Mistral, Gemini and GPT-3.5 often cease scoring without completing all the tasks, maintaining low scores until time runs out (*e.g.*, (5,b), (2,c) and (6,d)). We further examine their actions during these periods and find that they always perform **incorrect actions and waiting** alternately. Since `wait` is a valid action, repeatedly alternating between waiting and incorrect actions does not lead to task failure, but neither does it contribute to an increase in scores. To find out whether agents wait for good reasons, we ask them to explain each action via the chain-of-thought prompting strategy, and they often believe `wait` can pause incorrect actions. However, they find it hard to adjust their incorrect actions based on feedback after waiting, resulting in them being trapped in infinite loops.

5 Conclusion

In this paper, we introduce TIMEARENA, a text-based simulated environment designed to incorporate the notion of time. TIMEARENA extends beyond simply acknowledging the dependency of actions by also considering their duration, an essential factor in time modeling. Using TIMEARENA, we evaluate the multitasking and parallel processing capability of language agents. Our findings indicate that language agents still have significant room for improvement when completing multiple tasks in dynamic environments in a minimal time, highlighting an area for future research.

Limitations

In TIMEARENA, we implement detailed descriptions of tasks and environments, along with fine-grained textual feedback to simulate interactions. However, TIMEARENA is still designed as a textual simulation for LLMs, lacking visual information that might be necessary for agents to succeed in real-world tasks. For example, in the laboratory work scenario, it is challenging to completely represent chemical reactions through text due to their complexity. The number of tasks and scenarios is limited, while the number of multitasking scenarios that allow parallel processing is large in real life. Moreover, in TIMEARENA, agents interact with the environment only through actions that are explicitly presented in action prompts, rather than exploring freely. Also, whether an action occupies an agent sometimes depends on specific conditions. For instance, the action `cook beef` is classified as non-occupying in TIMEARENA, implying that it does not engage agents continuously. Yet, in reality, this action requires attention, such as turning the beef to prevent burning, a detail TIMEARENA overlooks, potentially reducing the realism of our simulation.

Ethical Statement

We hereby acknowledge that all authors of this work are aware of the provided ACL Code of Ethics and honor the code of conduct.

Use of Human Annotations Our institution recruited three annotators to implement the task creation for three scenarios. We ensure the privacy rights of the annotators are respected during the annotation process. The annotators receive compensation exceeding the local minimum wage and have consented to tasks generated for TIMEARENA for research purposes.

Risks The TIMEARENA in our experiment is created by human annotators, and we further examine them to guarantee that they are devoid of socially harmful or toxic language. However, evaluating the data quality of tasks is based on common sense, which can vary among individuals from diverse backgrounds.

Acknowledgement

We would like to thank Xintao Wang, Ruihan Yang, Tinghui Zhu from Fudan University for their valuable comments and suggestions for the manuscript. We would also like to thank Peter Jansen from University of Arizona and Bodhisattwa Prasad Majumder from Allen Institute for AI for fruitful discussions that helped shape this project at an early stage. This work is funded by Science and Technology Commission of Shanghai Municipality Grant (No. 22511105902).

References

Saaket Agashe, Yue Fan, and Xin Eric Wang. 2023. Evaluating multi-agent coordination abilities in large language models.

Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. 2023. Playing repeated games with large language models.

James F Allen. 1991. Planning as temporal reasoning. *KR*, 91:3–14.

Jiangjie Chen, Siyu Yuan, Rong Ye, Bodhisattwa Prasad Majumder, and Kyle Richardson. 2023a. Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena. *arXiv preprint arXiv:2310.05746*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. 2023. Timebench: A comprehensive evaluation of temporal reasoning abilities in large language models.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. 2023. Mindagent: Emergent gaming interaction.

Xinyu Hua, Zhe Hu, and Lu Wang. 2019. Argument generation with retrieval, planning, and realization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2661–2672, Florence, Italy. Association for Computational Linguistics.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Xiangzhe Kong, Jialiang Huang, Ziquan Tung, Jian Guan, and Minlie Huang. 2021. Stylized story generation with style-guided planning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2430–2436, Online. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents.

Puneet Mathur, Rajiv Jain, Franck Dernoncourt, Vlad Morariu, Quan Hung Tran, and Dinesh Manocha. 2021. TIMERS: Document-level temporal relation extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 524–533, Online. Association for Computational Linguistics.

Puneet Mathur, Vlad Morariu, Verena Kaynig-Fittkau, Jiuxiang Gu, Franck Dernoncourt, Quan Tran, Ani Nenkova, Dinesh Manocha, and Rajiv Jain. 2022. DocTime: A document-level temporal dependency graph parser. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 993–1009, Seattle, United States. Association for Computational Linguistics.

Mistral AI team. 2023. Mixtral of experts. Accessed: 2023-12-15.

Gabriel Mukobi, Hannah Erlebach, Niklas Lauffer, Lewis Hammond, Alan Chan, and Jesse Clifton. 2023. Welfare diplomacy: Benchmarking language model cooperation. *arXiv preprint arXiv:2310.08901*.

OpenAI. 2022. Chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. 2018. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2182–2192, Melbourne, Australia. Association for Computational Linguistics.

Stuart J Russell and Peter Norvig. 2010. *Artificial intelligence a modern approach*. London.

Roger C Schank and Robert P Abelson. 1975. Scripts, plans, and knowledge. In *IJCAI*, volume 75, pages 151–157.

Roger C Schank and Robert P Abelson. 2013. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

Chao Shang, Guangtao Wang, Peng Qi, and Jing Huang. 2022. Improving time sensitivity for question answering over temporal knowledge graphs. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8017–8026, Dublin, Ireland. Association for Computational Linguistics.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023. Taskbench: Benchmarking large language models for task automation.

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Oliviero Stock. 1998. *Spatial and temporal reasoning*. Springer Science & Business Media.

Qingyu Tan, Hwee Tou Ng, and Lidong Bing. 2023. Towards benchmarking and improving the temporal reasoning capability of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14820–14835, Toronto, Canada. Association for Computational Linguistics.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemini Team and Google. 2023. Gemini: A family of highly capable multimodal models.

Siddharth Vashishtha, Benjamin Van Durme, and Aaron Steven White. 2019. Fine-grained temporal relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2906–2919, Florence, Italy. Association for Computational Linguistics.

Lluis Vila. 1994. A survey on temporal reasoning in artificial intelligence. *Ai Communications*, 7(1):4–28.

Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. 2023a. Openchat: Advancing open-source language models with mixed-quality data. *arXiv preprint arXiv:2309.11235*.

Haoyu Wang, Hongming Zhang, Yuqian Deng, Jacob Gardner, Dan Roth, and Muhao Chen. 2023b. Extracting or guessing? improving faithfulness of event temporal relation extraction. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 541–553, Dubrovnik, Croatia. Association for Computational Linguistics.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yuqing Wang and Yun Zhao. 2023. Tram: Benchmarking temporal reasoning for large language models. *arXiv preprint arXiv:2310.00835*.

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023c. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Yue Wu, Xuan Tang, Tom M. Mitchell, and Yuanzhi Li. 2023. Smartplay: A benchmark for llms as intelligent agents.

Sen Yang, Xin Li, Lidong Bing, and Wai Lam. 2023. Once upon a *time* in *graph*: Relative-time pretraining for complex temporal reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11879–11895, Singapore. Association for Computational Linguistics.

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*.

Siyu Yuan, Jiangjie Chen, Ziquan Fu, Xuyang Ge, Soham Shah, Charles Jankowski, Yanghua Xiao, and Deqing Yang. 2023. Distilling script knowledge from large language models for constrained language planning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4303–4325, Toronto, Canada. Association for Computational Linguistics.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2023. Building cooperative embodied agents modularly with large language models.

Michael Zhang and Eunsol Choi. 2021. SituatedQA: Incorporating extra-linguistic contexts into QA. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7371–7387, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2023. Sotopia: Interactive evaluation for social intelligence in language agents.

A Details of TIMEARENA

A.1 Tasks

TIMEARENA contains 30 tasks in cooking, household activity, and laboratory work scenarios. To illustrate how to complete a task, we show the flow chart for each task in Figure 6, Figure 7 and Figure 8.

A.2 Actions

The environment implements 46 actions, and each action has a description. We show the details of these actions in Table 3.

A.3 Action Types

As shown in Table 4, we define 4 incorrect action types and 2 correct action types for analyzing why agents fail in multitasking.

A.4 Greedy-Like Algorithm

We show the greedy-like algorithm in Algorithm 1.

B Examples of TIMEARENA

B.1 Tasks

Table 5, 6 and 7 present some examples of task combinations in TIMEARENA for a better understanding.

B.2 Interaction

Table 5 shows an example of interaction between an agent and the environment in the cooking scenario.

B.3 ReAct

We ask models to think before each action. Table 9 shows the prompt of the ReAct method.

B.4 Reflexion

We ask models to reflect on their wrong actions and retry. Table 10 shows the prompt of the Reflexion method.

B.5 Self-plan

Table 11 shows the prompt of the self-plan method.

Algorithm 1: Greedy-Like Algorithm for Minimal Time Calculation

Input: Set of actions $\mathcal{A}$, Durations $\mathcal{T}$, Dependencies $p(\mathcal{A})$.
Output: Minimal time $\mathcal{T}_{min}$.

$T_{optimal} \leftarrow +\infty$.
Define non-occupied actions $\mathcal{A}*$ and occupied actions $\mathcal{A}'$ from $\mathcal{A}$.
Define the permutations of $\mathcal{A}*$ as $\{\mathcal{A}*'\}$.
foreach $A*' \in \{\mathcal{A}*'\}$ **do**
    $\mathcal{A} \leftarrow$ concatenate$(\mathcal{A}*', \mathcal{A}')$.
    Initialize Action_list as an empty list.
    foreach $a_i \in \mathcal{A}$ **do**
        $P \leftarrow$ BFS$(a_i, p(a_i))$ to collect prerequisites.
        foreach $p_i \in P$ **do**
            if $p_i \in \mathcal{A}$ **then**
                Action_list.append(p_i).
                Remove p_i from $\mathcal{A}$.
            end
        end
        Action_list.append(a_i).
    end
    $\mathcal{T}_{min} \leftarrow 0$.
    while *not empty* $\mathcal{A}*'$ *or* $\mathcal{A}'$ **do**
        foreach $a_i \in$ *Action_list* **do**
            if *check_dependency(a_i)* **then**
                if $a_i \in \mathcal{A}*'$ **then**
                    $\mathcal{T}_{min} \leftarrow \mathcal{T}_{min} + 1$.
                    Remove a_i from $\mathcal{A}*'$.
                else
                    $\mathcal{T}_{min} \leftarrow \mathcal{T}_{min} + \mathcal{T}(a_i)$.
                    Remove a_i from $\mathcal{A}'$.
                end
                break.
            end
        end
        Increment $\mathcal{T}_{min}$ by 1 if no action is performed.
    end
    if $T_{optimal} > \mathcal{T}_{min}$ **then**
        $T_{optimal} = \mathcal{T}_{min}$
    end
end

Action	Description
pick OBJ	Pick the unpicked item
cook OBJ1 in OBJ2	Cook the raw item until it's cooked through
chop OBJ	Chop the whole item into sliced pieces
fry OBJ1 in OBJ2	Fry the raw item until it is fried to perfection
wash OBJ	Wash the dirty item to make clean
bake OBJ1 in OBJ2	Bake the raw item in the oven until it's roasted
activate OBJ	Activate the inactive device to turn it active
pour OBJ1 into OBJ2	Pour the liquid in item into the empty container until it is full
brew OBJ1 with OBJ2	Brew the dry item leaves with the container until they're steeped
gather OBJ	Gather the scattered items until it is collected
scrape OBJ1 into OBJ2	Scrape the contents from the full item into th empty item
place OBJ1 into OBJ2	Place the unplaced item into the right place
fill OBJ1 with OBJ2	Fill the container with something
hoe OBJ	Hoe the uncultivated item until it is cultivated and ready for planting
weed_with OBJ	Weed with the item
set_up OBJ	Set up the item that is not set yet until it is already set
iron OBJ	Iron the wrinkled item until they are smooth
put OBJ1 on OBJ2	Put the item on the right place
add OBJ1 to OBJ2	Add one item to the container
rinse OBJ	Rinse the dry item
find OBJ	Find the missed item so that it is found and can be used
heat OBJ	Heat the cool item until it is hot
dilute OBJ	Dilute the concentrated item until it is diluted
cut OBJ	Cut the whole item into divided pieces
dissolve OBJ1 in OBJ2	Dissolve the solid item in the liquid until it is dissolved
polish OBJ	Polish the rusty item until it is polished
empty OBJ	Empty the full item until it is empty
hanging OBJ	Hang the item
water OBJ1 by OBJ2	Water the item by something
trim OBJ	Trim the overgrown item
plant OBJ	Plant the uncultivated item until it is planted
store OBJ	Store the unstored item
stir OBJ1 with OBJ2	Stir the separate liquid in item with something until it is homogeneous
soak OBJ1 in OBJ2	Soak the dry item in something until it is wet
mop OBJ	Mop the dirty item until it is clean
read OBJ	Read the unknown item
fold OBJ	Fold the spread item until it is tidy
crush OBJ	Crush the intact item until it is crushed
cool OBJ	Cool the hot item until it is cool
dry OBJ	Dry the item until it is dry
wipe OBJ	Wipe the dirty item until it is clean
put OBJ1 in OBJ2	Put the item in something
label OBJ	Give the ambiguous item a label
crystallize OBJ	Crystallize the fluid item until it is crystallized
filter OBJ	Filter the mixed item until it is refined
wait	Pass the current time without doing anything

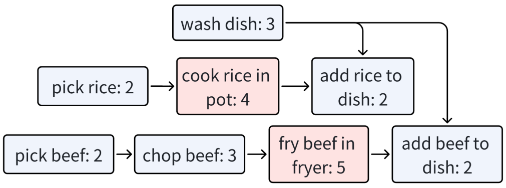
Table 3: Details of actions with descriptions.

Type	Subtype	Explanation	Example: Make tea
Incorrect Actions	Invalid Action/Object	An action does not in the action space or non-existent objects are visited.	<Valid Actions> activate; wash; brew with; pour into <Objects> tea(dry); kettle(inactive); teapot(dirty); cup(dirty) <Trajectory> T=1: clean teapot T=2: brew tea with teapot T=3: wash teapot T=4: wash kettle T=5: wash teapot T=6: activate kettle T=7: wait ...
	Repeating Completed Action	An action is in the action space and matches the objects, but it has already been completed.	
	Dependency Violation	An action is in the action space and matches the objects, but the necessary prerequisite actions have not been completed.	
	Object-Mismatched Action	An action is in the action space and the object is available, but they do not match.	
Correct Actions	Valid Action	An action is in the action space and matches the objects.	
	Wait	An action is used to pass the current time.	

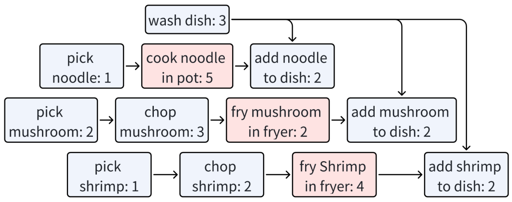
Table 4: Action types and their explanations with an example.

```
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Prepare a noodle dish, which consists of cooked noodle, fried mushrooms and shrimp
.
<Valid Actions and Usages>
- pick OBJ: Pick the unpicked item.
- cook OBJ1 in OBJ2: Cook the raw item until it's cooked through.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wash OBJ: Wash the dirty item to make clean.
- wait: pass the current time without doing anything.

**All Available Objects (OBJ)**
noodle; mushroom; shrimp; fryer; pot; dish

**The Initial States of Objects**
noodle: unpicked; mushroom: unpicked; shrimp: unpicked; fryer: empty; pot: empty;
dish: dirty
```

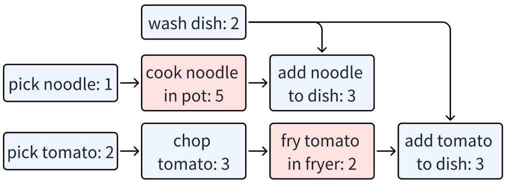
Table 5: An example of # Task=1 scenario.



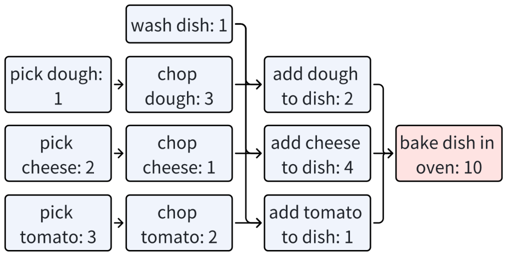
(a) The first task in the cooking scenario.



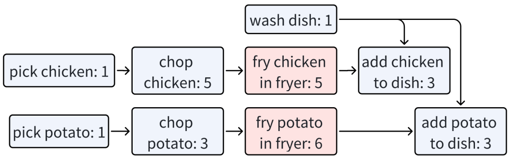
(b) The second task in the cooking scenario.



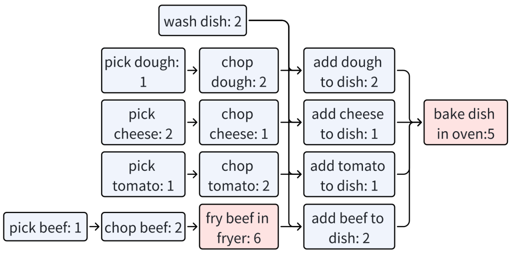
(c) The third task in the cooking scenario.



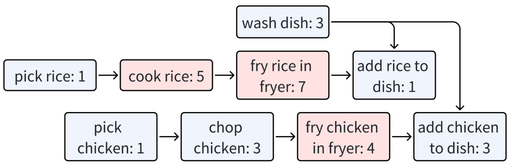
(d) The fourth task in the cooking scenario.



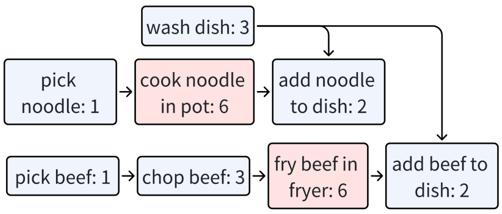
(e) The fifth task in the cooking scenario.



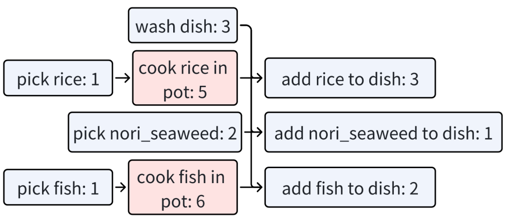
(f) The sixth task in the cooking scenario.



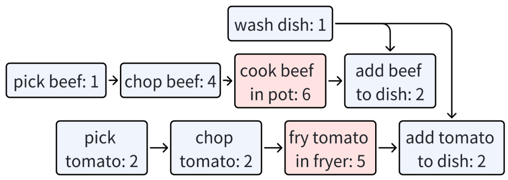
(g) The seventh task in the cooking scenario.



(h) The eighth task in the cooking scenario.

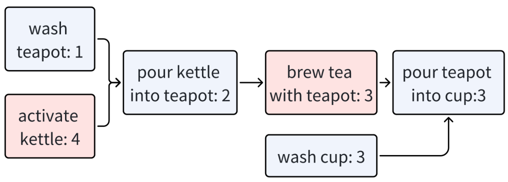


(i) The ninth task in the cooking scenario.

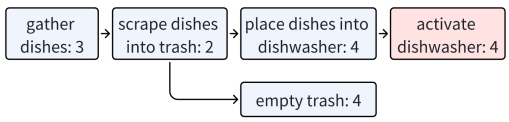


(j) The tenth task in the cooking scenario.

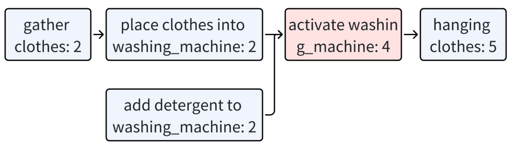
Figure 6: The action dependencies and durations for the ten tasks in the cooking scenario. Actions that occupy the agent, preventing them from doing anything else, are indicated with a blue background. In contrast, actions not occupying the agent, allowing for parallel tasks, are marked with a red background.



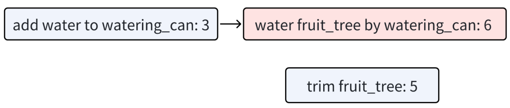
(a) The first task in the household activity scenario.



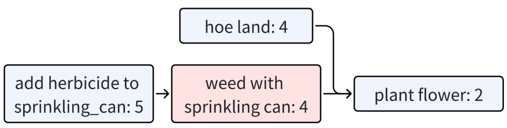
(b) The second task in the household activity scenario.



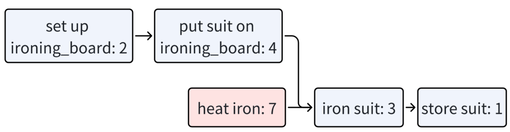
(c) The third task in the household activity scenario.



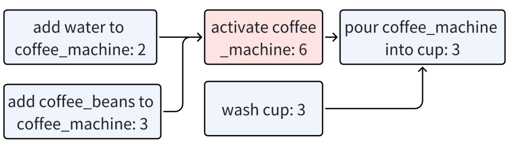
(d) The fourth task in the household activity scenario.



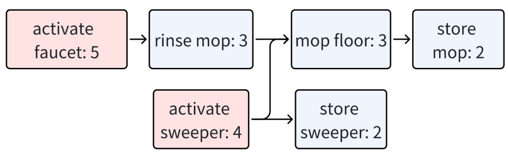
(e) The fifth task in the household activity scenario.



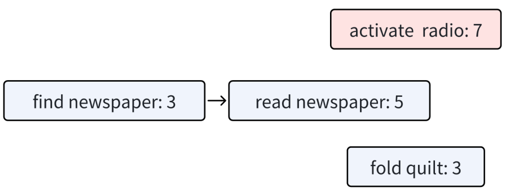
(f) The sixth task in the household activity scenario.



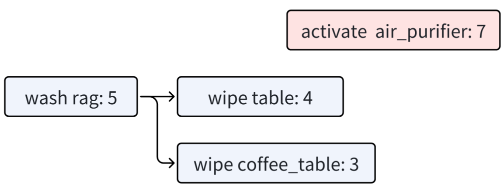
(g) The seventh task in the household activity scenario.



(h) The eighth task in the household activity scenario.

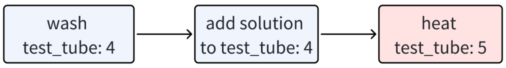


(i) The ninth task in the household activity scenario.

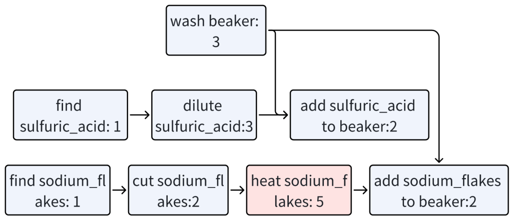


(j) The tenth task in the household activity scenario.

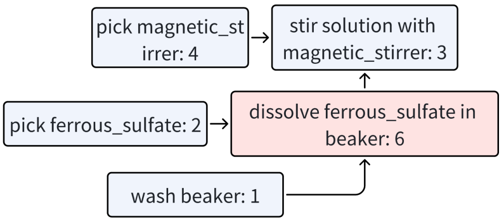
Figure 7: The action dependencies and durations for the ten tasks in the household activity scenario. Actions that occupy the agent, preventing them from doing anything else, are indicated with a blue background. In contrast, actions that do not occupy the agent, allowing for parallel tasks, are marked with a red background.



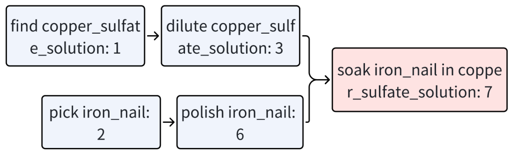
(a) The first task in the laboratory work scenario.



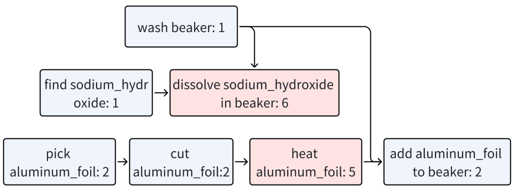
(b) The second task in the laboratory work scenario.



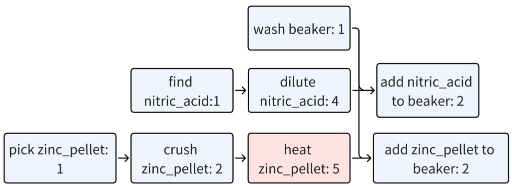
(c) The third task in the laboratory work scenario.



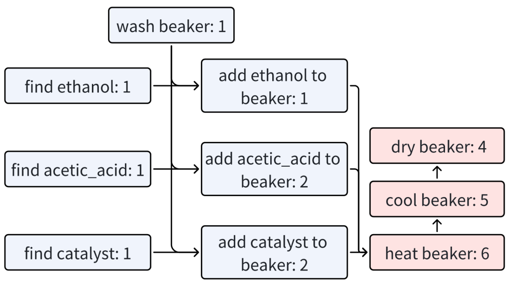
(d) The fourth task in the laboratory work scenario.



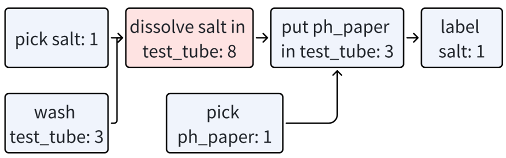
(e) The fifth task in the laboratory work scenario.



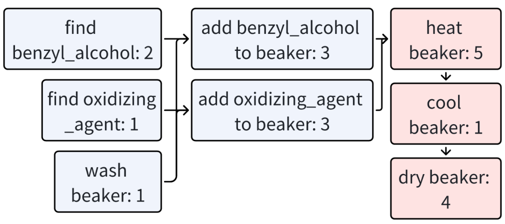
(f) The sixth task in the laboratory work scenario.



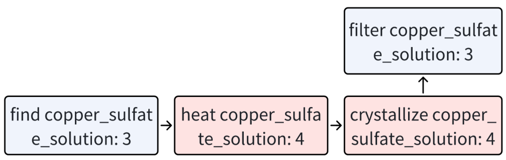
(g) The seventh task in the laboratory work scenario.



(h) The eighth task in the laboratory work scenario.



(i) The ninth task in the laboratory work scenario.



(j) The tenth task in the laboratory work scenario.

Figure 8: The action dependencies and durations for the ten tasks in the laboratory work scenario. Actions that occupy the agent, preventing them from doing anything else, are indicated with a blue background. In contrast, actions that do not occupy the agent, allowing for parallel tasks, are marked with a red background.

```
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Prepare and bake a cheese and tomato pizza
<Valid Actions and Usages>
- pick OBJ: Pick the unpicked item.
- chop OBJ: Chop the whole item into sliced pieces.
- wash OBJ: Wash the dirty item to make clean.
- add OBJ1 to OBJ2: Add one item to the container.
- bake OBJ1 in OBJ2: Bake the raw item in the oven until it's roasted.
- wait: pass the current time without doing anything.

**Task 2**
<Description>
- Prepare chicken and potato stir-fry, which consists of fried chicken and fried
potato.
<Valid Actions and Usages>
- pick OBJ: Pick the unpicked item.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wash OBJ: Wash the dirty item to make clean.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
dish_1; dish_2; dough; cheese; tomato; oven; chicken; potato; fryer

**The Initial States of Objects**
dish_1: dirty; dish_2: dirty; dough: unpicked; cheese: unpicked; tomato: unpicked;
oven: empty; chicken: unpicked; potato: unpicked; fryer: empty
```

Table 6: An example of # Task=2 scenario.

```
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., ”wash OBJ”).
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., ”
heat OBJ”).
- Follow the ”Valid Actions” format for your output (e.g., ”wash cup”).
- If no action is required, use ”wait” to skip the current time.
- Output the action explicitly (e.g., ”wash cup”).
- Select object names (OBJ) from the list of Available Objects (e.g., use ”rice”
instead of ”cooked rice”).

**Task 1**
<Description>
- Prepare a garden bed for planting flowers by using sprinkling can filled with
herbicide, hoeing, and weeding
<Valid Actions and Usages>
- add OBJ1 to OBJ2: Add one item to the container.
- weed_with OBJ: Weed with the item.
- hoe OBJ: Hoe the uncultivated item until it is cultivated and ready for planting.
- plant OBJ: Plant the uncultivated item until it is planted
- wait: pass the current time without doing anything.

**Task 2**
<Description>
- Iron a suit and store it properly
<Valid Actions and Usages>
- set_up OBJ: Set up the item that is not set yet until it is already set.
- put OBJ1 on OBJ2: Put the item on the right place.
- heat OBJ: Heat the cool item until it is hot.
- iron OBJ: Iron the wrinkled item until they are smooth.
- store OBJ: Store the unstored item\nwait: pass the current time without doing
anything.

**Task 3**
<Description>
- Make a cup of coffee
<Valid Actions and Usages>
- add OBJ1 to OBJ2: Add one item to the container.
- activate OBJ: Activate the inactive device to turn it active.
- wash OBJ: Wash the dirty item to make clean.
- pour OBJ1 into OBJ2: Pour the liquid in item into the empty container until it is
full.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
sprinkling_can; herbicide; land; flower; ironing_board; suit; iron; coffee_beans;
coffee_machine; water; cup

**The Initial States of Objects**
sprinkling_can: empty; herbicide: not added; land: uncultivated; flower:
uncultivated; ironing_board: not set yet; suit: not put on right place; iron: cool;
coffee_beans: not added; coffee_machine: empty; water: not added; cup: dirty
```

Table 7: An example of # Task=3 scenario.

```
<|Environment|>:
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Make a dish of beef fried rice, which consists of cooked rice and fried beef.
<Valid Actions and Usages>
- wash OBJ: Wash the dirty item to make clean.
- pick OBJ: Pick the unpicked item.
- cook OBJ1 in OBJ2: Cook the raw item until it's cooked through.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
rice; beef; pot; fryer; dish

**The Initial States of Objects**
rice: unpicked; beef: unpicked; pot: empty; fryer: empty; dish: dirty

<|Environment|>:
In t=1, your action is:

<|Agent|>:
pick dish

<|Environment|>:
You are doing ``wash dish``, it will take 2 minutes.
rice is picked.

In t=3, your action is:

<|Agent|>:
cook rice in pot

<|Environment|>:
You are doing ``cook rice in pot``, it will take 4 minutes.

In t=4, your action is:

<|Agent|>:
wait

<|Environment|>:
You wait for one minute.

...
```

Table 8: An example of interaction in cooking scenario.

```
<|Environment|>:
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Make a dish of beef fried rice, which consists of cooked rice and fried beef.
<Valid Actions and Usages>
- wash OBJ: Wash the dirty item to make clean.
- pick OBJ: Pick the unpicked item.
- cook OBJ1 in OBJ2: Cook the raw item until it's cooked through.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
rice; beef; pot; fryer; dish

**The Initial States of Objects**
rice: unpicked; beef: unpicked; pot: empty; fryer: empty; dish: dirty

Please think about the interaction history between the agent and the environment,
consider the states of the agent and objects and the task instructions with the goal
 of minimizing all task completion time.
Try to identify the most efficient action (i.e., parallel performing) to take next.
If there are other actions that can be executed, try not to wait.
Finally, output your thoughts on the next action.
...
```

Table 9: Prompt of ReAct method in cooking scenario.

```
<|Environment|>:
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Make a dish of beef fried rice, which consists of cooked rice and fried beef.
<Valid Actions and Usages>
- wash OBJ: Wash the dirty item to make clean.
- pick OBJ: Pick the unpicked item.
- cook OBJ1 in OBJ2: Cook the raw item until it's cooked through.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
rice; beef; pot; fryer; dish

**The Initial States of Objects**
rice: unpicked; beef: unpicked; pot: empty; fryer: empty; dish: dirty

<|Agent|>:
chop rice in pot

<|Environment|>:
Invalid action!

You are an advanced reasoning agent capable of improving through self-reflection.
Review and reflect on the historical interactions between the agent and the
environment.
Please diagnose a possible reason for the failure and devise a new, concise plan
that aims to mitigate the failure.
...
```

Table 10: Prompt of Reflexion method in cooking scenario.

```
<|Environment|>:
As an AI agent, your objective is to efficiently complete a series of tasks as
described. You must adhere to the specific requirements and constraints of each task
, including dependencies and timing. Efficiency is key; complete all tasks in the
shortest possible time. I will provide instructions regarding actions and objects.

**Action Protocol**:
- You can perform only one action at a time.
- After each observation from the environment, output an action based on that
observation and the instructions.
- Actions fall into two categories:
- Continuous Actions: Perform these actions until completion (e.g., "wash OBJ").
- Autonomous Actions: These progress over time, allowing simultaneous tasks (e.g., "
heat OBJ").
- Follow the "Valid Actions" format for your output (e.g., "wash cup").
- If no action is required, use "wait" to skip the current time.
- Output the action explicitly (e.g., "wash cup").
- Select object names (OBJ) from the list of Available Objects (e.g., use "rice"
instead of "cooked rice").

**Task 1**
<Description>
- Make a dish of beef fried rice, which consists of cooked rice and fried beef.
<Valid Actions and Usages>
- wash OBJ: Wash the dirty item to make clean.
- pick OBJ: Pick the unpicked item.
- cook OBJ1 in OBJ2: Cook the raw item until it's cooked through.
- chop OBJ: Chop the whole item into sliced pieces.
- fry OBJ1 in OBJ2: Fry the raw item until it is fried to perfection.
- add OBJ1 to OBJ2: Add one item to the container.
- wait: pass the current time without doing anything.

**All Available Objects(OBJ)**
rice; beef; pot; fryer; dish

**The Initial States of Objects**
rice: unpicked; beef: unpicked; pot: empty; fryer: empty; dish: dirty

Given the list of valid actions, available objects, and the task descriptions (goal)
, please perform the following steps:
- Identify and list all of the necessary actions required to accomplish the task's
goal.
- For each action, determine and note the specific objects that are required.
- Assess and map out any dependencies between actions, indicating which actions must
 precede others.
- Arrange the actions in a logical sequence that respects the dependencies and leads
 efficiently towards completing the task.
- If any action has multiple dependencies, list them in order of priority based on
the task's constraints and goal.
- Present the final action sequence in a clear and ordered list, ensuring that the
progression of steps will achieve the task's objective.

The key to efficiency:
- When completing tasks, some actions are non-occupied actions (Type 2), meaning you
 can perform other actions simultaneously.
- To maximize efficiency, adhere to the following principle: always start the non-
occupied action you anticipate will be the most time-consuming as early as possible.
- You should perform actions during idle times as much as possible to minimize the
time spent doing nothing.

...
```

Table 11: Prompt of self-plan method in cooking scenario.