Ask-before-Plan: Proactive Language Agents for Real-World Planning

Xuan Zhang[1], **Yang Deng**[2*], **Zifeng Ren**[1], **See-Kiong Ng**[1], **Tat-Seng Chua**[1]
[1]National University of Singapore [2]Singapore Management University
{xuanzhang,renzifeng}@u.nus.edu ydeng@smu.edu.sg
{seekiong,dcscts}@nus.edu.sg

Abstract

The evolution of large language models (LLMs) has enhanced the planning capabilities of language agents in diverse real-world scenarios. Despite these advancements, the potential of LLM-powered agents to comprehend ambiguous user instructions for reasoning and decision-making is still under exploration. In this work, we introduce a new task, Proactive Agent Planning, which requires language agents to predict clarification needs based on user-agent conversation and agent-environment interaction, invoke external tools to collect valid information, and generate a plan to fulfill the user's demands. To study this practical problem, we establish a new benchmark dataset, Ask-before-Plan. To tackle the deficiency of LLMs in proactive planning, we propose a novel multi-agent framework, Clarification-Execution-Planning (CEP), which consists of three agents specialized in clarification, execution, and planning. We introduce the trajectory tuning scheme for the clarification agent and static execution agent, as well as the memory recollection mechanism for the dynamic execution agent. Extensive evaluations and comprehensive analyses conducted on the Ask-before-Plan dataset validate the effectiveness of our proposed framework.[1]

1 Introduction

Since long, people have been engaged in a never-ending trek to devise intelligent agents cloning appearance, behaviors, and even mindsets similar to human beings (Descartes and Cress, 1998; Dreyfus, 1992). The emergence of large language models (LLMs) and their integration into autonomous agents exhibits the potential for logical reasoning, decision-making, and problem-solving capabilities (Wang et al., 2023b; Xi et al., 2023). LLM-

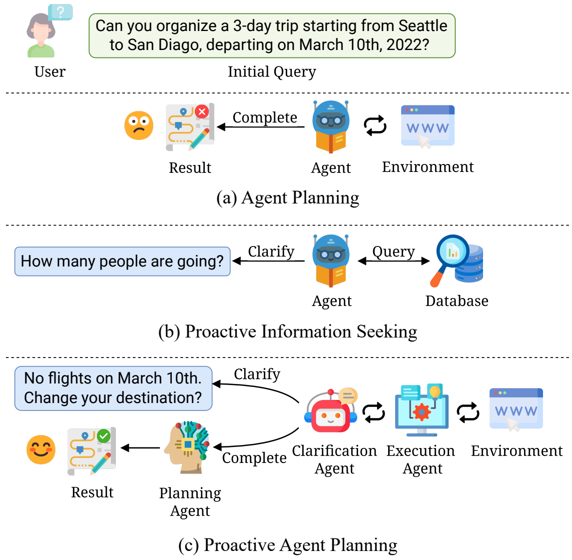


Figure 1: Illustrations of different problems.

based language agents have been explored in various real-world planning problems, such as travel planning (Xie et al., 2024), web navigation (Deng et al., 2023a), online shopping (Yao et al., 2022), etc. Nevertheless, they struggle to *digest* real-world instructions from time to time, particularly when the context is ambiguous or the user's intention is unclear. Figure 1(a) exemplifies a case of *Agent Planning*, where the language agent fails to produce a desired plan when encountering an unclear user instruction.

Inspired by proactive information-seeking systems (Deng et al., 2023b; Liao et al., 2023) that can proactively ask clarification questions for clarifying the query uncertainty, latest studies on language agents (Qian et al., 2024; Zhang et al., 2024b) emphasize intention clarifications by incorporating implicit intention understanding into downstream applications (Figure 1(b)). Despite their effectiveness in clarifying the uncertainty within user queries, there are several challenges that remain to be solved in agent planning scenarios: (1) The exclusive reliance on query understanding for clarification without considering

[*]Corresponding author.
[1]The data and code will be released via https://github.com/magicgh/Ask-before-Plan.

Findings of the Association for Computational Linguistics: EACL 2024, pages 10836–10863
November 12-16, 2024 ©2024 Association for Computational Linguistics

the observations from the environment interaction; and (2) The negligence of the effect of user intention disambiguation in subsequent planning.

Given these important but neglected aspects, we formulate a novel task named **Proactive Agent Planning**. In addition to apprehending the user's implicit intentions in proactive information seeking, Proactive Agent Planning further requires language agents to predict clarification needs depending on the environmental observations, invoke external tools to collect valid information based on the user-agent conversation, and finalize decisions to accomplish the common goal conditioned on both user-agent conversation and agent-environment interaction history (Figure 1(c)). To study this practical problem, we first establish a new benchmark dataset, **Ask-before-Plan**, based on an agent planning dataset, TravelPlanner (Xie et al., 2024), which is used for evaluating the capability of agents for real-world travel planning.

To tackle the aforementioned challenges, we propose a novel multi-agent framework, named Clarification-Execution-Planning (`CEP`), to encompass user instruction clarification into downstream agent planning. Specifically, the `CEP` involves three agents with distinct roles in *Clarification*, *Execution*, and *Planning*. The *Clarification* agent is responsible for understanding the uncertainty of user instructions and asking users clarifying questions to uncover their underlying intentions. The *Execution* agent leverages various tools to interact with the environment, gathering necessary information for the *Clarification* agent. The *Planning* agent produces the final plan by aggregating the clarification process for accomplishing the initial user instruction. To supplement the deficiency of simply prompting LLMs to ask clarification questions or perform complex tool learning, we devise *Trajectory Tuning* for fine-tuning the clarification and execution agents. Furthermore, we employ self-reflection to improve the reasoning process of the execution agent. However, the redundancy of self-reflection in multi-turn conversations may increase the time complexity of inference and introduce more noise into the context. To this end, we propose the memory recollection mechanism to optimize the memory utility for the execution agent in long-context reasoning.

In summary, our contributions are as follows:

- We introduce a new and practical problem of **Proactive Agent Planning** to study the challenges of LLM-powered language agents in handling unclear user instructions.
- We propose a novel multi-agent framework, namely `CEP`, which consists of clarification, execution, and planning agents, to address the underlying challenges in the Proactive Agent Planning problem.
- We construct the first dataset for studying Proactive Agent Planning, namely **Ask-before-Plan**. Extensive evaluations and comprehensive analyses in diverse settings validate the effectiveness of the proposed `CEP` framework.

2 Related Work

Language Agents Language-based agents (Xi et al., 2023; Wang et al., 2023b; Deng et al., 2024a; Zhang et al., 2024a) aim to perform real-world tasks that require professional expertise or extensive training by utilizing LLMs to conduct reasoning (Yao et al., 2023; Shinn et al., 2023), memory storage and retrieval (Wen et al., 2023; Zhong et al., 2024), and tool use (Qin et al., 2024; Schick et al., 2023). The rapid development and application of language agents span various domains, including web agents (Deng et al., 2023a; Yao et al., 2022; Deng et al., 2024b), game agents (Wang et al., 2023a; Zhu et al., 2023), and medical agents (Li et al., 2024a; Schmidgall et al., 2024), etc. Existing studies typically assume that the user instructions are clear enough for language agents to execute the task, while the instructions tend to be succinct and brief in actual scenarios, potentially leading to ambiguity and uncertainty.

Asking Clarification Questions The problem of asking clarification questions (Aliannejadi et al., 2021) typically involves two phases: Clarification Need Prediction and Clarification Question Generation. Early studies develop ranking-based methods (Rao and Daumé III, 2018; Xu et al., 2019) to select clarification questions from a candidate pool, or apply end-to-end solutions (Deng et al., 2022) for generating questions on the fly. Recently, LLMs have also been employed to generate clarifying questions (Kuhn et al., 2022; Deng et al., 2023c; Zhang et al., 2024b; Chen et al., 2024). Another line of research investigates environmental reward feedback to formulate questions (Zamani et al., 2020; Pyatkin et al., 2023; Andukuri et al., 2024). Qian et al. (2024) explore how language agents understand implicit user intentions but do not clarify users' needs based on their predefined preferences and consider the whole language agent problem, such as planning and task-solving.

3 Ask-before-Plan Dataset

Existing benchmarks on language agents either only focus on their capabilities of asking clarification questions, *e.g.*, *Tell Me More* (Qian et al., 2024) and *Clamber* (Zhang et al., 2024b), or simply assume that all the user instructions are clear and explicit, *e.g.*, *TravelPlanner* (Xie et al., 2024) and *Mind2Web* (Deng et al., 2023a). In order to integrally investigate agents' competence in asking clarification questions as well as planning and interacting with the environment, we construct the **Ask-before-Plan** dataset. Built on the TravelPlanner benchmark, our dataset incorporates uncertain user instructions that require clarifications into real-world travel planning scenarios.

3.1 Annotation & Quality Control

We aim to incorporate uncertain user instructions with proactive information-seeking dialogues into tool-augmented planning, allowing the agent to ask clarification questions when faced with ambiguous or vague user instructions. As shown in Figure 2, the dataset construction process is bifurcated into the following two phases: (1) **Indefinite Detail Formation**, and (2) **Proactive Conversation Generation**.

Indefinite Detail Formation Each instruction in *TravelPlanner* precisely describes the task and all pertinent details. To produce unclear instructions with indefinite details, we focus on two typical scenarios observed in proactive information-seeking dialogues: (1) clarifying under-specified queries, and (2) managing over-specified queries (Wu et al., 2023; Deng et al., 2023b):

1. We create the under-specified user instruction with missing details based on attributes such as *origin*, *destination by arrival days*, *departure date*, *duration*, *number of people*, and *budget*, by discarding corresponding information from the original complete instruction.
2. To construct over-specified instructions, we develop unfeasible details based on features such as *accommodation*, *cuisine*, and *transportation* preferences, *destination by arrival days*, and *budget*. We exhaustively iterate over all possible values within the TravelPlanner environment to select those that lack directly matched information for instruction rewriting.

Consequently, we range the size of indefinite details from 0 to 3 for each instruction to formulate the final dataset. Given the interdependencies among these details, topological sort is employed to reorder the details according to the dependency graph, and the priority of each detail is assigned accordingly. The descriptions of indefinite detail creation and instruction modification are presented in Appendix A.1.

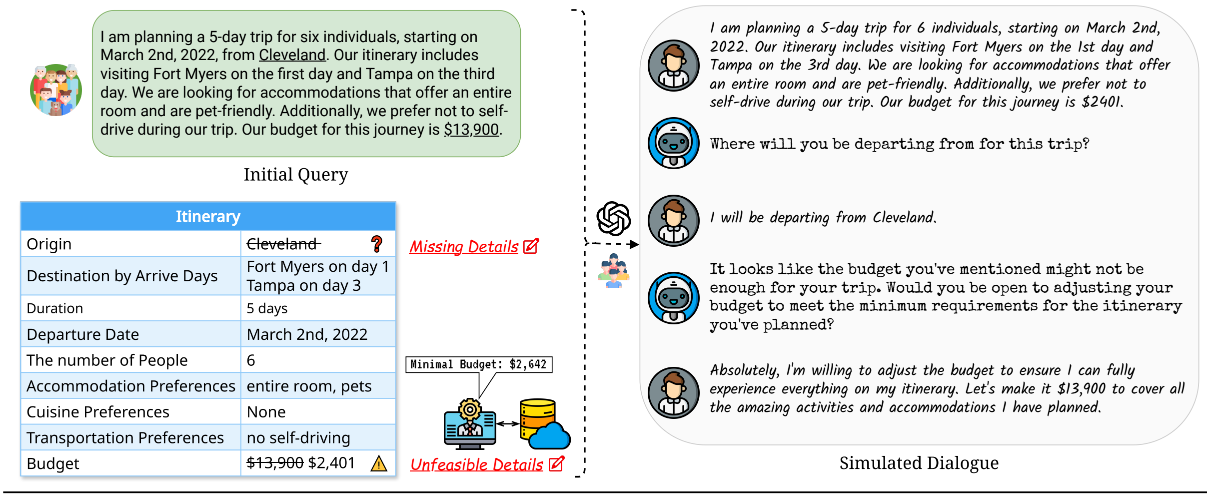
Proactive Conversation Generation After acquiring the revised instruction and selecting indefinite details, we utilize GPT-3.5 to simulate the proactive conversation between users and agents. In the user simulation, we adopt different tones, including succinct and passionate response patterns. To summarize, we follow the pipeline below to generate the real-world conversation:

1. **Ambiguity Observation**: The description of an unclarified indefinite detail is presented to GPT-3.5 (Agent) as the observation.
2. **Question Generation**: A question is generated to inquire about the indefinite detail based on the initial instruction and observation.
3. **Draft Answer**: The draft answer is provided to GPT-3.5 (User) to guide the clarification of the indefinite detail.
4. **Natural Response Generation**: A natural user response is generated based on the question and the draft answer.
5. **Quality Control**: The human examiner verifies that the question covers the indefinite detail and that the response provides clarification.

The detailed implementation of these simulations, including the prompts used, is documented in Appendix A.2. The methodology achieved a pass rate of 87.2% on the human examination.

3.2 Dataset Statistics

After the dataset construction, we obtain 1,000 samples for both the training and testing datasets. Indefinite details were randomly selected from 11 distinct candidate types, as previously defined. The distribution of indefinite detail sizes is structured as follows: 10% of the data contains no indefinite details, and 30% of the data includes 1, 2, or 3 indefinite details each. The overall dataset statistics are presented in Table 1. The environment setting is presented in Appendix A.3.



(1) Indefinite Detail Formation (2) Proactive Conversation Generation

Figure 2: Data annotation process.

		Train	Test
Level	Easy	331	348
	Medium	336	333
	Hard	333	319
Duration	3 days	335	308
	5 days	337	351
	7 days	328	341
Missing Details	Origin	219	192
	Destination by Arrival Days	379	360
	The number of People	200	204
	Duration	199	201
	Departure Date	199	213
	Budget	384	401
	Total	1,204	1,183
Unfeasible Details	Destination by Arrival Days	189	176
	Accommodation Preferences	182	185
	Cuisine Preferences	18	20
	Transportation Preferences	20	24
	Budget	187	212
	Total	596	617
Total	# Dialogues	1,000	1,000
	# Turns	2,800	2,800

Table 1: Dataset statistics of **Ask-before-Plan**.

3.3 Problem Definition

Regarding the problem of **Proactive Agent Planning**, given an initial user instruction q_0 potentially containing T vague or unfeasible details, $D = \{d_1, \ldots, d_T\}$, we define three subtasks:

- **Clarification Need Prediction & Clarification Question Generation**: For a user-agent dialogue up to $t-1$ turns, we define the conversation history, $C_{t-1} = \{(q_0), (a_1, q_1), \ldots, (a_{t-1}, q_{t-1})\}$, the agent-environment interaction history based on C_{t-1}, denoted as $E_{t-1} = \{f_1, o_1, \ldots, f_n, o_n\}$, where f_i is the function call and o_i is the observation from the environment, and the unclarified detail set for this turn, $D_t = \{d_{t+1}, \ldots, d_T\}$. The clarification agent is tasked with predicting a binary label b_t, indicating the need for clarification based on the vagueness and feasibility of environmental responses E_{t-1}, and eliciting unclarified details in D_t by generating a clarification question a_t if needed. The conversation will advance according to the topological priority assigned to each detail, with those sharing the same priority addressed randomly.

- **Tool Learning**: Given the ongoing conversation $C_t = \{(q_0), (a_1, q_1), \ldots, (a_{t-1}, q_{t-1}), a_t\}$, the agent needs to generate the complete interaction chain $E_t^{i-1} = \{f_1, o_1, \ldots, f_{i-1}, o_{i-1}\}$ within the present information. A correct function call is defined as having accurate function names and valid parameters. This stage is used to interact with the environment as well as provide observation for the next-turn clarification.

- **Agent Planning**: Upon completion of the clarified dialogue C_t and the last interaction history E_t, the planning agent must generate a valid plan in JSON format with pre-defined fields such as *day*, *current city*, *transportation*, etc.

4 Method

We introduce our multi-agent framework, namely Clarification-Execution-Planning (CEP), which consists of three agents: clarification agent, execution agent, and planning agent.

Specifically, we investigate two settings of execution agents to interact with the environment using tools, namely static and dynamic interaction. For the static interaction, the agent needs to generate the whole valid interaction of tool calls E_t^{i-1} based on the current conversation C_t in one-step

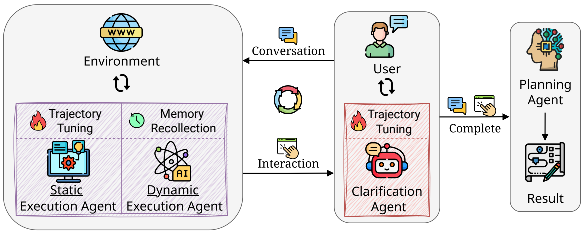


Figure 3: Overview of the CEP framework.

inference. We regard the dynamic interaction as a multi-step inference. In this situation, the agent is expected to generate one action f_i depending on the current conversation C_t, with previous interaction history $E_t^{i-1} = \{f_1, o_1, \ldots, f_{i-1}, o_{i-1}\}$.

As illustrated in Figure 3, we introduce the trajectory tuning scheme for the clarification agent and static execution agent, along with memory recollection for the dynamic execution agent. Finally, the planning agent generates a valid plan based on the conversation history and interaction results.

4.1 Trajectory Tuning

Concerning the status quo, LLMs struggle to spontaneously clarify users' intentions during the conversation (Deng et al., 2023b) and accurately obtain necessary information via tool utilization without hallucination (Li et al., 2024b), even after the instruction tuning or providing the well-crafted prompts. To resolve these issues and align the current language-based agents with our task, we propose to use the trajectory (C_{t-1}, E_{t-1}) to finetune the clarification agent and the execution agent in the static setting. We sample $t-1$ details according to topological order (see Appendix A.1) to craft C_{t-1} and derive the valid interaction chain E_{t-1} from C_{t-1} based on ground truth.

Clarification Agent As introduced in Section 3.3, there are two steps to be achieved for the clarification agent: (1) Clarification Need Prediction. In this step, the agent generates a boolean indicator b_t to predict the need for clarification based on the ongoing conversation C_{t-1} and the last turn interaction E_{t-1}. (2) Clarification Question Generation. If b_t is true, the model needs to generate a clarifying question a_t for a specific detail d_t. In this case, we create a conversation sequence $[f_{\text{prompt}}(C_{t-1}, E_{t-1}, b_t), a_t{}^{[b_t=1]}]$ and tokenize it into the input tokens $\mathbf{x}_t$, where $f_{\text{prompt}}(\cdot)$ is a function to apply the prompt template.

Static Execution Agent In the tool learning subtask, we apply trajectory tuning for the static setting. Given the current-turn conversation C_t and the complete interaction chain E_t, we construct the sequence $[f_{\text{prompt}}(C_t), E_t]$ and transform it into the input $\mathbf{x}_t$.

Training Objective Finally, we train the agent using the trajectory sequence $\mathbf{x}_t$ for turn t in the autoregressive manner:

$$\mathcal{L} = \max_{\theta} \frac{1}{|T|} \sum\nolimits_{t=1}^{T} \sum\nolimits_{i=1}^{N_t} \log P_\theta^t(x_i^t | \mathbf{x}_{<i}^t) \quad (1)$$

where θ denotes the model parameter to be learned, T denotes the number of total conversation turns, and N_t denotes the number of tokens in turn t.

4.2 Memory Recollection

In view of dynamic tool interaction, Reflexion (Shinn et al., 2023) has been validated as an effective approach to improve the reasoning capability of LLMs by incorporating the self-reflective feedback into the ReAct framework (Yao et al., 2023). However, when applying this approach to Proactive Agent Planning, there is a potential issue: the dynamic execution agent may encounter similar types of exceptions and repeatedly generate identical rationales across different turns. To alleviate this disadvantage, we introduce the **Memory Recollection** mechanism to *reuse* self-reflective feedback from previous turns.

Given the conversation context C_t at turn t, the execution agent generates a tool call f_i during the i-th interaction, based on previous interactions E_t^{i-1} and all reflective text in the memory bank R_t^i, *i.e.*, $f_i = \text{LLM}(C_t, E_t^{i-1}, R_t^i)$. When f_i is invalid, the agent will generate a rationale r_i and store it into R_t^{i+1} for self-reflection at the next interaction:

$$R_t^{i+1} = \begin{cases} R_t^i & \text{if } f_i \text{ is valid} \\ \text{concat}(R_t^i, r_i) & \text{else} \end{cases} \quad (2)$$

Given that C_t shares common clarified details across different turns, the memory recollection accumulates the reflective feedback from previous turns, *i.e.*, $R_{t+1} = R_t$. This mechanism not only prevents the model from repeating mistakes in history turns but also reduces the inference time for upcoming turns. The theoretical time complexity analysis and corresponding experimental results are presented in Appendix C.1.

4.3 Overall Framework

Given the conversation C_t of turn t, the execution agent first generates the whole interaction chain

E_t. Next, based on C_t and E_t, the clarification agent is required to predict the clarification need and ask a question about a detail from D_t if necessary. This process continues iteratively until all indefinite details have been recovered by the clarification agent. Once all such details are clarified, the planner agent will generate a valid plan in the format of JSON, based on the conversation C_T and the interaction E_T from C_T.

5 Experiment

5.1 Experimental Setups

Baselines We adopt the following baselines for subtasks in the problem of Proactive Agent Planning: (1) Clarification: We first design a rule-based environment-only method and adopt four conversation-only methods, including Proactive, Proactive CoT (ProCoT) (Deng et al., 2023c), and Direct prompting, as baselines. For the reason that there is no existing approach that concurrently considers the environment and conversation when asking clarification questions, we further directly prompt the GPT-3.5 with the conversation history and interaction log, along with In-context Learning (ICL) prompting, as additional baselines. (2) Tool learning: We adopt the general brute-force algorithm and four baselines, including Direct prompting (GPT-3.5) and finetuned ToolLLM (LLaMA-2-7B) (Qin et al., 2024) for the static execution setting, as well as ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023) for the dynamic setting. (3) Planning: Our baselines include Direct prompting, Zero-shot CoT (Kojima et al., 2023) for static settings, as well as ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023) for dynamic settings.[2] Additionally, Greedy Search is included as a baseline for comparison. Details of these baselines are presented in Appendix B.1.

Evaluation Metrics Similarly, we employ evaluation metrics for the three subtasks as follows: (1) In the clarification subtask, Clarification Accuracy (Clarif. Acc) is adopted for evaluating the performance of Clarification Need Prediction, and meanwhile, we setup three metrics to score the quality of clarification question generation, including Rule-based, BLEU, and GPT scores. (2) For the tool learning subtask, following previous studies (Wang et al., 2024), we also adopt the same metrics for evaluation, including Well-formed, API Match, and Correctness. Furthermore, we add two metrics to assess the Repeat Rate and Pass Rate of tool use. (3) Concerning the planning subtask, we also adopt the same evaluation metrics as TravelPlanner (Xie et al., 2024), including Delivery Rate, Commonsense Pass Rate, Hard Constraint Pass Rate, and Final Pass Rate. Details of these evaluation metrics are presented in Appendix B.2.

[2] ReAct and Reflexion are not tested for Mistral-7B due to the inability to generate valid JSON plans.

	Clarif. Acc		Rule-based Score		BLEU	GPT Score	
	Micro	Macro	Micro	Macro		Micro	Macro
Environment-only	70.4	17.7	21.5	8.1	1.0	40.1	19.4
Conversation-only							
Proactive (GPT-3.5)	62.3	6.1	9.7	3.4	3.7	0.9	0
ProCoT (GPT-3.5)	33.7	10.6	3.3	2.6	2.2	2.4	1.8
Direct (Mistral-7B)	59.4	24.6	56.8	50.8	47.9	65.8	59.3
Direct (LLaMA-3-8B)	**76.8**	**48.5**	**70.5**	**64.6**	**53.4**	**80.7**	**75.5**
Environment + Conversation							
Direct (GPT-3.5)	47.0	16.9	20.8	17.4	8.2	8.6	6.2
ICL (GPT-3.5)	65.7	29.4	2.1	0.6	8.8	2.7	0.9
CEP (Mistral-7B)	82.8	51.7	54.2	37.0	44.5	73.1	58.6
CEP (LLaMA-3-8B)	**99.4**	**98.2**	**69.7**	**55.8**	**57.2**	**85.8**	**77.0**

Table 2: Evaluation of clarification.

Implementation Details For all experiments involving GPT-3.5, we use `gpt-3.5-turbo-0125`. Mistral-7B, LLaMA-2-7B, and LLaMA-3-8B are adopted for finetuning and inference. More implementation details are presented in Appendix B.3.

5.2 Overall Evaluation

We list our experiments on clarification, tool learning, and planning subtasks, shown in Tables 2, 3, 4. To assess the capability of agents on these three subtasks individually, we use the ground truth of prior subtasks as input for the subsequent subtask. Evaluations of the integral framework are discussed in Section 5.2.4.

5.2.1 Evaluation on Clarification

From Table 2, the results of the environment-only method surpass Proactive and ProCoT, pointing out the importance of using environment observation as a supervised signal in clarification for language agents.

Comparing four conversation-only methods, Proactive performs well in turn-grained (Micro Clarif. Acc), while ProCoT strengthens in instance-grained clarification (Macro Clarif. Acc). Upon careful inspection, we observe that Proactive prefers to clarify the user's instruction, whereas ProCoT tends to directly execute the instruction, which is discussed in Section 5.3. Besides, ProCoT performs better in clarification

	Well-formed	API Match	Repeat Rate	Correctness			Pass Rate	
				P	R	F1	Micro	Macro
Brute-force	100	98.8	0	77.2	90.3	81.9	45.7	22.0
Static Setting								
Direct (GPT-3.5)	99.9	88.9	0.07	72.8	62.0	64.7	7.1	2.3
ToolLLM (LLaMA-2-7B)	99.7	82.9	2.4	65.9	66.1	63.3	16.3	3.4
CEP (Mistral-7B)	99.4	93.4	0.15	91.7	90.1	90.1	57.6	27.3
CEP (LLaMA-3-8B)	**100**	**99.3**	**0.04**	**97.9**	**98.1**	**97.9**	**89.0**	**78.4**
Dynamic Setting								
ReAct (GPT-3.5)	66.2	33.3	14.3	42.8	15.6	21.1	**1.4**	0
Reflexion (GPT-3.5)	70.5	42.2	11.1	44.4	18.6	24.0	1.0	0
CEP (GPT-3.5)	**73.3**	45.3	**9.8**	45.0	19.2	24.7	1.1	0
ReAct (Mistral-7B)	49.0	**50.0**	11.8	**58.1**	**24.6**	**32.3**	1.2	0
Reflexion (Mistral-7B)	48.1	46.3	15.6	54.2	21.8	28.8	1.2	0
CEP (Mistral-7B)	46.9	42.4	18.1	49.0	19.5	25.6	1.3	0

Table 3: Evaluation of tool learning.

question generation with higher GPT scores. Both Direct prompting methods deliver better results than the previous two baselines while achieving BLEU and GPT scores that are closer to their Environment + Conversation counterparts. However, they fail to distinguish the need for clarifying the conversation, highlighting the importance of observing both environment and conversation.

For approaches using both environment and conversation observations, Direct (GPT-3.5) surpasses Proactive and ProCoT in question quality and macro clarification accuracy. However, it still performs worse than the environment-only method, regardless of Clarification Need Prediction or Clarification Question Generation. ICL (GPT-3.5) performs better in clarification need prediction but worse in question generation, supporting the argument that in-context learning is an unstable approach for the clarification subtask. Finally, our proposed CEP framework with either Mistral-7B or LLaMA-3-8B outperforms all other methods. Specifically, finetuned LLaMA-3-8B is better than finetuned Mistral-7B, mainly due to the larger parameter size and better pretraining data.

5.2.2 Evaluation on Tool Learning

As shown in Table 3, we evaluate tool learning in two settings, namely static and dynamic.

Discussion on Static vs. Dynamic Settings First of all, Brute-force outperforms all methods in the dynamic setting. In general, agents in the dynamic setting represent worse results compared with agents in the static setting, indicating that in a complex environment similar to the real-world situation, agents are more likely to fail to handle different exceptions during the interaction.

Comparison on Static Setting In the static setting, we observe the same phenomenon in the clarification subtask, *i.e.*, LLaMA-3-8B performs better, and even surpasses the Brute-force baseline. Zero-shot GPT-3.5 presents the undesired performance on API match, correctness, and pass rate. Interestingly, ToolLLM also produces unexpected results, suggesting that the pretraining knowledge from ToolBench did not translate into improved performance on our task.

Comparison on Dynamic Setting In the dynamic setting, due to the input token limitation of LLaMA-3-8B, we could only conduct experiments over GPT-3.5 and Mistral-7B for three different strategies. We find that GPT-3.5 is prone to generating actions without syntax errors. But when it comes to evaluating the quality of the generated actions, Mistral performs better. Secondly, we find the performance degradation trend of Mistral from the strategy *ReAct* to our proposed memory recollection mechanism, which is in contrast to the observation we find in GPT-3.5, probably because of the reasoning ability drops when processing the context with length over 8K. The results of GPT-3.5 prove the effectiveness of memory recollection. It can also be concluded that in a dynamic setting, all agents fail to consistently generate correct actions throughout the entire interaction. More analysis on dynamic execution agents is presented in Appendix C.

5.2.3 Evaluation on Planning

In the planning task, greedy search achieves a delivery rate of 100% and meets most constraints, but this does not result in a valid plan. For Mistral, we only test the Direct and CoT strategies because the agent gets trapped in a fail-retry loop when using ReAct and Reflexion. This scenario mirrors the performance drop observed in the tool learning task with a super-long context. For GPT-3.5, we observe the same trend in TravelPlanner (Xie et al., 2024), the performance degrades from direct prompting to Reflexion, demonstrating the difficulty for most advanced agents to finish the planning task. Among these results, only GPT-3.5 with direct prompting successfully generates one passed plan out of 1,000 samples in the test set, indicating the significant challenge of this problem.

5.2.4 Evaluation on the Integral Framework

Finally, we evaluate the integral CEP framework holistically rather than assessing the performance of each subtask independently. To this end, we adopt the execution interactions generated by the SOTA model in Table 3, CEP (LLaMA-3-8B), for

	Delivery Rate	Commonsense Pass Rate		Hard Constraint Pass Rate		Final Pass Rate
		Micro	Macro	Micro	Macro	
Greedy Search	100	76.9	0	64.5	46.7	0
Direct (Mistral-7B)	86.6	44.8	0.4	4.0	0.9	0
CoT (Mistral-7B)	61.5	29.8	0	2.4	0.1	0
Direct (GPT-3.5)	**98.6**	**63.7**	**0.7**	**19.0**	**5.1**	**0.1**
CoT (GPT-3.5)	77.5	50.0	0.6	16.2	5.2	0
ReAct (GPT-3.5)	68.7	38.0	0	3.2	0.6	0
Reflexion (GPT-3.5)	61.5	33.9	0	3.1	0.4	0

Table 4: Evaluation of planning.

Clarification						
	Clarif. Acc		Rule-based Score		GPT Score	
	Micro	Macro	Micro	Macro	Micro	Macro
$\text{CEP}_{\text{independent}}$ (LLaMA-3-8B)	99.4	98.2	69.7	55.8	85.8	77.0
$\text{CEP}_{\text{integral}}$ (LLaMA-3-8B)	97.3	92.9	68.4	54.7	85.1	76.0

Planning						
	Delivery Rate	Commonsense Pass Rate		Hard Constraint Pass Rate		Final Pass Rate
		Micro	Macro	Micro	Macro	
$\text{CEP}_{\text{integral}}$	98.8	64.3	1.0	19.2	5.0	0.1
$\text{CEP}_{\text{integral}}$ w/o Clarification	93.3	53.3	0.3	8.4	3.1	0

Table 5: Evaluation of the integral framework.

the subsequent clarification and the final planning. Results are reported in Table 5.

As for the clarification subtask, we use CEP (LLaMA-3-8B) based on the environment interaction generated by the execution agent and the current conversation history. Compared to the version supervised by ground-truth data, *i.e.*, $\text{CEP}_{\text{independent}}$ (LLaMA-3-8B), the performance of $\text{CEP}_{\text{integral}}$ (LLaMA-3-8B) slightly drops due to incorrectly-predicted tool usage, highlighting the robustness of our proposed CEP framework.

For the final planning subtask, we analyze the effect of user intention disambiguation by ablating the clarification agent. We adopt the best planning model from Table 4, *i.e.*, Direct (GPT-3.5), as the planning agent. Specifically, we feed the initial instruction without clarification into the planning agent, denoted as w/o Clarification, for comparison. The observations indicate that the planning agent fails to formulate valid plans without clarification, stating that user intention understanding is essential for effective plan delivery and meeting constraints. To intuitively illustrate the importance of clarification in agent planning, we present a detailed case study in Appendix D.

5.3 Detailed Analysis

Analysis of Clarification Recall We further examine the clarification accuracy from two angles: *clearness* and *vagueness* judgment recall (definitions are presented in Appendix B.2). It can be observed from Figure 4 that: (1) CEP

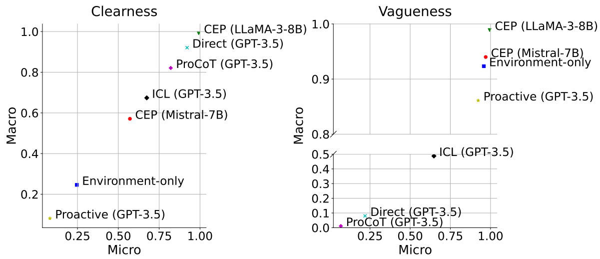


Figure 4: Analysis of clarification recall.

Constraint Type	Greedy Search			$\text{CEP}_{\text{integral}}$			$\text{CEP}_{\text{integral}}$ w/o Clarif.		
	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Commonsense Constraint									
Within Sandbox	100	100	100	38.8	38.1	42.3	23.6	22.2	21.0
Complete Information	100	100	100	89.7	89.8	74.6	67.0	55.0	48.0
Within Current City	100	100	100	69.8	76.3	77.7	62.9	64.3	62.1
Reasonable City Route	100	100	100	68.7	74.2	67.4	31.6	25.8	28.5
Diverse Restaurants	0	0	0	65.2	69.4	72.4	66.7	72.1	73.7
Diverse Attractions	100	100	100	93.1	93.7	92.2	89.7	90.4	89.0
Non-conf. Transportation	93.4	92.5	91.8	74.4	70.3	89.3	64.9	55.6	68.0
Minimum Nights Stay	20.4	24.9	22.9	6.0	5.1	6.0	31.6	30.0	34.5
Hard Constraint									
Budget	99.7	99.4	100	5.2	7.8	2.5	3.4	4.8	0.9
Accommodation	-	41.3	32.0	-	33.8	34.8	-	16.4	14.4
Cuisine	-	5.9	0	-	32.8	19.8	-	8.4	8.4
Transportation	-	-	55.0	-	-	37.2	-	-	15.6

Table 6: Analysis of plan constraints.

(LLaMA-3-8B) outperforms all other methods in both clearness and vagueness judgment. (2) Proactive (conversation-only) and environment-only approaches are inclined to request clarification, indicating that these agents struggle to make accurate judgments when either type of observation is absent. (3) ProCoT, Direct, and ICL (GPT-3.5) tend to not clarify the conversation, suggesting that full observations or invoking agents to engage in deeper reasoning might bring about hallucinations in judgment. (4) CEP (Mistral-7B) performs worse than CEP (LLaMA-3-8B) in both clearness and vagueness judgment, correlating with findings from static tool learning that LLaMA-3-8B's larger parameter size and better pretraining data lead to superior performance.

Analysis of Planning Constraints We report the detailed planning constraint pass rates for Greedy Search, $\text{CEP}_{\text{integral}}$ with and without clarification in Table 6. The description of these constraints is shown in Appendix B.2. Our findings include: (1) Compared to Greedy Search, $\text{CEP}_{\text{integral}}$ underperforms on most constraints except for *diverse restaurants* and *cuisine*, highlighting the limitations of off-the-shelf language agents in Proactive Agent Planning. (2) Clarification plays a critical role in enabling agents to meet most commonsense and all hard constraints, as evidenced by the lower pass rates of $\text{CEP}_{\text{integral}}$ without clarification. However, an exception is the constraint *minimum* *nights stay*, which checks whether the accommodation options entailed in a plan satisfy the minimum night requirements. This constraint demands mathematical understanding of the plan, while the lengthy context of clarification dialogues may introduce some noise into the planning process.

Analysis of Framework Efficiency To delve into the time complexity and communication overhead of the proposed multi-agent framework, we measure the execution time for $\texttt{CEP}_{\text{integral}}$ from Table 5. For each data sample involving multiple conversation turns and interaction iterations, we calculate the total time spent on three subtasks separately: an average of 14.17 seconds for tool learning, 5.40 seconds for clarification, and 8.78 seconds for planning. The experimental results demonstrate that $\texttt{CEP}_{\text{integral}}$ is efficient in real-world applications, as clarification requires significantly less time than tool learning and planning.

6 Conclusion

In this work, we introduce a new and practical problem of Proactive Agent Planning to address the challenges faced by LLM-powered language agents in handling unclear user instructions. To tackle these obstacles, we propose a novel multi-agent framework, CEP, which consists of clarification, execution, and planning agents. This framework features trajectory tuning and memory recollection to address challenges faced by current agents in requesting clarifications and executing tools within lengthy conversation contexts and complex environmental interactions. Additionally, we construct the first dataset specifically for studying the concerned problem, named Ask-before-Plan. Extensive evaluations and comprehensive analyses across diverse settings substantiate the effectiveness of the proposed CEP framework.

Limitations

User Simulation To simplify the evaluation on the clarification subtask, we employ static user simulation rather than real-time user simulation for dialogues. This approach allows us to focus exclusively on evaluating the agent's ability for clarification question generation, without the latent interference from incorrect user simulation. We believe that our proposed task could be easily extended to include dynamic real-time user simulation in future work.

Agent Planning The effectiveness of our proposed framework, CEP, has not been verified on other benchmarks, as we are the first to introduce the problem of Proactive Agent Planning and there are no other suitable datasets available. Due to the limited computing resources, we are unable to conduct experiments with larger-size models. In addition, while cooperation and division of labor among different LLM-based agents improve planning capabilities in uncertain situations (Song et al., 2024; Xie and Zou, 2024), our work does not thoroughly explore this. Our primary goal is to establish and benchmark this new and practical problem, as well as leave room for future research on this challenging problem.

Ethical Consideration

The Ask-before-Plan dataset was built from the TravelPlanner dataset (Xie et al., 2024), which is publicly available. We have obtained permission from the authors of TravelPlanner to utilize the dataset for further construction.

Acknowledgement

This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant (No. MSS24C004). This research/project is supported by A*STAR, CISCO Systems (USA) Pte. Ltd, and National University of Singapore under its Cisco-NUS Accelerated Digital Economy Corporate Laboratory (Award I21001E0002) and the National Research Foundation, Singapore, under its Industry Alignment Fund Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Mohammad Aliannejadi, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev. 2021. Building and evaluating open-domain dialogue corpora with clarifying questions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4473–4484. Association for Computational Linguistics.

Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D Goodman. 2024. Star-gate: Teaching language models to ask clarifying questions. *ArXiv preprint*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yue Chen, Chen Huang, Yang Deng, Wenqiang Lei, Dingnan Jin, Jia Liu, and Tat-Seng Chua. 2024. STYLE: improving domain transferability of asking clarification questions in large language model powered conversational agents. In *Findings of the Association for Computational Linguistics, ACL 2024*, pages 10633–10649.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023a. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*.

Yang Deng, Wenqiang Lei, Wai Lam, and Tat-Seng Chua. 2023b. A survey on proactive dialogue systems: problems, methods, and prospects. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23.

Yang Deng, Wenqiang Lei, Wenxuan Zhang, Wai Lam, and Tat-Seng Chua. 2022. PACIFIC: Towards proactive conversational question answering over tabular and textual data in finance. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6970–6984. Association for Computational Linguistics.

Yang Deng, Lizi Liao, Liang Chen, Hongru Wang, Wenqiang Lei, and Tat-Seng Chua. 2023c. Prompting and evaluating large language models for proactive dialogues: Clarification, target-guided, and non-collaboration. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics.

Yang Deng, An Zhang, Yankai Lin, Xu Chen, Ji-Rong Wen, and Tat-Seng Chua. 2024a. Large language model powered agents in the web. In *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024*, pages 1242–1245.

Yang Deng, Xuan Zhang, Wenxuan Zhang, Yifei Yuan, See-Kiong Ng, and Tat-Seng Chua. 2024b. On the multi-turn instruction following for conversational web agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8795–8812. Association for Computational Linguistics.

R. Descartes and D.A. Cress. 1998. *Discourse on Method (Third Edition)*. HPC Classics Series. Hackett Publishing Company.

Hubert L. Dreyfus. 1992. *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Advances in neural information processing systems*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2022. Clam: Selective clarification for ambiguous questions with generative language models. *ArXiv preprint*.

Junkai Li, Siyu Wang, Meng Zhang, Weitao Li, Yunghwei Lai, Xinhui Kang, Weizhi Ma, and Yang Liu. 2024a. Agent hospital: A simulacrum of hospital with evolvable medical agents. *ArXiv preprint*.

Zelong Li, Wenyue Hua, Hao Wang, He Zhu, and Yongfeng Zhang. 2024b. Formal-llm: Integrating formal language and natural language for controllable llm-based agents. *ArXiv preprint*.

Lizi Liao, Grace Hui Yang, and Chirag Shah. 2023. Proactive conversational agents in the post-chatgpt world. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23. Association for Computing Machinery.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022. Fp8 formats for deep learning. *ArXiv preprint*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

Valentina Pyatkin, Jena D. Hwang, Vivek Srikumar, Ximing Lu, Liwei Jiang, Yejin Choi, and Chandra Bhagavatula. 2023. ClarifyDelphi: Reinforced clarification questions with defeasibility rewards for social and moral situations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1088–1113. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR 2024*.

Sudha Rao and Hal Daumé III. 2018. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2737–2746. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*.

Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Reis, Jeffrey Jopling, and Michael Moor. 2024. Agentclinic: a multimodal agent benchmark to evaluate ai in simulated clinical environments. *ArXiv preprint*.

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

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. 2024. Adaptive in-conversation team building for language model agents. *ArXiv preprint*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. Llms in the imaginarium: tool learning through simulated trial and error. *ArXiv preprint*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *ArXiv preprint*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023b. A survey on large language model based autonomous agents. *ArXiv preprint*.

Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. 2023. Dilu: A knowledge-driven approach to autonomous driving with large language models. *ArXiv preprint*.

Zeqiu Wu, Ryu Parish, Hao Cheng, Sewon Min, Prithviraj Ammanabrolu, Mari Ostendorf, and Hannaneh Hajishirzi. 2023. InSCIt: Information-seeking conversations with mixed-initiative interactions. *Transactions of the Association for Computational Linguistics*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *ArXiv preprint*.

Chengxing Xie and Difan Zou. 2024. A human-like reasoning framework for multi-phases planning task with large language models. *ArXiv preprint*.

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. In *ICML 2024*.

Jingjing Xu, Yuechen Wang, Duyu Tang, Nan Duan, Pengcheng Yang, Qi Zeng, Ming Zhou, and Xu Sun. 2019. Asking clarification questions in knowledge-based question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1618–1629. Association for Computational Linguistics.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*.

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

Hamed Zamani, Susan T. Dumais, Nick Craswell, Paul N. Bennett, and Gord Lueck. 2020. Generating clarifying questions for information retrieval. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 418–428. ACM / IW3C2.

An Zhang, Yang Deng, Yankai Lin, Xu Chen, Ji-Rong Wen, and Tat-Seng Chua. 2024a. Large language model powered agents for information retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024*, pages 2989–2992.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. 2024b. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *ACL 2024*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 17.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *ArXiv preprint*.

A Dataset Specifications

To adapt TravelPlanner to our proposed task, we modify the dataset and environment for proactive information seeking. In the following sections, we share our implementation of the **Ask-before-Plan** dataset along with the environment.

A.1 Indefinite Detail Formation

To simplify the dataset and tailor it to our task, we specify the destination cities and related arrival days, rather than providing the destination state and the number of cities to visit. In this case, we present only the departure date for each data sample in lieu of the date range. Besides, we combine the room type and the house rule under local constraints as the *accommodation*. Table 7 shows the description of elements in the dataset. To construct the unfeasible details, we apply the following procedures for each feature:

- Accommodation: We search all combinations of room types and house rules for every destination in the environment to identify candidates with no matching information, and then randomly select one of these accommodation preferences as the unfeasible detail.
- Cuisine: We iterate all possible cuisine preferences in each destination. If some cuisine preferences are not available in any destination, we randomly choose one of them as the unfeasible detail.
- Transportation: We assign the preference "No flight" or "No self-driving" as the unfeasible detail if the corresponding transportation mode is not available for the route described in the instruction. It is guaranteed in the environment that both preferences will not be unavailable simultaneously.
- Destination by Arrival Days: We randomly modify one of the destinations to be unreachable from the previous city in the travel route. To preserve the consistency of destinations, cities within the same state as the original destination are preferentially chosen for modification.
- Budget: We begin by estimating the lower bound of the budget for the user's travel plan by summing the accommodation and transportation expenses. To determine the minimum accommodation cost, we select the least expensive accommodation in the given city that satisfies the *minimum nights* constraint. The minimum transportation cost is calculated by choosing the cheapest option between "No flight" and "No self-driving". Ultimately, we subtract a random integer, ranging from 10 to 10% of the final estimated lower bound, from the total approximate budget.

Upon obtaining all the indefinite details, we select between 0 and 3 for each instruction. Afterwards, we apply the topological sort algorithm, based on the directed acyclic graph illustrated in Figure 5, to regulate the dependency between these details. Eventually, we utilize the prompts outlined in E.1 and employ `gpt-3.5-turbo-0125` to rewrite the original instruction.

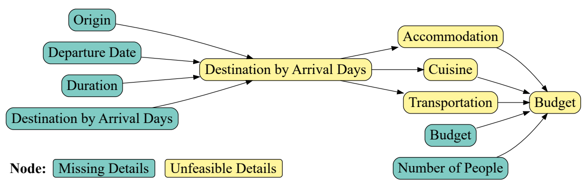


Figure 5: Dependency graph for indefinite details.

A.2 Proactive Conversation Generation

We apply `gpt-3.5-turbo-0125` to implement conversation generation, with the specific prompts utilized detailed in Section E.1. Notably, to accurately guide GPT-3.5 in generating role-playing dialogues, we incorporate both XML and Python-type structures into the prompts. In the quality control phase, we initially use keyword matching to ensure that relevant keywords are included in both the questions and responses. Finally, we select 20% of the data points at random for proofreading.

Destination by Arrival Days Definition: *An array depicts the destination city with the day to arrive.* Example in the Natural Language Instruction: *We plan to visit Dallas on the 1st day and Houston on the 3rd day...*
Duration Definition: *The number of travel days.* Example in the Natural Language Instruction: *We are planning a 3-day trip...*
Departure Date Definition: *Date of departure from the origin city.* Example in the Natural Language Instruction: *Would you be able to organize a trip on March 18th, 2022...*
Number of People Definition: *The total number of individuals on the trip.* Example in the Natural Language Instruction: *Please organize a trip for 2 individuals...*
Budget Definition: *The budget for the trip in integers.* Example in the Natural Language Instruction: *... we have a budget of $1600...*
Origin Definition: *The departure city of the trip.* Example in the Natural Language Instruction: *Would you mind creating a travel plan starting from St. Louis...*
Accommodation Definition: *An optional array describes the accommodation preferences, consisting of the room type and the house rule. Possible values for the room type include "shared room", "not shared room", "private room", and "entire room". Possible values for the house rule include "parties", "smoking", "children under 10", "pets", and "visitors".* Example in the Natural Language Instruction: *... Our accommodation preference is a place that allows parties...*
Transportation Definition: *An optional string indicates transportation preferences, such as "No flight" and "No self-driving".* Example in the Natural Language Instruction: *... Additionally, we prefer not to use flights for transportation...*
Cuisine Definition: *An optional array represents cuisine preferences, with possible values including "Chinese", "American", "Italian", "Mexican", "Indian", "Mediterranean", and "French".* Example in the Natural Language Instruction: *... We are interested in exploring Chinese, Indian, Mexican, and Italian cuisines...*

Table 7: Description of elements in the dataset.

A.3 Environment Settings

To align the TravelPlanner environment with our proposed task, we modify it by removing tools such as *NotebookWrite* and *CitySearch*. We introduce a new tool named *BudgetEstimator* to accommodate the unfeasible detail, *budget*. Furthermore, we adjust the required parameters of each tool to better suit our dataset setup. The description of modified tools in our environment is as follows:

- **AccommodationSearch**: Instead of merely searching for accommodation options in the specified city, an optional parameter, *filters*, is introduced to describe the user's accommodation preferences. This parameter corresponds directly to the dataset element *accommodation*.
- **RestaurantSearch**: We add an optional parameter, *cuisines*, to capture the user's culinary preferences, corresponding to the dataset element *cuisine*. This enhancement enables the tool to list all restaurants in the selected city and verify the availability of the user's preferred cuisine.
- **BudgetEstimator**: This tool is used to calculate the minimum budget required for the trip based on the cheapest accommodation and transportation options. The parameters include the departure city, destination cities, dates of arrival in each city, and the number of people.

B Details of Experimental Setups

B.1 Baselines

We describe the details of the adopted baselines. The prompts for the clarification, execution, and planning subtasks are exhibited in Sections E.2, E.3, and E.4.

Clarification For the clarification subtask, we adopt the following baselines:

- Environment-only: To implement the generation of clarification questions based on environmental feedback, we prepare predefined questions tailored to two scenarios: one where a tool invocation is not recorded in the interaction log, and another where a tool invocation suggests that the current travel plan conversation is infeasible. For each interaction history, if one of these scenarios occurs, we add the corresponding questions to the candidate pool and finally sample one question from this pool to ask the user.

- Proactive (Deng et al., 2023c): This method provides alternative choices for agents to decide whether to ask a question for disambiguation or take no action.

- ProCoT (Deng et al., 2023c): This approach distinguishes itself from the *Proactive* baseline by involving reasoning about the next action prior to making a decision.

- Direct: Given the conversation history and the current interaction log, the agent is prompted to finish Clarification Need Prediction and Clarification Question Generation without additional strategies. In the conversation-only setting, environmental information is excluded from the prompts.

- ICL (Brown et al., 2020): Based on Direct prompting, two examples for clarification need prediction and one for question generation are selected from the training dataset and incorporated into the context.

In Clarification Need Prediction, we record the first "Yes" or "No" from the agent's response as the final judgment. If the response does not contain "Yes" or "No", we consider the prediction to be "No".

Execution We adopt the following baselines in the execution subtask:

- Brute-force: We provide the brute-force solution, as a representative of the traditional algorithm, for the execution subtask. As the name suggests, this solution invokes all tools, regardless of syntax correctness, based on the information from the conversation history.

- Direct: In this method, the agent is prompted to generate all possible tool invocations according to the conversation history. For GPT-3.5, we provide an example to enable in-context learning.

- ToolLLM (Qin et al., 2024): Following the Direct prompting, we train ToolLLaMA on our Ask-before-Plan dataset in the static setting for 3 epochs, after pretraining on ToolBench for 2 epochs, to match the total number of training epochs with `CEP`.

- ReAct (Yao et al., 2023): This prompting strategy necessitates that the agent integrate environmental observations into its reasoning process. In the execution subtask, the ReAct agent produces step-by-step tool commands based on the conversation history and its previous interactions with the environment, cycling through *thought*, *action*, and *observation*.

- Reflexion (Shinn et al., 2023): This approach is similar to ReAct, but the agent is required to reflect on the incorrect tool invocations and provide a high-level explanation of the error in order to improve its reasoning.

Planning We use the same baseline setting as described in the TravelPlanner sole-planning mode, which is outlined as follows:

- Greedy Search: We borrow the greedy search algorithm from TravelPlanner and adapt it to our planning subtask. To illustrate, we remove destination selection since our dataset specifies the city to visit for 5- or 7-day travel plans.

- Direct: In this baseline, the agent directly generates the plan given the conversation and interaction history.

- CoT (Kojima et al., 2023): Compared with the direct baseline, we add the prompt "Let's think step by step" to elicit reasoning capability.

- ReAct (Yao et al., 2023): Similar to ReAct in the execution subtask, we only provide the agent with a cost calculation environment to assist in generating the travel plan.

- Reflexion (Shinn et al., 2023): Based on ReAct, we introduce the self-reflection module, similar to the execution subtask, for guiding the agent to generate a valid travel plan.

Unlike TravelPlanner, which first generates a natural language plan and then uses GPT-4 to convert it into a JSON format, our planning subtask requires directly generating a JSON travel plan.

B.2 Evaluation Metrics

Considering the conversational nature of clarification and execution subtasks, we introduce two different evaluation strategies: *micro* and *macro*. The micro strategy calculates the average per conversation turn, whereas the macro strategy calculates the average per data sample. In the planning subtask, the definition of *micro* and *macro* strategies is consistent with TravelPlanner (Xie et al., 2024), which is the ratio of passed constraints to the total number of constraints and the ratio of passed constraints among all tested plans, respectively.

Clarification We incorporate the following metrics to evaluate the clarification subtask:

- Clarification Accuracy: The percentage of correct Clarification Need Prediction. A prediction is considered correct if the agent accurately identifies when a conversation requires clarification.

- Clearness Recall: The ratio of conversation turns predicted to be clear to the total number of turns that do not require clarification. This measures the agent's ability to identify unambiguous user inquiries.

- Vagueness Recall: The ratio of conversation turns predicted to be vague to the total number of turns that require clarification. This metric evaluates the agent's capability to identify ambiguous user inquiries.

- Rule-based Score: The percentage of generated questions containing the predefined keywords for respective indefinite details.

- BLEU (Papineni et al., 2002): This metric measures the similarity between the generated question and the ground truth in terms of n-gram overlap. We set the maximum n-gram order to 4 without smoothing.

- GPT Score: We utilize the OpenAI model `gpt-4-turbo-2024-04-09` with zero temperature to judge the correctness of the generated questions. Prompts are shown in Section E.2.

Note that the last three metrics are exclusively used to assess the correctness of the generated questions. Consequently, we perform these calculations only when the Clarification Need Prediction is correct. Given that multiple indefinite details may require clarification within a single conversation turn, for these metrics, we calculate the highest score across all candidate indefinite details.

Execution For conversation turn t, we denote the generated tool calls as F^t_{gen} and the ground truth tool calls as F^t_{gt}. We define $\text{set}(\cdot)$ as a function that eliminates duplicate elements from an array. Following Wang et al. (2024), the metrics for the execution subtask are as follows:

- Well-formed: The percentage of F^t_{gen} without syntax errors.

- API Match: This metric calculates the ratio of matched API names between $\text{set}(F^t_{\text{gen}})$ and $\text{set}(F^t_{\text{gt}})$ to the total number in $\text{set}(F^t_{\text{gt}})$.

- Repeat Rate: The percentage of repeated tool calls in F^t_{gen}.

- Correctness: We measure the precision, recall, and F1 score between $\text{set}(F^t_{\text{gen}})$ and $\text{set}(F^t_{\text{gt}})$.

- Pass Rate: This metric in the micro strategy evaluates the full match of tool execution, *i.e.*, $\text{set}(F^t_{\text{gen}}) = \text{set}(F^t_{\text{gt}})$. This macro metric evaluates that $\forall t, \text{set}(F^t_{\text{gen}}) = \text{set}(F^t_{\text{gt}})$.

Planning In this subtask, we adapt the evaluation metrics from TravelPlanner (Xie et al., 2024) to our setting, shown as follows:

- Delivery Rate: The percentage of valid JSON plans within a limited interaction step.

- Commonsense Constraint Pass Rate: This metric evaluates whether the planning agent could incorporate commonsense, incorporating *within sandbox*, *complete information*, *within current city*, *reasonable city route*, *diverse restaurants*, *diverse attractions*, *non-conflict transportation*, *minimum nights stay* into the plan without explicit instructions.

- Hard Constraint Pass Rate: This metric measures whether a generated plan satisfies all hard constraints, including *budget*, *accommodation*, *cuisine*, *transportation*.

- Final Pass Rate: The metric reports the percentage of valid JSON plans that satisfy all constraints.

To differentiate, we combine the hard constraints *room rule* and *room type* from TravelPlanner into *accommodation* to fit our dataset. Since we specify the destinations and their arrival days in our dataset, we do not evaluate the number of visiting cities or the state of destinations in our commonsense constraint assessment. Additionally, the *minimum nights stay* constraint in our setting is designed to verify whether the accommodation option provided by the agent aligns with the user's expected duration of stay in a city.

B.3 Implementation Details

We choose Mistral-7B-Instruct[3], ToolLLaMA-2-7B[4], and LLaMA-3-8B-Instruct[5] for finetuning and inference using a single NVIDIA A5000 24GB. To facilitate finetuning, we apply LoRA (Hu et al., 2022) in 4-bit quantization and train for 5 epoches with a maximum context length of 4096. To optimize the open-source LLM inference, FP8 E5M2 KV Cache (Micikevicius et al., 2022) is exploited.

In dynamic execution and planning evaluations, we exclude LLaMA-3-8B due to the limited context window length. To maximize the utilization of the context window, we truncate the environment feedback in *Clarification* and *Execution* subtasks to 3 items. In terms of dynamic prompting strategies such as ReAct and Reflexion, we set the maximum interaction steps to 30 and the maximum action retries to 3. Our temperature is set to 0 for all experiments.

For the integral framework, we feed the clarification need prediction and execution interactions generated by `CEP` (LLaMA-3-8B), along with the ground truth conversation, into the final planning.

[3] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

[4] https://huggingface.co/ToolBench/ToolLLaMA-2-7b-v2

[5] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

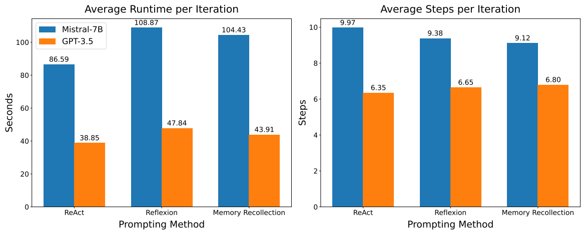


Figure 6: Average runtime and steps in dynamic execution.

C Additional Study on Dynamic Execution Agents

We discuss the time complexity and execution status of the dynamic execution agents in this section.

C.1 Analysis of Time Complexity

To precisely outline the memory recollection, let C_t denote the conversation at turn t, associated with the clarified detail set $D'_t = \{d_0, d_1, \ldots, d_t\}$, where d_0 pertains to the details entailed in the initial query q_0. It can be observed that C_{t-1} and C_t share the common clarified details, defined as $D'_t - D'_{t-1} = \{d_0, d_1, \ldots, d_{t-1}\}$. We postulate that an agent, operating under deterministic parameters, consistently replicates specific errors when invoking tools on these shared details. In the Reflexion framework, if the rationales generated during conversation turn t are represented by R_t, then the cumulative number of generated rationales across T turns is expressed as $\sum_{t=1}^{T} |R_t|$. Conversely, in memory recollection, as we *reuse* the past rationales, this can be reduced to $\max_{t=1}^{T} |R_t| = |R_T|$. In conclusion, the memory recollection mechanism effectively reduces the number of generated rationales, thereby diminishing the time complexity of rationale inference from linear to constant time.

Figure 6 showcases a comparison in average runtime and execution steps between ReAct, Reflexion, and Memory Recollection. We observe that Reflexion consumes more time than ReAct for both Mistral-7B and GPT-3.5, while the Memory Recollection module alleviates this effect. Moreover, Mistral-7B, on average, takes longer inference time than GPT-3.5. We also examine the mean execution steps per iteration for Mistral-7B and GPT-3.5, revealing a starkly contrasting trend: Mistral-7B shows a decrease in steps from ReAct to Memory Recollection, while GPT-3.5 exhibits an increase. This phenomenon, detailed in Table 3, suggests that more execution steps lead to improved performance, as the agent engages more

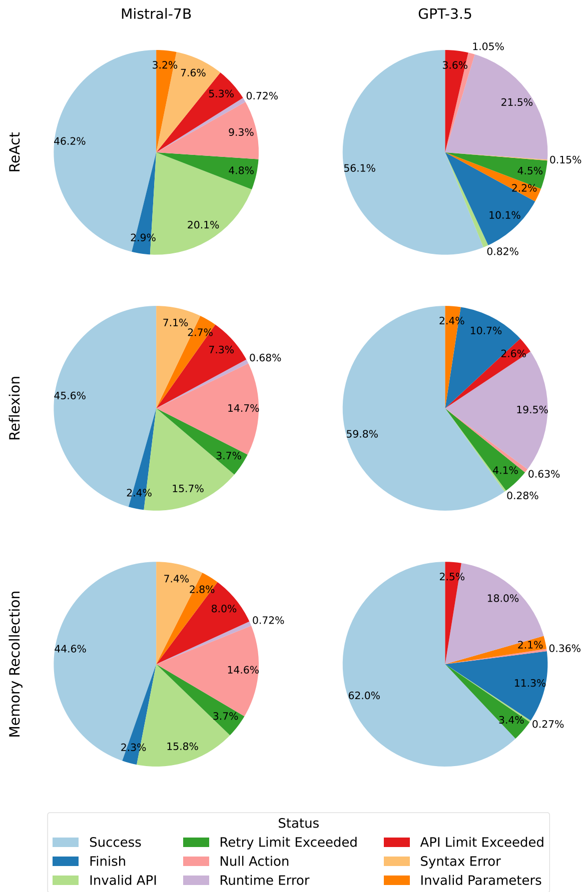


Figure 7: Execution status distributions. Statuses with proportions less than 0.1% are ignored.

with the environment and gathers more information.

C.2 Analysis of Execution Status

Figure 7 presents the distribution of each execution status for Mistral-7B and GPT-3.5. From a holistic perspective, we notice that Reflexion eliminates the errors consisting of *Invalid API*, *Retry Limit Exceeded*, and *Invalid Parameters*. In the case of Mistral-7B, prompting strategies like Reflexion and Memory Recollection lead to lower *Success* and *Finish* rates, while the probability of exceptions such as *API Limit Exceeded* and *Null Action* rises. This is in line with our observation in Table 3. In contrast, the performance improvements of GPT-3.5 are reflected in higher *Success* and *Finish* rates, and lower error rates.

D Case Study

We display two detailed case studies to explore the impact of clarification on subsequent planning subtasks. We compare plans generated by $\texttt{CEP}_{\text{integral}}$ with and without clarification in Tables 8 and 9. Note that each table includes the initial user instruction, the type of indefinite detail, the question generated by finetuned LLaMA-3-8B, the ground truth response, and the planning outcome. Discrepancies resulting from the clarification are highlighted in yellow. From Table 8, it is evident that without asking for the missing destination detail, the agent incorrectly confines the itinerary to the origin city, Nashville, on the first day, leading to a confusing itinerary. In Table 9, if the agent overlooks that the previously specified accommodation preferences in Houston are unavailable and fails to inquire about the user's opinion, an incomplete travel plan will be generated.

E Prompt List

E.1 Data Annotation

We present the prompts for original instruction revision during the indefinite detail formation stage in Table 10. Prompts and examples involved in proactive conversation generation are shown in Tables 11, 12, and 13.

E.2 Clarification Subtask

Since the clarification subtask necessitates the interaction history from the execution stage as input, we provide the relevant tool description in Snippet 1. Prompts of conversation-only methods, including Proactive and ProCoT, as well as the direct strategy, are exhibited in Snippets 2 and 3. Furthermore, we also specify the prompts for GPT-4 evaluation in Table 14.

E.3 Execution Subtask

The tool description used in execution is showcased in Snippet 4. We demonstrate the prompts for static and dynamic settings in Snippets 5 and 7. The error prompts for dynamic interaction are presented in Snippet 6.

E.4 Planning Subtask

The prompts for static planning, dynamic planning, and errors in dynamic interaction are displayed in Snippets 8, 10, and 9.

Instruction: Could you suggest a 5-day travel itinerary for a solo traveler starting from Nashville, departing on March 13th, 2022, with a budget of $2123?
Missing Detail: Destination by arrival days.
Question: Which cities are you interested in visiting during your 5-day trip, and on which specific days do you plan to arrive in each city?
Response: I can't wait to explore the vibrant streets of San Diego on day 1 and then soak up the glitz and glamour of Los Angeles on day 3!

	$\text{CEP}_{\text{integral}}$	$\text{CEP}_{\text{integral}}$ w/o Clarification
Day	1	1
Current City	from Nashville to San Diego	Nashville
Transportation	Flight Number: F3973475, from Nashville to San Diego, Departure Time: 09:14, Arrival Time: 11:22, Cost: $485	N/A
Breakfast	Open Yard, San Diego	Biscuit Love Gulch, Nashville
Attraction	Cabrillo National Monument, San Diego; La Jolla Shores Park, San Diego	Country Music Hall of Fame and Museum, Nashville; Ryman Auditorium, Nashville
Lunch	Bikaner Sweets, San Diego	Hattie B's Hot Chicken, Nashville
Dinner	Chawlas 2, San Diego	Martin's Bar-B-Que Joint, Nashville
Accommodation	Spacious Room in Large 2 Bedroom Prewar Apartment, San Diego	Cozy Loft in East Nashville, Nashville

Table 8: Comparison of travel plans for the missing detail, destination by arrival days.

Instruction: I am planning a solo trip starting from Bloomington. The itinerary includes visiting Dallas on day 1, Amarillo on day 3, and Houston on day 5. I will be departing on March 5th, 2022. I am looking for shared room accommodations that allow pets.
Unfeasible Detail: Accommodation.
Question: It seems like the accommodation preferences you've specified might not be feasible for this trip. Is there a different type of accommodation you would consider for your solo trip?
Response: Any accommodation preference is acceptable.

	$\text{CEP}_{\text{integral}}$	$\text{CEP}_{\text{integral}}$ w/o Clarification
Day	5	5
Current City	from Amarillo to Houston	from Amarillo to Houston
Transportation	Flight Number: F3822285, from Amarillo to Houston, Departure Time: 06:31, Arrival Time: 08:05, Cost: $103	Taxi: from Amarillo to Houston, Duration: 8 hours 49 mins, Distance: 965 km, Cost: 965
Breakfast	N/A	Jalapenos, Houston
Attraction	Downtown Aquarium, Houston; Space Center Houston, Houston	Downtown Aquarium, Houston; Space Center Houston, Houston
Lunch	Jalapenos, Houston	The BrewMaster - The Mix Fine Dine, Houston
Dinner	Royal Mart, Houston	"Pebble Street, Houston
Accommodation	Superhost 3 bedroom DISCOUNT, Houston	N/A

Table 9: Comparison of travel plans for the unfeasible detail, accommodation.

Role	Content
system	Given a JSON with data where 'org' denotes the departure city, 'dest' contains the destination cities to be visited along with the arrival day, and 'housing' contains the room type and house rules. Please generate a natural language query beginning with '{begin_word}' and including the arrival day if available.
user	JSON: {"org": "Gulfport", "dest": ["Charlotte on day 1"], "departure_date": "2022-03-05", "budget": 1800} QUERY:
assistant	Please design a travel plan from Gulfport heading to Charlotte on day 1, departing on March 5th, 2022, with a budget of $1800.
user	JSON: {"org": "Omaha", "days": 5, "people_number": 7, "housing": ["pets"], "budget": 35300} QUERY:
assistant	Could you provide a 5-day travel itinerary for a group of seven, starting in Omaha? Our budget is set at $35,300, and it's essential that our accommodations be pet-friendly since we're bringing our pets.
user	JSON: {"dest": ["Atlanta on day 1", "Augusta on day 3", "Savannah on day 5"], "days": 7, "departure_date": "2022-03-01", "people_number": 2, "housing": ["entire room", "parties"], "cuisine": ["Italian", "Indian"], "transportation": "no self-driving"} QUERY:
assistant	I'm looking for a week-long trip for 2 individuals. We plan to visit Atlanta on the 1st day, Augusta on the 3rd day, and Savannah on the 5th day. For accommodations, we'd prefer an entire room that allows parties. We intend to depart on March 1st, 2022, and will navigate our journey without self-driving. In terms of food, we're enthusiasts of Italian food, and we'd also appreciate indulging in genuine Indian cuisine.

Table 10: Prompts for original instruction revision.

Type	System Message
Missing Detail Observation	You are an intelligent agent designed to interact with users to clarify and specify their requests. When given a user's initial query and a specific detail that is missing, your task is to generate a natural, conversational question to obtain that specific missing information from the user. Your response should use coreference or omission to refer back to the initial query, minimizing direct repetition of its details.
Unfeasible Detail Observation	You are an intelligent agent designed to interact with users to clarify and specify their requests based on the search results from external tools. When a user's initial query includes unfeasible details, as determined by these search results, your task is to inform the user that the initial query cannot be fulfilled due to these details. Then, generate a natural, conversational question to obtain an alternative option from the user. Your response should use coreference or omission to refer back to the initial query, minimizing direct repetition of its details.
Succinct User Response	You are an intelligent agent designed to act as a real human user talking to a travel agent. When asked for details or clarifications about your travel plans, reply succinctly and directly using only the provided draft answers, ensuring your responses are natural, human-like, and creative without repeating the question.
Passionate User Response	You are an intelligent agent designed to act as a real human user talking to a travel agent. When asked for details or clarifications about your travel plans, reply diversely and passionately using only the provided draft answers, ensuring your responses are natural, human-like, and creative without repeating the question.

Table 11: System messages for simulated conversations.

Destination by Arrival Days

Instruction: *Could you provide a 5-day travel itinerary for a group of seven, starting in Omaha? Our budget is set at $35,300, and it's essential that our accommodations be pet-friendly since we're bringing our pets.*
Observation: *<missing_detail> Destinations and arrive days of the trip </missing_detail>*
Question: *Could you specify which cities you plan to visit during the trip, and the specific days you plan to arrive in each city?*
Thought: *<draft_answer> destinations_and_arrive_days = ... </draft_answer>*
Answer: *We plan to go Seattle on the 1st day.*

Duration

Instruction: *Please design a travel plan departing from Gulfport and heading to Charlotte on day 1, departing on March 5th, 2022, with a budget of $1800.*
Observation: *<missing_detail> Number of days for the trip </missing_detail>*
Question: *Sorry for the confusion, but could you please clarify the number of days you plan to spend on this trip?*
Thought: *<draft_answer> number_of_days_for_trip = ... </draft_answer>*
Answer: *5 day.*

Departure Date

Instruction: *I'm looking for a week-long trip for 2 individuals. We plan to visit Atlanta on the 1st day, Augusta on the 3rd day, and Savannah on the 5th day. For accommodations, we'd prefer an entire room that allows parties. We don't like driving during our journey.*
Observation: <missing_detail> Departure date of the trip </missing_detail>
Question: *I think I missed the departure date for your trip. Could you provide that information?*
Thought: *<draft_answer> departure_date = ... </draft_answer>*
Answer: *March 1st, 2022.*

Number of People

Instruction: *Could you provide a 5-day travel itinerary, starting in Omaha? Our budget is set at $35,300, and it's essential that our accommodations be pet-friendly since we're bringing our pets.*
Observation: *<missing_detail> Number of people on the trip </missing_detail>*
Question: *I'm not sure about the number of people in your group. Would you mind sharing that information?*
Thought: *<draft_answer> total_number_of_people_including_me = ... </draft_answer>*
Answer: *We are a group of seven.*

Budget

Instruction: *Please design a travel plan departing from Gulfport and heading to Charlotte on day 1, departing on March 5th, 2022.*
Observation: *<missing_detail> Budget of the trip </missing_detail>*
Question: *It seems you haven't mentioned the expected budget for this trip. Could you provide that information?*
Thought: *<draft_answer> budget_of_trip = ... </draft_answer>*
Answer: *Our budget for this trip is $36,000.*

Origin

Instruction: *We plan to visit South Bend on the 1st day, Ithaca on the 3rd day departing on March 5th, 2022 for a 5-day trip. Our budget is $1800.*
Observation: *<missing_detail> Departure city of the trip </missing_detail>*
Question: *Sorry, I am not sure about the departure city for your trip. Could you provide that information?*
Thought: *<draft_answer> departure_city = ... </draft_answer>*
Answer: *Ann Arbor.*

Table 12: Examples of missing details in proactive conversation generation.

Budget

Instruction: *Could you provide a 5-day travel itinerary for a group of seven, starting in Omaha? Our budget is set at $35,300, and it's essential that our accommodations be pet-friendly since we're bringing our pets.*
Observation: *<search_result> The current budget ... is not sufficient for the trip. The minimum budget required is ... </search_result>*
Question: *It seems like the budget you've mentioned might be insufficient for your trip. Would you be open to considering a higher budget?*
Thought: *<draft_answer> Adjust the budget to ... </draft_answer>*
Answer: *My Budget could be $40,000.*

Accommodation

Instruction: *Please design a travel plan departing from Gulfport and heading to Charlotte on day 1, departing on March 5th, 2022. I prefer to stay in private rooms that allow parties.*
Observation: *<search_result> The current accommodation preference ... is not available for the trip. </search_result>*
Question: *The accommodation preference you've mentioned might be a bit restrictive. Would you be open to considering other options?*
Thought: *<draft_answer> ... </draft_answer>*
Answer: *Sure, how about any rooms that allow parties?*

Transportation

Instruction: *I'm looking for a week-long trip for 2 individuals. We plan to visit Atlanta on the 1st day, Augusta on the 3rd day, and Savannah on the 5th day. For accommodations, we'd prefer an entire room that allows parties. We don't like driving during our journey.*
Observation: *<search_result> The current transportation preference ... is not available for the trip. </search_result>*
Question: *It seems like navigating the entire trip without self-driving might be quite challenging. Would you be open to considering other transportation options?*
Thought: *<draft_answer> ... </draft_answer>*
Answer: *Any transportation preference is acceptable.*

Cuisine

Instruction: *Could you provide a 5-day travel itinerary for a group of seven, starting in Omaha? We plan to go Seattle on the 1st day. We enjoy Chinese and Italian cuisine.*
Observation: *<search_result> The current cuisine preference ... is not available in any of the cities you plan to visit. </search_result>*
Question: *Sorry, I do not find any Chinese restaurants in cities you plan to visit. Could you provide alternative cuisine preferences?*
Thought: *<draft_answer> ... </draft_answer>*
Answer: *I am open to any cuisine preference.*

Destination by Arrival Days

Instruction: *We plan to visit South Bend on the 1st day, Ithaca on the 3rd day departing from Richmond on March 5th, 2022 for a 5-day trip. Our budget is $1800.*
Observation: *<search_result> ... is unreachable by all available transportation methods. </search_result>*
Question: *Sorry, I couldn't find a way to arrive in Ithca. Could you provide an alternative destination?*
Thought: *<draft_answer> My alternative destination is ... </draft_answer>*
Answer: *Charlotte could be my alternative destination.*

Table 13: Examples of unfeasible details in proactive conversation generation.

Message Type	Content
System Message	You are a helpful assistant skilled at evaluating questions.
User Message for Missing Details	Please check if the following question exclusively asks for [...], rather than [...]. Provide a simple "Yes" or "No" answer. Question: [...]
User Message for Unfeasible Details	Please check if the question indicates that the initial [...] is/are unfeasible and requests changes to the [...], rather than [...]. Provide a simple "Yes" or "No" answer. Question: [...]

Table 14: Prompts for clarification evaluation.

SNIPPET 1: TOOL DESCRIPTION IN CLARIFICATION

1. AccommodationSearch(city, filters)
– Description: Discover accommodations in your desired city with specific filters.
– Parameters:
  – city (str, required): The name of the city where you're seeking accommodation.
  – filters (list[str], required): A list of filters to refine your search. Choices include "shared room", "not shared room ", "private room", "entire room", "parties", "smoking", "children under 10", "pets", "visitors". If the user does not specify any accommodation filters, assign an empty list "[]" to the parameter.

2. RestaurantSearch(city, cuisines)
– Description: List all restaurants in your chosen city, regardless of cuisine type, and check if any of the cuisines you specify are unavailable there.
– Parameters:
  – city (str, required): The name of the city where you're seeking restaurants.
  – cuisines (list[str], required): A list of desired cuisines to check for availability. Available options include " Chinese", "American", "Italian", "Mexican", "Indian", "Mediterranean", and "French". If the user does not specify any cuisines, assign an empty list "[]" to the parameter.

3. AttractionSearch(city)
– Description: Find attractions in a city of your choice.
– Parameters:
  – city (str, required): The name of the city where you're seeking attractions.

4. DistanceMatrix(origin, destination, mode)
– Description: Estimate the distance, time, and cost between two cities.
– Parameters:
  – origin (str, required): The departure city of your journey.
  – destination (str, required): The destination city of your journey.
  – mode (str, required): The method of transportation. Choices include "self–driving" and "taxi".

5. FlightSearch(origin, destination, date):
– Description: A flight information retrieval tool.
– Parameters:
  – origin (str, required): The city you'll be flying out from.
  – destination (str, required): The city you aim to reach.
  – date (str, required): The date of your travel in "YYYY–MM–DD" format.

6. BudgetEstimator(origin, destination, dates, people_number)
– Description: Calculate the minimal estimated budget for the trip. Use this tool to verify if the budget provided by the user is sufficient.
– Parameters:
  – origin (str, required): The departure city of your trip.
  – destination (list[str], required): A list of cities you plan to visit during your trip.
  – dates (list[str], required): A list of dates corresponding to the departure from the origin and each of the destinations. The first date is the departure from the origin, and each subsequent date corresponds to the departure from the respective city in the destination list. The last date in this list is the departure from the final destination back to the origin city. All dates should be formatted as "YYYY–MM–DD".
  – people_number (int, required): The number of people on the trip.

Snippet 2: Conversation-only Prompts

System Message

Your current task is to determine the user's intentions and satisfy their needs based on the provided conversation between the user and the assistant.

Proactive User Message

Conversation: [...]
Based on the conversation, you have two options: ask a clarifying question or take no action. Choose the appropriate option to formulate your answer, starting with either "The clarifying question is" or "No action should be taken". Note that if the user does not specify details regarding accommodation, cuisine, or transportation in the initial query, it indicates that the user does not have specific preferences that need clarification in the following conversation.

ProCoT User Message

Conversation: [...]
Based on the conversation, first determine whether the user's request is ambiguous. A request is ambiguous if it contains missing or unfeasible details. If it is ambiguous, ask a clarifying question. If it is not ambiguous, no action is needed. Your response should start with an analysis of the ambiguity and then conclude with either "Therefore, the request is not ambiguous. No action should be taken." or "Therefore, the request is ambiguous. The clarifying question is". Note that if the user does not specify details regarding accommodation, cuisine, or transportation in the initial query, it indicates that the user does not have specific preferences that need clarification in the following conversation.

Snippet 3: Direct Clarification Prompts

System Message

Your current task is to determine the user's intentions and satisfy their needs based on the provided conversation between the user and the assistant, along with the interaction trajectory involving tool use between the agent and the environment. The interaction trajectory includes the following tools:

[*tool description*]

If certain tools are not called in the interaction trajectory, it indicates a lack of the required parameters needed to call those tools. For each category, including accommodations, dining, attractions, transportation, and budget, at least one relevant tool should be used during the interaction to gather sufficient information to help the user provide a clear and feasible request. Note that if the user does not specify details regarding accommodation, cuisine, or transportation in the initial query, it indicates that the user does not have specific preferences that need clarification in the following conversation.

Clarification Need Prediction

Conversation: [...]
Interaction trajectory: [...]
Please determine whether the user's request needs clarification. A request needs clarification if the user's intention contains missing or unfeasible details based on the tool parameters and call results in the interaction trajectory. If the user's intention requires clarification, answer "Yes"; if it is clear and feasible, answer "No".
Answer:

Clarification Question Generation

Please ask the user one clarification question to gather more information about a specific detail. Do not attempt to solve the task.
Question:

Snippet 4: Tool Description in Execution

1. AccommodationSearch(city, filters)
– Description: Discover accommodations in your desired city with specific filters.
– Parameters:
  – city (str, required): The name of the city where you're seeking accommodation.
  – filters (list[str], required): A list of filters to refine your search. Choices include "shared room", "not shared room ", "private room", "entire room", "parties", "smoking", "children under 10", "pets", "visitors". If the user does not specify any accommodation filters, assign an empty list "[]" to the parameter.
– Example: AccommodationSearch("Berlin", ["private room", "parties"]) would return private rooms in Berlin that allow parties.

2. RestaurantSearch(city, cuisines)
– Description: List all restaurants in your chosen city, regardless of cuisine type, and check if any of the cuisines you specify are unavailable there.
– Parameters:
  – city (str, required): The name of the city where you're seeking restaurants.
  – cuisines (list[str], required): A list of desired cuisines to check for availability. Available options include " Chinese", "American", "Italian", "Mexican", "Indian", "Mediterranean", and "French". If the user does not specify any cuisines, assign an empty list "[]" to the parameter.
– Example: RestaurantSearch("Dublin", ["Chinese", "Italian", "French"]) returns all restaurants in Dublin and informs you if any of the Chinese, Italian, or French cuisines are unavailable.

3. AttractionSearch(city)
– Description: Find attractions in a city of your choice.
– Parameters:
  – city (str, required): The name of the city where you're seeking attractions.
– Example: AttractionSearch("London") would return attractions in London.

4. DistanceMatrix(origin, destination, mode)
– Description: Estimate the distance, time, and cost between two cities.
– Parameters:
  – origin (str, required): The departure city of your journey.
  – destination (str, required): The destination city of your journey.
  – mode (str, required): The method of transportation. Choices include "self–driving" and "taxi".
– Example: DistanceMatrix("Paris", "Lyon", "self–driving") would provide driving distance, time, and cost between Paris and Lyon.

5. FlightSearch(origin, destination, date):
– Description: A flight information retrieval tool.
– Parameters:
  – origin (str, required): The city you'll be flying out from.
  – destination (str, required): The city you aim to reach.
  – date (str, required): The date of your travel in "YYYY–MM–DD" format.
– Example: FlightSearch("New York", "London", "2022–10–01") would fetch flights from New York to London on October 1, 2022.

6. BudgetEstimator(origin, destination, dates, people_number)
– Description: Calculate the minimal estimated budget for the trip. Use this tool to verify if the budget provided by the user is sufficient.
– Parameters:
  – origin (str, required): The departure city of your trip.
  – destination (list[str], required): A list of cities you plan to visit during your trip.
  – dates (list[str], required): A list of dates corresponding to the departure from the origin and each of the destinations. The first date is the departure from the origin, and each subsequent date corresponds to the departure from the respective city in the destination list. The last date in this list is the departure from the final destination back to the origin city. All dates should be formatted as "YYYY–MM–DD".
  – people_number (int, required): The number of people on the trip.
– Example: BudgetEstimator("London", ["Paris", "Rome", "Madrid"], ["2022–09–01", "2022–09–05", "2022–09–10", "2022–09–15"], 2) would return the minimal estimated budget for a trip from London to Paris, from Paris to Rome, from Rome to Madrid, and from Madrid back to London on September 1, 5, 10, and 15, 2022, respectively, for two people.

SNIPPET 5: STATIC EXECUTION PROMPTS

System Message

Based on the provided conversation between the user and the assistant, generate function calls to collect valid information related to accommodations, dining, attractions, transportation, and budget. The available functions are detailed below:

[*tool description*]

Please ensure that nested function use is avoided, escape symbols are not included in the string, and functions are only called when all required parameters are available. Your response should include all available function calls, specifying both the function name and its parameters, with each function on a separate line.

Example

Conversation: [{'user': 'Could you create a 3–day travel plan for 7 people from Ithaca to Portland on day 1, from March 8th, 2022?'}, {'assistant': 'Sorry, I couldn\'t find a way to arrive in Portland. Could you provide an alternative destination?'}, {'user': 'Charlotte.'}]

Response:
AccommodationSearch("Charlotte", [])
RestaurantSearch("Charlotte", [])
AttractionSearch("Charlotte")
DistanceMatrix("Ithaca", "Charlotte", "taxi")
DistanceMatrix("Ithaca", "Charlotte", "self–driving")
FlightSearch("Ithaca", "Charlotte", "2022–03–08")
DistanceMatrix("Charlotte", "Ithaca", "taxi")
DistanceMatrix("Charlotte", "Ithaca", "self–driving")
FlightSearch("Charlotte", "Ithaca", "2022–03–10")
BudgetEstimator("Ithaca", ["Charlotte"], ["2022–03–08", "2022–03–10"], 7)

User Message

Conversation: [...]
Response:

SNIPPET 6: ERROR PROMPTS IN DYNAMIC EXECUTION

Invalid Action

Invalid action: [...]. Valid actions include AccommodationSearch(city, filters), RestaurantSearch(city, cuisines), AttractionSearch(city), DistanceMatrix(origin, destination, mode), FlightSearch(origin, destination, date), BudgetEstimator(origin, destination, dates, people_number), and Finish(). Do not include any comment or description in the action.

Invalid Parameters

Invalid parameters for [*action*]. Please ensure that all parameters are provided in the correct format.

Null Action

Your action has been filtered due to content restrictions. Please ensure your action does not begin with ['\n', ' Thought', 'Action', 'Observation']. Ensure that the action is permitted in this environment, and try again.

Snippet 7: Dynamic Execution Prompts

System Message

Based on the provided conversation between the user and the assistant, collect valid information related to accommodations, dining, attractions, transportation, and budget. Solve this task by alternating between "Thought", " Action", and "Observation" steps. "Thought" can reason about the current situation, and "Action" can have 7 different types:

[*tool description*]

7. Finish()
– Description: Use this function to indicate the task's completion once all the necessary information has been collected.
– Example: Call Finish() after gathering all necessary information related to accommodations, dining, attractions, transportation, and budget.

Please ensure that nested function use is avoided, escape symbols are not included in the string, and functions are only called when all required parameters are available. Each action should call a single function once, using the valid function name and all required parameters. You should take as many steps as possible until you have gathered the necessary information to complete the task using Finish(). If the user's request is vague or infeasible, avoid making assumptions and strictly use the provided information. Do not add any description or comment to the action. Additionally, do not include line breaks in your response.

User Message for ReAct

Conversation: [...]
[...]

User Message for Reflexion

In previous attempts, you tried to use tools to interact with the environment to gather valid information on accommodations, dining, attractions, transportation, and budget given the user–assistant conversation but were unsuccessful. The reflections below offer suggestions to help you avoid past mistakes. Use these insights to refine your strategy for effectively and efficiently utilizing tools to collect the necessary information.
Reflections:
[...]
Conversation: [...]
[...]

Self-reflection Generation

You are an advanced reasoning agent capable of self–improvement through reflection. You will review a previous attempt where you failed to effectively utilize a tool to gather valid information about accommodations, dining, attractions, transportation, and budget given the user–assistant conversation. Analyze the reasons for the mistake, referencing the tool documentation, the observation, and the action you have taken. Then, formulate a concise, high–level explanation to address and prevent similar errors in the future. Keep your reflections in complete sentences without any line breaks.

Tool documentation:
[...]
Ensure that each action uses only one non–nested tool and contains no comments or descriptions.

Observation: [...]
Action: [...]

Reflection:

Snippet 8: Static Planning Prompts

System Message

You are a proficient planner tasked with generating a detailed travel plan in JSON format, which is an array of objects, based on the interaction trajectory and the user–assistant conversation. Your plan must strictly adhere to the format provided in the example, incorporating specific details such as flight numbers (e.g., "F0123456"), restaurant names, and accommodation names. Ensure all information in your plan is derived solely from the provided data and aligns with common sense. Attraction visits and meals are expected to be diverse. Use the symbol "−" for unnecessary details, such as "eat at home" or "on the road". For instance, you do not need to plan after returning to the departure city. When traveling to two cities in one day, ensure that "current_city" aligns with the format "from A to B" in the example. If transportation details indicate a journey from one city to another (e.g., from A to B), update the "current_city" to the destination city (in this case, B) the following day. Use ";" to separate different attractions, formatting each as "Name, City". Make sure all flight numbers and costs are appended with a colon (e.g., "Flight Number:" and "Cost:"), consistent with the example. Your JSON plan should include the following fields: ["day", " current_city", "transportation", "breakfast", "attraction", "lunch", "dinner", "accommodation"]. Escape symbols should be used in the string when necessary. Additionally, remove any "$" symbols and comments from the plan.

***** Example *****
Conversation: [{'user': 'Could you create a 3–day travel plan for 7 people from Ithaca to Portland on day 1, from March 8th, 2022?'}, {'assistant': 'Sorry, I couldn't find a way to arrive in Portland. Could you provide an alternative destination?'}, {'user': 'Charlotte.'}, {'assistant': 'It seems you haven't mentioned the expected budget for this trip. Could you provide that information?'}, {'user': 'Yes, my expected budget is $30,200.'}]
Travel Plan: [{"day": 1, "current_city": "from Ithaca to Charlotte", "transportation": "Flight Number: F3633405, from Ithaca to Charlotte, Departure Time: 05:38, Arrival Time: 08:10", "breakfast": "Nagaland's Kitchen, Charlotte ", "attraction": "The Charlotte Museum of History, Charlotte;", "lunch": "Cafe Maple Street, Charlotte", "dinner": " Bombay Vada Pav, Charlotte", "accommodation": "Affordable Spacious Refurbished Room in Bushwick!, Charlotte "}, {"day": 2, "current_city": "Charlotte", "transportation": "–", "breakfast": "Olive Tree Cafe, Charlotte", "attraction ": "The Mint Museum, Charlotte;Romare Bearden Park, Charlotte;", "lunch": "Birbal Ji Dhaba, Charlotte", "dinner": "Pind Balluchi, Charlotte", "accommodation": "Affordable Spacious Refurbished Room in Bushwick!, Charlotte"}, {"day": 3, "current_city": "from Charlotte to Ithaca", "transportation": "Flight Number: F3786160, from Charlotte to Ithaca, Departure Time: 20:48, Arrival Time: 22:34", "breakfast": "Subway, Charlotte", "attraction": "Books Monument, Charlotte;", "lunch": "Olive Tree Cafe, Charlotte", "dinner": "Kylin Skybar, Charlotte", " accommodation": "–"}]
***** Example Ends *****

User Message for Direct

Interaction trajectory: [...]
Conversation: [...]
Travel Plan:

User Message for Zero-shot CoT

Interaction trajectory: [...]
Conversation: [...]
Travel Plan: Let's think step by step. First,

Snippet 9: Error Prompts in Dynamic Planning

Invalid Plan

The subplan cannot be parsed into JSON format; please check. Only a one–day plan is supported.

Error Plan

The subplan cannot be parsed into JSON format due to the syntax error; please check.

Invalid Action

Invalid action. Valid actions include CostEnquiry(subplan) and Finish(final_json_plan). Please ensure that the parameter is provided in the correct format. Do not include any comments, descriptions, or line breaks in your response.

Snippet 10: Dynamic Planning Prompts

System Message

You are a proficient planner tasked with generating a detailed travel plan in JSON format, which is an array of objects, based on the interaction trajectory and the user−assistant conversation. Your plan must strictly adhere to the format provided in the example, incorporating specific details such as flight numbers (e.g., "F0123456"), restaurant names, and accommodation names. Ensure all information in your plan is derived solely from the provided data and aligns with common sense. Attraction visits and meals are expected to be diverse. Use the symbol "−" for unnecessary details, such as "eat at home" or "on the road". For instance, you do not need to plan after returning to the departure city. When traveling to two cities in one day, ensure that "current_city" aligns with the format "from A to B" in the example. If transportation details indicate a journey from one city to another (e.g., from A to B), update the "current_city" to the destination city (in this case, B) the following day. Use ";" to separate different attractions, formatting each as "Name, City". Make sure all flight numbers and costs are appended with a colon (e.g., "Flight Number:" and "Cost:"), consistent with the example. Your JSON plan should include the following fields: ["day", "current_city", "transportation", "breakfast", "attraction", "lunch", "dinner", "accommodation"]. Escape symbols should be used in the string when necessary. Additionally, remove any "$" symbols and comments from the plan.
Solve this task by alternating between "Thought", "Action", and "Observation" steps. The "Thought" phase involves reasoning about the current situation. The "Action" phase can be of two types:
1. CostEnquiry(subplan): This function calculates the cost of a detailed subplan, for which you need to input the number of people and plan in JSON format. The subplan should encompass a complete one−day plan and include the following fields: ["people_number", "day", "current_city", "transportation", "breakfast", "attraction", "lunch", "dinner", "accommodation"]. An example will be provided for reference.
2. Finish(final_json_plan): Use this function to indicate the completion of the task. You must submit a final, complete plan in JSON as the argument.

***** Example *****
Conversation: [{'user': 'Could you create a 3−day travel plan for 7 people from Ithaca to Portland on day 1, from March 8th, 2022?'}, {'assistant': 'Sorry, I couldn't find a way to arrive in Portland. Could you provide an alternative destination?'}, {'user': 'Charlotte.'}, {'assistant': 'It seems you haven't mentioned the expected budget for this trip. Could you provide that information?'}, {'user': 'Yes, my expected budget is $30,200.'}]
You can call CostEnquiry like CostEnquiry({"people_number": 7, "day": 1, "current_city": "from Ithaca to Charlotte", "transportation": "Flight Number: F3633405, from Ithaca to Charlotte, Departure Time: 05:38, Arrival Time: 08:10", "breakfast": "Nagaland's Kitchen, Charlotte", "attraction": "The Charlotte Museum of History, Charlotte", "lunch": "Cafe Maple Street, Charlotte", "dinner": "Bombay Vada Pav, Charlotte", "accommodation": "Affordable Spacious Refurbished Room in Bushwick!, Charlotte"})
You can call Finish like Finish([{"day": 1, "current_city": "from Ithaca to Charlotte", "transportation": "Flight Number: F3633405, from Ithaca to Charlotte, Departure Time: 05:38, Arrival Time: 08:10", "breakfast": "Nagaland's Kitchen, Charlotte", "attraction": "The Charlotte Museum of History, Charlotte;", "lunch": "Cafe Maple Street, Charlotte", "dinner": "Bombay Vada Pav, Charlotte", "accommodation": "Affordable Spacious Refurbished Room in Bushwick!, Charlotte"}, {"day": 2, "current_city": "Charlotte", "transportation": "−", "breakfast": "Olive Tree Cafe, Charlotte", "attraction": "The Mint Museum, Charlotte;Romare Bearden Park, Charlotte;", "lunch": "Birbal Ji Dhaba, Charlotte", "dinner": "Pind Balluchi, Charlotte", "accommodation": "Affordable Spacious Refurbished Room in Bushwick!, Charlotte"}, {"day": 3, "current_city": "from Charlotte to Ithaca", "transportation": "Flight Number: F3786160, from Charlotte to Ithaca, Departure Time: 20:48, Arrival Time: 22:34", "breakfast": "Subway, Charlotte", "attraction": "Books Monument, Charlotte;", "lunch": "Olive Tree Cafe, Charlotte", "dinner": "Kylin Skybar, Charlotte", "accommodation": "−"}])
***** Example Ends *****

You must use Finish(final_json_plan) to indicate that you have finished the task. Each action only calls one function once, without any comments or descriptions. Do not include line breaks in your response.

User Message for React

Interaction trajectory: [...]
Conversation: [...]
[...]

User Message for Reflexion

You have attempted to give a subplan before and failed. The following reflection(s) give a suggestion to avoid failing to answer the query in the same way you did previously. Use them to improve your strategy for correctly planning.
Reflections:
[...]
Interaction trajectory: [...]
Conversation: [...]
[...]