Enhancing LLM-Based Agents via Global Planning and Hierarchical Execution

Junjie Chen[1], Haitao Li[1], Jingli Yang, Yiqun Liu[1], Qingyao Ai[1*]

[1]DCST, Tsinghua University, Beijing, 100084, China.

*Corresponding author(s). E-mail(s): aiqingyao@gmail.com;

Abstract

Intelligent agent systems based on Large Language Models (LLMs) have shown great potential in real-world applications. However, existing agent frameworks still face critical limitations in task planning and execution, restricting their effectiveness and generalizability. Specifically, current planning methods often lack clear global goals—leading agents to get stuck in local branches—or produce non-executable plans. Meanwhile, existing execution mechanisms struggle to balance complexity and stability, and their limited action space restricts their ability to handle diverse real-world tasks. To address these limitations, we propose GoalAct, a novel agent framework that introduces a continuously updated **global planning** mechanism and integrates a **hierarchical execution** strategy. GoalAct decomposes task execution into high-level skills, including searching, coding, writing and more, thereby reducing planning complexity while enhancing the agents' adaptability across diverse task scenarios. We evaluate GoalAct on LegalAgentBench, a benchmark with multiple types of legal tasks that require the use of multiple types of tools. Experimental results demonstrate that GoalAct achieves state-of-the-art (SOTA) performance, with an average improvement of 12.22% in success rate. These findings highlight GoalAct's potential to drive the development of more advanced intelligent agent systems, making them more effective across complex real-world applications. Our code can be found at https://github.com/cjj826/GoalAct.

Keywords: Large Language Model, Agent, Global Planning, Hierarchical Execution

1 Introduction

In recent years, intelligent agent systems based on Large Language Models (LLMs) have shown remarkable potential in practical applications [1, 2]. This advancement comes mainly from two aspects: (1) LLMs have continuously improved their core capabilities, such as instruction-following, tool utilization, programming, writing, and logical reasoning [3–5]; (2) the ongoing advancement of agent frameworks (e.g., Plan-and-Solve [6], ReAct [7], CodeAct [8]) has enabled agents to autonomously perform task planning (what to do) and task execution (how to do), facilitating the achievement of target objectives through interactions with the environment.

Although existing agent frameworks demonstrate a certain level of autonomy in **task planning and execution**, substantial limitations remain within these two critical stages:

Firstly, at the planning level, current methods either lack clear global goals or generate plans that are hard to execute. Some frameworks, such as ReAct, adopt an incremental reasoning approach of "Thought-Action-Observation", focusing only on the immediate step without a comprehensive global perspective. Consequently, these frameworks frequently become stuck in local optima during tasks involving multiple branches. In contrast, methods like Plan-and-Execute [9] attempt to enhance task-solving capabilities by first generating a global plan and subsequently executing it, while dynamically adjusting the plan based on feedback obtained during execution. However, such methods often fail to effectively integrate concrete executable actions into the global plan, resulting in plans that exceed the agents' action space.

Secondly, at the execution level, existing methods face a trade-off between complexity and stability, while also being constrained by a limited action space. Frameworks like ReAct primarily rely on **text or json formats** to invoke external tools. While these formats offer a straightforward mechanism for tool interaction, they lack the capability to handle complex logic structures, such as loops and conditional branches. CodeAct attempts to unify the agent's action space using **python code**, allowing for more sophisticated tool invocation logic. However, despite its greater expressive power, it increases execution complexity. In real-world scenarios where tool invocation outcomes are inherently unpredictable, highly complex invocation logic is more prone to errors, making the system less stable. Moreover, not all tasks can be effectively solved using code alone, such as advanced writing tasks (e.g., legal document generation [10]) and logical/mathematical reasoning (e.g., Lateral Thinking Puzzles [11]).

These limitations lead to discrepancies between the agents' intended decisions and their actual executed behaviors, thereby constraining both the effectiveness and generalizability of these agents when addressing complex real-world tasks. To address the above issues, we propose the **GoalAct** framework, aiming to enhance LLM-based agents via global planning and hierarchical execution:

- **Global planning:** we introduce a continuously updatable global planning mechanism that tightly couples planning and execution, enabling agents to establish clearer long-term goals while ensuring the feasibility of plans.
- **Hierarchical execution:** we argue that the complexity of real-world actions cannot be effectively managed by **existing single-stage execution methods**, which

require agents to simultaneously determine the appropriate skills or methods, select relevant tools, and configure their parameters. Inspired by human practice—first identifying high-level skills, then selecting suitable tools, and finally refining execution details—we propose a hierarchical execution framework that decomposes task execution into distinct high-level skills, such as searching, coding, writing, reasoning and more. This hierarchical structure offers two key advantages: (1) it significantly simplifies global planning, as the plan only needs to specify appropriate high-level skills and their objectives rather than low-level details; and (2) it is inherently scalable, enabling the dynamic addition and selection of skills to flexibly adapt to diverse and evolving task scenarios.

To evaluate the effectiveness of GoalAct, we conducted experiments using LegalAgentBench [12], a novel benchmark that poses no risk of data leakage and requires external tool invocation and legal knowledge for task completion. Experimental results demonstrate that GoalAct achieves state-of-the-art (SOTA) performance, with an average improvement of 12.22% in success rate.

2 Related Work

2.1 Large Language Models

Large Language Models (LLMs) have experienced rapid development in recent years, significantly advancing natural language understanding and generation capabilities. Models like ChatGLM [13, 14], Qwen [15] and GPT [4] demonstrate remarkable abilities in language comprehension, multi-turn dialogue management, and instruction following [4, 16]. Techniques such as contextual learning, in-context instruction tuning, and chain-of-thought reasoning have enabled LLMs to handle complex reasoning tasks more effectively [17–19]. These advances have led to LLMs being integrated into various downstream applications, from knowledge-based question answering to tool-use scenarios, thereby driving the development of more sophisticated intelligent agents.

2.2 LLM-Based Agents

Leveraging the advancements of LLMs, researchers have built agents capable of reasoning, planning, and interacting within complex environments. Approaches such as ReAct [7] combine reasoning and acting to allow agents to interact iteratively with external tools, enhancing the agent's factuality and problem-solving capabilities. Voyager [20] introduces code-based action execution, enabling an LLM agent to operate effectively within dynamic game environments by generating executable python code to navigate and interact within Minecraft. Similarly, TaskWeaver [21] translates user instructions into executable code for efficient task completion in structured data analysis environments. Other frameworks like CodeAct [8] emphasize tool integration, enabling agents to leverage external coding environments for complex problem-solving tasks, significantly enhancing their operational flexibility. Despite these promising developments, current LLM-based agent frameworks still face limitations in task planning and execution, highlighting ongoing research opportunities.

3 Methodology

Figure 1 shows the framework of our GoalAct. Next, we introduce it in detail.

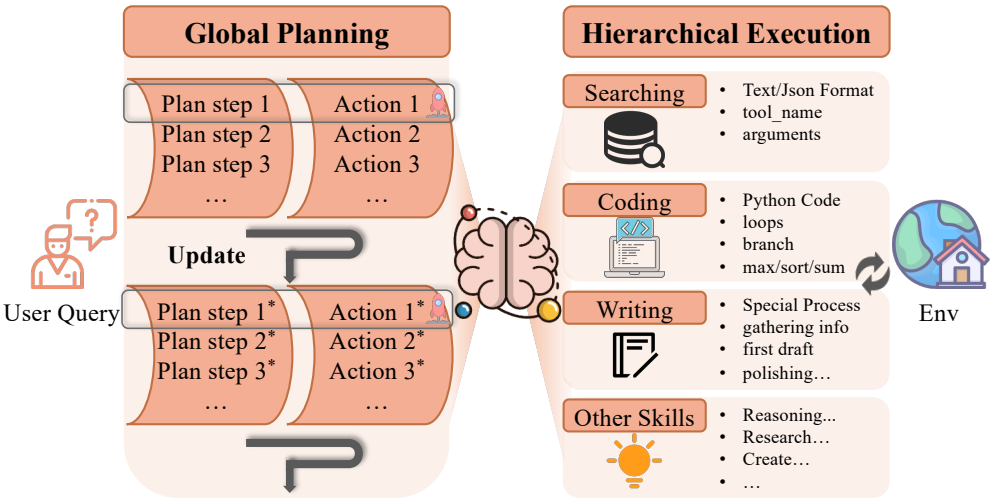

Fig. 1 The framework of our GoalAct. It emphasizes continuously updatable **global planning**, which harmonizes the clarity of long-term goals with actionable steps, and **hierarchical execution**, which decomposes task execution into high-level skills.

3.1 Global Planning

In our GoalAct framework, we continuously maintain and update a global plan G, defined as Equation (1) shows:

$$G = (P_1A_1, P_2A_2, P_3A_3, \ldots, P_nA_n) \tag{1}$$

Here, P_i denotes the i-th plan step, and A_i represents its corresponding action. Since providing detailed action descriptions directly within the global plan is challenging, we only require the specification of high-level skills, which will be introduced next in the section 3.2. The final step, A_n, is always designated as $Finish$.

At each time step t, the agent updates G according to Equation (2):

$$G_t = \pi(Q \mid T \mid S_t) \tag{2}$$

where π is the update policy (detailed in the Table 4), Q is the user query, T is the set of available tools, and S_t is the historical record at time t, as Equation (3) shows:

$$S_t = (P_1A_1O_1, P_2A_2O_2, \ldots, P_{t-1}A_{t-1}O_{t-1}) \tag{3}$$

Here, O_i represents the observation from the execution of A_i. When $t = 1$, S_t is empty.

Compared to existing methods, GoalAct tightly integrates global planning with execution through the continuously updated global plan. By dynamically adjusting the plan based on real-time feedback, it ensures both a coherent overall strategy and practical executability, ultimately enhancing the agents across diverse scenarios.

3.2 Hierarchical Execution

To reduce the complexity of considering executable actions within global planning and to enable the agent to flexibly adapt its action complexity based on task difficulty, we introduce a hierarchical execution framework. As stated in section 1, we argue that existing single-stage execution methods struggle to manage real-world action complexity, as they require agents to simultaneously determine appropriate skills or techniques, select tools and configure them. Inspired by human action processes—first identifying necessary skills, then selecting appropriate tools, and finally fine-tuning execution parameters—our framework firstly structures execution into distinct skills, allowing global planning to focus on high-level skills rather than low-level details, while maintaining flexibility and extensibility, thereby ensuring a well-organized execution process and enhancing the agents. Next, we will introduce several representative skills:

Searching: This skill primarily functions as a tool invocation mechanism for information retrieval, utilizing a text or json formats similar to the ReAct. Its key advantage lies in its simplicity and stability in complex external environments, where simpler probing strategies tend to be more effective, as they facilitate error analysis and behavioral refinement. However, its major limitation is its restricted expressiveness, as it cannot handle complex tool calls or data processing logic, such as conditional branches and loops. Nevertheless, this approach remains valuable in some simpler scenarios.

Coding: This skill leverages python code to invoke tools and process data. Its strength lies in its ability to efficiently implement complex logic, including branching and looping, thereby expanding the agent's action space and enhancing execution efficiency. However, this increased action space comes at the cost of greater complexity. While ongoing research [20] aims to enable agents to generate more robust code, external environments remain inherently unpredictable, and even the most sophisticated code can produce unforeseen bugs. In such cases, a simpler Searching approach often serves as a more effective exploration method.

Writing: In real-world applications, many specialized tasks , which involve specific and intricate logic, cannot be effectively addressed through either Searching or Coding. For example, writing legal documents typically requires gathering legal articles, legal knowledge, and similar cases, followed by composing a preliminary draft and finally structuring the output according to specific format requirements. In such scenarios, although coding excels in handling complex branching and conditional logic, it does not directly contribute to completing the task.

Beyond these skills, expanding the skill pack according to task-specific requirements is essential. For instance, reasoning tasks may require specialized reasoning skills, scientific research tasks may demand research skills, and artistic creation tasks may call for creative skills. Notably, this expansion process is easy and highly iterative, enabling continuous adaptation and refinement to meet evolving task demands.

Table 1 Detailed Statistics of LegalAgentBench Tasks.

Attribute	1-hop	2-hop	3-hop	4-hop	5-hop	Writing	ALL
# Task	80	80	60	40	20	20	300
Avg. length per query	88.29	87.90	99.37	118.33	110.25	1059.95	160.65
Avg. length per answer	74.20	40.84	45.53	63.48	86.20	678.75	99.24
# Avg. key_answer per query	1.88	1.44	1.20	1.40	2.25	10.25	2.14
Avg. length of key_answer	10.59	5.94	6.07	6.59	6.93	12.58	9.28
# Avg. key_middle per query	0.13	1.45	2.87	4.78	5.60	6.20	2.42
Avg. length of key_middle	9.20	9.72	10.95	11.35	11.25	7.21	10.24

4 Experiment

4.1 Experimental Setup

4.1.1 Tasks Selection

To validate the effectiveness of GoalAct, it is essential to ensure the reliability of the evaluation benchmark. However, existing well-known datasets may have already been used in training, posing a risk of data leakage. Moreover, we aim to assess the agent's performance in an unfamiliar environment. Therefore, we select **LegalAgentBench** [12] as our benchmark. As a newly released dataset, it eliminates the risk of data leakage while covering a comprehensive range of task difficulties. Additionally, solving its tasks requires external data and legal knowledge, making it a suitable and robust test for the capabilities of agents.

The data in LegalAgentBench has been meticulously verified by human experts and includes a total of **300 tasks across six different types**. Table 1 presents the detailed statistics of these tasks. Among them, **1-hop to 5-hop** tasks represent logical reasoning problems with varying solution path lengths, where a higher hop count indicates increased task complexity. Beyond logical reasoning tasks, the dataset also features a **writing** category, specifically **writing a defense document**. In this scenario, the agent must query basic information about the plaintiff, defendant, and their lawyer, while simultaneously retrieving relevant legal knowledge and relevant articles to construct a defense against the complaint.

4.1.2 Baselines

We evaluated three well-known LLMs on LegalAgentBench: GLM-4-Plus [13], Qwen-max [15], GPT-4o-mini [22] (gpt-4o-mini-2024-07-18). All LLMs are evaluated through API calls. To ensure the reproducibility of the results, we set the temperature to 0.

For each LLM, we implemented four different methods. (1) **Plan-and-Solve**: Outline a complete plan and execute it step by step. (2) **Plan-and-Execute**: Develop a multi-step plan and complete it sequentially. After completing a task, the LLM can reassess the plan and make appropriate adjustments. (3) **ReAct**: Perform reasoning incrementally through the "thought-action-observation" process, integrating reasoning and tool usage. (4) **CodeAct**: Utilize Python code as the agent's action space, enabling the invocation of multiple tools within a single execution.

Table 2 The success rate of different methods on LegalAgentBench. P-S represents the Plan-and-Solve method, and P-E represents the Plan-and-Execute method. The best results are highlighted in bold.

Model	Method	1-hop	2-hop	3-hop	4-hop	5-hop	Writing	ALL
GPT-4o-mini	P-S	0.7117	0.3375	0.2750	0.2583	0.1250	0.6314	0.4196
	P-E	0.7444	0.3771	0.3250	0.2417	0.1417	0.6681	0.4503
	ReAct	0.9333	0.6500	0.4000	0.4208	0.2583	0.6087	0.6161
	CodeAct	0.9058	0.7938	0.4167	0.3875	0.1683	0.4202	0.6275
	GoalAct	**0.9556**	**0.8771**	**0.6250**	**0.5625**	**0.4517**	**0.7981**	**0.7720**
Qwen-max	P-S	0.8469	0.4958	0.4083	0.3792	0.2333	0.4836	0.5381
	P-E	0.8594	0.5896	0.3583	0.4083	0.3017	0.5539	0.5695
	ReAct	0.9062	0.7917	0.6333	0.5833	0.6083	0.6662	0.7422
	CodeAct	0.8442	0.8583	0.6333	0.7583	0.5117	0.6522	0.7594
	GoalAct	**0.9531**	**0.9062**	**0.8167**	**0.7917**	**0.6117**	**0.8220**	**0.8603**
GLM-4-Plus	P-S	0.8519	0.4667	0.4167	0.3583	0.1167	0.7522	0.5406
	P-E	0.8419	0.5000	0.3667	0.3458	0.1167	0.7679	0.5363
	ReAct	0.9131	0.8104	0.6417	0.6167	0.4300	0.7659	0.7499
	CodeAct	0.9221	0.7812	0.4167	0.5458	0.2333	0.5836	0.6648
	GoalAct	**0.9506**	**0.9187**	**0.8500**	**0.7375**	**0.6983**	**0.8643**	**0.8710**

When given a task, the model first determines which tools are needed, and then uses the selected tools to gradually solve the task. When the LLM outputs *Finish* or reaches the maximum iteration limit $T = 10$, it summarizes the current trajectory and provides the final answer. We include two examples for each process to guide the model in using the tools and following the specified output format.

4.1.3 Metrics

We use **Success Rate** as the metric, which measures the proportion of key_answer elements included in the LLM's output. Assume a dataset $\mathcal{D}$ consisting of N data points, where each data point includes a keyword set $\mathcal{K}_i$ and a model output $\mathcal{O}_i$. The success rate s_i for the i-th data point is computed as Equation 4 shows:

$$s_i = \frac{|\mathcal{M}_i|}{|\mathcal{K}_i|} \tag{4}$$

where $\mathcal{M}_i = \{k \in \mathcal{K}_i \mid k \text{ appears in } \mathcal{O}_i\}$. The notation $|\cdot|$ represents the number of elements in a set. We report the average success rate across all tasks in the results.

4.2 Experimental Results

4.2.1 Main Results

Table 2 presents the results of different methods on LegalAgentBench, from which we draw the following conclusions:

- **GoalAct achieves SOTA performance across different LLM series and varying task difficulties.** Compared to the second-best method, GoalAct

Table 3 An ablation study was conducted to evaluate the effectiveness of GoalAct with the GLM-4-Plus as the base LLM. The notation 'w/o' indicates experiments where specific modules were removed.

Method	1-hop	2-hop	3-hop	4-hop	5-hop	Writing	ALL
GoalAct	**0.9506**	**0.9187**	**0.8500**	**0.7375**	**0.6983**	**0.8643**	**0.8710**
w/o global plan	0.9281	0.7375	0.8000	0.6208	0.6950	0.8454	0.7896
w/o searching	0.8950	0.8063	0.7500	0.6708	0.6100	0.8521	0.7906
w/o coding	0.8381	0.7833	0.6333	0.6208	0.4833	0.8447	0.7304
w/o writing	0.9381	0.9000	0.7667	0.6708	0.5767	0.8247	0.8264

improves average performance by **14.45% on GPT-4o-mini**, **10.09% on Qwen-max**, and **12.11% on GLM-4-Plus**, with an overall average improvement of **12.22%**. These results demonstrate the broad effectiveness of GoalAct.

- **The inherent limitations of various baseline methods undermine their performance.** **P-S** executes a static global plan, which limits its adaptability in complex scenarios, leading to poor performance. **P-E** dynamically updates its global plan, demonstrating improvements over P-S in some scenarios. However, as its global plan does not account for task execution, the execution feasibility of its plan in some tasks is weak, ultimately resulting in performance degradation. **ReAct** exhibits relatively strong performance, but the absence of global planning and its reliance on action formats constrained by text or json impose an upper bound on its effectiveness in certain scenarios. **CodeAct** extends the action space through python code; however, in complex scenarios, invoking multiple tools within a single code execution often results in errors. Moreover, experimental results show that CodeAct performs poorly in Writing tasks, highlighting its significant limitations.
- **As the task difficulty increases, the performance gain of GoalAct over the second-best method becomes more pronounced.** Specifically, the relative improvements on average are **3.26% for 1-hop**, **7.98% for 2-hop**, **20% for 3-hop**, **9.86% for 4-hop**, and **15.5% for 5-hop**. Furthermore, in Writing tasks, GoalAct achieves a 12.74% improvement, further validating its strong adaptability and generalization across different task types.

4.2.2 Ablation Study

To evaluate the effectiveness of **Global Planning and Hierarchical Execution**, we conducted an ablation study by systematically removing key components of GoalAct with GLM-4-Plus as the base LLM. Table 3 presents the results, from which we observe the following: Removing the **global plan** reduces average performance by **8.14%**, **8.04%** for **Searching**, **14.06%** for **Coding**, and **4.46%** for **Writing** (with Writing tasks specifically declining by **3.96%**). These ablation results validate the rationality of GoalAct's design and highlight the necessity of synergy among its components to achieve optimal performance.

4.2.3 Case Study

Figure 2 presents a specific example of the competitive agent frameworks in our experiment, including ReAct, CodeAct, and GoalAct, with GLM-4-Plus as the base LLM. We observe that ReAct has restricted action space due to the text or json formats and its tendency to repeatedly attempt solutions, often getting stuck in local branches. CodeAct leverages code to express complex logic, which enhances its action space. However, its approach of invoking multiple tools within a single piece of code introduces execution difficulties, as each tool's output may contain uncertainties. In contrast, our proposed method, GoalAct, effectively avoids the local branch issue by implementing a global planning strategy. Additionally, its hierarchical execution mechanism enables the agent to flexibly choose appropriate skills based on task complexity, utilizing searching for simpler tasks and coding for more complex ones. These observations demonstrate that GoalAct offers superior performance and greater generalizability.

5 Conclusion

This paper presents GoalAct, which enhances LLM-based agents by integrating global planning and hierarchical execution. Experimental results on the LegalAgentBench benchmark demonstrate that GoalAct achieves state-of-the-art (SOTA) performance, with an average improvement of 12.22% over existing methods. In the future, GoalAct can be integrated with mechanisms such as agent reflection [23] and memory [24] to facilitate the development of more advanced and intelligent agent systems.

References

[1] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., *et al.*: A survey on large language model based autonomous agents. Frontiers of Computer Science **18**(6), 186345 (2024)

[2] Lai, H., Liu, X., Iong, I.L., Yao, S., Chen, Y., Shen, P., Yu, H., Zhang, H., Zhang, X., Dong, Y., *et al.*: Autowebglm: A large language model-based web navigating agent. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 5295–5306 (2024)

[3] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems **36**, 68539–68551 (2023)

[4] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)

[5] Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al.: Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437 (2024)

[6] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R.K.-W., Lim, E.-P.: Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. arXiv preprint arXiv:2305.04091 (2023)

[7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023)

[8] Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., Ji, H.: Executable code actions elicit better llm agents. In: Forty-first International Conference on Machine Learning (2024)

[9] Topsakal, O., Akinci, T.C.: Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In: International Conference on Applied Engineering and Natural Sciences, vol. 1, pp. 1050–1056 (2023)

[10] Li, H., Ye, J., Hu, Y., Chen, J., Ai, Q., Wu, Y., Chen, J., Chen, Y., Luo, C., Zhou, Q., et al.: Casegen: A benchmark for multi-stage legal case documents generation. arXiv preprint arXiv:2502.17943 (2025)

[11] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., *et al.*: Agentbench: Evaluating llms as agents. In: International Conference on Learning Representations (ICLR) (2024)

[12] Li, H., Chen, J., Yang, J., Ai, Q., Jia, W., Liu, Y., Lin, K., Wu, Y., Yuan, G., Hu, Y., et al.: Legalagentbench: Evaluating llm agents in legal domain. arXiv preprint arXiv:2412.17259 (2024)

[13] GLM, T., Zeng, A., Xu, B., Wang, B., Zhang, C., Yin, D., Zhang, D., Rojas, D., Feng, G., Zhao, H., et al.: Chatglm: A family of large language models from glm-130b to glm-4 all tools. arXiv preprint arXiv:2406.12793 (2024)

[14] Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al.: Glm-130b: An open bilingual pre-trained model. arXiv preprint arXiv:2210.02414 (2022)

[15] Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al.: Qwen technical report. arXiv preprint arXiv:2309.16609 (2023)

[16] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., *et al.*: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)

[17] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., *et al.*: Training language models to follow instructions with human feedback. Advances in neural information processing systems **35**, 27730–27744 (2022)

[18] Wei, J., Bosma, M., Zhao, V.Y., Guu, K., Yu, A.W., Lester, B., Du, N., Dai, A.M., Le, Q.V.: Finetuned language models are zero-shot learners. arXiv preprint arXiv:2109.01652 (2021)

[19] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., *et al.*: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems **35**, 24824–24837 (2022)

[20] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291 (2023)

[21] Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., et al.: Taskweaver: A code-first agent framework. arXiv preprint arXiv:2311.17541 (2023)

[22] Hurst, A., Lerer, A., Goucher, A.P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al.: Gpt-4o system card. arXiv preprint arXiv:2410.21276 (2024)

[23] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems **36**, 8634–8652 (2023)

[24] Zhong, W., Guo, L., Gao, Q., Ye, H., Wang, Y.: Memorybank: Enhancing large language models with long-term memory. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 19724–19731 (2024)

Appendix

Table 4 The prompt used in the GoalAct for global planning.

The prompt used in the GoalAct for global planning.
You are a planner skilled in solving complex tasks. You use the three alternating steps of "Thinking, Acting, and Observing" to solve question-answer tasks based on the provided data tables. Thinking involves reasoning about the current situation and determining the next subproblem to solve the current issue. Acting is performing an operation based on the results of your thinking. It must be one of the following four types: 1. Searching: Retrieve a record from a data table based on information, or filter multiple records that meet specific attribute values. 2. Coding: If the problem is too complex to be solved by querying alone, you may attempt to program a solution. You can filter, sort, sum, or iterate over queried data. 3. Writing: If content generation (such as a defense statement) is needed, you should attempt writing to resolve the issue. 4. Finish: Provide the final answer and terminate the task. The action must be detailed within square brackets [] and should accurately identify the correct table and returned fields. Observing is the information obtained after an action. If all observations sufficiently answer the problem, provide the final response and end the task. You should take the necessary steps. Ensure that your response strictly follows the format above. Specifically, the action must be one of the listed types, and all actions should terminate with an end. Problem to be solved: {question} The data tables involved include: {table_used_prompt} Available API_tool: {tool_prompt} Here are some reference examples: {memory} Reference examples end. Question: {question} Existing planning chain: {scratchpad} Please continue to think and execute based on the existing planning chain logic. Ensure that you avoid repeating any previous thought paths or actions already taken. If the current planning chain sufficiently resolves the issue, directly output the action as "Finish" Please output the result in the following JSON format, which can be parsed using Python's json.loads function. Provide only the problem decomposition result without explanation or direct answers: "json [{{ "Thinking": "", "Action": "" }}] "

Task: I am a newly practicing legal professional and would like to know the number of cases handled in 2021 by the law firm with the shortest establishment time, located in the registered address area of the defendant in the highest-value case involving Anhui Ankang Law Firm.

ReAct

Thought 1: Retrieve information on the highest-value case involving Anhui Ankang Law Firm, including the defendant's name.
Action 1:
"get_legal_document_law_list"…
Observation 1: **…**
Thought 2: Retrieve the registered address of Huaibei Mining Holdings Co., Ltd.
Action 2:
"get_company_register"…
Observation 2:
{"Company Address": "No. 276 Renmin Middle Road, Huaibei City, Anhui Province"}
Thought 3: Retrieve the law firm with the shortest establishment time in Huaibei City, Anhui Province.
Action 3:
{ "action": "get_lawfirm_info_list", "action_input": { "prov": "Anhui Province", "city": "Huaibei City", "county": "", "columns": ["Law Firm Name", "Establishment Date"] } }
Observation 3:
No data found for the specified identifier.
Thought 4: Retry …
Action 4: "get_lawfirm_info_list"…
Observation 4:
No data found for the specified identifier.
….

Get stuck in local branch!

CodeAct

Thought 1: This problem needs to be solved step by step. First…
Code 1:

```
law_list =
get_legal_document_law_list(identifier="Anhui Ankang Law Firm")

sorted_cases = rank(keys=[case['Case Number'] for case in law_list], values=[case['Monetary Amount'] for case in law_list], is_desc=True)

highest_amount_case = sorted_cases[0]
defendant = law_list[0]['Defendant']
defendant_register_info =
get_company_register(identifier=defendant)

defendant_address =
defendant_register_info['Company Address']

address_info =
get_address_info(identifier=defendant_address)
lawfirm_list = get_lawfirm_info_list(prov="Anhui Province", city="Ankang City",
county=address_info['District'])…
```

Observation 1:
<error>Code execution failed due to the following error

Use too many tools in a single code!

GoalAct: Global Planning and Hierarchical Execution

Global Plan: […]
Plan Step 1: First, we need to query all cases involving **Anhui Ankang Law Firm** and identify the case with the highest monetary amount involved.
Action 1: **Coding**[Query the **LegalDoc** table to retrieve all cases involving **Anhui Ankang Law Firm**, returning the **case number** and **monetary amount** fields. Then, sort the cases by **monetary amount** and find the case with the highest amount.]
Observation 1: The case with the highest monetary amount is **(2020) Wan 0603 Min Chu 2364**, with a monetary value of **259,221.38**.
Global Plan (updated): […]
Plan Step 1 : Next, we need to query the **defendant company name** in the case with the highest monetary amount.
Action 2: Searching[Search the **LegalDoc** table for the case with **case number (2020) Wan 0603 Min Chu 2364** and return the **defendant** field]
Observation 2: The defendant company in the case is **Huaibei Mining Holdings Co., Ltd.** ...
….

Fig. 2 A specific example of ReAct, CodeAct and GoalAct in our experiments.