ParaCook: On Time-Efficient Planning for Multi-Agent Systems

Anonymous ACL submission

Abstract

Large Language Models (LLMs) exhibit strong reasoning abilities for planning long-horizon, real-world tasks, yet existing agent benchmarks focus on task completion while neglecting time efficiency in parallel and asynchronous operations. To address this, we present ParaCook, a benchmark for time-efficient collaborative planning. Inspired by the Overcooked game, ParaCook provides an environment for various challenging interaction planning of multi-agent systems that are instantiated as cooking tasks, with a simplified action space to isolate the core challenge of strategic parallel planning. Through a comprehensive evaluation of state-of-the-art LLMs, we find that current approaches achieve suboptimal plans, which struggle with parallel actions or coordination. Our analysis also reveals LLMs' potential on abstract tasks where they can focus on high-level parallel optimization. ParaCook provides a scalable evaluation framework with adjustable complexity, establishing a foundation for developing and assessing time efficiency-aware multi-agent planning.

1 Introduction

Large Language Models (LLMs) have empowered agents with remarkable **planning** capabilities in complex and interactive tasks (Yang et al., 2025; Shinn et al., 2023). Planning integrates task knowledge and breaks down long-horizon goals into subtasks, enabling overall coherence rather than just local optimality. However, time efficiency remains a major challenge, as it is widely accepted that the usability of LLM agents depends not only on improvements in raw model intelligence or accuracy, but on maximizing Agentic ROI (Liu et al., 2025).

This investigates the research question that **long-horizon tasks should not only be distributed correctly, but also scheduled efficiently**. Efficient scheduling is non-trivial because complex tasks always involve both parallel and sequential steps. For a single agent, its schedule can exploit concurrency between sub-tasks that do not block each other to reduce idle time, such as during water boiling, the agent can switch to another action like chopping vegetables. For multi-agent system, different agents can leverage the collaboration to distribute workload to operate on independent subtasks, such as cooking different dishes. These two forms of parallelism, intra- and inter-agent, jointly determine the time efficiency of multi-agent planning.

However, existing agent benchmarks mainly focus on correctness and rule-following rather than time efficiency in parallel scheduling. While several studies have considered concurrent scenarios, their coverage remains limited. AsyncHow (Lin et al., 2024a) oversimplifies tasks and assumes unlimited resources. Robotouille (Gonzalez-Pumariega et al., 2025) focuses only on single-agent asynchronous planning. CookBench (Cai et al., 2025) offers long-horizon, multi-step tasks, but is too comprehensive to isolate time efficiency evaluation. As a result, current benchmarks do not fully expose opportunities for parallelism, leaving a notable gap in evaluating whether LLMs can truly exploit concurrency to minimize overall task completion time.

To address this gap, we propose ParaCook, a new benchmark that focuses on evaluating whether LLMs can effectively schedule tasks and coordinate agents to minimize overall completion time. We focus on cooking, an everyday scenario that naturally involves both sequential and parallel tasks.Inspired by the Overcooked game, ParaCook simplifies action spaces and task rules, allowing models to concentrate on effective task scheduling and optimal time utilization. Our benchmark supports multiple agents cooperating, emphasizing both inter-agent parallelism and intra-agent time utilization. To further isolate planning from low-level actions, we introduce corresponding abstract tasks that focus solely on high-level task allocation and optimization of execution time. These abstract

tasks enable direct evaluation of LLMs' planning capabilities and preserve the challenges of efficient multi-agent scheduling.

Through comprehensive experiments, we find that top models like GPT-5 achieve only a 65% average success rate with significant performance drops on complex tasks, while humans maintain perfect success. Models also exhibit substantially longer completion times and higher movement costs than human baselines. On abstract planning tasks, top LLMs achieve near-optimal performance (within 1-7% of optimum), demonstrating strong high-level reasoning capabilities.

Our contributions can be summarized as follows: (i) ParaCook, the first benchmark for time-efficient multi-agent planning with systematic parallelism evaluation; (ii) A scalable framework with adjustable complexity control; (iii) Comprehensive analysis revealing critical gaps between current LLMs and human performance.

2 Related Work

In this section, we introduce relevant studies on LLM as planning agents, their extension to multi-agent cooperation, and benchmarks for evaluating planning quality.

2.1 Planning for Agents

LLMs have emerged as powerful planner due to their strong reasoning capabilities.

Single-agent system planning. Planning enables an agent to decompose complex tasks and refine actions through iterative reasoning (Yao et al., 2023; Yang et al., 2025). Mainstream studies focus on task decomposition (Shen et al., 2023; Wang et al., 2023; Gao et al., 2023; Chen et al., 2023), or self-improvement with reflection on historical error in memory (Shinn et al., 2023). Hybrid paradigms combine symbolic planners utilizing code or graph (Liu et al., 2023; Cao et al., 2025; Zhang et al., 2025b), further improve generalization.

Multi-agent system planning. Multi-agent systems (MAS) have shown great progress in complex tasks, where agents with diverse profiles collaborate. Planning becomes even more pronounced in MAS, requiring collaboration and ROI. A commander, meta-agent, or manager agent is often responsible for iter-agent planning, like role assignment, and coordination among agents. Early frameworks (Wu et al., 2023; Hong et al., 2024) established structured workflows under centralized control. More flexible planning strategies encourage hierarchical and partially decentralized planning, as in AgentOrchestra (Zhang et al., 2025c) and Agent-Oriented Planning (Li et al., 2025).

Benchmark	MAS	Interact.	IntraP.	InterP.	Time	Step
AsyncHow	✓	×	✓	✓	✓	Short
Robotouille	×	✓	✓	×	✓	Long
WORFBENCH	✓	×	×	✓	×	Short
Collab-Overcooked	✓	✓	×	✓	×	Long
CookBench	×	✓	✓	×	×	Long
ParaCook (Ours)	✓	✓	✓	✓	✓	Long

Table 1: Comparison of agent planning benchmarks. IntraP: Intra-Agent Parallelism; InterP: Inter-Agent Parallelism; Env. Interact.: Environment Interaction; Time Eval.: Time Efficiency Evaluation as primary metric.

2.2 Parallelable Agent

Recent efforts have explored making LLM-based agents parallelizable to enhance efficiency. Adaptive reasoning frameworks enable concurrent thought processes (Pan et al., 2025; Zhang et al., 2024), while DAG-based and divide–aggregate methods support simultaneous tool use (Zhang et al., 2025a; Zhu et al., 2025). At the system level, asynchronous planning–acting architectures and graph-based schedulers (Zhang et al., 2025b) facilitate concurrent execution and coordination. Despite these advances, research remains disparate across reasoning and scheduling, , with no unified benchmark for evaluating agents' ability to exploit parallelism effectively.

2.3 Benchmarks for Agent Planning

Rigorous benchmarks are essential for systematic evaluation, especially in multi-agent settings. Most existing benchmarks focus on single-agent or strictly sequential planning, emphasizing solution correctness and rule-following. AsyncHow (Lin et al., 2024b) evaluates asynchronous planning by computing an optimal schedule, but assumes unlimited resources to achieve maximal parallelism, and lacks environment interaction. Robotouille (Gonzalez-Pumariega et al., 2025) simulates cooking tasks with proper time consumption, but only focuses on single-agent asynchronous planning. WORFBENCH (Qiao et al., 2024) decomposes tasks into workflow graphs but evaluates graph similarity rather than execution time. Collab-Overcooked (Sun et al., 2025)and CookBench (Cai et al., 2025) provide comprehensive challenges for MAS. However, they also do not support explicit time efficiency evaluation. As is summarized in

Table 1, existing benchmarks still fall short of systematically testing how effectively LLMs can exploit both intra-agent and inter-agent parallelism to minimize overall completion time, motivating our design of ParaCook.

3 Formulation

In this section, we formulate the research question of parallel planning for agent systems, introducing basic concepts and key properties.

Task Decomposition and Dependence First, we define the parallel planning uniformly for single- and multi-agent systems. A complex task $\mathcal{I}$ can be decomposed into subtasks with dependencies, formalized as a Directed Acyclic Graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ represents subtasks, while the edge set $\mathcal{E} = \{(u, v)\}$ represents dependencies, i.e., u must complete before v starts. Formally, the predecessors of v is denoted as $Pa_{\mathcal{G}}(\mathrm{v}) = \{u \mid (u, v) \in \mathcal{E}\}$. In terms of a multi-agent system with m participants, $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$, each vertex includes an extra attribute of agent identity to indicate the operator of each subtask, $v_j.id \leftarrow a_j$. To build agent systems that achieve tasks efficiently, our study concentrates on task success and time efficiency.

Task Success Task success requires that the task decomposition $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is valid, all dependency constraints are satisfied, and every subtask $v \in \mathcal{V}$ is correctly executed by its assigned agent $v.id$.

Time Efficiency In consideration of the time efficiency, each subtask has a *theoretical execution time*, $v_j.time \leftarrow t(v)$. Subtasks involve inherent execution delays that proceed automatically, requiring only waiting rather than continuous involvement. This allows parallel execution for single-agent system on long-horizon tasks. For example, when boiling water, the heating process requires only waiting without further action. These are denoted by a delay function $d(u, v)$, modeling the minimum waiting interval between completing u and starting v. multi-agent system's parallel execution includes (i) single-agent parallelism as defined by $d()$ and (ii) inter-agent parallelism: different agents execute independent subtasks simultaneously as defined by $Pa_{\mathcal{G}}()$.

To reflect the physical world, we account for the stochastic nature of execution times. Our notion of *actual execution time*, $t'(v)$ incorporates inter-task transitions. For example, the time required for an agent to be prepared for the next subtask. Such setup time between subtasks depends on the agent's assigned task sequence in the plan.

4 Benchmark: ParaCook

Based on our problem formulation, we propose ParaCook, the first benchmark that focuses on time-efficient parallel planning and multi-agent scheduling. We adopt "cooking" as a natural testbed, as it inherently exhibits sequential dependencies and asynchronous waiting, making it both challenging and well-suited for parallel planning.

4.1 Environment

The environment of ParaCook is a 2D grid world that simulates a kitchen, supporting agent–environment and agent–agent interactions while providing feedback. At the beginning of each episode, the environment initializes an agent team, the task to be performed, and the map configuration.

Action space supported by our environment are

- **MoveTo.** Specify a target coordinate for the agent to move to. Our environment computes the shortest path and executes the movement. The movement time is proportional to the distance.
- **Interact.** Pick up or put down an item at the current location. The execution time is negligible.
- **Process.** Perform a continuous operation at a workstation, such as chopping or washing. The execution time depends on the specific operation.
- **Wait.** Do nothing for a specified period, typically when blocked or waiting for a predecessor.
- **Finish.** Declare task accomplished.

Instantiation Formulated concepts in Section 3 are instantiated in our cooking environment.

- Task decomposition $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ models the workflow of multiple orders, where nodes correspond to individual cooking steps and edges indicate dependency relations among them.
- Time delay $d()$ corresponds to cooking and waiting times between steps.
- Parallel execution enables simultaneous cooking operations and cross-dish coordination.
- Theoretical execution time $t()$
- Actual execution time $t'()$ incorporates the travel time required for an agent to move to the next workstation for the following subtask.

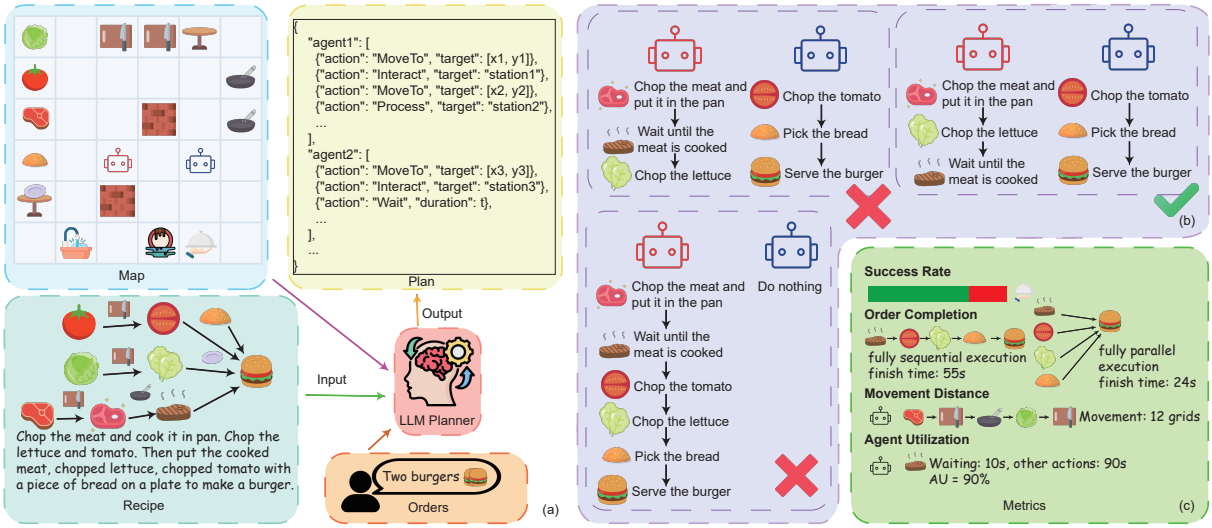

Figure 1: Overview of the ParaCook benchmark, showing (a) the benchmark pipeline, (b) comparison of planning strategies, and (c) evaluation metrics for model performance.

4.2 Task

The agent team is instructed to propose action sequences to complete all dishes, given the input specified order. We structure tasks based on recipes and orders, creating controllable yet meaningful concurrency challenges.

Recipes The recipes describe the steps required to prepare a dish in natural language. For example, *Put chopped lettuce and chopped tomato together on a plate to make a salad.* Our recipes vary in planning difficulty.

- Simple recipes require only basic operations such as chopping and plating, providing a baseline for sequential planning.
- Intermediate recipes involve multiple operations (e.g., cut then cook) with asynchronous waiting periods, requiring agents to schedule efficiently.
- Complex recipes contain a larger number of ingredients that each require separate processing, demanding long-horizon planning and careful coordination to minimize idle time.

Orders The orders are combinations of dish sequences. Orders with more dishes are more difficult, as the action horizon increases while conflicts and constraints of resources arise, including plate reuse and workstation availability. Thus, effective planning requires understanding and coordinating the steps of each dish to maximize time efficiency.

Therefore, our task design enables concurrency (i) between steps within a dish, different steps can be executed simultaneously, and (ii) across dishes, idle time slots of earlier dishes can be used to prepare for later dishes.

4.3 Map

The configuration of the environment maps enables dynamic and flexible control over map size, workstation arrangement, agent count, and initial positions, ensuring high scalability. More importantly, to support efficiency-oriented evaluation, the map can be configured by tuning the balance between workstations and agents, enabling controlled parallelism that ensures sufficient concurrent executions without excessive parallelization.

4.4 Difficulty Control

Based on the diverse and complex settings introduced above, the difficulty D of a task can be formalized as

$$D = f(C_{\text{recipe}}, C_{\text{order}}, C_{\text{map}}), \tag{1}$$

where C_{recipe} denotes the operating complexity of selected dishes, C_{order} denotes the composition order of dishes. C_{map} indicates the challenges in spatial map configuration. Therefore, ParaCook enables fine-grained difficulty control, facilitating systematic benchmarking of planning capabilities.

5 Metrics

ParaCook evaluates plans along two primary dimensions: *correctness* and *efficiency*. Correctness is measured by the dataset-level **Success Rate (SR)**, while efficiency is primarily captured by the task-level **Order Completion Time (OCT)**. Additional metrics, including **Movement Distance (MD)** and **Agent Utilization (AU)**, provide fine-grained analyses of execution cost and coordination.

5.1 Primary Metrics

Success Rate (SR) At the dataset level, we define a run as successful if all dishes in the given order are completed correctly:

$$SR = N_{\text{success}}/N_{\text{total}}, \tag{2}$$

where N_{success} is the number of successful runs and N_{total} is the total number of test cases.

Order Completion Time (OCT) For a single successful task, OCT is the total elapsed time until completion of all orders, which the real time returned by the environment simulation.

$$OCT = T_{\text{actual}}. \tag{3}$$

To aggregate across samples, we use two variants:

$$pOCT = 1/N_{\text{total}} \sum_{i=1}^{N_{\text{total}}} T_i^*, \tag{4}$$

$$nOCT = 1/N_{\text{success}} \sum_{i=1}^{N_{\text{success}}} OCT_i/T_{\max}^i, \tag{5}$$

where $T_i^* = OCT_i$ if task i succeeds, and $T_i^* = T_{\max}^i$ otherwise. Here, $T_{\max}^i$ is the predefined upper bound for task i. $pOCT$ penalizes failures with maximal time, while $nOCT$ normalizes efficiency among successful runs.

5.2 Auxiliary Metrics

Movement Distance (MD) For a task, MD is the mean travel distance of all agents:

$$MD = 1/M \sum_{j=1}^{M} d_j, \tag{6}$$

where d_j is the distance traveled by agent j, and M is the number of agents. Similarly, to aggregate across tasks, we calculate the penalized form:

$$pMD = 1/N_{\text{total}} \sum_{i=1}^{N_{\text{total}}} D_i^*, \tag{7}$$

with $D_i^* = MD_i$ for successful tasks and $D_i^* = D_{\max}^i$ otherwise. $D_{\max}^i$ represents the predefined upper bound of movement distance for task i.

Agent Utilization (AU) For each agent j, utilization is defined as the proportion of active working time:

$$u_j = T_j^{\text{work}}/T_j^{\text{total}}, AU = 1/M \sum_{j=1}^{M} u_j. \tag{8}$$

Dataset-level AU is averaged over successful runs.

In summary, **SR** evaluates correctness, and **OCT** evaluates efficiency with penalized and normalized variants. **MD** and **AU** serve as auxiliary analyses, capturing execution cost and coordination quality.

6 Experiments

This section presents our empirical implementations and main findings from our results.

6.1 Experimental Setup

Dataset We manually annotated six categories of recipes with increasing levels of difficulty, and each category contains several different dishes. For orders, a set of dishes was randomly sampled from a given recipe category, with the number of dishes ranging from one to four to control the complexity of the task. To encourage parallelism, each kitchen map was equipped with two stations of each type, placed randomly while ensuring connectivity. The number of agents varied between one and three. For each configuration, we used five random seeds to generate different orders and maps.

Agents To evaluate different methods, we consider the I/O and CoT prompting methods. I/O produces a complete plan directly from the initial state, which specifies the actions for each agent. CoT (Wei et al., 2022) encourages reasoning while planning, explicitly predicting intermediate states. We evaluate state-of-the-art API LLMs, including GPT-5, Claude-Opus-4.1, Gemini-2.5-Pro (Comanici et al., 2025), DeepSeek-V3.2-Exp (DeepSeek-AI, 2025), and Qwen3-Max-Preview.

6.2 Results

Table 2 presents overall results, while Table 5 shows more detailed scores. Our results answer the following three key questions.

ParaCook is challenging enough to differentiate SOTA LLMs, which vary on SR. In terms of *Success Rate (SR)*, GPT-5 achieves the highest success rate across all difficulty levels, with an average SR of 65.0%. It maintains strong performance on Easy (80.8%) and Medium (69.2%) tasks, and remains competitive even on Hard tasks (45.0%). In contrast, Gemini-2.5-Pro (47.4% on average) and DeepSeek-V3.2-Exp (45.3%) form the second tier, showing moderate performance but a noticeable drop in the Hard setting (27.1% and 21.7%, respectively). Claude-Opus-4.1 (13.1%) and Qwen3 (2.2%) perform poorly, failing almost completely on Medium and Hard tasks.

ParaCook serves as a suitable, still challenging testbed for time efficiency for planning capabilities of SOTA LLMs. In terms of pOCT, GPT-5 again demonstrates the best performance, with the

Model	Success Rate (SR, %)				Penalized Order Completion Time (pOCT)			
	Easy	Medium	Hard	Avg.	Easy	Medium	Hard	Avg.
GPT-5	**80.83**	**69.17**	**45.00**	**65.00**	**137.98**	**330.80**	**416.35**	**295.04**
Gemini-2.5-Pro	60.00	55.07	27.14	47.40$_{-17.60}$	191.94	390.33	486.94	356.40$_{+61.36}$
DeepSeek-V3.2-Exp	66.67	47.50	21.67	45.28$_{-19.72}$	176.89	467.98	521.19	388.69$_{+93.65}$
Claude-Opus-4.1	26.67	12.50	0.00	13.06$_{-51.94}$	268.54	585.65	–	–
Qwen3-Max-Preview	6.67	0.00	0.00	2.22$_{-62.78}$	291.97	–	–	–

Table 2: Success Rate (SR) and penalized Order Completion Time (pOCT) of different models across three difficulty levels and their averages. **Bold** indicates the best performance in each column. Red subscripts in the Avg. column show the performance gap relative to the best model (GPT-5).

lowest average planning overhead (295.0), consistently faster than other models at all difficulty levels. Gemini-2.5-Pro (356.4) and DeepSeek-V3.2-Exp (388.7) occupy the middle range, showing reasonable efficiency but still substantially slower than GPT-5. Claude-Opus-4.1 (356.7) and Qwen3 (289.4) exhibit unstable efficiency: despite occasionally achieving low overhead on Easy tasks, they either fail to complete tasks at higher difficulty levels or produce unreliable results, limiting their practical applicability.

In detail, Table 5 illustrates that time efficiency is related to nOCT, MD and AU. Specifically, GPT-5's superior efficiency is reflected in strong auxiliary metrics, maintaining minimal nOCT and pMD across all tasks, and high agent utilization (AU) above 86%. Gemini-2.5-Pro and DeepSeek-V3.2-Exp are less efficient, with pMD values rising to 178.60 and 195.28 on Medium tasks, respectively. Claude-Opus-4.1 performs the worst, with pMD exceeding 230 and significantly lower utilization, confirming that inefficient scheduling and the failure to parallelize actions directly lead to longer completion times.

SOTA LLM planners demonstrably lag behind human performance in success rate, time efficiency, and spatial optimization. We provide human performance to answer the question: *How do LLM planners compare to human performance?* Figure 2 compares human participants with top-performing LLMs on a curated subset of tasks (whose detailed results are presented in Table 6). Humans achieve a perfect 100% success rate across all difficulty levels, highlighting their robustness in complex, long-horizon scenarios where LLMs may fail. The performance gap becomes particularly evident on Hard tasks. For instance, the top-performing model, GPT-5 with CoT, achieved a 57% SR, whereas humans made no errors.

In terms of efficiency, while top LLMs are competitive on easy tasks, they lag significantly as complexity increases. On Hard tasks, humans completed orders with an nOCT of 15.38, whereas GPT-5 required a longer time of 17.61. The disparity in spatial efficiency is even more stark, with humans having a pMD of just 47.75, compared to GPT-5's 143.55, indicating far more optimized movement from humans. Interestingly, AU is slightly lower for humans (87.13%) than for GPT-5 (94.34%). This is not due to inefficiency, but because human plans are executed so quickly that the necessary waiting time between dependent tasks constitutes a larger proportion of the total duration. Overall, these results demonstrate that while LLMs can produce correct and parallelized plans for less complex tasks, their execution strategies remain less temporally and spatially optimized than those of humans, particularly for high coordination demands.

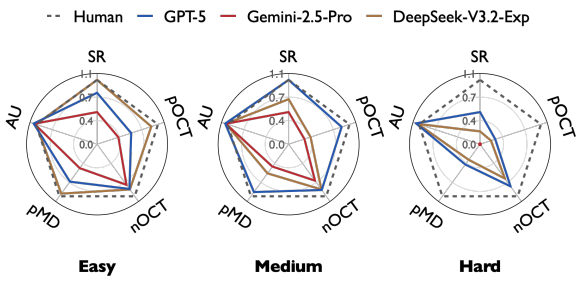

Figure 2: Results on the human-evaluated subset for LLM-human comparison. Detailed scores are in Table 6 and 7.

7 Analysis

7.1 Performance across Task Complexity

Figure 3 shows the performance by varying two key complexity dimensions: the number of agents and the number of orders.

Impact of Agent Number: A team with more agents requires better parallel planning, but increasing coordination complexity and resource

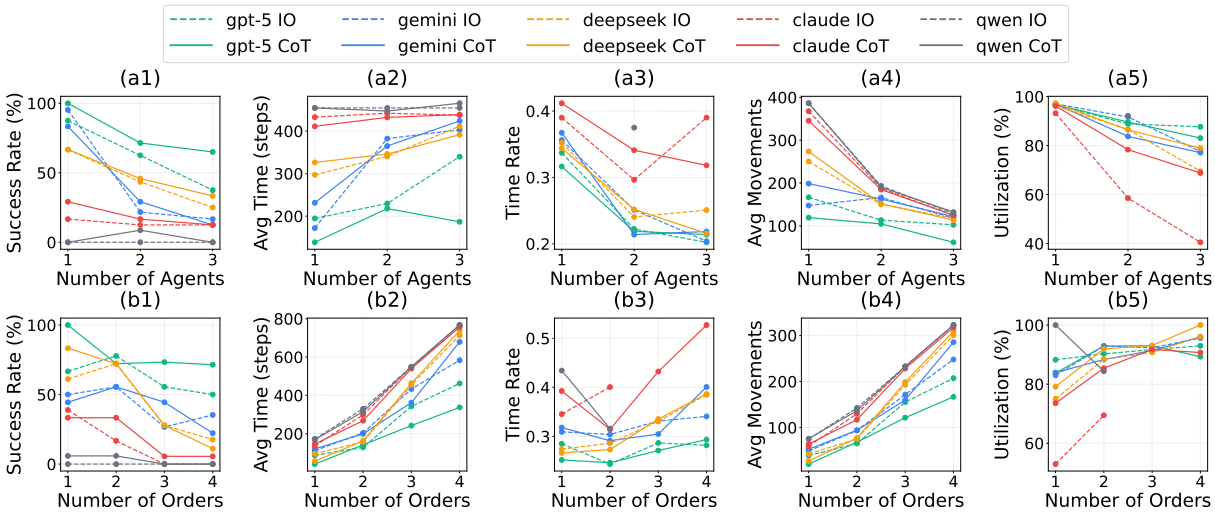

Figure 3: Model performance across different task complexities. Top row (a1-a5): varying number of agents. Bottom row (b1-b5): varying number of orders. Each column corresponds to the five metrics.

contention. As shown in Figure (a1), the SR consistently declines as the number of agents increases, highlighting the inherent difficulty of multi-agent coordination. This is also reflected in the pOCT (Figure (a2)), which generally rises with the failure rate. A notable exception is GPT-5 CoT, whose pOCT drops at three agents, suggesting its successful plans achieve exceptional parallel efficiency that outweighs the penalty from a slightly lower SR. The nOCT offers deeper insight (Figure (a3)). The sharp drop from one to two agents demonstrates that an additional agent effectively enhances efficiency through parallelism. However, this gain diminishes from two to three agents, where the nOCT curve flattens. We attribute this to resource contention; with a limited number of workstations (e.g., two cutting boards), a third agent cannot work in parallel on the same task type, capping the achievable speedup. Furthermore, the anomalous rise in Claude IO's nOCT, coupled with its significantly lower AU (Figure (a5)), suggests a "false parallelism" where one agent works while others wait, failing to leverage the multi-agent setup. Also, the expected decrease in pMD (Figure (a4)) confirms better task locality with more agents.

Impact of Order Number: Increasing order number raises cost and lowers success, but sometimes allows more parallelism opportunity. Next, we examine the effect of the planning horizon. As anticipated, increasing the number of orders extends the task sequence, leading to a general decline in SR (Figure (b1)). Consequently, the overall cost metrics, pOCT (Figure (b2)) and pMD (Figure (b4)), rise steadily as more work requires more time and movement. The nOCT trend is more nuanced (Figure (b3)); the metric remains stable or even decreases when moving from one to two orders. This is because sufficient initial resources allow agents to prepare two orders in parallel, so the actual completion time T_{actual} increases only modestly. In contrast, the normalization factor, $T_{\max}$, is set significantly higher for a two-order task. This combination results in a stable or smaller nOCT ratio, indicating scalable plan quality in scenarios with low complexity. Lastly, AU generally improves with more orders (Figure (b5)), as more subtasks effectively reduce agent idle time in successful runs.

7.2 Performance with CoT

Does Chain-of-Thought prompting universally improve model performance? Our findings indicate that **the effectiveness of CoT is heavily dependent on the model's underlying reasoning capability.**

For a strong model like GPT-5, CoT acts as a consistent amplifier, enhancing performance across the board and raising the SR on Hard tasks from 45% to 57%. In contrast, for moderately capable models, CoT's influence is unstable. Gemini, for instance, showed a counterintuitive decline in success on Easy and Medium tasks when using CoT. DeepSeek presented a mixed picture, with CoT only providing a decisive improvement on Hard tasks, where the SR rose from 22% to 41%. For weaker models such as Claude and Qwen, CoT functioned as a limited rescue mechanism, signif-

Model	Method	Easy		Medium		Hard		Average	
		SR↑	pOCT↓	SR↑	pOCT↓	SR↑	pOCT↓	SR↑	pOCT↓
GPT-5	IO	80.83	137.98	69.17	330.80	45.00	416.35	65.00	295.04
	CoT	84.17$_{+3.34}$	129.59$_{-8.39}$	77.03$_{+7.86}$	283.03$_{-47.77}$	57.39$_{+12.39}$	359.16$_{-57.19}$	72.86$_{+7.86}$	257.26$_{-37.78}$
Gemini-2.5-Pro	IO	60.00	191.94	55.07	390.33	27.14	486.94	47.40	356.40
	CoT	55.83$_{-4.17}$	201.43$_{+9.49}$	47.22$_{-7.85}$	419.10$_{+28.77}$	37.50$_{+10.36}$	447.49$_{-39.45}$	46.85$_{-0.55}$	356.01$_{-0.39}$
DeepSeek-V3.2-Exp	IO	66.67	176.89	47.50	467.98	21.67	521.19	45.28	388.69
	CoT	67.50$_{+0.83}$	178.42$_{+1.53}$	45.83$_{-1.67}$	470.74$_{+2.76}$	40.83$_{+19.16}$	465.84$_{-55.35}$	51.39$_{+6.11}$	371.67$_{-17.02}$
Claude-Opus-4.1	IO	26.67	268.54	12.50	585.65	0.00	–	13.06	427.10
	CoT	55.00$_{+28.33}$	215.99$_{-52.55}$	14.58$_{+2.08}$	577.52$_{-8.13}$	0.00$_{+0.00}$	–	23.19$_{+10.13}$	396.76$_{-30.34}$
Qwen3-Max-Preview	IO	6.67	291.97	0.00	–	0.00	–	2.22	291.97
	CoT	10.83$_{+4.16}$	286.73$_{-5.24}$	0.00$_{+0.00}$	–	0.00$_{+0.00}$	–	3.61$_{+1.39}$	286.73$_{-5.24}$

Table 3: Results with CoT. Blue subscripts indicate improvements (higher SR or lower pOCT), red indicates degradation. The δ(CoT-IO) row shows the difference between CoT and IO methods.

Model	SR↑	pOCT↓	nOCT↓
cp_sat	100.00	100.28	1.0000
GPT-5	100.00	103.76	1.0241
Gemini-2.5-Pro	100.00	99.61	1.0179
DeepSeek-Reasoner	97.00	87.10	1.0288
Claude-Opus-4.1	100.00	108.94	1.0856
Qwen3-Max-Preview	78.00	113.14	1.1190

Table 4: Model performance on abstract tasks.

icantly improving success on Easy tasks but remaining insufficient for Medium or Hard problems. Therefore, CoT is not a universally beneficial strategy. It reliably enhances already strong models but can destabilize moderately capable ones and offers only limited support to weaker ones.

7.3 Abstract Task

Method To further investigate the essence of suboptimal temporal efficiency, we designed abstract planning tasks that isolate planning from action execution. In this setup, tasks are formulated as in section 3. The LLM's objective is to generate a schedule, allocating and ordering subtasks for a given number of agents, that minimizes the total completion time. We benchmarked the generated plans against the provably optimal solutions derived from a constraint programming solver. We analyze three aspects, which are Success Rate, nOCT, and pOCT, with a penalty of 1.2 times the optimal time to any invalid plan.

Results Table 4 shows results on abstract tasks, where we summarize two key findings.

SOTA LLMs have strong inherent potential in reasoning for parallel planning. For instance, GPT-5, Gemini-2.5-Pro, and Claude-Opus all achieved perfect SR scores, with nOCT scores of 1.0174, 1.0161, and 1.0721 respectively, indicating their schedules were on average only 1-7% slower than the theoretical optimum. This demonstrates a strong inherent capability for reasoning about complex temporal dependencies and optimizing for parallelism when not constrained by environmental interaction.

The capability hierarchy observed in the ParaCook environment largely persists in this abstract setting. Qwen3-Max-Preview's SR fell to 79%, and its successfully generated plans were, on average, nearly 11% less efficient than the optimal solution (nOCT of 1.1190). This suggests that the performance differences among LLMs are at least partially rooted in their fundamental optimization and reasoning abilities, not solely in their capacity to handle the specific rules of the embodied environment.

Crucially, the contrast between near-optimal performance on abstract tasks and the inefficiency in real ParaCook highlights the need for structured approaches as a bridge, such as hierarchical planning frameworks that separate high-level scheduling from detailed action execution.

8 Conclusion

In this work, we introduce ParaCook, a benchmark for evaluating time-efficient planning in multi-agent systems. We systematically test state-of-the-art LLMs across varied task complexities and find that GPT-5 achieves the best overall performance in both success rate and completion time. Human–LLM comparisons reveal that, while top models approach human performance in simple scenarios, they still lag in complex coordination and fine-grained temporal optimization. Results on abstract tasks further confirm LLMs' strong reasoning ability for high-level scheduling. ParaCook establishes a scalable foundation for advancing time-efficient and coordination-aware LLM agents.

Limitations

We acknowledge the limitations of this work. (i) Our current work mainly focuses on establishing a benchmark and evaluating existing methods, and does not propose a novel solution to overcome the parallel planning challenges. (ii) The scope of Agent methods tested was limited, primarily focusing on IO and CoT strategies. Our future work will focus on these two directions.

References

Muzhen Cai, Xiubo Chen, Yining An, Jiaxin Zhang, Xuesong Wang, Wang Xu, Weinan Zhang, and Ting Liu. 2025. Cookbench: A long-horizon embodied planning benchmark for complex cooking scenarios. *arXiv preprint arXiv:2508.03232*.

Zouying Cao, Runze Wang, Yifei Yang, Xinbei Ma, Xiaoyong Zhu, Bo Zheng, and Hai Zhao. 2025. PGPO: Enhancing agent reasoning via pseudocode-style planning guided preference optimization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 14966–14985, Vienna, Austria. Association for Computational Linguistics.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3290 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

DeepSeek-AI. 2025. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. 2025. Robotouille: An asynchronous planning benchmark for LLM agents. In *The Thirteenth International Conference on Learning Representations*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. 2025. Agent-oriented planning in multi-agent systems. In *The Thirteenth International Conference on Learning Representations*.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony G Cohn, and Janet B Pierrehumbert. 2024a. Graph-enhanced large language models in asynchronous plan reasoning. In *Forty-first International Conference on Machine Learning*.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony Cohn, and Janet B. Pierrehumbert. 2024b. Graph-enhanced large language models in asynchronous plan reasoning. *Preprint*, arXiv:2402.02805.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Weiwen Liu, Jiarui Qin, Xu Huang, Xingshan Zeng, Yunjia Xi, Jianghao Lin, Chuhan Wu, Yasheng Wang, Lifeng Shang, Ruiming Tang, and 1 others. 2025. The real barrier to llm agent usability is agentic roi. *arXiv preprint arXiv:2505.17767*.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. 2025. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv: 2504.15466*.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024. Benchmarking agentic workflow generation. *arXiv preprint arXiv:2410.07869*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. In *Advances in Neural Information Processing Systems*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. 2025. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. *Preprint*, arXiv:2502.20073.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Dongjie Yang, Chengqiang Lu, Qimeng Wang, Xinbei Ma, Yan Gao, Yao Hu, and Hai Zhao. 2025. Plan your travel and travel with your plan: Wide-horizon planning and evaluation via llm. *arXiv preprint arXiv:2506.12421*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Bin Zhang, Hangyu Mao, Lijuan Li, Zhiwei Xu, Dapeng Li, Rui Zhao, and Guoliang Fan. 2024. Sequential asynchronous action coordination in multi-agent systems: A stackelberg decision transformer approach. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 59559–59575. PMLR.

Shaokun Zhang, Jieyu Zhang, Dujian Ding, Jiale Liu, Mirian Del Carmen Hipolito Garcia, Ankur Mallick, Daniel Madrigal, Menglin Xia, Victor Rühle, Qingyun Wu, and Chi Wang. 2025a. Ecoact: Economic agent determines when to register what action. In *Workshop on Reasoning and Planning for Large Language Models*.

Shiqi Zhang, Xinbei Ma, Zouying Cao, Zhuosheng Zhang, and Hai Zhao. 2025b. Plan-over-graph: Towards parallelable llm agent schedule. *arXiv preprint arXiv:2502.14563*.

Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. 2025c. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving. *arXiv preprint arXiv:2506.12508*.

Dongsheng Zhu, Weixian Shi, Zhengliang Shi, Zhaochun Ren, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. 2025. Divide-then-aggregate: An efficient tool learning method via parallel tool invocation. *arXiv preprint arXiv:2501.12432*.

A detailed experimental results

Table 5, 6 and 7 provide detailed experimental results.

B Prompts

Prompt template for I/O Testing - Part 1

You are given the input map JSON, recipes, and orders, along with the Overcooked multi-agent parallel planning rules described below. Your goal is to generate a detailed action plan (Action List) for guiding each agent to complete dish preparation. The action plan must strictly follow the specified format and constraints.
Core Principles: Maximize Efficiency: Minimize the total time required to complete all orders. This is the most critical goal. Maximize Parallelism: Ensure multiple agents are working simultaneously whenever possible to reduce idle time. Ensure Accuracy: Adhere 100% to all action definitions, rules, and constraints outlined below.
Input Content: Map JSON: Describes kitchen layout, station coordinates, initial items, and agent positions. Recipes: Describes the preparation workflow and required ingredients for the dishes. Orders: Describe the dishes that need to be completed in order.
Output Requirements: For each agent, output an ordered action list (e.g., agent1, agent2). Each action is a dictionary containing action type and parameters. Please strictly follow the output standard JSON format action list. Do not add any additional explanations or content!
Output Format Example: {{ "plan": {{ "agent1": [{{"action": "MoveTo", "target": [x1, y1]}}, {{"action": "Interact", "target": "station_name1"}}, ...], "agent2": [{{"action": "MoveTo", "target": [x2, y2]}}, {{"action": "Process", "target": "station_name2"}}, ...], ... }} }}
Task:
{task}

Model	Method	Easy					Medium					Hard				
		SR↑	pOCT↓	nOCT↓	pMD↓	AU↑	SR↑	pOCT↓	nOCT↓	pMD↓	AU↑	SR↑	pOCT↓	nOCT↓	pMD↓	AU↑
GPT-5	IO	80.83	137.98	29.16	67.84	90.16	69.17	330.80	25.38	144.21	83.75	45.00	416.35	27.34	185.76	94.70
	CoT	84.17	129.59	26.84	64.34	91.10	77.03	283.03	24.92	129.32	86.52	57.39	359.16	23.56	164.00	91.98
Gemini-2.5-Pro	IO	60.00	191.94	31.24	90.03	92.73	55.07	390.33	27.68	164.62	84.23	27.14	486.94	30.56	210.18	87.71
	CoT	55.83	201.43	32.06	93.33	93.69	47.22	419.10	27.05	178.60	87.05	37.50	447.49	30.38	198.51	89.91
DeepSeek-V3.2-Exp	IO	66.67	176.89	30.01	83.49	89.52	47.50	467.98	26.31	193.85	84.89	21.67	521.19	26.44	229.07	95.43
	CoT	67.50	178.42	29.29	83.63	90.52	45.83	470.74	25.16	195.28	81.89	40.83	465.84	26.08	206.53	94.52
Claude-Opus-4.1	IO	26.67	268.54	31.96	122.12	70.95	12.50	585.65	35.64	236.99	52.33	0.00	–	–	–	–
	CoT	55.00	215.99	33.20	98.98	83.87	14.58	577.52	37.61	233.45	66.69	0.00	–	–	–	–
Qwen3-Max-Preview	IO	6.67	291.97	31.33	132.63	61.54	0.00	–	–	–	–	0.00	–	–	–	–
	CoT	10.83	286.73	33.55	130.47	90.27	0.00	–	–	–	–	0.00	–	–	–	–

Table 5: Results of different methods on state-of-the-art models across three difficulty levels.

Model	Method	Easy					Medium					Hard				
		SR↑	pOCT↓	nOCT↓	pMD↓	AU↑	SR↑	pOCT↓	nOCT↓	pMD↓	AU↑	SR↑	pOCT↓	nOCT↓	pMD↓	AU↑
Human	–	100.00	44.00	19.48	32.17	93.31	100.00	73.67	16.24	57.17	87.96	100.00	66.67	15.31	51.08	90.92
GPT-5	IO	80.00	78.60	22.70	44.60	97.09	100.00	85.20	18.53	62.15	91.60	50.00	270.50	19.00	130.55	95.67
	CoT	90.00	53.20	19.08	36.05	96.35	100.00	91.50	20.36	59.58	88.54	40.00	304.20	17.61	143.55	94.34
Gemini-2.5-Pro	IO	50.00	126.90	24.84	69.70	96.46	50.00	288.33	23.19	132.58	90.66	0.00	–	–	–	–
	CoT	60.00	117.70	21.71	64.40	94.09	50.00	278.17	21.31	127.08	86.30	50.00	247.50	18.26	125.75	93.58
DeepSeek-V3.2-Exp	IO	100.00	51.80	23.19	34.35	93.39	70.00	204.10	18.77	102.15	90.61	20.00	379.40	23.36	171.75	92.95
	CoT	100.00	49.30	22.32	33.85	93.63	90.00	137.80	21.54	76.80	90.17	50.00	263.70	18.90	130.50	99.78

Table 6: Results on the human-evaluated subset for LLM-human comparison.

Prompt template for I/O Testing - Part 2

Environment Rules and Constraints:
Agent Rules: No Collision: Agents do not consider collision boxes between each other; their movement paths and positions can overlap at any time. Single Item Hold: An agent can only hold one item at a time (e.g., an ingredient, a plate, a pot). Item exchange must be done via surfaces like tables; direct passing is not allowed. Cannot hold multiple ingredients or containers at once. Positioning: Agents can only stand on empty floor tiles; actions must be performed on adjacent empty ground to target stations; movement can only occur through empty ground. At any time, an agent's coordinates can never overlap with a station's coordinates. Agents can only interact or process with workstations that are adjacent in the four cardinal directions (up, down, left, right).
Environment & Item Rules: Station Exclusivity: Fixed stations like cutting boards or sinks can only be used by one agent at a time for a Process action. Ingredient Dispensing: Ingredients can only be obtained from designated dispensers. Each dispenser provides a specific type of ingredient. All types of ingredients can be directly held without the need for additional containers. Cooking Process: Stoves can only hold cookware (pots/pans), not ingredients directly. Cooking starts automatically once cookware is placed on a stove and contains ingredients. Picking it up pauses cooking; placing it back on any stove resumes it. Cooked food cannot be picked up by hand; it must be transferred in a container. Serving Process: All food items must be placed on a plate before being submitted at the serving window. The order in which the ingredients are placed on the plate is not important. Dishes must be served in the exact order specified in the Orders list. Plate Cycle: Dirty plates return to the dirty plate return station some time after a dish is served. A dirty plate cannot hold any items and must be washed at a sink to become a clean plate. Time Consumption: Move: 1 unit per tile Interact: INTERACT_TIME units Chopping: PROCESS_CUT_TIME units Pot Cooking: PROCESS_POT_COOK_TIME units Pan Cooking: PROCESS_PAN_COOK_TIME units Washing Plates: PROCESS_WASH_PLATE_TIME units Dirty Plate Return: RETURN_DIRTY_PLATE_TIME units

Prompt template for I/O Testing - Part 3

Action Definitions: MoveTo(coordinate): format: {{"action": "MoveTo", "target": [x, y]}} Interact(target_name): format: {{"action": "Interact", "target": "station_name"}} Process(target_name): format: {{"action": "Process", "target": "station_name"}} Wait(duration): format: {{"action": "Wait", "duration": t}} Finish(): format: {{"action": "Finish"}}
Suggestions: Tasks must be reasonably allocated to achieve multi-agent parallel collaboration and minimize total time consumption. Action sequence must completely cover the entire process from raw material acquisition, processing, assembly to serving. Always notice the timepoint when each action starts and ends to ensure no conflicts in agent actions and get the most efficient plan.

Model	Method	Easy					Medium					Hard				
		SR↑	pOCT↑	nOCT↑	pMD↑	AU↑	SR↑	pOCT↑	nOCT↑	pMD↑	AU↑	SR↑	pOCT↑	nOCT↑	pMD↑	AU↑
Human	–	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
GPT-5	IO	0.80	0.56	0.86	0.72	1.04	1.00	0.86	0.88	0.92	1.04	0.50	0.25	0.81	0.39	1.05
	CoT	0.90	0.83	1.02	0.89	1.03	1.00	0.81	0.80	0.96	1.01	0.40	0.22	0.87	0.36	1.04
Gemini-2.5-Pro	IO	0.50	0.35	0.78	0.46	1.03	0.50	0.26	0.70	0.43	1.03	0.00	–	–	–	–
	CoT	0.60	0.37	0.90	0.50	1.01	0.50	0.26	0.76	0.45	0.98	0.50	0.27	0.84	0.41	1.03
DeepSeek-V3.2-Exp	IO	1.00	0.85	0.84	0.94	1.00	0.70	0.36	0.86	0.56	1.03	0.20	0.18	0.66	0.30	1.02
	CoT	1.00	0.89	0.87	0.95	1.00	0.90	0.53	0.75	0.74	1.03	0.50	0.25	0.81	0.39	1.10

Table 7: Normalized results relative to human performance within each difficulty level. All metrics are normalized such that human performance equals 1.0 for each difficulty separately, with higher values indicating better performance. For pOCT, nOCT, and pMD (originally lower-is-better metrics), inverse ratios are used.

Prompt template for CoT Testing - Part 1

You are given the input map JSON, recipes, and orders, along with the Overcooked multi-agent parallel planning rules described below. Your goal is to generate a **step-by-step reasoning process** (Chain-of-Thought, CoT) that leads to a detailed action plan for guiding each agent to complete dish preparation. The reasoning must explicitly explain the allocation of subtasks, parallel coordination, and timing decisions.
Core Principles: Maximize Efficiency: Minimize the total time required to complete all orders. This is the most critical goal. Maximize Parallelism: Ensure multiple agents are working simultaneously whenever possible to reduce idle time. Ensure Accuracy: Adhere 100% to all action definitions, rules, and constraints outlined below.
Input Content: Map JSON: Describes kitchen layout, station coordinates, initial items, and agent positions. Recipes: Describes the preparation workflow and required ingredients for the dishes. Orders: Describe the dishes that need to be completed in order.
Output Requirements: For each agent, output an ordered action list alongside the CoT reasoning steps. Each step should include: reasoning about which subtask to execute, dependencies, and timing considerations. Each action is a dictionary containing action type and parameters. Please strictly follow the output standard JSON format for action lists and reasoning steps. Do not add any additional explanations outside of the CoT reasoning.
Output Format Example: {{ "CoT": ["Step 1: Agent1 moves to ingredient dispenser to pick up tomato, reasoning: starting first ingredient to minimize idle time", "Step 2: Agent2 moves to counter to prepare plate, reasoning: parallel work to maximize efficiency", ...], "plan": {{ "agent1": [{{"action": "MoveTo", "target": [x1, y1]}}, {{"action": "Interact", "target": "station_name1"}}, ...], "agent2": [{{"action": "MoveTo", "target": [x2, y2]}}, {{"action": "Process", "target": "station_name2"}}, ...], ... }} }}
Task:
{task}

Prompt template for CoT Testing - Part 2

Environment Rules and Constraints:
Agent Rules: No Collision: Agents do not consider collision boxes between each other; movement paths and positions can overlap. Single Item Hold: Agents can only hold one item at a time. Exchanges must be done via surfaces; direct passing is not allowed. Positioning: Agents can only stand on empty floor tiles; movement can only occur through empty ground; coordinates cannot overlap with stations. Interactions: Only with adjacent workstations in four cardinal directions.
Environment & Item Rules: Station Exclusivity: Fixed stations can only be used by one agent at a time. Ingredient Dispensing: Ingredients obtained only from designated dispensers. Cooking Process: Stoves hold cookware, start automatically when ingredients are inside; cooked food must be transferred in a container. Serving Process: Food must be plated before submission; served in order of Orders list. Plate Cycle: Dirty plates return to the dirty plate return station; washed at a sink to become clean. Time Costs: Move: 1 unit per tile Interact: INTERACT_TIME units Chopping: PROCESS_CUT_TIME units Pot Cooking: PROCESS_POT_COOK_TIME units Pan Cooking: PROCESS_PAN_COOK_TIME units Washing Plates: PROCESS_WASH_PLATE_TIME units Dirty Plate Return: RETURN_DIRTY_PLATE_TIME units
Action Definitions: MoveTo(coordinate): format: {{"action": "MoveTo", "target": [x, y]}} Interact(target_name): format: {{"action": "Interact", "target": "station_name"}} Process(target_name): format: {{"action": "Process", "target": "station_name"}} Wait(duration): format: {{"action": "Wait", "duration": t}} Finish(): format: {{"action": "Finish"}}
Suggestions: Tasks must be reasonably allocated to achieve multi-agent parallel collaboration and minimize total time consumption. Action sequence must completely cover the entire process from raw material acquisition, processing, assembly to serving. Always notice the timepoint when each action starts and ends to ensure no conflicts in agent actions and get the most efficient plan.

C Menu Recipes

This section provides a comprehensive overview of all recipes in the dataset, organized by food category. The dataset contains 20 recipes across 6 categories: Burger, Burrito, Pasta, Salad, Sashimi, and Sushi. Each recipe includes detailed preparation instructions.

C.1 Burger

Burgers are a classic dish consisting of cooked meat patties served with bread. The dataset includes five burger variations, ranging from basic burgers to more elaborate versions with additional toppings such as lettuce, tomato, and cheese. All burger recipes require cooking the meat in a pan and assembling the ingredients on a plate.

Basic Burger (burger_basic) First chop the meat and cook it in pan. Then put the cooked meat with a piece of bread on a plate to make a basic burger.

Burger with Lettuce (burger_lettuce) Chop the meat and cook it in pan. Chop the lettuce. Then put the cooked meat, chopped lettuce with a piece of bread on a plate to make a burger with lettuce.

Full Burger (burger_full) Chop the meat and cook it in pan. Chop the lettuce and tomato. Then put the cooked meat, chopped lettuce, chopped tomato with a piece of bread on a plate to make a full burger.

Burger with Cheese (burger_cheese) Chop the meat and cook it in pan. Then put the cooked meat with a piece of bread and a slice of cheese on a plate to make a burger with cheese.

Burger with Cheese and Lettuce (burger_cheese_lettuce) Chop the meat and cook it in pan. Chop the lettuce. Then put the cooked meat, chopped lettuce with a piece of bread and a slice of cheese on a plate to make a burger with cheese and lettuce.

C.2 Burrito

Burritos are Mexican-inspired dishes that combine cooked rice with protein, wrapped in a tortilla. The dataset features three burrito variations with different protein options: meat, chicken, and mushroom. All burrito recipes require cooking rice in a pot and the protein in a pan before assembly.

Burrito with Meat (burrito_meat) Chop and cook the meat in pan, cook the rice in pot, then put cooked meat and cooked rice together with a raw tortilla to a plate to make a burrito with meat.

Burrito with Chicken (burrito_chicken) Chop and cook the chicken in pan, cook the rice in pot, then put cooked chicken and cooked rice together with a raw tortilla to a plate to make a burrito with chicken.

Burrito with Mushroom (burrito_mushroom) Chop and cook the mushroom in pan, cook the rice in pot, then put cooked mushroom and cooked rice together with a raw tortilla to a plate to make a burrito with mushroom.

C.3 Pasta

Pasta dishes combine cooked pasta with various sauces and toppings. The dataset includes four pasta variations featuring tomato, meat, mushroom, and seafood. All pasta recipes require cooking the pasta in a pot and preparing the sauce or protein in a pan.

Pasta with Tomato (pasta_tomato) Cook the pasta in pot, chop the tomato and cook it in pan, then put cooked pasta and cooked tomato together to a plate to make pasta with tomato pasta.

Pasta with Meat (pasta_meat) Cook the pasta in pot, chop the meat and cook it in pan, then put cooked pasta and cooked meat together to a plate to make pasta with meat sauce.

Pasta with Mushroom (pasta_mushroom) Cook the pasta in pot, chop the mushroom and cook it in pan, then put cooked pasta and cooked mushroom together to a plate to make pasta with mushroom sauce.

Seafood Pasta (pasta_seafood) Cook the pasta in pot, chop the fish and prawn and cook them in pan respectively, then put cooked pasta, cooked fish and cooked prawn together to a plate to make seafood pasta.

C.4 Salad

Salads are fresh vegetable dishes that require no cooking. The dataset contains three salad variations of increasing complexity, from a basic lettuce salad to a full salad with multiple vegetables. All salad recipes only require chopping vegetables and plating them.

Basic Salad (salad_basic) Put chopped lettuce on a plate to make a salad.

Advanced Salad (salad_advanced) Put chopped lettuce and chopped tomato together to a plate to make a salad.

Full Salad (salad_full) Put chopped lettuce, chopped tomato and chopped cucumber together to a plate to make a salad.

C.5 Sashimi

Sashimi is a Japanese dish featuring raw seafood that is simply chopped and plated. The dataset includes two sashimi variations: fish and shrimp. These are the simplest recipes in the dataset, requiring only chopping and plating.

Sashimi with Fish (sashimi_fish) Chop the fish and put the chopped fish to a plate to make sashimi with fish.

Sashimi with Shrimp (sashimi_shrimp) Chop the shrimp and put the chopped shrimp to a plate to make sashimi with shrimp.

C.6 Sushi

Sushi is a Japanese dish that combines cooked rice, nori (seaweed), and various fillings. The dataset features three sushi variations with fish, cucumber, or both. All sushi recipes require cooking rice in a pot and assembling with raw nori and chopped ingredients.

Fish Sushi (sushi_fish) First chop the fish and cook the rice. Then put the chopped fish and cooked rice and a piece of nori on a plate to make a fish sushi.

Cucumber Sushi (sushi_cucumber) First chop the cucumber and cook the rice. Then put the chopped cucumber and cooked rice and a piece of nori on a plate to make a cucumber sushi.

Full Sushi (sushi_full) First chop the fish and cucumber and cook the rice. Then put the chopped fish, chopped cucumber and cooked rice and a piece of nori on a plate to make a full sushi.

D Dataset Extensibility

ParaCook is designed not as a fixed dataset but as a flexible framework. Both tasks and map configuration support extensions. New recipes and cooking tools can be added to create richer workflows, while recipe complexity, order size, and resource constraints can be systematically varied. Maps can be scaled or altered during execution, and stochastic events (e.g., overcooking, fires) can be introduced to test robustness and adaptive planning. In addition, Maps also supports particular collaborative planning with specific designs. For instance, a row of tables may divide the kitchen, placing different types of workstations on each side, so that agents on each side can only perform certain tasks and must pass items through the central tables for coordination. This extensibility keeps the framework useful for evaluating complex planning as agents advance.