TPS-Bench: Evaluating AI Agents' Tool Planning & Scheduling Abilities in Compounding Tasks

Hanwen Xu*, **Xuyao Huang***, **Yuzhe Liu**, **Zhijie Deng**†
School of Computer Science, Shanghai Jiao Tong University
{dctnorin, huangxuyao, 2263692082, zhijied}@sjtu.edu.cn
GitHub 🤗 Hugging Face

Abstract

Large language model (LLM) agents have exhibited strong problem-solving competence across domains like research and coding. Yet, it remains underexplored whether LLM agents can tackle compounding real-world problems that require a diverse set of tools to complete. Given a broad, heterogeneous tool repository, LLM agents must not only select appropriate tools based on task planning analysis but also strategically schedule the execution order to ensure efficiency. This paper introduces **TPS-Bench** to benchmark the ability of LLM agents in solving such problems that demand **T**ool **P**lanning and **S**cheduling. TPS-Bench collects 200 compounding tasks of two difficulty levels, based on a tool repository containing hundreds of model context protocol (MCP) tools. In particular, each task is composed of multiple subtasks, such as web search, map navigation, calendar checking, etc., and each subtask can be completed by a basic tool. Our evaluation emphasizes both task completion rate and efficiency. The empirical studies on popular closed-source and open-source LLMs indicate that most models can perform reasonable tool planning, but differ in scheduling. For example, GLM-4.5 achieves an outperforming task completion rate of 64.72% with extensive sequential tool calls, hence suffering from significantly long execution time. By contrast, GPT-4o prioritizes parallel tool calls but achieves only a 45.08% completion rate. Considering reinforcement learning (RL) can be a viable way to improve the scheduling efficiency without compromising performance, we perform an initial study on Qwen3-1.7B and witness a 14% reduction in execution time alongside a 6% gain in task completion rate based on only 597 RL training samples.

1 Introduction

Large language model (LLM) agents have demonstrated strong capabilities in solving various problems that require tool use (Wu et al., 2024; Qin et al., 2023; Nakano et al., 2022) , including deep research (OpenAI, 2025), coding (Yang et al., 2024), and data analysis (Hong et al., 2024), etc. For example, GPT-o3 (OpenAI, 2025) can suitably solve the web search tasks in AssistantBench (Yoran et al., 2024) using the Google search tool, and GPT-5 (OpenAI, 2025) excels in using edit tools to tackle coding tasks in SWE-Bench (Jimenez et al., 2024).

However, real-world scenarios often involve *compounding tasks* that demand the collaboration of various tools, which present new challenges to LLM agents. As exemplified in Figure 1, a compounding task can involve multiple subtasks, each of which can be addressed by a basic tool. For example, the request "*Please check the dates of upcoming holidays, recommend travel destinations for me, list a detailed itinerary, and arrange hotel reservation.*" can be decomposed into subtasks such as calendar checking, web search, itinerary planning, hotel reservation, etc. Tackling such a task requires LLM agents not only to engage in task comprehension and tool planning to identify necessary tools for completing the task, but also to employ strategic scheduling so that independent subtasks can be parallelized while dependent ones adhere to a sequential workflow.

This paper introduces **TPS-Bench** for assessing the **T**ool **P**lanning and **S**cheduling ability of LLM agents. It consists of 200 compounding tasks defined on hundreds of model context protocol (MCP) tools. The involved subtasks include weather report, calendar checking, web search, map navigation, etc. According to compounding complexity, we organize TPS-Bench into two levels: TPS-Bench-Easy, which features simple and

*Equal contribution.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 34949–34961
July 2-7, 2026

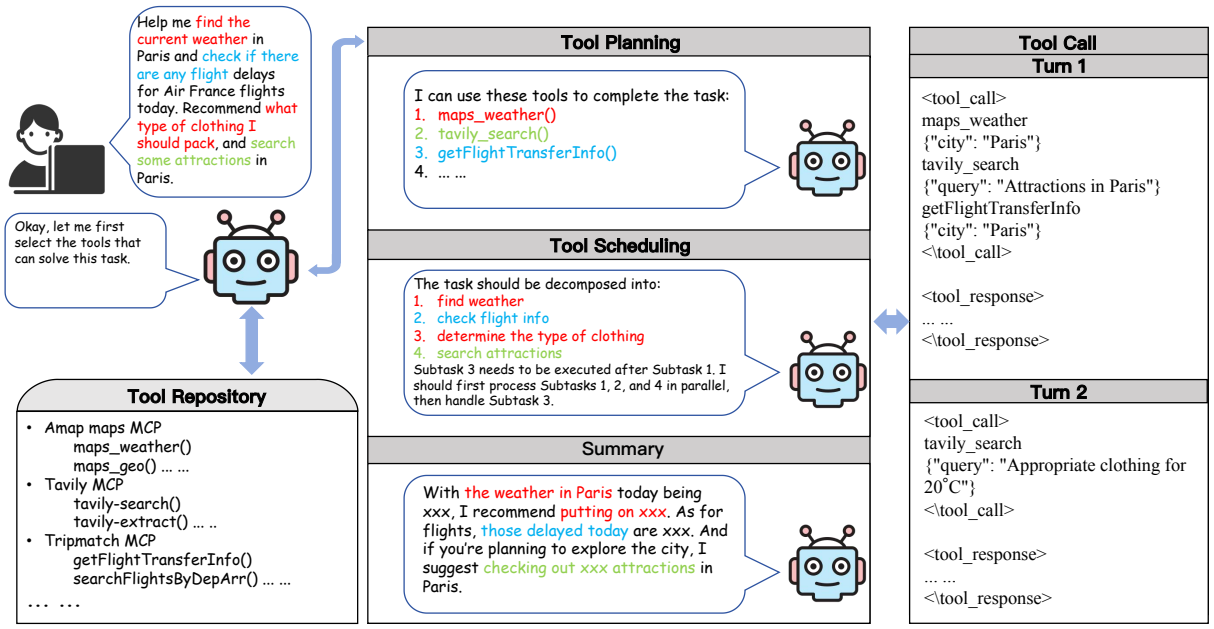

Figure 1: TPS-Bench assesses the tool planning and scheduling ability of LLM agents for solving compounding tasks. As shown, the LLM agent needs to first select tools capable of solving the task from a tool repository. Next, the LLM agent decomposes the original task into multiple subtasks and identifies their dependency relationships (subtasks that have no dependencies are marked by different colors in the figure). After that, the LLM agent performs tool calls and collects the tool responses in multiple turns, after which the final answer is delivered.

weakly relevant subtasks such as "*check tomorrow's weather and recommend autumn-appropriate recipes...*", and TPS-Bench-Hard, which comprises subtasks more compactly and implicitly: "*check the price fluctuations of Apple phones over the past year and recommend phones under 1,000 US dollars...*". Considering the involved tasks are open and no standard answers exist, we use LLM-as-a-judge (Zheng et al., 2023) to evaluate task completion rate. In addition, we systematically measure token usage and time consumption to assess the efficiency.

We perform extensive empirical studies with closed-source and open-source LLMs, including GPT-4o (OpenAI, 2024), Kimi-K2 (Bai et al., 2025), DeepSeek-R1 (DeepSeek-AI, 2025), GLM-4.5 (Zeng et al., 2025), QwQ-32B (Qwen, 2025), Qwen3-32B (Yang et al., 2025), and Qwen3-1.7B (Yang et al., 2025). We notice that most models exhibit reasonable performance in tool planning but behave differently in scheduling. E.g., GLM-4.5 tends to employ sequential tool execution, achieving the highest task completion rate of 64.72% for TPS-Bench-Hard, at an average cost of 217.8s and 14k tokens per query. In contrast, GPT-4o prioritizes parallel scheduling, which leads to 76.84s and 9k tokens per query, but the task completion rate is rarely 45.08%. Besides, Qwen3-32B stands as a strong open-source baseline for TPS-Bench. It consumes 8.0k average token usage and achieves a 56.72% task completion rate on TPS-Bench-Hard.

We then investigate reinforcement learning (RL)-based post-training to improve the task scheduling ability of LLMs for both task completion and efficiency, considering its effectiveness displayed in agentic workflows (Zhang et al., 2025). In particular, we construct a training set of 597 samples, and use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to train the cost-effective Qwen3-1.7B (Yang et al., 2025). With only 125 training iterations, we witness a 6% improvement in task completion rate alongside a 14% reduction in execution time.

2 Related Work

Evaluation of LLM-based agents. As LLMs pave a promising path for the development of general AI agents (Xi et al., 2025), recent years have witnessed a surge in benchmarks designed to evaluate the capabilities of LLM-based agents. These efforts span multiple dimensions: some studies assess agents' general problem-solving abilities across diverse environments (Liu et al., 2023; Yu

et al., 2025; Li et al., 2024; Trivedi et al., 2024; Barres et al., 2025) for broad-capability assessment. Some focus on evaluating LLM tool use effectiveness (Huang et al., 2023; Chen et al., 2023; Ye et al., 2024) as a key interactive capability. Others specialize in researching domain-specific agents' abilities (Jimenez et al., 2023; Guo et al., 2024; Xie et al., 2024) for professional-grade performance. However, these benchmarks primarily emphasize task completion effectiveness but pay limited attention to the efficiency metrics, such as tool usage, token consumption, and execution time, which are significant factors for the cost-effectiveness and scalability of agents in real-world tasks.

Efficiency in AI Agents. Efficient Agents (Wang et al., 2025a) is closest to our work. However, it concentrates exclusively on developing strategies to enhance the operational efficiency of agents in single-task or limited-tool scenarios. This gap is particularly critical as currently agents increasingly operate in strictly token-constrained or latency-sensitive environments (Miao et al., 2023). Recent studies have introduced various methods for resource-aware inference and efficiency optimization, such as adaptive computation (Zhou et al., 2024), cost-efficient tool use, and latency-aware decoding (Wang et al., 2025b). Our work further fills the aforementioned benchmark gap by proposing a dedicated efficiency-oriented benchmark system, which is specifically tailored to the unique functional demands and operational characteristics of our target multi-task, multi-tool agent settings.

Benchmark Construction and Task Designation. Several studies have explored benchmarks and task design for evaluating LLM agents. BIG-bench (Srivastava et al., 2023) provides diverse and challenging tasks for multi-faceted evaluation. SmartPlay (Wu et al., 2023) takes a game-based approach to evaluate LLM agents across 6 distinct games, each designed to challenge various key capabilities. More recently, AgentBoard (Chang et al., 2024) introduced a unified benchmark for multi-turn LLM agents, featuring 9 diverse tasks and various environments that require partially observable and multi-round interactions. However, these benchmarks predominantly rely on tasks with isolated objectives or simple sequential structures. To address this gap, TPS-Bench explicitly incorporates compound tasks with intricate subtask dependencies and clear parallelization potential, requiring agents not only to select appropriate tools but also to strategically schedule their execution for optimal efficiency. Furthermore, TPS-Bench introduces graded task difficulty levels—a feature notably absent in existing benchmarks.

MCP for Tool-Augmented Agents. The model context protocol (MCP) has emerged as a pivotal standard for enabling seamless and dynamic interactions between LLM agents and external tools, thereby expanding the scope and complexity of tasks agents can undertake (Singh et al., 2025; Ray, 2025). Although potential security concerns remain, recent studies have outlined future directions to address these challenges (Hou et al., 2025). In line with these developments, our work applies diverse MCP tools across categories, constructing a realistic, heterogeneous tool-fetching environment that closely mirrors complex real-world application scenarios. By strategically integrating MCP tools with a wide range of distinct, complementary functionalities, TPS-Bench provides a solid framework for assessing and advancing tool-augmented agents while reinforcing tool-planning and scheduling capabilities.

3 TPS-Bench

As illustrated in Figure 1, the tasks in TPS-Bench are a reasonable composition of multiple subtasks based on a large and diverse tool repository. We present more details below.

3.1 Benchmark Construction

To construct TPS-Bench, we need to first collect a wide range of subtasks. Considering the subtasks are actually determined by the tools at hand, we first collect and organize 15 model context protocol (MCP) servers, as shown in Figure 2 (right), each of which provides a set of complementary tools designed to work together. These MCP servers provide 141 tools spanning multiple domains.

Given such tools, we prompt LLMs to construct the corresponding subtasks that can be solved with the tool repository, as shown in Figure 3. Specifically, the descriptions of the tools are fed into the LLMs for subtask generation. Then, we compose these subtasks, forming two levels of compounding tasks with varying complexity. In detail, the tasks in **TPS-Bench-Easy** are the combination of simple and weakly relevant subtasks, and the number of subtasks is confined to be at most five. For **TPS-Bench-Hard**, we combine subtasks with strict dependency relationships and set the number limit of

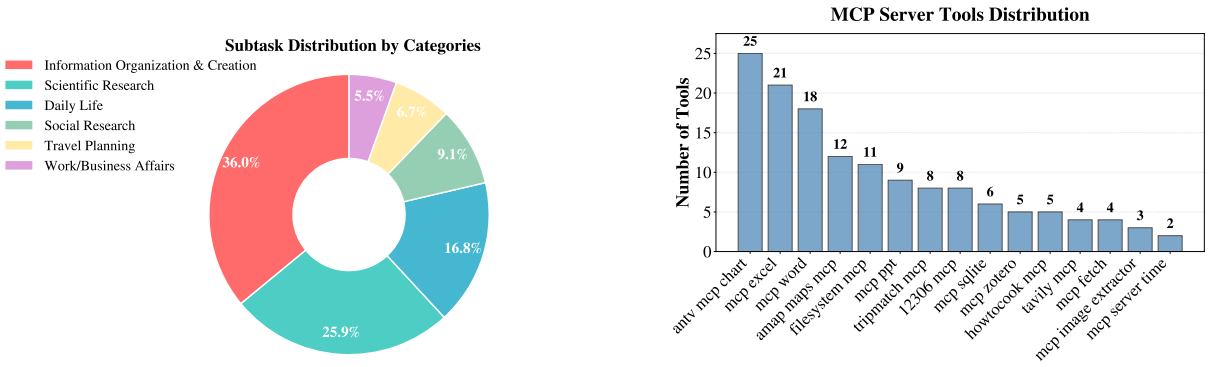

Figure 2: Statistics of subtasks and tools in TPS-Bench. **Left:** Distribution of all subtask categories in TPS-Bench. **Right:** Number of tools in each MCP Server used in TPS-Bench.

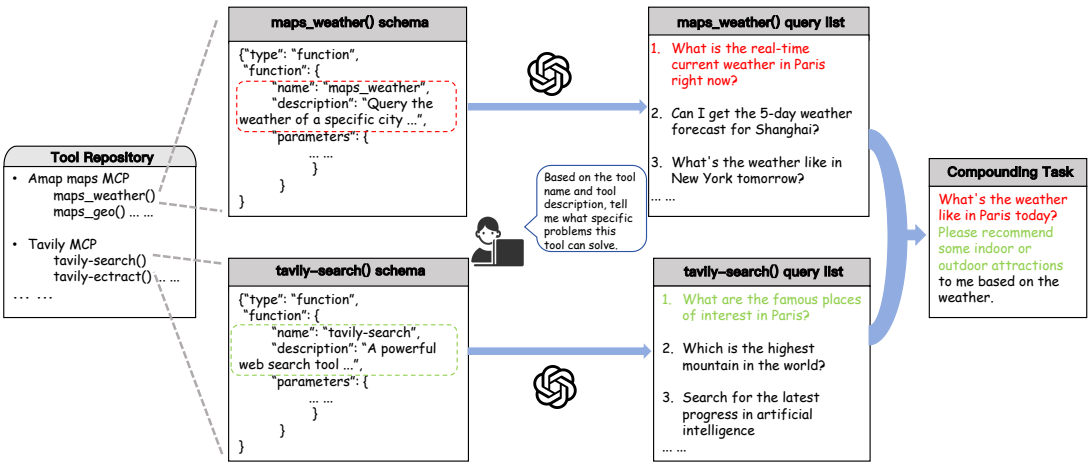

Figure 3: Task construction workflow in TPS-Bench. We send tool names and descriptions to LLMs to generate solvable subtasks, which are then combined by LLMs and inspected manually to form compounding tasks.

subtasks to 50. The combination is implemented by LLMs and then inspected manually.

3.2 Evaluation Process

To evaluate on TPS-Bench, the LLM agent first needs to perform tool planning—selecting appropriate tools for the task based on the name and description of the tools in our repository. We confine the number of selected tools to no more than **10** to ensure the task completion is efficient in terms of token consumption. Then, the LLM agent needs to decompose the provided task into a set of subtasks and systematically identify the underlying dependencies among them. Based on these dependency relations, the agent determines which tools should be invoked in the current execution turn. We encourage that independent subtasks are executed in parallel to improve efficiency. Given the received tool responses, the LLM agent then determines whether there is a need to launch another turn of tool calls. If not, it will summarize the context and return the result to the user.

3.3 Metrics

Task Completion Rate Considering that TPS-Bench tasks are relatively open-ended, we use LLM-as-a-judge (Zheng et al., 2023) to evaluate task completion rate. To be specific, we introduce a third-party LLM to first decompose the task into a sequence of subtasks, then judge whether each subtask has been completed, and finally calculate the task completion rate and provide a summary. The complete prompt can be found in Section A.

Tool Selection Score To assess LLM agents' task planning capabilities, we also evaluate the tool selection score by LLM-as-a-judge. After tool selection, we retrieve the corresponding tool description from the selected tools and submit the query and tool description to LLM for judgment.

We also estimate token and end-to-end time

consumption, as well as the number of tool call turns, to measure the efficiency of the scheduling. Regarding time, for open-source LLMs served by ourselves, the evaluation is conducted on vLLM (Kwon et al., 2023) with four NVIDIA A100 GPUs. Otherwise, we report the time measured with the official APIs.

4 Experiment

The section presents the evaluation results of mainstream LLMs on TPS-Bench.

4.1 Setup

LLMs We consider diverse LLMs in our analysis, including GPT-4o (OpenAI, 2024), Kimi-K2 (Bai et al., 2025), DeepSeek-R1 (DeepSeek-AI, 2025), GLM-4.5 (Zeng et al., 2025), QwQ-32B (Qwen, 2025), Qwen3-32B (Yang et al., 2025), and Qwen3-1.7B (Yang et al., 2025).

LLM-as-a-judge We employ Gemini-2.5-Flash (Comanici et al., 2025) as the LLM-as-a-judge to evaluate both the task completion rate and tool selection score. To verify its reliability, we provide the correlation between human evaluations and the LLM-as-a-judge in Figure 4. As shown, the Pearson correlation coefficient is 0.8375. We also measure the correlation between the number of subtasks decomposed by the LLM and by humans for the same tasks, and observe a Pearson coefficient of 0.7590. The strong positive correlations (0.8375 and 0.7590) demonstrate that LLM performance is closely aligned with human judgments. This indicates that TPS-Bench provides a reliable benchmark for evaluating both task success and decomposition quality.

4.2 Main Results

Reasoning and Planning Abilities Table 1 reports the tool selection score and task completion rate of 7 representative models. The results show that large reasoning models (LRMs) like Kimi-K2, DeepSeek-R1, and GLM-4.5 achieve a task completion rate of 52.29%, 62.03%, and 64.72%, respectively, which have comprehensively outperformed GPT-4o. Among them, GLM-4.5 has maintained the highest completion rate of 64.72% on TPS-Bench-Hard, demonstrating both robust reasoning and consistent decision-making abilities across diverse tasks. However, LRMs clearly consume significantly more tokens and time—for instance, DeepSeek-R1 takes an average of 343s and 8k tokens to complete a single task. It reflects the performance differences between models with varying planning and reasoning capabilities on compounding tasks. Furthermore, we found that although Qwen3-32B has a much smaller parameter size than Kimi-K2 and GPT-4o, it still maintains competent performance in the task completion rate of 56.72%. Such efficiency, coupled with its relatively lightweight architecture, may be attributed to its training on agentic data.

Different Scheduling Strategies Affect Efficiency Table 1 indicates that most models exhibit reasonable performance in tool planning but behave differently in scheduling strategies. In particular, GLM-4.5 requires an average of up to 35 tool call turns per task on TPS-Bench-Hard. As shown in Table 1, GLM-4.5 performs a considerable number of tool call turns under both difficulty levels. We note that GLM-4.5 tends to perform tool execution in a strictly sequential manner, and it achieves a task completion rate as high as 64.72%. However, GLM-4.5 executes an average of 35 tool call turns per task, takes 217.8 seconds, and uses as high as 12.6k input tokens, exhibiting poor efficiency. In contrast, GPT-4o tends to execute tool execution in parallel, with an average of 2 tool call turns per task. While the time required is significantly reduced, its task completion rate is only 45.08%. As for QwQ-32B, it struggles to clearly distinguish dependencies among subtasks and often invokes multiple tools simultaneously, even when certain tools rely on the outputs of others. This behavior results in only one tool call turn being produced and leads to the low task completion rate of 29.36%.

Cost To better capture the trade-off between monetary cost and task performance, we further collect and report the cost-of-pass for each model in Table 2. The cost-of-pass denoted as $v(m, t)$, represents the expected monetary cost of employing a model m to complete a task t. This metric allows a fair comparison between models with distinct pricing structures and completion capabilities. It is formulated as the ratio between the overall cost of completing a task once, $C_m(t)$, and the corresponding task completion rate, $R_m(t)$:

$$v(m, t) = \frac{C_m(t)}{R_m(t)}. \quad (1)$$

The total cost of completing a task once, $C_m(t)$,

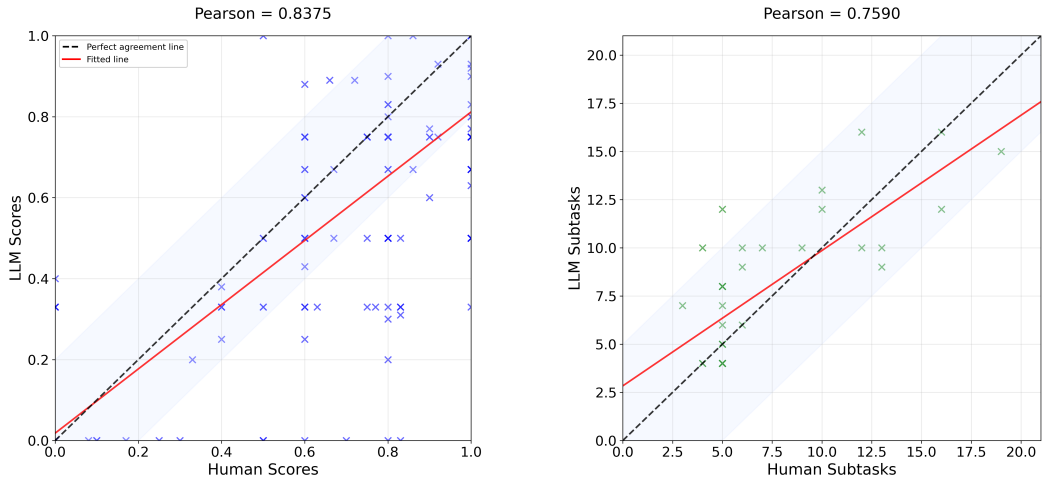

Figure 4: **Left:** Pearson correlation between LLM scores and Human scores on Task Completion Rate. The fitted line (red) shows the linear regression trend, while the dashed black line indicates great agreement between LLMs and humans. **Right:** Pearson correlation between LLM and Humans regarding the number of subtasks decomposed from the same task.

Model	Effectiveness		Efficiency			
	Tool Selection Score↑	Task Completion Rate↑	# Input Tokens↓	# Output Tokens↓	Turns↓	Time↓
TPS-Bench-Easy						
GPT-4o (OpenAI, 2024)	94.09%	52.44%	7.2k	0.8k	2.2	51.2
Kimi-K2 (Bai et al., 2025)	92.87%	73.54%	6.5k	0.8k	3.7	103.2
DeepSeek-R1 (DeepSeek-AI, 2025)	92.19%	68.52%	6.5k	1.1k	2.8	329.0
GLM-4.5 (Zeng et al., 2025)	87.72%	64.91%	11.1k	1.2k	9.5	123.0
QwQ-32B (Qwen, 2025)	85.36%	32.65%	6.5k	1.1k	1.0	217.6
Qwen3-32B (Yang et al., 2025)	93.03%	61.60%	6.3k	0.9k	2.5	202.3
Qwen3-1.7B (Yang et al., 2025)	78.03%	33.35%	7.3k	1.1k	2.3	22.2
TPS-Bench-Hard						
GPT-4o (OpenAI, 2024)	87.12%	45.08%	7.2k	1.3k	2.5	76.84
Kimi-K2 (Bai et al., 2025)	84.49%	52.29%	6.5k	1.1k	19.7	216.5
DeepSeek-R1 (DeepSeek-AI, 2025)	84.41%	62.03%	6.7k	1.4k	6.0	343.4
GLM-4.5 (Zeng et al., 2025)	79.31%	64.72%	12.6k	1.5k	35.0	217.8
QwQ-32B (Qwen, 2025)	70.74%	29.36%	6.7k	1.6k	1.05	171.0
Qwen3-32B (Yang et al., 2025)	85.36%	56.72%	6.7k	1.3k	3.1	226.2
Qwen3-1.7B (Yang et al., 2025)	65.26%	26.75%	7.8k	2.2k	2.4	42.0

Table 1: The main results of testing 7 representative models on TPS-Bench are presented here. We report tool selection score and task completion rate (higher is better), and token usage (Input Tokens, Output Tokens; lower is better). Regarding execution time, for the Qwen3-1.7B model, the evaluation was conducted locally based on vLLM, whereas for all other models, the time was obtained via API calls in their default settings.

is given by:

$$C_m(t) = n_{\text{in}}(m,t) \cdot c_{\text{in}}(m) + n_{\text{out}}(m,t) \cdot c_{\text{out}}(m), \quad (2)$$

where $n_{\text{in}}(m,t)$ and $n_{\text{out}}(m,t)$ denote the number of input and output tokens consumed by model m on task t, while $c_{\text{in}}(m)$ and $c_{\text{out}}(m)$ are their respective per-token costs.

As the table, GPT-4o incurred the highest cost, reaching 52.2×10^{-3}\$, and the highest cost-of-pass, reaching 138.0×10^{-3}\$, primarily due to its premium pricing. In contrast, QwQ-32B required a low cost, yet its cost-of-pass was six times higher because of its low task completion rate.

4.3 Analysis and Ablation

The strong performance of our models on TPS-Bench primarily depends on two factors: the ability to select appropriate tools and the ability to schedule them correctly in sequential and parallel workflows. To gain a deeper understanding of how each of these factors affects the model's tool planning and scheduling capabilities, we conducted two corresponding ablation studies.

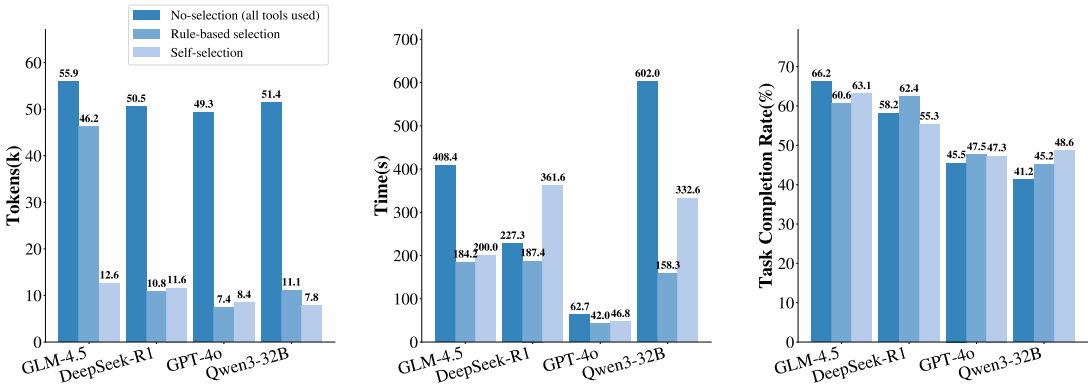

Figure 5: Evaluation of three tool selection strategies, no-selection, rule-based selection, and self-selection, was performed by applying each strategy to the tasks in TPS-Bench-Hard. The four models, GLM-4.5, DeepSeek-R1, GPT-4o, and Qwen3-32B, are tested with all three strategies across the benchmark. Efficiency is reflected by the total number of tokens consumed and the time required to complete each task, while effectiveness is reflected by the task completion rate.

Model	TPS-Bench-Easy		TPS-Bench-Hard	
	Cost↓	Cost-of-Pass↓	Cost↓	Cost-of-Pass↓
GPT-4o	52.6	100.3	62.2	138.0
Kimi-K2	5.9	8.0	6.5	12.4
DeepSeek-R1	5.8	8.5	6.7	10.8
GLM-4.5	9.8	8.5	11.3	17.5
QwQ-32B	2.7	15.3	3.2	21.2
Qwen3-32B	4.3	7.0	5.5	9.7
Qwen3-1.7B	0.8	2.4	1.3	4.9

Table 2: The average monetary cost of the seven models on TPS-Bench, including the average cost(10^{-3}\$) and cost-of-pass for tasks at each level. More information about the prices of each model can be found in Appendix B.

The Impact of Tool Selection Strategy In previous experiments, we all had the model select tools on its own, and we refer to this strategy as "self-selection". To investigate the impact of different tool selection strategies, we designed two alternative tool selection strategies for comparison.

- **No selection (all tools used) :** In this setting, no tool selection is performed before task execution; instead, all available tool schemas are provided to the model directly.
- **Rule-based selection:** In this setting, the task content is compared with the tool names and descriptions using word-based similarity measures. The top-10 most similar tools are then selected and provided to the model for task execution.

For this ablation experiment, we employed GLM-4.5, DeepSeek-R1, GPT-4o, and Qwen3-32B. As Figure 5, it shows different tool selection strategies do not lead to substantial differences in task completion rate, but notable differences are observed in the total number of tokens consumed and the time required. The no-selection strategy, which inputs a large number of tool schemas, results in a token count significantly higher than the other two strategies. Additionally, since the model needs to select tools on its own, both the no-selection and self-selection strategies require more time. These findings indicate that although the task completion rate is largely unaffected by the choice of tool selection strategy, the efficiency of each strategy varies significantly, emphasizing the importance of appropriate tool selection.

Tool Scheduling To compare the advantages and disadvantages of serial and parallel tool scheduling in terms of both effectiveness and efficiency, we conducted experiments using four models. As Figure 6 shows, serial scheduling strategies generally consume more tokens and time than parallel strategies. For instance, DeepSeek-R1's token usage increased from 11.6k to 12.6k, and time consumption rose from 361.6 seconds to 423.6 seconds. During serial execution, the model engages in additional reasoning steps, allowing the model to evaluate subtask dependencies, potentially avoiding cascading errors. Thereby, the task completion rate is improved, for example, GLM-4.5 increased from 63.1% to 71.8%. Serial scheduling can effectively help the model reduce errors caused by incorrect judgments of subtask dependencies. These results

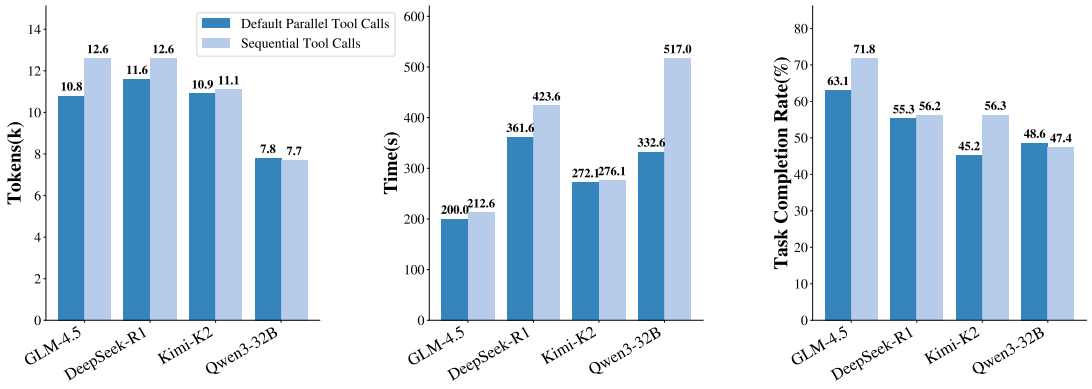

Figure 6: Evaluation of two tool scheduling strategies, default parallel tool calls and sequential tool calls, was conducted on TPS-Bench-Hard. We tested four models, GLM-4.5, DeepSeek-R1, Kimi-K2, and Qwen3-32B, applying both scheduling strategies across the benchmark to ensure comparability. Differences between the two strategies are reflected in the number of tokens consumed, the time required, and the task completion rate, thereby illustrating the trade-offs between efficiency and effectiveness when tools are invoked in parallel versus sequentially.

Model	Method	TPS-Bench-Hard					
		Tool Sel. Score	Task Comp. Rate	# Input Tokens	# Output Tokens	Turn	Time
Qwen3-1.7B	origin	65.26%	26.75%	7.8k	2.2k	2.4	42.0
	GRPO	82.34%	35.17%	7.5k	1.6k	1.9	34.8

Table 3: Results of Qwen3-1.7B on TPS-Bench-Hard after tuning on 597 RL samples

reveal a trade-off between efficiency and effectiveness, requiring a balance between computational cost and task reliability in scheduling choices.

5 Reinforcement Learning

To improve the scheduling ability without compromising performance, especially in complex and multi-tool scenarios, considering the trade-off between model performance during sequential tool calls and efficiency in parallel execution, this section presents an initial study that uses Qwen3-1.7B (Yang et al., 2025) to enhance scheduling efficiency through RL training.

Training Details We utilize Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to train Qwen3-1.7B on 597 training samples over five epochs with actor learning rate of 1e-6. For each step, 5 roll-out completions are generated for each sample, with all other hyperparameters set to the default values from veRL (Sheng et al., 2024). The reward signal is derived from Gemini-2.5 flash (Comanici et al., 2025), which scores the model on task completion and the degree of parallelism. This specific reward design directly incentivizes the model to balance effectiveness and efficiency.

Result Analysis As shown in Table 3, with only 597 training samples, the GRPO-trained Qwen3-1.7B model demonstrates remarkable improvements across key metrics on the TPS-Bench-Hard. This is evidenced as the model improves the average task completion rate by 6% while simultaneously reduces execution time by 14% in TPS-Bench-Hard. Notably, compared with the baseline, the model requires fewer tool call turns and generates fewer output tokens, which indicates more parallel tool execution, suggesting that reinforcement learning can effectively guide the model toward more structured and parallel execution patterns.

6 Limitation

We propose TPS-Bench to evaluate models' tool planning and scheduling on compound tasks. Through TPS-Bench, our work advances LLM agents' tool planning and scheduling capabilities, providing a new direction for addressing complex real-world problems effectively and efficiently.

Future research will gravitate toward two key directions: richer tool selection, and adaptive tool selection by large models based on user memory and cost. This study pertaining to these directions remains to be supplemented and expanded.

Acknowledgements

This work was supported by Shanghai Key Technology R&D Program "New Generation of Information Technology" (No. 25511103700), NSF of China (Nos. 62306176, 92470118), CCF-ALIMAMA TECH Kangaroo Fund (NO. CCF-ALIMAMA OF 2025010), and Ant Group.

References

Y. Bai, Kimi Team, Y. Bao, and et al. 2025. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and 1 others. 2023. T-eval: Evaluating the tool utilization capability of large language models step by step. *arXiv preprint arXiv:2312.14033*.

G. Comanici and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. https://arxiv.org/abs/2507.06261. ArXiv preprint.

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

Zishan Guo, Yufei Huang, and Deyi Xiong. 2024. Ctooleval: A chinese benchmark for llm-powered agent evaluation in real-world api interactions. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 15711–15724.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Xiangru Tang, and 8 others. 2024. Data interpreter: An llm agent for data science. *Preprint*, arXiv:2402.18679.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and 1 others. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, and 1 others. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards efficient generative large language model serving: A survey from algorithms to systems. *ACM Computing Surveys*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

OpenAI. 2024. GPT-4o system card. *Preprint*, arXiv:2410.21276. Please cite as OpenAI (2024).

OpenAI. 2025. Introducing deep research. https://openai.com/index/introducing-deep-research/. Accessed 2025-09-23.

OpenAI. 2025. Introducing gpt-5. https://openai.com/index/introducing-gpt-5/. Accessed 2025-09-24.

OpenAI. 2025. Openai o3 and o4-mini system card. https://openai.com/index/o3-o4-mini-system-card/. Accessed: 2025-09-24.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Qwen. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Partha Pratim Ray. 2025. A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. *Authorea Preprints*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. A survey of the model context protocol (mcp): Standardizing context to enhance large language models (llms).

Aarohi Srivastava, Abhinav Rastogi, Rao, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*.

Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. 2025a. Efficient agents: Building effective agents while reducing cost. *Preprint*, arXiv:2508.02694.

Zili Wang, Tianyu Zhang, Lei Zhu, Haoli Bai, Lu Hou, Shiming Xiang, Xianzhi Yu, and Wulong Liu. 2025b. Faster and better llms via latency-aware test-time scaling. *arXiv preprint arXiv:2505.19634*.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. In *Advances in Neural Information Processing Systems*, volume 37, pages 25981–26010. Curran Associates, Inc.

Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. 2023. Smartplay: A benchmark for llms as intelligent agents. *arXiv preprint arXiv:2310.01557*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Preprint*, arXiv:2405.15793.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, and 1 others. 2024. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. Assistantbench: Can web agents solve realistic and time-consuming tasks? *Preprint*, arXiv:2407.15711.

Peijie Yu, Yifan Yang, Jinjian Li, Zelong Zhang, Haorui Wang, Xiao Feng, and Feng Zhang. 2025. c^3-bench: The things real disturbing llm based agent in multi-tasking. *arXiv preprint arXiv:2505.18746*.

Aohan Zeng and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. Aflow: Automating agentic workflow generation. *Preprint*, arXiv:2410.10762.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, and 1 others. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

A PROMPT FOR TPS-Bench

A.1 Prompt for Tool Selection

This section presents the prompt designed for the tool selection module. The primary objective of this prompt is to instruct the AI assistant to act as a tool selection expert, capable of identifying and outputting the most relevant tools from a predefined list based on a given user query. The detailed prompt content is shown as Figure 7.

Prompt for Tool Selection

You are a tool selection expert. Please select the most relevant tools from the provided tool list based on the user's query.

User query: {query}

IMPORTANT:

- Select only the tools that are necessary and relevant to complete the task
- Maximum 10 tools, but you can select fewer if appropriate
- Output format must be exactly like this (one tool name per line):

|tool_name_1|

|tool_name_2|

|tool_name_3|

|...|

No explanations, no analysis, no additional text.

Figure 7: Prompt format for tool selection.

A.2 Prompt for Tool Scheduling

This section introduces the prompt for guiding tool selection and task execution logic, which serves as the core instruction for the assistant to handle tool scheduling, parallel processing, and summarize the results of the task. The detailed prompt content is shown as Figure 8.

A.3 Prompt for Evaluating Task Completion Rate

This prompt is designed to evaluate how well an AI model completes a given task by decomposing it into fine-grained, verifiable subtasks and assessing their completion individually as Figure 9.

A.4 Prompt for Evaluating Tool Selection Score

This prompt is designed to assess the appropriateness of the model's tool selection, establishing a standardized framework to measure whether the selected tools align with the user's query and task requirements. The detailed prompt content is shown as Figure 10.

Table 4: Pricing of Each Model per 1M Tokens of **July 30, 2025** (Unit: $)

Model Name	Input Pricing (per 1M tokens)	Output Pricing (per 1M tokens)
GPT-4o	5.00	20.00
Kimi-K2	0.60	2.50
DeepSeek-R1	0.55	2.19
GLM-4.5	0.60	2.50
Qwen3-32B	0.28	2.79
QwQ-32B	0.28	0.84
Qwen3-1.7B	0.042	0.42

B PRICE OF MODELS

Table 4 presents the token-based pricing of seven mainstream large language models as of July 30, 2025, with the pricing unit specified as US dollars per 1 million tokens.

C DATA CONSTRUCTION

We provide additional details on data construction beyond Section 3.1. Starting from the tool-grounded subtask composition pipeline illustrated in Figure 3, we generated approximately 1,000 candidate tasks. To ensure data quality, five annotators manually reviewed and screened these instances. We excluded tasks if they (1) could be solved using only one tool, (2) could not be addressed by the current tool library, or (3) were overly complex, excessively verbose, or logically inconsistent.

From the screened task pool, we selected 100 tasks to construct TPS-Bench-Easy and 100 tasks to construct TPS-Bench-Hard, resulting in a final evaluation benchmark of 200 tasks. In addition to the 200 evaluation tasks, we further selected 597 task instances from the candidate pool for RL training. These 597 instances were used only as RL training data and were kept separate from the evaluation benchmark.

Prompt for Tool Scheduling

You are a helpful assistant specialized in efficient task execution and parallel processing.

Tool Usage Instructions:
If you need to search for something, please set max-result==5.

Task Execution Framework:
When solving a user's problem, please strictly follow these steps:

1. Task Analysis and Decomposition
- Break down the user's question into specific, manageable sub-tasks
- Each sub-task should have a single, clear objective
- Ensure sub-tasks are granular enough to be independently executed and verified
- Consider different decomposition strategies: functional modules, data flow, temporal sequence

2. Dependency Analysis
For each sub-task, identify:
- **Data dependencies**: Tasks requiring output from other tasks
- **Logical dependencies**: Tasks with sequential execution requirements
- **Resource dependencies**: Tasks competing for the same resources
- Create a dependency graph to visualize relationships

3. Execution Planning
- **Sequential tasks**: Must be executed in order due to dependencies
- **Parallel tasks**: Can be executed simultaneously with resource considerations
- **Priority assignment**: Critical path tasks get higher priority
- **Resource allocation**: Balance CPU, memory, and network usage
- **Error handling**: Define fallback strategies for failed parallel tasks

4. Execution Monitoring
- Track progress of each sub-task
- Monitor resource usage and performance
- Implement timeout mechanisms
- Prepare for dynamic plan adjustments

5. Task Completion Summary
- When you complete tasks, provide detailed descriptions of each sub-task performed
- Explain the specific actions taken and results achieved for every sub-task
- Include any challenges encountered and how they were resolved
- **IMPORTANT: Always provide your complete answers and solutions in the final response, regardless of whether you save them to docx, pptx, or other files. Users should be able to understand your complete solution process and results without needing to check external files**
- Summarize the overall completion status and final outcomes

CRITICAL EXECUTION REQUIREMENTS:
- **SILENT EXECUTION**: Do NOT output intermediate steps, progress updates, or ask "what should I do next?"
- **COMPLETE ALL WORK**: Execute all necessary sub-tasks in one continuous session without stopping
- **FINAL SUMMARY ONLY**: Only provide the comprehensive final summary with all results, data, and conclusions
- **NO INTERRUPTIONS**: Never pause to ask for user input or next steps - complete everything autonomously
- **COMPREHENSIVE OUTPUT**: Include all research findings, analysis, data, and conclusions in your final response

Execute all tasks silently and provide only the complete final summary with all results.

Before each operation, explain your plan and reasoning in natural language, then proceed to call the tool(s). You may call multiple tools in parallel for independent tasks, but always consider resource constraints and error handling.

Figure 8: Prompt format for tool scheduling.

Prompt for Evaluating Task Completion Rate

You are a professional evaluation assistant. Your task is to assess the task completion rate of AI models.

Task Context

{query}

Instructions

Step 1: Task Decomposition

Based on the task context above, break down the overall task into granular, indivisible subtasks. Each subtask should be:

- Specific and measurable
- Independently verifiable
- Cannot be further subdivided

Please list all subtasks clearly with numbers (1, 2, 3, etc.)

Step 2: Model Response Analysis

Model Response:

\{result\}

Special Case: Empty Response

If the model response (result) is empty, null, or contains no meaningful content, then all subtasks are incomplete.

Special Case: Tool Calls

If the model response has |<tool_call>| tags, then the relative subtasks are incomplete.

Step 3: Completion Assessment

For each subtask you identified in Step 1, analyze whether it was completed in the model response:

- Completed successfully: Provide evidence from the response
- Not completed: Explain why it wasn't addressed
- Partially completed: Explain what was done and what was missing

Step 4: Final Scoring

Calculate the completion percentage based on fully completed subtasks only.

Output Format

Please provide a detailed analysis in the following format:

SUBTASK DECOMPOSITION:
1. First subtask
2. Second subtask
...
Total subtasks: X

COMPLETION ANALYSIS:
1. Subtask 1 - Explanation with evidence
2. Subtask 2 - Explanation with evidence
...

FINAL RESULT:
- Completed subtasks: X out of Y
- Completion rate: XX\%
- Summary: Brief explanation of overall performance

Figure 9: Prompt for evaluating task completion rate.

Prompt for Evaluating Tool Selection Score

You are a tool selection evaluation expert. Please evaluate the quality of the model's tool selection based on the following criteria:

Evaluation Task:
User Query:
\{query\}
Model Selected Tools:
\{selected tools\}
Available Tools List:
\{available tools\}

Evaluation Criteria:

1. **Tool Matching Assessment (50 points)**
- Do the selected tools directly address the specific needs in the user query?
- Do the selected tool functions match the task requirements?
- Are there any obvious missing essential tools?
- Is the logic of tool selection reasonable?

2. **Efficiency Assessment (30 points)**
- Were redundant or duplicate-function tools selected?
- Is the number of tools reasonable (maximum 10)?
- Is there a more concise tool combination solution?

3. **Completeness Assessment (20 points)**
- Does it cover all subtasks in the user query?
- Can the tool combination form a complete solution?

Scoring Rules:
- Total score: 100 points, allocated according to the above weights
- Exceeding 10 tools: deduct 5 points for each additional tool
- Missing critical tools: deduct 10 points for each missing tool
- Selecting irrelevant tools: deduct 8 points for each irrelevant tool

Output Format:
{{
"total score": X,
}}

Please conduct the evaluation strictly according to the above criteria and provide an objective, detailed analysis.

Figure 10: Prompt for evaluating task selection rate.