CtxWF: Context Window Focus with Global Management for LLM Agents in Multi-Document Workspace

Anonymous ACL submission

Abstract

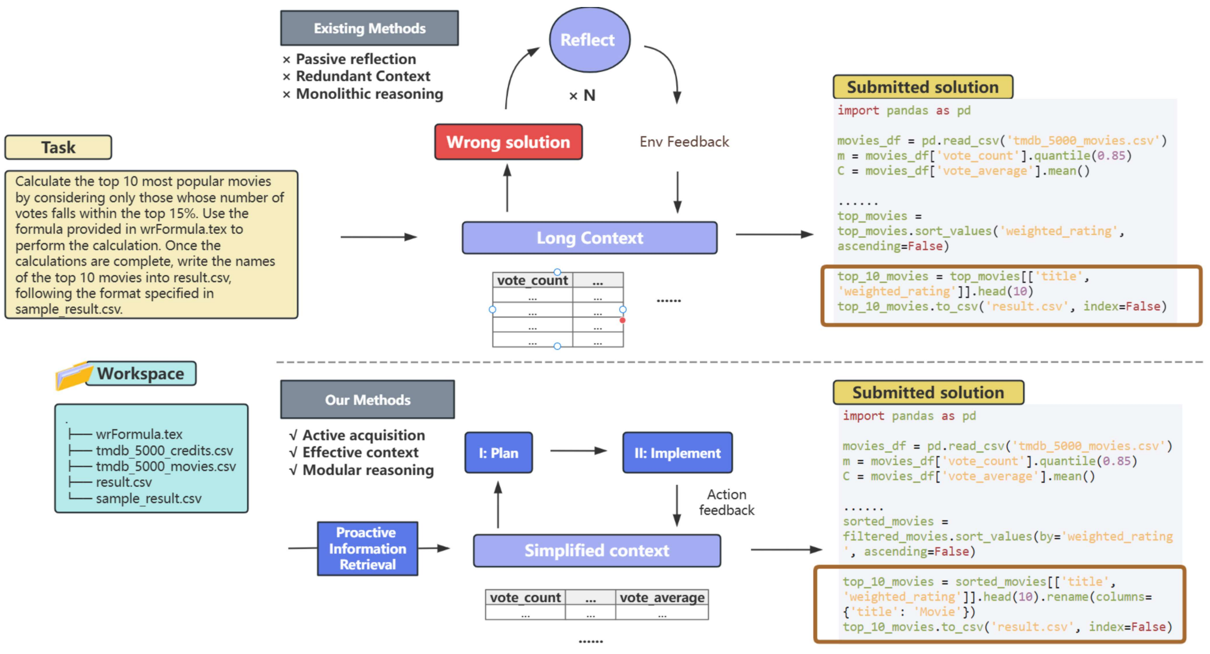
LLM-based agent systems have achieved remarkable progress in automatically solving natural language processing tasks. However, real-world tasks often involve working within a multi-file workspace that requires exploratory implementation of specific objectives, demanding LLMs to acquire, process, and manage substantial information from workspace data sources. Due to the limited attention span of LLMs, excessive or disorganized information can lead to distraction from core objectives during reasoning, ultimately resulting in suboptimal outputs. To enhance LLMs' capability in handling complex real-world tasks, inspired by human problem-solving strategies, we propose CtxWF, a context window-focused agent that resolves long-term complex tasks through global context management and concentrated execution of short-term sub-tasks. CtxWF features three key innovations: (1) Proactive acquisition of essential contextual information prior to task resolution, (2) Single-responsibility specialization of LLM reasoning to reduce context window requirements, (3) Refinement of environmental feedback for context updates to enhance information quality post short-term task execution. We showcase the effectiveness of CtxWF on agent-based data science tasks, where it achieves state-of-the-art accuracy across multiple models. The GPT-4o-powered CtxWF attains an accuracy of 42.26%, representing a 10.01% improvement over baseline methods.

1 Introduction

Remarkable progress has been observed in recent Large Language Models (LLMs) for various natural language processing tasks, while LLM-based agent systems further extend these capabilities. However, compared to simply transforming instructions into executable code (Yu et al., 2018; Lin et al., 2018; Chen et al., 2021; Huang et al., 2024a; Lu et al., 2022), research on leveraging LLMs for complex real-world programming tasks remains insufficient. Taking real-world data analysis tasks as an example, the contextual information required for tasks is not always pre-organized as natural language instructions, but rather needs to be actively explored by LLMs within the workspace. For instance, in the task illustrated in Figure 1, LLMs need to proactively retrieve the algorithmic formula stored in the `wrFormula.tex` file, compute the top 10 most popular movies from the `tmdb_5000_movies.csv` file, and store the results in result.csv following the format of `sample_result.csv`.

When tackling complex real-world data analysis tasks, humans actively utilize existing tools (e.g., Notepad, Excel) to interact with their environment (external world). Through this interaction, they acquire task-relevant information into working memory (Baddeley, 1992) for cognitive reasoning (Alderson-Day and Fernyhough, 2015). The brain filters task-related information, designs task plans based on this filtered data, focuses attention on specific implementations, and self-regulates (Zavershneva and van der Veer, 2018; Luria, 1965; Dick and Overton, 2009) through environmental feedback. For example, given the task description and workspace files in Figure 1, humans first identify useful files, extract necessary information (e.g., the algorithm formula from `wrFormula.tex` and table schemas of CSV files) into working memory, then derive code solutions, and iteratively refine them by observing execution feedback (e.g., console outputs).

Recent studies have explored methods utilizing LLMs for interactive environment planning and action. In these approaches, environmental outcomes are fed back to the LLMs in text form, enabling LLMs to generate domain-specific actions or plans, which are then executed by a controller (Liu et al., 2024b; Ahn et al., 2022; Nakano



Task

Calculate the top 10 most popular movies by considering only those whose number of votes falls within the top 15%. Use the formula provided in wrFormula.tex to perform the calculation. Once the calculations are complete, write the names of the top 10 movies into result.csv, following the format specified in sample_result.csv.

```
.
├── wrFormula.tex
├── tmdb_5000_credits.csv
├── tmdb_5000_movies.csv
├── result.csv
└── sample_result.csv
```

```python
import pandas as pd

movies_df = pd.read_csv('tmdb_5000_movies.csv')
m = movies_df['vote_count'].quantile(0.85)
C = movies_df['vote_average'].mean()

......
top_movies =
top_movies.sort_values('weighted_rating',
ascending=False)

top_10_movies = top_movies[['title',
'weighted_rating']].head(10)
top_10_movies.to_csv('result.csv', index=False)
```

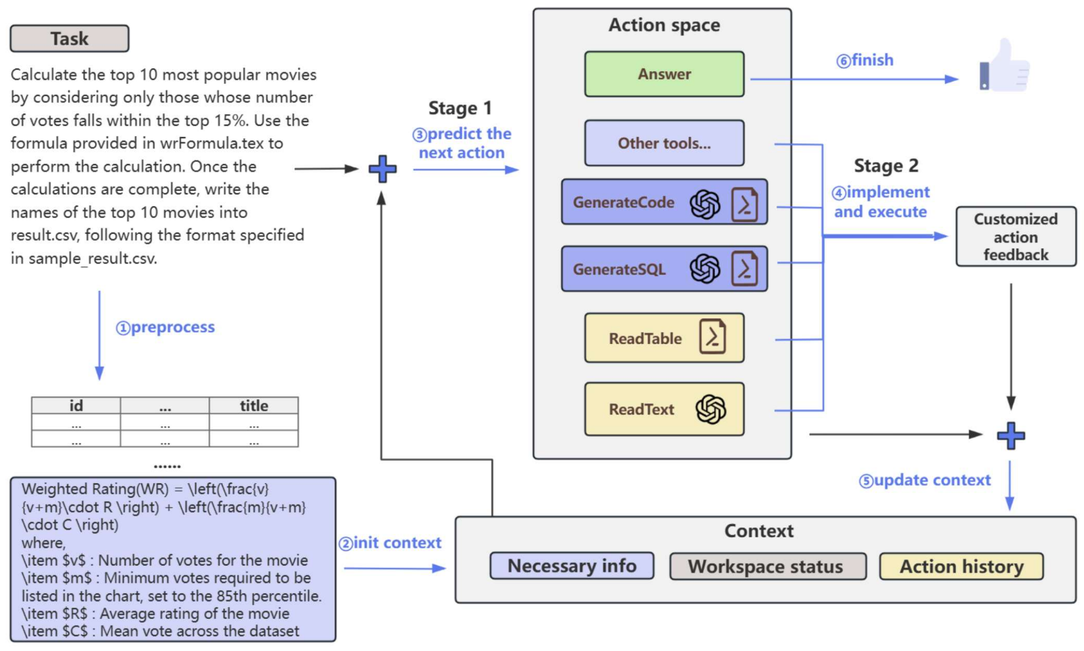
```python
import pandas as pd

movies_df = pd.read_csv('tmdb_5000_movies.csv')
m = movies_df['vote_count'].quantile(0.85)
C = movies_df['vote_average'].mean()

......
sorted_movies =
filtered_movies.sort_values(by='weighted_rating
', ascending=False)

top_10_movies = sorted_movies[['title',
'weighted_rating']].head(10).rename(columns=
{'title': 'Movie'})
top_10_movies.to_csv('result.csv', index=False)
```

Figure 1: Comparison between existing methods and our approach.

et al., 2021; Yao et al., 2020; Huang et al., 2022). Alternative approaches treat code generation as the primary interaction mechanism between agents and environments (Qiao et al., 2023; Wang et al., 2024b), avoiding domain-specific action design while enhancing problem-solving versatility. However, these agents over-rely on LLMs, **burdening them with multiple responsibilities during reasoning** (e.g., information supplementation, task planning, and implementation) and **flooding agent contexts with lengthy, disorganized multi-turn dialogue histories**. As shown in Figure 1's existing methods: (1) Initial solutions often lack CSV schema information, leading to hallucinated code (e.g., falsely assuming a "VoteCount" column in `tmdb_5000_movies.csv`); (2) When LLMs perform reflective reasoning in a single iteration while simultaneously handling multiple responsibilities, their reasoning contexts become overly lengthy and disorganized, which often leads to erroneous outcomes; (3) Multi-round reflection, while eventually prompting schema checks, suffers from inefficiency, high error rates, and redundant context accumulation. Moreover, some hallucinated code executions may not generate explicit errors, resulting in executable but incorrect solutions.

To address these challenges, we propose CtxWF, inspired by human attentional focus during problem-solving, which employs three key strategies: (1) Proactively acquiring task-critical information as contextual background before LLM reasoning; (2) Decomposing complex long-term task planning into multiple short-term phases, separating planning from implementation to simplify reasoning responsibilities; (3) Distilling feedback from successful short-term task executions to filter noise and guide LLMs toward correct paths upon errors. These mechanisms enable CtxWF to dynamically plan and execute phased tasks while maintaining clean, focused reasoning contexts with sufficient background information. Our approach uses off-the-shelf LLMs as reasoning engines without model training or fine-tuning, operating fully autonomously without human intervention.

We demonstrate CtxWF's effectiveness on DA-Code (Huang et al., 2024b), a real-world complex data analysis benchmark reflecting practical scenarios. Compared to the baseline DA-Agent (Huang et al., 2024b), CtxWF achieves superior performance across multiple LLMs and task difficulty levels. The GPT-4o-powered CtxWF reaches an accuracy of 42.26%, outperforming DA-Agent by 10.01%. The results demonstrate that CtxWF fully unleashes the potential of LLMs in solving complex data analysis tasks.



Task
Calculate the top 10 most popular movies by considering only those whose number of votes falls within the top 15%. Use the formula provided in wrFormula.tex to perform the calculation. Once the calculations are complete, write the names of the top 10 movies into result.csv, following the format specified in sample_result.csv.
①preprocess
id
...
title
Weighted Rating(WR) = \left(\frac{v}{v+m}\cdot R \right) + \left(\frac{m}{v+m} \cdot C \right) where, \item v : Number of votes for the movie \item m : Minimum votes required to be listed in the chart, set to the 85th percentile. \item R : Average rating of the movie \item C : Mean vote across the dataset
②init context
Stage 1
③predict the next action
Action space
Answer
Other tools...
GenerateCode
GenerateSQL
ReadTable
ReadText
⑥finish
Stage 2
④implement and execute
Customized action feedback
⑤update context
Context
Necessary info
Workspace status
Action history

Figure 2: The Operational Workflow of CtxWF.

2 Method

CtxWF focuses on three core design components to manage reasoning contexts: (1) The agent actively acquires critical information required for current task objectives as background context during reasoning, (2) When utilizing LLMs for single-step reasoning, the agent concentrates on well-defined responsibilities to reduce context length during inference, and (3) The agent refines feedback information obtained from environmental interactions to ensure beneficial updates rather than noise injection into subsequent reasoning contexts. Specifically, as shown in Figure 2, CtxWF initially obtains preliminary reasoning contexts during the preprocessing phase. Subsequently, it employs LLMs for interactive environment planning and action execution: first abstractly deducing the objective of the next step and matching it with corresponding actions from the action space, then performing secondary planning for phase-specific implementation based on action objectives. The framework continuously refines environmental feedback generated by action execution to update agent contexts, thereby dynamically creating, maintaining, and adjusting high-level action plans (Yao et al., 2023). The full prompt template is provided in Appendix A.

2.1 Proactive Information Acquisition

In real-world data analysis tasks, agents must contend with diverse data sources within their workspace when solving problems. To address this challenge, we have designed a mechanism that enables agents to proactively acquire essential information before generating code-based (or SQL-based) solutions through LLMs. As illustrated in Figure 2, prior to action prediction, CtxWF leverages the planning capabilities of LLMs (Wei et al., 2022) and builds on an LLM as a natural language planner during preprocessing. Prompted by action descriptions, the planner generates action chains in order to retrieve task-relevant information to update the agent context (Lu et al., 2023). Specifically for SQL generation, CtxWF first acquires database schema information pertinent to the current task as reasoning context before initiating SQL generation. See Appendix A.1 for the full prompt structure.

2.2 Two-Stage Planning Framework

The objective of two-stage planning is to decompose a complex task into correct action sequences, resulting in a task solution. To empower agents to adapt to diverse complex tasks while maintaining specialized reasoning responsibilities for LLMs in each step, our system has customized a set of generic Actions. We retain executable code generation as the core action, as it allows integrating various LLM behaviors into a unified action space (Wang et al., 2024b), thereby liberating agent capabilities from the constraints of manually designed predefined tool sets. Additionally, in data

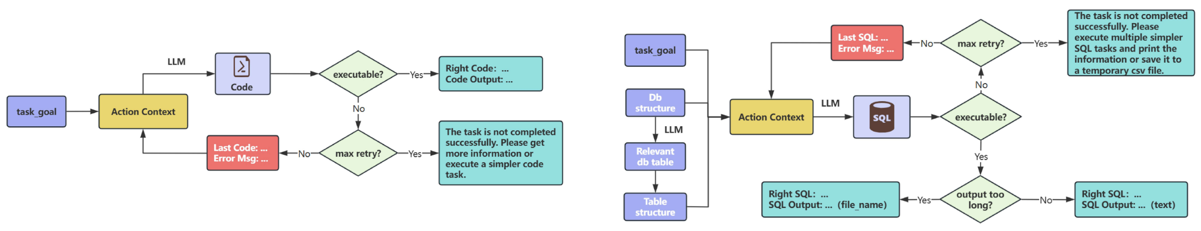


Figure 3: The Operational Workflow of the GenerateCode action (left) and the GenerateSQL action (right).

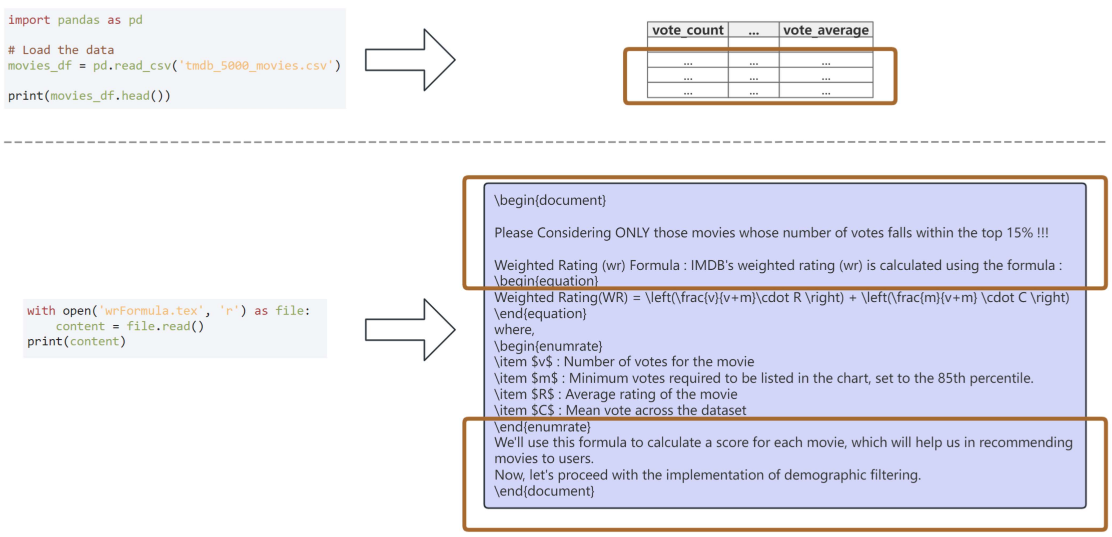
analysis tasks, operations such as text content retrieval, tabular file structure acquisition, and SQL query execution are frequently required. However, LLMs often indiscriminately fetch entire file contents when accessing such information. This necessitates our specialized design for these actions, rather than relying entirely on executable code. We provide high-level specifications of these actions below, with their concrete implementations detailed in our experiments.

- **ReadTable(file_path):** This action retrieves structural metadata (e.g., column names, data types) and a data preview from structured files.
- **ReadText(file_path, task_goal):** Designed to search target content in text files using LLMs' natural language processing capabilities.
- **GenerateCode(task_goal):** This action generates code through LLMs and executes it within a Docker sandbox to accomplish specified parameterized objectives. The operational workflow is illustrated in Figure 3 (left).
- **GenerateSQL(file_path, task_goal):** This action first retrieves all table names from the target database, obtains relevant table schemas, then generates and executes SQL queries to achieve specified tasks. The operational workflow is illustrated in Figure 3 (right).
- **Decompress(file_path):** This action automatically selects appropriate decompression methods based on file extensions to handle compressed files.
- **Answer(output):** This action submits final task results, which may include filenames, text information, or failure notifications.

As illustrated in Figure 2, the primary objective of Stage 1 is to enable LLMs to perform high-level task planning at the holistic level. This stage continuously generates subsequent action purposes based on the agent's contextual state and final task objective, while predicting the next executable action from the action space to achieve the designated purpose. Conversely, Stage 2 focuses on enabling LLMs to conduct detailed task planning for implementing concrete operations required by the current action. The separation of abstract high-level planning (Stage 1) from concrete implementation planning (Stage 2) stems from their distinct responsibilities. It is well-established that LLMs have limited attention capacity. Ambiguous responsibilities or excessively lengthy contextual information may induce hallucinations and related issues. Our hierarchical approach allows LLMs to concentrate all computational attention on singular responsibilities, simultaneously reducing the required contextual information for reasoning. Furthermore, detailed task implementation often proves challenging to resolve through single-pass LLM reasoning. This decoupled architecture facilitates human-controlled refinement of specific action implementations (e.g., code generation, SQL command formulation) during the reasoning process.

The detailed prompt for the first stage is provided in Appendix A.2.1. Briefly, the prompt comprises four key components:

- **Action Space:** A predefined set of actions controlling LLMs' behavior, primarily categorized into information retrieval, code generation, and auxiliary actions.
- **Files Info:** Real-time file information in the current workspace represented as a directory tree structure.



```python
import pandas as pd

# Load the data
movies_df = pd.read_csv('tmdb_5000_movies.csv')

print(movies_df.head())
```

vote_count	...	vote_average
...	...	...
...	...	...
...	...	...

```python
with open('wrFormula.tex', 'r') as file:
    content = file.read()
print(content)
```

```
\begin{document}

Please Considering ONLY those movies whose number of votes falls within the top 15% !!!

Weighted Rating (wr) Formula : IMDB's weighted rating (wr) is calculated using the formula :
\begin{equation}
Weighted Rating(WR) = \left(\frac{v}{v+m}\cdot R \right) + \left(\frac{m}{v+m} \cdot C \right)
\end{equation}
where,
\begin{enumrate}
\item $v$ : Number of votes for the movie
\item $m$ : Minimum votes required to be listed in the chart, set to the 85th percentile.
\item $R$ : Average rating of the movie
\item $C$ : Mean vote across the dataset
\end{enumrate}
We'll use this formula to calculate a score for each movie, which will help us in recommending
movies to users.
Now, let's proceed with the implementation of demographic filtering.
\end{document}
```

Figure 4: LLMs employ programmatic code execution to retrieve tabular data schemas or textual content.

- **Task:** Textual description of the current objective.

- **Action History:** Record of the agent's real-time progress and achieved outcomes.

2.3 Refinement of Environmental Feedback Information

In traditional approaches, agents directly store each interaction record with LLMs in their context. While practitioners recognize that richer information can enhance LLMs' global comprehension, the indiscriminate inclusion of excessive information into the reasoning context can introduce substantial noise. Compared to allowing LLMs to freely access vast amounts of irrelevant information, CtxWF refines contextual information fed back to the agent when implementing the actions. This approach effectively reduces the context window size required for LLMs' reasoning processes, enabling LLMs to comprehend contextual background information more effortlessly during inference. Additionally, we incorporate scenario-specific guidance to redirect LLMs back to valid trajectories when erroneous behaviors occur, avoiding unproductive error correction cycles. For each action implementation, we meticulously monitor its output to retain task-beneficial information while filtering irrelevant content. Implementation details of these actions are provided in Appendix A.3.

The customization objectives for key actions are outlined below:

ReadTable: This action filters out raw data in structured files, retaining only structural metadata. As illustrated in the upper section of Figure 4, while LLMs attempt to retrieve table information through code execution, only the structural metadata in the output proves operationally meaningful. The substantial data entries highlighted in red, when incorporated into the LLM's reasoning context, not only fail to facilitate downstream data processing but actively introduces noise into the reasoning process. The automated filtration mechanism embedded in this action systematically eliminates such extraneous data elements.

ReadText: This action selectively outputs task-relevant content from specified files, eliminating extraneous text. This mechanism is critical when processing multiple lengthy text files, as excessive file content occupying the LLM's context often leads to attention dispersion, task failure, and unnecessarily consuming tokens. In the lower section of Figure 4, while LLMs attempt to extract calculation formulas through code, the irrelevant text segments highlighted in red provide no operational value. This action systematically filters such non-essential content through targeted pattern matching.

GenerateCode: In Figure 3 (left), when code execution errors occur, this action loads only the previous code snippet and its corresponding error into a temporary action context (distinct from the agent's planning context). The action then asks LLMs to debug and regenerate the code, focusing each reasoning cycle on resolving a single error. Since excessive debug cycles typically indicate overcomplicated initial code proposals, making it difficult for LLMs to pinpoint error messages, the agent prompts LLMs to re-plan task objectives upon reaching a predefined threshold of debugging cycles.

GenerateSQL: This action follows the same feedback design principles as GenerateCode (Figure 3, right).

3 Experiments

3.1 Dataset and Baseline

DA-Code is a code generation benchmark specifically designed for evaluating LLM-based agents in data science tasks. Distinct from conventional code generation benchmarks, this benchmark is designed to enable agents to explore data and leverage programming capabilities to solve challenging objectives, rather than simply translating explicit natural language instructions into code. Unlike existing benchmarks like DS-1000 (Lai et al., 2023) and HumanEval (Chen et al., 2021), which primarily focus on directly converting natural language instructions into executable code, DA-Code establishes a more realistic scenario that simulates real-world data science tasks under given requirements and workspace constraints. DA-Code tasks not only feature inherently complex solutions but also incorporate diverse data sources (databases, spreadsheets, documents, codebases, etc.) containing multifaceted information and data from authentic programming scenarios. Moreover, these information sources may be saturated with noise and extraneous information. We constructed a subset DA-Code-100 containing 100 randomly sampled tasks for evaluation, with difficulty levels distributed as 23 easy, 60 medium, and 17 challenging tasks.

To address the challenges posed by the DA-Code benchmark where no existing agent framework has demonstrated sufficient capability, the authors of DA-Code developed DA-Agent, an LLM-based agent framework specialized for complex data analysis through dynamic environment interactions. DA-Agent demonstrates superior performance compared to prevailing agent frameworks including OpenHand (Wang et al., 2024c), AutoGen (Wu et al., 2024), and X-Agent (Team, 2023) in comprehensive evaluations. Notably, while DA-Agent replaces Python code with Bash commands for environmental information retrieval, LLMs still acquire substantial noise through Bash operations. For instance, using `head` command to read CSV files could capture redundant lines, and `cat` command outputs the entire file contents indiscriminately.

3.2 Overall Performance

3.2.1 Experimental Setup

We employed two state-of-the-art open-source models, Qwen2.5-72B-Instruct (Yang et al., 2024) and DeepSeek-V3 (Liu et al., 2024a), as open-source representatives, along with the closed-source model GPT-4o-2024-08-06 (Achiam et al., 2023) as base testing models. All models were configured with a temperature of 0, a maximum of 20 action steps, and a 60-second timeout per action execution. We evaluated each LLM-based agent on DA-Code-100 through three rounds of testing. For each task, we calculated both the average score (aggregating performance across three trials) and the peak score (highest achievement in any trial). The final evaluation metrics Avg@3 and Max@3 were subsequently derived by computing the mean values of these task-specific scores across all benchmark tasks.

3.2.2 Experimental Result

As shown in Table 1, we compared the performance of CtxWF against baseline methods across various base LLMs. The results demonstrate that CtxWF achieves superior evaluation metrics across almost all models and difficulty levels, except for a minor 2.04% decrease in maximum score at medium difficulty on Qwen2-72B-Instruct. This indicates our method's enhanced capability to leverage LLMs' reasoning potential in most scenarios. Compared with existing methods (Figure 1) that rely on multiple reflection iterations to supplement LLMs' reasoning context, our proactive information acquisition mechanism obtains most of the task-relevant contextual information before LLMs initiate their reasoning process. This approach directly reduces the likelihood of LLM hallucinations at their source, while also preventing context overload issues caused by storing extensive intermediate reflection steps as historical records in the agent's reasoning context. Furthermore, while reflection-based methods prove effective for detectable er-

Model	Easy		Medium		Hard		Total	
	Avg@3	Max@3	Avg@3	Max@3	Avg@3	Max@3	Avg@3	Max@3
DA-Agent (Qwen2.5-72B)	40.77	44.62	22.94	35.70	12.12	21.69	25.21	35.37
DA-Agent (DeepSeek-V3)	44.50	50.27	25.62	29.12	14.87	18.69	28.13	32.21
DA-Agent (gpt-4o)	38.66	49.67	21.81	29.19	13.31	19.48	24.24	32.25
CtxWF (Qwen2.5-72B)	**49.11**	55.48	25.09	33.66	15.13	**25.84**	28.92	37.35
CtxWF (DeepSeek-V3)	47.38	**57.66**	**29.05**	39.50	**18.98**	22.69	**31.55**	40.82
CtxWF (gpt-4o)	45.03	54.25	27.95	**42.80**	18.31	24.15	30.24	**42.26**

Table 1: Performance comparison between CtxWF and baselines on selected LLMs. Avg@3 denotes the agent's mean accuracy rate across three testing trials. Max@3 reflects the peak accuracy rate observed during these trials.

rors (e.g., code execution failures), they struggle to supplement missing task-critical information in other scenarios. For instance, in Figure 1, while `sample_result.csv` specifies "Movie" as the required column header, reflection-based methods would submit results immediately after successful code execution. In contrast, CtxWF proactively acquires the structure of `sample_result.csv` and explicitly specifies column headers during file preservation.

Notably, CtxWF (GPT-4o) achieves a peak accuracy of 42.26%, representing a 10.01 percentage-point improvement over DA-Agent (GPT-4o), which substantiates that our methodology enables more effective exploitation of LLMs' latent capabilities. Furthermore, CtxWF (DeepSeek-V3) attains an average accuracy of 31.55%, outperforming its DA-Agent counterpart by 3.42 percentage points, which suggests more consistent performance in complex data analysis tasks.

3.3 Ablation Study

3.3.1 Experimental Setup

We investigate the performance degradation in average accuracy and peak accuracy when removing key functional designs of GPT-4o-based CtxWF for task processing (Table 2), using three rounds of testing on the DA-Code-100 dataset. The experimental configurations are designed as follows:

- **w/o Preprocess1:** The proactive information acquisition mechanism operates in two phases: Preprocess1 initializes contextual information before action prediction, while Preprocess2 retrieves table schema information before SQL generation. This experiment removes the first preprocessing phase.
- **w/o Preprocess1 & 2:** This configuration eliminates both preprocessing phases designed for active information acquisition.
- **w/o Refinement of Env Feedback:** This setup removes the environment feedback refinement module, simulating scenarios where LLMs freely access information (printing first five rows when reading tables or full text files). It also disables guidance mechanisms when code/SQL generation fails.
- **w/o Two-Stage Planning & Refinement of Env Feedback:** Since feedback refinement requires separation of abstract planning and detailed implementation, removing two-stage planning consequently disables the feedback refinement capability.

Configuration	ΔAvg@3	ΔMax@3
w/o Preprocess1	-1.44%	-5.02%
w/o Preprocess1 & 2	-3.16%	-5.35%
w/o Refinement of Env Feedback	-3.54%	-3.57%
w/o Two-Stage Planning & Refinement of Env Feedback	-3.66%	-7.4%

Table 2: Ablation Study

3.3.2 Experimental Result

The experimental results analysis demonstrates that each module in the proposed method contributes significantly to model performance. The comparative analysis of ablation study data yields supplementary insights:

- The complete removal of the proactive information acquisition functionality causes more severe accuracy degradation compared to solely removing the preprocessing 1, indicating that the proactive information acquisition mechanism benefits LLMs' reasoning capabilities across diverse scenarios.

- The simultaneous removal of both two-stage planning and environmental feedback optimization modules leads to the most substantial performance drop (-7.4% in max@3 metric), proving that the synergistic gain between the hierarchical planning mechanism and feedback refinement module effectively enhances the model's upper-bound capability in handling complex tasks. Moreover, the compounded accuracy deterioration from removing both components exceeds that of solely eliminating environmental feedback optimization, suggesting that the two-stage planning mechanism itself plays a pivotal standalone role.

4 Related Work

LLM-based Agent Systems: Agent systems constructed with LLMs have significantly enhanced the performance of LLMs in solving various complex tasks. Currently, there are four primary agent design patterns: (1) Reflection: Enabling agents to review and revise based on self-generated outputs or environmental feedback. SELF-REFINE (Madaan et al., 2023), ReACT (Yao et al., 2023), and Reflexion (Shinn et al., 2023) demonstrate that post-generation reflection effectively improves LLM performance, though they primarily focus on enhancing reasoning through iterative multi-step feedback. (2) Tool Invocation: Expanding LLM capabilities beyond pure NLP tasks by invoking external APIs. Gorilla (Patil et al., 2024) and ToolLLM (Qin et al., 2024) improve API-calling accuracy through API dataset construction and model fine-tuning, while Chameleon (Lu et al., 2023) enhances LLM performance via plug-and-play module integration. (3) Planning: Leveraging LLMs' reasoning abilities to automate task decomposition and execution planning. Methods like CoT (Wei et al., 2022), PoT (Chen et al., 2023), and SCoT (Li et al., 2025) enhance reasoning performance by generating intermediate reasoning steps before final solutions. (4) Multi-Agent Collaboration: Coordinating multiple role-playing LLMs to accomplish complex tasks. Systems like ChatDev (Qian et al., 2024) and AutoGen (Wu et al., 2024) simulate real-world collaborative environments to solve intricate problems. However, existing approaches inadequately address the critical role of context quality during LLM reasoning. CtxWF innovatively improves reasoning performance by integrating reflection (error feedback mechanisms during code/SQL execution and guided regeneration after failures), tools (action invocation), planning (preprocessing and action planning phases), and multi-agent principles (single-responsibility role design). These integrations collectively ensure high-quality context—sufficient information, minimal noise, and task-specific focus—during every LLM reasoning step.

Code as Action: LLMs have achieved remarkable results on code generation benchmarks. Given code's universality, many systems employ code as the primary agent-environment interaction medium. VOYAGER (Wang et al., 2024a) enables automated exploration in Minecraft through code-based interactions. CodeAct (Wang et al., 2024b) exclusively uses code for multi-step task solving, while OpenHands (Wang et al., 2024c) extends this paradigm for coding-specific agents. However, prioritizing generality through pure code generation often sacrifices accuracy. CtxWF addresses this trade-off through three key strategies: (1) guiding LLMs to proactively acquire task-critical information before code generation, (2) encouraging task decomposition when handling overly complex problems, (3) controlling feedback quality post environment interaction. These mechanisms collectively enhance LLMs' performance in real-world data analysis tasks resolution.

5 Conclusion

In summary, we present a novel agent design methodology that addresses the limitations of current LLM-based agents by focusing on the management of global context. Our approach enables agents to concentrate on single objectives during each reasoning cycle while maintaining high-quality reasoning context, thereby achieving superior overall performance. The proposed method demonstrates promising capabilities in automatically resolving complex real-world problems, exhibiting remarkable effectiveness on the challenging DA-Code benchmark for real-world data analysis. It establishes new state-of-the-art accuracy across multiple models and evaluation metrics. We anticipate that incorporating additional optimization strategies and novel algorithms into the existing design framework could further enhance performance across broader application scenarios in future research.

6 Limitations

Domain Limitations: Our approach requires customizing the workflows of agents to replace certain decision-making processes of LLMs, which compromises some degree of generality. However, such trade-offs are inevitable when constructing stable and effective agents, as there exists an inherent tension between generality and determinism.

Inaccuracy Issues: Despite our meticulous design of the agent architecture for accuracy, several failure patterns persist: (1) During two-stage planning, LLMs frequently generate overly complex subsequent action objectives. Even with customized prompting strategies to induce simpler task decomposition, LLMs often reiterate identical problematic goals; (2) When executing ReadText actions, LLMs occasionally extract only partial task-relevant content, with omitted critical information leading to subsequent operations deviating from file-specified requirements; (3) Our current assumption of a compact action space (to preserve LLM context window capacity) may limit handling of complex tasks requiring extensive actions. Potential solutions could involve dynamic retrieval of action sets via RAG (Retrieval-Augmented Generation) (Gao et al., 2023) prior to task planning.

These limitations highlight promising directions for future agent research. While substantial optimization opportunities remain, we maintain these shortcomings do not diminish the significance of our contributions. The current CtxWF framework, despite employing basic code-generated actions and a simplistic error feedback mechanism for LLM reflection, already demonstrates competitive performance. We posit that integrating domain expert workflows (e.g., data scientists' problem-solving patterns) into agent architectures could enable LLMs to generate more effective solutions - a valuable direction for subsequent research.

7 Ethics Statement

The system is designed to augment rather than replace data scientists. By incorporating human cognitive decision-making processes into the agent design architecture, our approach strategically enhances LLM performance in data analysis through controlled restriction of certain decision-making authorities. All datasets employed in this study were sourced from repositories with explicit MIT open-source licenses. The complete implementation codebase, including evaluation datasets, will subsequently be released under the same MIT license to ensure reproducibility and community accessibility.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Ben Alderson-Day and Charles Fernyhough. 2015. Inner speech: Development, cognitive functions, phenomenology, and neurobiology. *Psychological bulletin*, 141(5):931.

Alan Baddeley. 1992. Working memory. *Science*, 255(5044):556–559.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Anthony Steven Dick and Willis F. Overton. 2009. Self- and social-regulation social interaction and the development of social understanding and executive functions.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR.

Yiming Huang, Zhenghao Lin, Xiao Liu, Yeyun Gong, Shuai Lu, Fangyu Lei, Yaobo Liang, Yelong Shen, Chen Lin, Nan Duan, and Weizhu Chen. 2024a. Competition-level problems are effective LLM evaluators. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13526–13544, Bangkok, Thailand. Association for Computational Linguistics.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024b. DA-code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13487–13521, Miami, Florida, USA. Association for Computational Linguistics.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 18319–18345. PMLR.

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025. Structured chain-of-thought prompting for code generation. *ACM Trans. Softw. Eng. Methodol.*, 34(2).

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024b. Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977*.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, volume 35, pages 2507–2521. Curran Associates, Inc.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 43447–43478. Curran Associates, Inc.

Aleksandr Romanovich Luria. 1965. Ls vygotsky and the problem of localization of functions. *Neuropsychologia*, 3(4):387–392.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

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

X Team. 2023. Xagent: An autonomous agent for complex task solving. *XAgent blog*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024c. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep CALM and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8736–8754, Online. Association for Computational Linguistics.

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*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Ekaterina Zavershneva and Rene van der Veer. 2018. Thinking and speech.

A Method

A.1 Proactive Information Acquisition

Prompt for the preprocessing steps before proceeding to the next action planning:

```
# ROLE
You are an assistant who evaluates
    whether the current code task
    requires more file information
    according to the rules. If the rules
     are violated, you can only use the
    actions provided in the ACTION SPACE
     to acquire all the necessary info
    that has not been acquired.

# ACTION SPACE
{retrieval_action_space}

# Rules
1. You need to ensure that I have
    already obtained the necessary file
    information before executing the
    current code task.
2. You should first obtain the relevant
    information about the file before
    saving content to a file.
3. You should ensure that you have
    obtained the format information for
    the specified file.

# Current directory
{files_info}

# Current code task
{current_task}

# RESPONSE FORMAT
1. thought: Based on the information I
    listed above, do reasoning to
    evaluate the code task.
2. actions: All the signature of the
    actions you need.

‘‘‘json
{
    "thought": "thought",
    "actions": ["signature"]
}
‘‘‘
```

Prompt for selecting relevant database table names from the database before generating SQL:

```
# ROLE
You are a database expert, skilled at
    identifying the tables in a database
     that need to be examined further
    based on the current task goal.

# Database table name
‘‘‘
{tables}
‘‘‘

# Current task goal
{current_task}

# RESPONSE FORMAT
1. thought: Based on the information I
    listed above, do reasoning to
    evaluate the task.
2. tables: All the name of the tables
    you need to be examined further.

‘‘‘json
{
    "thought": "thought",
    "tables": []
}
‘‘‘
```

A.2 Stage 1

A.2.1 Prompt for planning

The prompt used in the first stage to predict the next action:

```
# ROLE
You are a data scientist proficient in
    data analysis, skilled at using code
     to solve data-related problems. You
     can only use the actions provided
    in the ACTION SPACE to determine the
     next action to do. The maximum
    number of the actions you can take
    is {max_steps}.

# ACTION SPACE
{action_space}

# KNOWN FACTS
## Current directory
{files_info}
## Final task
{task}
## Completed action so far
{action_history}

# ATTENTION
1. You need to fully understand the
    action space and its arguments
    before using it.
2. You should first understand the known
     facts before handling the task.
3. You only need to execute the action
    for the same argument once.
4. Before finishing the task, ensure all
     instructions are met and verify the
     existence and correctness of any
    generated files.
5. If a task goal fails multiple times,
    try breaking it down into multiple
    simpler subtasks, and print the
    results of the subtasks or save them
     to a temporary file. Finally, merge
     these files.

# RESPONSE FORMAT
For each task input, your response
    should contain:
1. Based on the information I listed
    above , do reasoning about what the
    next action should be. (prefix "
    Thought: ").
2. One action string in the ACTION SPACE
     (prefix "Action: ").
```

A.2.2 Action space

ReadTable

```
## ViewTable Action
* Signature: ViewTable(file_path="path/
    to/table_file")
* Description: This action will get the
    table structure and a portion of the
     data of the table file located at '
    file_path'.
* Constraints:
  - The table file must be accessible
      and in a tabular data format (e.g
      ., .csv, .tsv).
* Example: ViewTable(file_path="info.csv
    ")
```

ReadText

```
## ReadTextFile Action
* Signature: ReadTextFile(file_path="
    path/to/file", task_goal="a detailed
     description of the information you
    want to obtain in the file")
* Description: This action will read the
     file and extract a **relevant
    section of text** from the file
    specified by 'file_path' based on
    the 'task_goal'.
* Example: ReadTextFile(file_path="info.
    txt", task_goal="the description for
     'money'")
```

GenerateCode

```
## CodeTaskExecutor Action
* Signature: CodeTaskExecutor(task_goal
    ="task_goal")
* Description: This action will generate
     and execute the program code to
    achieve the task goal.
* Example: CodeTaskExecutor(task_goal="
    Print the 'Hello, world!' string.")
```

GenerateSQL

```
## SQLTaskExecutor Action
* Signature: SQLTaskExecutor(file_path="
    path/to/database_file", task_goal="a
     detailed description of the task")
* Description: This action will generate
     and execute the SQL commands on the
     specified database file to achieve
    the task goal.
* Constraints:
  - The database file must be accessible
      and in a format compatible with
      SQLite (e.g., .sqlite, .db).
* Example: SQLTaskExecutor(file_path="
    data.sqlite", task_goal="Calculate
    the average of the quantities.")
```

Decompress

```
## Decompress Action
* Signature: Decompress(file_path="path/
    to/compressed_file")
* Description: This action will extract
    the contents of the compressed file
    located at 'file_path'. It supports
    .zip and .tar and .gz formats.
* Examples:
  - Example1: Decompress(file_path="data
      .zip")
  - Example2: Decompress(file_path="data
      .gz")
```

Answer

```
## Answer Action
* Signature: Answer(output="
    literal_answer_or_output_path")
* Description: This action denotes the
    completion of the entire task and
    returns the final answer or the
    output file/folder path. Make sure
    the output file is located in the
    initial workspace directory.
* Examples:
  - Example1: Answer(output="New York")
  - Example2: Answer(output="result.csv
      ")
  - Example3: Answer(output="FAIL")
```

A.3 Stage 2

The implementation details of some actions in the second stage are as follows:

ReadTable

```
import pandas as pd

pd.set_option('display.max_columns',
    None)
pd.set_option('display.expand_frame_repr
    ', False)

file_path = "{file_path}"
if file_path.endswith('.xlsx') or
    file_path.endswith('.xls'):
    df = pd.read_excel(file_path)
elif file_path.endswith('.tsv'):
    df = pd.read_csv(file_path, sep='\t
        ')
else:
    df = pd.read_csv(file_path)

print(df.head(1))
print('...')
print(f'[{len(df)} rows x {len(df.
    columns)} columns]')
```

ReadText

```
You are a helpful assistant in
    information retrieval. Now I need to
     obtain some information, and you
    should extract the relevant snippets
     from the file content based on the
    descriptions I provide.

The relevant snippets I need to obtain:
```
{task_goal}
```

The contents of the '{file_path}' file:
```
{file_content}
```
```

~~~
You should only respond in the format as
     described below:
RESPONSE FORMAT:
For each input, your response should
    contain:
1. One analysis of the query, reasoning
    to determine the required
    information (prefix "Thought: ").
2. One string of the relevant original
    content snippets (prefix "Content:
    ").

Thought: ...
Content:
```Plain Text
...
~~~
```

GenerateCode

Prompt for generating code:

```
# ROLE
You are a data scientist proficient in
    data analysis, skilled at using
    Python code to solve data-related
    problems. You can utilize some
    provided APIs to address the current
     task. If you need to print
    information, please use the print
    function.

# USEFUL APIS
{apis}

# KNOWN FACTS
## Current directory
{files_info}
## Final task
{task}
## Acquired information
{action_history}
## Current task
{current_task}
## Wrong code from the last round
{last_code_info}

# RESPONSE FORMAT
For each task input, your response
    should contain:
1. One analysis of the known facts,
    reasoning to complete the current
    task (prefix "Thought: ").
2. One executable piece of python code
    to achieve the current task (prefix
    "Code: ").
```python
...
```
```

GenerateSQL

Refer to Appendix A.1 for the prompt used to select relevant database table names.

Prompt for generating SQL:

```
# ROLE
You are a database expert skilled at
    achieving current task goal through
    SQL commands.

# KNOWN FACTS
## Current directory
{files_info}
## Final task
{task}
## Current task
{current_task}
## Acquired information
{action_history}
## Database table names
```
{tables}
```
## Relevant tables structure
{table_columns}
## Wrong sql command from the last round
{last_sql_info}

# RESPONSE FORMAT
1. thought: Based on the information I
    listed above, do reasoning to
    generate the SQL commands to achieve
     the current task goal.
2. sql_command: An SQL command string.
3. output: The file path where the
    results are saved as a CSV file.

```json
{
    "thought": "",
    "sql_command": "",
    "output": ""
}
```
```
```