How Can Input Reformulation Improve Tool Usage Accuracy in a Complex Dynamic Environment? A Study on τ-bench

Venkatesh Mishra[†*] **Amir Saeidi**[†*] **Satyam Raj**[†] **Mutsumi Nakamura**[†]
Gaowen Liu[‡] **Ali Payani**[‡] **Jayanth Srinivasa**[‡] **Chitta Baral**[†]

[†]Arizona State University [‡]Cisco Research
{vmishr23, ssaeidi1, chitta}@asu.edu, {gaoliu, apayani, jasriniv}@cisco.com

Abstract

Recent advances in reasoning and planning capabilities of large language models (LLMs) have enabled their potential as autonomous agents capable of tool use in dynamic environments. However, in multi-turn conversational environments like τ-bench, these agents often struggle with consistent reasoning, adherence to domain-specific policies, and extracting correct information over a long horizon of tool-calls and conversation. To capture and mitigate these failures, we conduct a comprehensive manual analysis of the common errors occurring in the conversation trajectories. We then experiment with reformulations of inputs to the tool-calling agent for improvement in agent decision-making. Finally, we propose the **Input-Reformulation Multi-Agent (IRMA)** framework, which automatically reformulates user queries augmented with relevant domain rules and tool suggestions for the tool-calling agent to focus on. The results show that IRMA significantly outperforms ReAct, Function Calling, and Self-Reflection by **16.1%**, **12.7%**, and **19.1%**, respectively, in overall pass^5 scores. These findings highlight the superior reliability and consistency of IRMA compared to other methods in dynamic environments.

1 Introduction

Recent advancements in Large Language Models (LLMs) [1–4] have created the potential for them to be used as reasoning agents with varied levels of autonomy in complex real-world tasks like customer support, scientific discovery, legal reasoning and enterprise operations [5–10]. However, such complex tasks require the need of reasoning and planning capabilities beyond just language processing: they require the ability on behalf of these agents to be able to invoke suitable tools[2] which can complete tasks through logic implemented in computer programs leading to deterministic outcomes. Recent research [11–13], which benchmarks the simulation of such real-world problem-solving settings, shows that LLM-agents significantly falter in correctly solving these tasks and commit errors that range from generative hallucinations to failure to adhere to context and domain-specific policy violations by incorrect reasoning about actions over extended interactions.

These shortcomings underscore the need for more fine-grained evaluations and methods that can diagnose and address the nuanced failure modes of LLM agents in complex, real-world interactions that employ natural language as a form of communication. Thus, our main focus in this work is to find

[*] Equal Contribution

[2]The terms 'tool-use', 'tool-calling' and 'function-calling' are used interchangeably in this paper

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Workshop on Multi-Turn Interactions in Large Language Models.

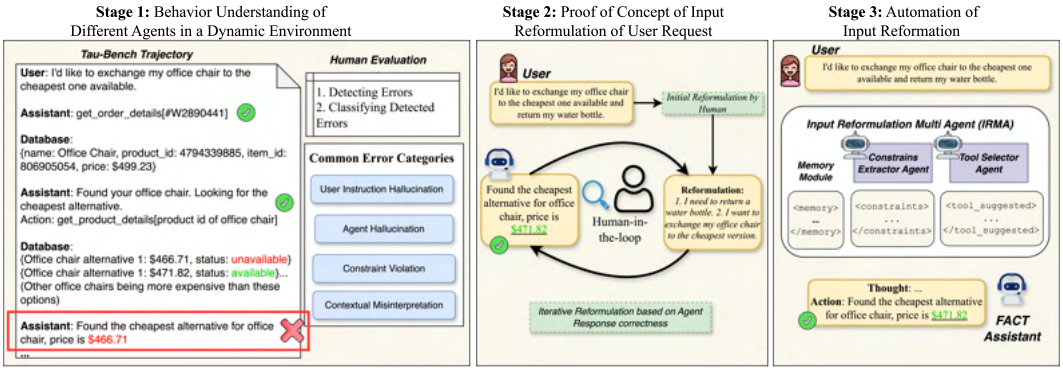

Figure 1: Overview of the tasks conducted for evaluating and improving tool-calling capabilities of language agents in τ-bench [11]. **Stage 1)** involved human evaluators manually evaluating simulated conversation trajectories to find common failure modes of the language agents. **Stage 2)** employs a human-in-the-loop approach to experiment with various prompt reformulations to improve agent correctness. **Stage 3)** automates this process through the IRMA framework, which leads to improved agent behavior.

and mitigate the causes of why language agents fail to solve simulations of real-world conversational requests that require complex reasoning and relevant information processing according to the situation at hand. To this end, we utilize τ- bench [11] as an appropriate test-bed for such investigation as it emulates realistic airline and retail dialogues. We define the reasoning about actions of language agents as the ability to generate context-aware inference and decision-making tokens for selecting the next best action (a tool-call in this context). Additionally, we define and evaluate the planning capabilities of the agents through decision-making for tool-calling over multiple tool-calls in the correct sequential manner to complete a goal.

To address the challenges, we propose a three-pronged sequential approach. *First*, we develop a comprehensive error classification that categorizes common reasoning and planning mistakes in a multi-turn tool-calling simulation. This taxonomy serves as a diagnostic guideline to systematically identify and understand the causes of failures for LLM agents. *Second*, we manually experiment with input reformulations of the user requests to evaluate whether the correct prompt reformulations can guide the tool-calling agents towards correct decision-making through appropriate tool-calling/response to the user. *Third*, we automate this prompt-reformulation process by building a multi-agent LLM framework (§5.2), called **Input-Reformulation Multi-Agent (IRMA)**, which further optimizes the input reformulation with augmentation of follow-up questions (§5.1). Before the tool-calling agent invokes or responds to any tool output, our automated framework supplies targeted guidance that ensures strict adherence to domain-specific rules and well-placed follow-up questions to extract accurate information, thereby enhancing its reasoning and planning capabilities in dynamic environments.

Our results show that the IRMA framework not only outperforms ReAct [14], Function Calling, and Self-Reflection [15] on pass@1, but also achieves **20%** and **22.4%** higher accuracy on Airline tasks compared to Gemini 1.5 Pro-FC and Claude 3.5 Haiku-FC, respectively. IRMA also demonstrates stronger reliability, with higher scores on pass^4 and pass^5 (Figure 3). In addition, IRMA solves tasks in fewer turns than competing methods, highlighting its efficiency (Figure 4). Lastly, IRMA shows greater robustness, with an increased performance gap on pass^5 after removing tasks affected by ground truth and instruction errors in the airline and retail domains.

2 Related Works

Tool-Integration for LLMs The ReAct framework, introduced by Yao et al. [14], is one of the first approaches to explore the potential of Large Language Models (LLMs) as tool-using agents by integrating reasoning and acting within LLMs. Toolformer [16] presents a fine-tuning approach to teach LLMs to invoke tool calls. ToolEVO [17] and ToolLLM [18] employ tree search algorithms for integrating and evaluating tool-learning capabilities in LLMs. ToolACE [19], AutoTools [20],

and APIGen [21] introduce automated frameworks designed to generate accurate, complex, and high-quality tool-learning data, with works like [22, 23] extending this to multi-turn interactive conversational settings.

Tool-Use Benchmarks LLMs have been extensively evaluated on invoking external functions in both single-turn and interactive multi-turn conversational test beds. API-Bench [24] and API-Bank [25] are two prominent benchmarks designed to evaluate the function-calling capabilities of LLMs in single-turn scenarios. NESTful [26] focuses on evaluating LLMs' ability to handle nested sequences of API calls. ToolQA-D [17] gauges robustness in changing API specifications. τ-bench [11] and ToolSandbox [12] emulate realistic dialogues requiring policy-compliant tool use over multi-turn user-agent interactions, where each step modifies an external environment. While these existing multi-turn benchmarks evaluate the overall success of tool-calling agents, they lack fine-grained analysis of reasoning errors while following complex domain rules—a gap our work addresses through the construction of a fine-grained error classification by evaluating τ-bench.

3 Problem Statement

To evaluate the tool-usage capabilities of current Large Language Models (LLMs), we adopt the benchmark provided by τ-bench [11]. This benchmark is specifically designed to assess language agents in realistic, multi-turn interaction settings. τ-bench includes tasks from two domains: (1) Airline, comprising 50 tasks centered around flight reservation scenarios, and (2) Retail, containing 115 tasks focused on shopping and order management. In this setup, both the user and the customer-service assistant are simulated by LLMs, enabling a controlled environment for analyzing interactive behavior.

Each task is framed as a Partially Observable Markov Decision Process (POMDP) (Details in Appendix A), where the assistant agent must generate appropriate function calls based on user inputs. These function calls are executed in an external environment, which then returns outputs that shape the ongoing dialogue. The interaction continues until the user ends the conversation, and the performance of the assistant is evaluated based on final rewards. These rewards reflect how closely the agent's actions align with gold-standard trajectories and how well it fulfills the user's goals.

A key challenge in τ-bench arises from the dynamic nature of user-agent interactions, where both user inputs and agent responses can vary across runs. This variability requires the agent to consistently execute correct action sequences, regardless of the conversational path. However, current results indicate that even state-of-the-art LLMs struggle to reliably complete these tasks as the number of trials increases. To address this limitation, we conduct a root-cause analysis of common agent errors (§4) and introduced IRMA, a multi-agent framework (§5) designed to improve agent reliability in this challenging setting.

4 Error-Classification

To identify the failure modes of LLMs, human evaluators conducted experiments using GPT-4o [27] as the base model for both the user and the assistant agent across all tasks in τ-bench [11]. Both ReAct and function-calling agent configurations were used to generate up to five trials per task in each domain. Evaluators manually reviewed the resulting multi-turn conversation trajectories from the retail and airline domains. While prior studies [28–30] have examined failures related to tool availability, definition errors, or tool set complexity, our analysis focuses specifically on the contextual reasoning limitations of LLMs in generating tool calls within dynamic, multi-turn interactions.

Although τ-bench provides a general taxonomy of failure types for the retail domain, our classification is more cause-oriented than effect-oriented. By framing errors in terms of their underlying causes, we can more effectively inform the design of targeted interventions, such as retrieval-augmented memory to mitigate context retention issues or follow-up question generation (§5.1) to reduce hallucinations from context drift. The following subsections (§4.1–§4.4) provide a detailed breakdown of the identified error types.

4.1 User Instruction Hallucination

User instruction errors occur when the LLM-simulated user deviates from the original task instruction, typically in the later stages of a conversation. These errors highlight the limitations of LLMs in maintaining instruction fidelity over long contexts, especially when multiple follow-up turns introduce competing directives. Another contributing factor is context drift, where the model increasingly relies on recent inputs or high-probability continuations, leading it to overlook or forget the initial user intent. An Example illustrating this error is provided in Figure 7 in Appendix D.

4.2 Agent Hallucination

Agent hallucination errors arise when the assistant agent generates incorrect or incomplete responses that fail to fully satisfy the user's request. For example, the agent may neglect to process all items specified by the user or incorrectly fulfill a request by selecting the wrong item or applying it to the wrong order. These errors reflect underlying challenges with LLM memory limitations [31] and the degradation of instruction-following abilities over long contexts [32]. As prior context accumulates, excessive or outdated information can distort the model's understanding, leading to hallucinated outputs and ultimately incorrect decisions [33].

4.3 Domain Policy Violation

Domain policy violations occur when tool-calling agents make decisions that contradict the domain-specific constraints defined for task completion. For instance, in Retail task 19 (Figure 8), the agent attempts to exchange the user's office chair and pet bed even when the order is no more in 'delivered' status: a prerequisite domain rule required to be satisfied for exchange. This leads to the agent violating the domain rule (see Figure 11): '*An order can only be exchanged if its status is 'delivered'...*' Such violations may also arise when the user issues an invalid request, and the agent proceeds to fulfill it without adhering to the applicable domain rules. This error is caused due to similar reasons as mentioned in §4.1 and §4.2.

4.4 Contextual Misinterpretation

Contextual misinterpretation errors occur when the tool-calling agent misunderstands the intent or nuance of the user's request and generates function calls using inappropriate tools for the given context. For example, if a user asks to *return* an item and receive a different one in exchange, a human familiar with the domain policies would recognize this as an *exchange* request. However, the LLM-based agent may misinterpret it as a simple *return*, failing to grasp the full context and thereby invoking the wrong tool.

5 Methodology

As outlined in the previous sections, complex dynamic environments such as τ-bench present reliability challenges. Specifically, the user simulator may hallucinate during interactions, generating questions that do not adhere to the provided instructions. In this study, we aim to improve the assistant agent's tool-calling performance in τ-bench by enabling more accurate decision-making. Unlike prior approaches that monitor and correct agent actions through verification or reflection, our method focuses on enhancing the quality of the agent's input before any action is taken. To achieve this, we first introduce a novel

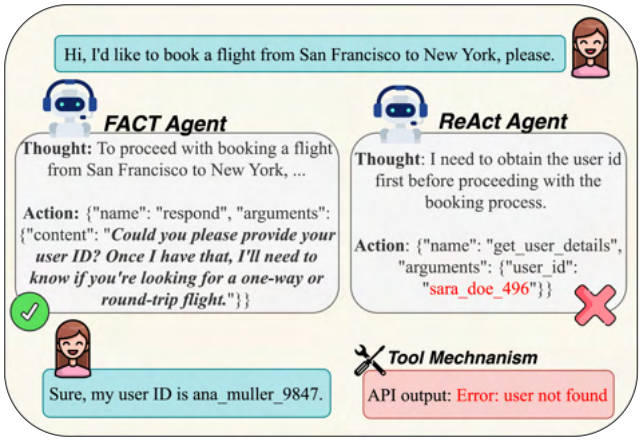

Figure 2: FACT agent demonstrates superior user guidance, avoiding tool-call errors encountered by the ReAct Agent.

prompting strategy: Follow-up Question Acting (FACT), designed to support decision-making in dynamic settings. We then present the Input Reformulation Multi-Agent (IRMA) framework that reformulates the agent's input to guide more effective and context-aware decisions.

5.1 FACT: Follow-up question ACTing

Although reasoning-based prompting techniques like ReAct outperform non-reasoning methods such as Act, they remain inefficient in dynamic environments. As shown in Figure 2, ReAct often calls a tool prematurely, triggers an error, and only then asks clarifying questions, leading to longer conversations and increased interaction issues. To overcome this, we introduce **Follow-up Question ACTing (FACT)**, a prompting method that first gathers information through targeted questions before calling a tool. Our results in Figure 4 show that FACT is more effective than ReAct and performs comparably to Function Calling. We refer readers to Appendix §E.1.

Another advantage of FACT is its ability to involve the user in the loop. When the user simulator hallucinates or provides misleading input, FACT detects the issue and hands off the conversation to a human, ensuring more robust handling of unreliable inputs. In summary, FACT is more efficient, reliable, and consistent than other methods in dynamic environments. However, in long conversations, it may forget domain rules and tools due to system prompt limitations, leading to domain violations. To address this, we propose the Input-Reformulation Multi-Agent Framework (IRMA), which restructures the user prompt to retain key information like domain rules and a relevant tool list within the assistant's input.

5.2 IRMA: Input-Reformulation Multi-Agent Framework

Our analysis reveals three key failure cases for assistant agents. First, in long conversations, the agent may forget the user's initial request and respond only partially. Second, it may violate domain rules by forgetting constraints from lengthy policy lists. Third, tool selection becomes harder over time, especially when tools have similar names (e.g., "search_direct_flight" vs. "search_onestep_flight"), leading to incorrect choices.

We hypothesize that combining user queries with crucial context, such as domain rules and relevant tools, can improve the assistant agent's decision-making. To test this, we conducted a *human-in-the-loop* experiment with prompt engineers who reformulated queries using additional policy and tool information. In most cases, the agent successfully completed the tasks, motivating us to automate this input reformulation process.

Based on this insight, we propose the **Input Reformulation Multi-Agent Framework (IRMA)**. In contrast to prior methods that focus on post-hoc correction of the agent's behavior, such as Self-Reflection, PlanGen [34], or other verification-based approaches, IRMA centers on enhancing the quality of the input provided to the assistant agent. This approach enhances decision-making at the input stage—before any action is taken—ensuring more accurate and context-aware responses. The framework comprises three core modules: memorization, constraints, and tool suggestion.

Memorization This module is independent of the language model and is responsible for storing the user queries throughout the interaction trajectory. It helps the agent retain awareness of the initial request and make decisions accordingly. The conversation history is maintained within `<memory>` tags.

Constraints One of the main reasons the agent makes incorrect decisions is domain policy violation. A key insight from the human-in-the-loop experiment was the positive impact of providing a concise list of domain constraints to guide the assistant agent's decisions. To address this challenge, we define a dedicated agent that generates a checklist of relevant domain constraints based on the user query. If the user query is a response to a follow-up question from the assistant, the agent is prompted to return "None". The generated constraint list is stored within `<constraints>` tags to ensure the assistant agent receives a structured and interpretable input prompt.

Tool Suggestion Although the number of available tools is limited, the assistant agent sometimes struggles to select the most relevant tool for a given user query. In some cases, after encountering an error or receiving an empty output, the agent may lose track of other parts of the user's request.

To mitigate this, we introduce a Tool Selector agent that generates a short list of tools most relevant to the user query, along with a one-line explanation for each suggestion. This list is stored within `<tool_suggested>` tags to help the assistant agent focus on selecting the most appropriate tool.

In summary, the IRMA framework aims to replicate the input reframing performed by researchers during the human-in-the-loop experiment. Unlike other techniques such as verification, self-reflection, or agentic verification methods, IRMA functions in a loop-free manner and focuses on strengthening the input by reformulating the user query. This approach not only improves accuracy but also offers better cost-effectiveness compared to alternative methods. In the next section, we provide a comparative analysis of IRMA against existing techniques.

6 Experiments

6.1 Experimental Setup

In this section, we introduce the baseline models and methods used for comparison, followed by a detailed analysis of the Input Reformulation Multi-Agent (IRMA) framework using various evaluation metrics.

Models and Methods We evaluated IRMA against a range of open-source and closed-source language models. The open-source models include Qwen2.5-32B, LLaMA3-70B, DeepSeek-V3, and Phi-4-14B, while the closed-source models comprise Claude 3.5, Gemini 1.5, and GPT-4o. In addition, we compared IRMA with three widely adopted prompting strategies: (1) ReAct, a reasoning-based prompting technique; (2) Function Calling, designed specifically to enhance a model's tool-calling capability; and (3) Self-Reflection, a method aimed at improving tool-use performance by addressing errors in the agent's actions.

Model	**Method**	**τ-Retail**	**τ-Airline**	**Overall**
Open-Source Models				
Qwen 2.5 32B	ReAct	24.4	25.0	24.7
Llama 3.1 70B	ReAct	50.4	26.0	38.2
DeepSeek v3[1]	ReAct	58.3	22.8	40.6
Phi-4 14B	ReAct	32.2	28.0	30.1
Close-Source Models				
Gemini 1.5 pro[1]	FC	54.9	25.2	40.1
Claude 3.5 Haiku[2]	FC	51.0	22.8	36.9
Claude 3.5 Sonnet[2]	FC	**62.6**	36.0	<u>49.3</u>
gpt-4o	FC	<u>60.5</u>	42.4	51.4
gpt-4o	ReAct	51.8	39.6	45.7
gpt-4o	SR	51.1	<u>44.8</u>	47.9
gpt-4o **(ours)**	IRMA	58.3	**47.2**	**52.75**

Table 1: Performance of various open and closed-source models in Pass^1 for retail and airline domains in τ-bench across 5 runs. 'SR' stands for the Self-Reflection agent. [1] from [35]; [2] from [36].

Evaluation To evaluate performance, we use the pass^k metrics, which measure the reliability and consistency of models across different prompting strategies. The pass^k metric (pronounced "pass hat k") is defined as the probability that all of the k independently sampled outputs successfully complete the task, averaged across all tasks. Specifically, if a task is run for n independent trials and c of those are successful (i.e., have a correct result with reward $r = 1$), an unbiased estimate of pass ^k can be computed using the following formula:

$$\text{pass\^{}k} = \mathbb{E}_{\text{task}} \left[\binom{c}{k} \Big/ \binom{n}{k} \right]$$

This metric provides insight into how likely a model is to succeed given multiple attempts, capturing both reliability and diversity in its outputs.

6.2 Experimental Results

As outlined in the τ-bench, in real-world scenarios—reliability and consistency are often more critical than the average success rate (measured by pass@1). We argue that an ideal agentic method should exhibit three key properties: (1) Accuracy, (2) Reliability, and (3) Consistency. Accordingly, we

Figure 3: Comparison of IRMA and other techniques across five runs with varying values of K. The figure shows a significant performance difference between IRMA and other methods on pass^5. Note that all methods use GPT-4o as the base model. See Appendix B for more details.

begin by comparing results using pass@1 to assess accuracy, and then evaluate the performance of state-of-the-art methods using pass^k to measure reliability and consistency.

IRMA outperforms other state-of-the-art methods in tool calling. We conducted evaluations of multiple methods—Function Calling (FC), ReAct, and Self-Reflection—each executed over five trials. These experiments were performed using the GPT-4o as backbone LLM. The results, presented in Table 1, show that the IRMA framework outperforms ReAct, Self-Reflection, and FC by 6.1%, 3.9%, and 0.4%, respectively, in overall pass@1 score. Additionally, in the airline tasks, which represent the most challenging scenarios within the dynamic environment, IRMA on GPT-4o achieves improvements of 20%, 22.4%, and 9.2% compared to Gemini 1.5 Pro-FC, Claude 3.5 Haiku-FC, and Claude 3.5 Sonnet-FC, respectively. These findings highlight IRMA's strong accuracy in real-world tasks and demonstrate its effectiveness over existing methods.

IRMA is more reliable and consistent than other methods in dynamic settings. The results in Table 1 show that the performance of IRMA on retail pass^1 is slightly lower than that of GPT-4o-FC. For this reason, we further explored the performance of other methods using pass^k to evaluate their reliability and consistency. The results in Figure 4 show that IRMA, compared with ReAct and FC on GPT-4o, is much more reliable and consistent, outperforming ReAct and FC by 16.1% and 12.6%, respectively, in overall scores on pass^5.

IRMA is more robust on tasks with GT and UI errors. As explained in the previous sections, τ-bench suffers from two major issues: (1) Ground Truth (GT) errors and (2) User Instruction (UI) errors. Figure 5 in the Appendix shows the distribution of these errors across the airline and retail tasks. We progressively removed tasks affected by these problems, and the results revealed that the performance of all three methods improved, with IRMA showing slightly greater gains compared to the others. We hypothesize that IRMA is more robust to hallucination-related issues. Specifically, in tasks with GT errors, IRMA tends to avoid incorrect tool calls or invalid actions and instead produces safe and accurate responses.

A key observation is the change in performance difference between IRMA and FC on pass^5. Before removing tasks with GT and UI errors, IRMA outperformed FC by 10%. However, after removing these problematic tasks, the performance gap widened to 16.1% on average. Similar patterns were observed for other methods as well, reinforcing the claim that IRMA is more robust and less sensitive to noisy supervision and ambiguous instructions compared to existing techniques.

IRMA solves tasks more efficiently and effectively, using fewer turns than others. One of the primary reasons assistant agents make incorrect decisions in the final turns is the length of the conversation, which often causes them to forget important rules and instructions. In an ideal scenario, an assistant should resolve the user's query with the fewest but most effective actions. To investigate this aspect, we analyzed the distribution of turns in successful task completions by IRMA, ReAct,

FC, and Self-Reflection, as shown in Figure 4 in the Appendix. The results show that, in retail tasks, IRMA completes tasks with 7.9 points fewer turns than Self-Reflection and 3.1 points fewer than FC. In airline tasks, IRMA requires 8.3 fewer turns than Self-Reflection, 1.1 fewer than FC, and 3.3 fewer than ReAct. These results demonstrate IRMA's superior efficiency compared to other state-of-the-art methods.

Input Reformulation framework vs Self-Reflection The central concept of IRMA is to reformulate the agent's input under the assumption that supplying sufficient and well-structured information enables the agent to act more reliably and consistently in real-world scenarios. To evaluate this, we implemented the Self-Reflection method (Appendix F), which analyzes the agent's previous actions and extracts relevant information from domain rules to guide future decisions (see section E.1 for implementation details). As shown in Figure 3, IRMA outperforms Self-Reflection in both airline and retail tasks, achieving a 3.9% higher overall score in pass@1. More notably, IRMA exceeds Self-Reflection by 19.1% in pass^5, highlighting its superior reliability in a real-world environment.

In summary, while ReAct and Self-Reflection perform well in certain settings, they fall short in complex, dynamic environments like τ-bench. Role-play methods, including verification techniques, are also inefficient, as real-world scenarios require assistant agents to act based on limited information, with each action affecting the environment. Although Function Calling was designed for tool use, our results show it lacks reliability in decision-making and offers limited controllability, even in GPT-4o with tailored system prompts. Combining FACT with GPT-4o-FC led to a 12% performance drop, highlighting the need for more robust approaches. In contrast, IRMA consistently delivers higher accuracy, reliability, and consistency in dynamic environments like τ-bench.

7 Conclusion

In this work, we investigate the limitations of state-of-the-art LLM-based tool-calling agents in complex, multi-turn environments, focusing on the retail and airline domains of τ-bench. Through a detailed analysis of conversation trajectories, we identify four major failure modes: *user instruction hallucination, agent hallucination*, *domain-policy violations*, and *contextual misinterpretation*, all of which stem from limitations in memory retention, contextual reasoning, and adherence to domain constraints across extended interactions. To address these challenges, we propose the Input Reformulation Multi-Agent (IRMA) framework, designed to enhance the structure of the assistant agent's input. Our results show that IRMA not only outperforms other methods in pass^1 but also demonstrates significantly higher reliability, achieving an overall score of 43% pass^in τ-bench. Moreover, by leveraging the FACT agent, IRMA exhibits greater efficiency in task completion. In conclusion, IRMA shows robust and consistent behavior in the unreliable and dynamic environment of τ-bench, highlighting its effectiveness in real-world tool-use scenarios.

8 Limitations

Although the Input Reformulation Multi-Agent (IRMA) framework demonstrated superior performance on τ-bench, several limitations remain. As shown in Figure 3, while IRMA exhibits greater reliability compared to other methods, its performance on pass^5 still hovers around 43%. This indicates that there is still considerable room for improving the reliability of tool-using agents in real-world scenarios. Another limitation of this work is that our experiments and analysis are restricted to the τ-bench benchmark. It would be valuable to evaluate IRMA across a broader range of real-world environments to assess its generalizability.

Moreover, our observations suggest that beyond the error taxonomy we proposed, τ-bench itself suffers from issues related to unfair reward modeling. Building a truly dynamic and reliable evaluation environment—especially one that can control for the correctness of user instructions—would have a significant impact on the field. Such an environment would enable more rigorous development and evaluation of agentic frameworks and encourage further research into robust, real-world agent behavior. Ultimately, we believe this work contributes meaningfully to the research community and provides a strong foundation for developing more reliable and consistent agentic methods for dynamic environments.

Ethics Statement

We have utilized AI assistants, specifically Grammarly and ChatGPT, to correct grammatical errors and rephrase sentences.

References

[1] Yashwanth Annepaka and Prasenjit Pakray. Large language models: a survey of their development, capabilities, and applications. *Knowledge and Information Systems*, 67:2967–3022, 2025. doi: 10.1007/s10115-024-02310-4. URL https://doi.org/10.1007/s10115-024-02310-4.

[2] Amir Saeidi, Shivanshu Verma, Aswin RRV, Kashif Rasul, and Chitta Baral. Triple preference optimization: Achieving better alignment using a single step optimization. *arXiv preprint arXiv:2405.16681*, 2024.

[3] Divij Handa, Mihir Parmar, Aswin RRV, Md Nayem Uddin, Hamid Palangi, and Chitta Baral. Guidedsampling: Steering llms towards diverse candidate solutions at inference-time. *arXiv preprint arXiv:2510.03777*, 2025.

[4] RRV Aswin, Jacob Dineen, Divij Handa, Md Nayem Uddin, Mihir Parmar, Chitta Baral, and Ben Zhou. Thinktuning: Instilling cognitive reflections without distillation. *arXiv preprint arXiv:2508.07616*, 2025.

[5] Aili Chen, Xuyang Ge, Ziquan Fu, Yanghua Xiao, and Jiangjie Chen. Travelagent: An ai assistant for personalized travel planning. *arXiv preprint arXiv:2409.08069*, 2024.

[6] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[7] Harmanpreet Singh, Nikhil Verma, Yixiao Wang, Manasa Bharadwaj, Homa Fashandi, Kevin Ferreira, and Chul Lee. Personal large language model agents: A case study on tailored travel planning. In Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 486–514, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-industry.37. URL https://aclanthology.org/2024.emnlp-industry.37/.

[8] Yingxuan Yang, Qiuying Peng, Jun Wang, and Weinan Zhang. Multi-llm-agent systems: Techniques and business perspectives. *arXiv preprint arXiv:2411.14033*, 2024.

[9] Venkatesh Mishra, Bimsara Pathiraja, Mihir Parmar, Sat Chidananda, Jayanth Srinivasa, Gaowen Liu, Ali Payani, and Chitta Baral. Investigating the shortcomings of llms in step-by-step legal reasoning.

[10] Divij Handa, David Blincoe, Orson Adams, and Yinlin Fu. Optagent: Optimizing query rewriting for e-commerce via multi-agent simulation. *arXiv preprint arXiv:2510.03771*, 2025.

[11] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

[12] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024.

[13] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. 2024.

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

[15] Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

[16] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36: 68539–68551, 2023.

[17] Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. Learning evolving tools for large language models. *arXiv preprint arXiv:2410.06617*, 2024.

[18] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[19] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*, 2024.

[20] Zhengliang Shi, Shen Gao, Lingyong Yan, Yue Feng, Xiuyi Chen, Zhumin Chen, Dawei Yin, Suzan Verberne, and Zhaochun Ren. Tool learning in the wild: Empowering language models as automatic tool agents, 2025. URL `https://arxiv.org/abs/2405.16533`.

[21] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37: 54463–54482, 2024.

[22] Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika Awalgaonkar, Haolin Chen, Thai Hoang, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.

[23] Fan Yin, Zifeng Wang, I Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T Le, Kai-Wei Chang, Chen-Yu Lee, et al. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. *arXiv preprint arXiv:2503.07826*, 2025.

[24] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

[25] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

[26] Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, et al. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*, 2024.

[27] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[28] Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. Tools fail: Detecting silent errors in faulty tools. *arXiv preprint arXiv:2406.19228*, 2024.

[29] Cailin Winston and René Just. A taxonomy of failures in tool-augmented llms. In *Proceedings of the International Conference on Automation of Software Test (AST)*, April 28–29 2025.

[30] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[31] Lianlei Shan, Shixian Luo, Zezhou Zhu, Yu Yuan, and Yong Wu. Cognitive memory in large language models. *arXiv preprint arXiv:2504.02441*, 2025.

[32] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

[33] Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. ToolBeHonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11388–11422, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.637. URL https://aclanthology.org/2024.emnlp-main.637/.

[34] Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, et al. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. *arXiv preprint arXiv:2502.16111*, 2025.

[35] Scaled Cognition. Apt-1: Adaptive prompt tuning for llms. https://www.scaledcognition.com/blog/apt-1, 2025. Accessed: 2025-05-19.

[36] Anthropic. Claude 3.5 models and computer use. https://www.anthropic.com/news/3-5-models-and-computer-use, 2024. Accessed: 2025-05-20.

A Task Definition in τ-bench

Following Yao et al. [11], each task in τ-bench is modelled as a *partially observable Markov decision process* (POMDP)

$$\mathcal{M} = \langle \mathcal{S},\ \mathcal{A},\ \mathcal{O},\ \mathcal{T},\ \mathcal{R},\ \mathcal{U} \rangle.$$

We briefly restate every component and specify how they instantiate in the `retail` and `airline` domains.

State space $\mathcal{S}$: The hidden state is factored into $\mathcal{S} = \mathcal{S}_{\text{db}} \otimes \mathcal{S}_{\text{user}}$ where $\mathcal{S}_{\text{db}}$ is a snapshot of the underlying database (orders, flights, balances *etc.*) and $\mathcal{S}_{\text{user}}$ stores the latent user context (identity, revealed preferences, dialogue progress).

Action space $\mathcal{A}$: The agent can either (i) invoke an API tool that queries or mutates the database ($\mathcal{A}_{\text{db}}$) or (ii) send a free-form respond message to the user ($\mathcal{A}_{\text{user}}$). Thus $\mathcal{A} = \mathcal{A}_{\text{db}} \cup \mathcal{A}_{\text{user}}$.

Observation space $\mathcal{O}$: After each action the environment returns either a JSON payload/error from the database ($\mathcal{O}_{\text{db}}$) or the next user utterance produced by an LLM simulator ($\mathcal{O}_{\text{user}}$), yielding $\mathcal{O} = \mathcal{O}_{\text{db}} \cup \mathcal{O}_{\text{user}}$.

Transition function $\mathcal{T}$: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{O}$ is deterministic for database tools (state is updated, observation is the tool output) and stochastic for *respond*, which calls the user simulator to sample the next utterance and potentially reveal more of the instruction.

Reward function $\mathcal{R}$: At dialogue termination we compare the execution log to a gold reference: (1) hashes of mutable tables must match, (2) all mandatory natural-language outputs must appear in the agent's responses. If both hold, $\mathcal{R} = 1$, otherwise 0.

Instruction space $\mathcal{U}$: Each task provides a fixed natural-language instruction $u \in \mathcal{U}$ describing the user goal, persona and constraints. The user simulator may disclose u incrementally; therefore the agent must act under partial observability.

This causal decomposition lets us pinpoint failure modes such as wrong tool arguments (action-level), policy violations (transition-level), or hallucinated user messages (observation-level).

B Pass^k Results

B.1 Airline results

Tables 2-4 refer to pass^k results of the baselines and our implemented methods. As explained in §6.2, IRMA performs better when there are no ground-truth or user instruction errors. All results are obtained using GPT-4o as the LLM in the agent frameworks.

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.396	0.2779	0.2279	0.200	0.180
IRMA	**0.452**	**0.3680**	**0.3280**	**0.308**	**0.300**
FC	0.424	0.3120	0.2660	0.232	0.200
Self-reflection	0.448	0.3140	0.2560	0.224	0.200

Table 2: Results on all Airline tasks.

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.4941	0.3735	0.3206	0.2882	0.2647
IRMA	**0.5706**	**0.4912**	**0.4471**	**0.4235**	**0.4118**
FC	0.5529	0.4353	0.3794	0.3353	0.2941
Self reflection	0.5167	0.3750	0.3139	0.2778	0.2500

Table 3: Results of different methods on all Airline tasks, excluding the tasks with ground-truth errors.

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.5226	0.4065	0.3516	0.3161	0.2903
IRMA	**0.6258**	**0.5387**	**0.4903**	**0.4645**	**0.4516**
FC	0.6000	0.4774	0.4161	0.3677	0.3226
Self reflection	0.5556	0.4146	0.3528	0.3111	0.2778

Table 4: Results of different methods on all Airline tasks, excluding the tasks with ground-truth errors and user instruction errors.

B.2 Retail results

Tables 5-7 represent the results of the baseline and our implemented methods in the Retail domain.

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.5182	0.3704	0.2999	0.2573	0.2260
IRMA	0.5826	**0.4783**	**0.4261**	**0.3948**	**0.3739**
FC	**0.6052**	0.4522	0.3643	0.3043	0.2609
Self-reflection	0.5113	0.3809	0.3017	0.2383	0.1826

Table 5: Results of different methods on all Retail tasks.

C Domain Policies

Figures 11 and 12 are the domain policies present for the retail and airline domains in the τ-bench. These rules are injected verbatim as the system prompt to every tool-calling agent. An agent that

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.5304	0.3804	0.3080	0.2643	0.2321
IRMA	0.5982	0.4911	0.4375	0.4054	0.3839
FC	0.6164	0.4616	0.3732	0.3125	0.2679
self-reflection	0.5250	0.3911	0.3098	0.2446	0.1875

Table 6: Results of different methods on all Retail tasks, excluding the tasks with ground-truth errors.

Method	Pass^1	Pass^2	Pass^3	Pass^4	Pass^5
ReAct	0.5562	0.4048	0.3200	0.2818	0.2476
ours	0.6248	0.5171	0.4629	0.4305	0.4095
FC	0.6381	0.4838	0.3933	0.3314	0.2857
self reflection	0.5562	0.4171	0.3305	0.2610	0.2000

Table 7: Results of different methods on all Retail tasks, excluding the tasks with ground-truth errors and user instruction errors.

violates any of them—even if it successfully fulfills the user's request—receives zero reward, so strict compliance is essential. The Tool-Calling Agent has to strictly operate under the constraints of these policies to correctly solve user requests.

D Failure Example

Figures 7 and 8 show an example of errors occurring in the conversational trajectories simulating task 19 (retail) of the user-agent interactions as enumerated in subsections of §4. Error 1 in Figure 7 shows an example of 'User Instruction Hallucination' occurring in the very first user turn. Error 2 in Figure 8 shows an example of 'Domain Policy Violation' error. The user instruction for Task 19 is provided in Figure 6. This 'instruction' represents the original user instruction provided to the LLM-simulated user. It is the 'script' the user has to follow to provide requests to the agent.

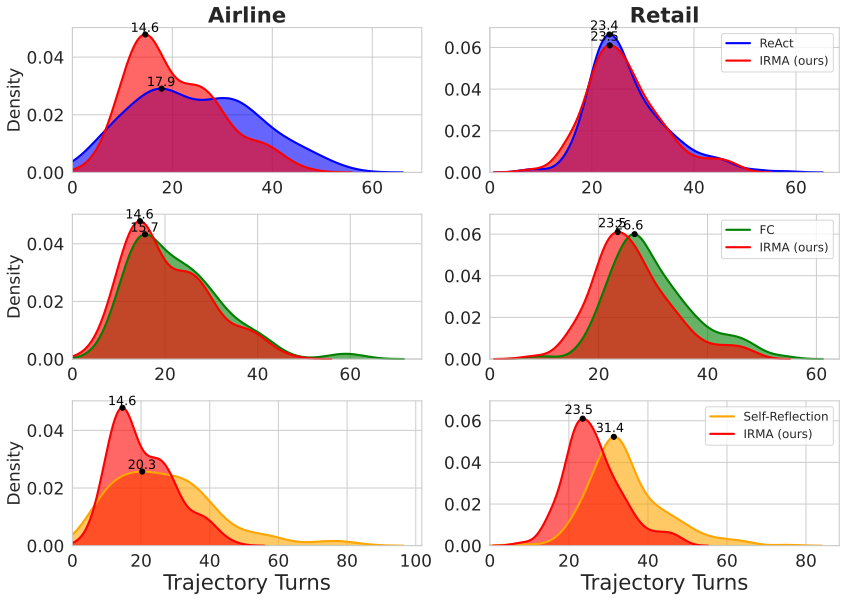

Figure 4: Comparison of IRMA and other methods based on the number of turns in successful tasks in the Airline and Retail domains.

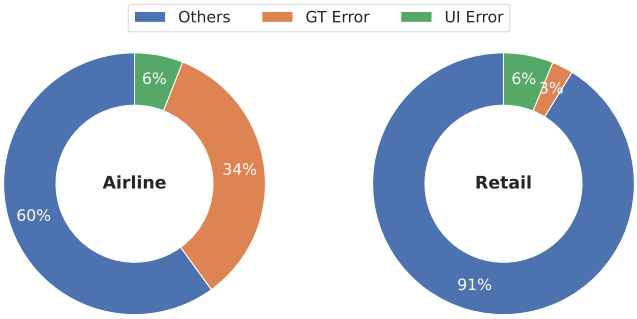

Figure 5: Error statistics across Airline and Retail tasks. **GT:** Ground Truth errors; **UI:** User Instruction errors.

E Input Reformulation Multi Agent framework

E.1 Follow-up question ACTing (FACT) Agent

The primary difference between FACT and other prompting techniques lies in the instruction section of the system prompt (refer to Figure 9).

F Self-Reflection Framework

To check the effectiveness of self-reflection as an alternative against the baselines and IRMA, we implement a multi-agent LLM self-reflection pipeline, consisting of a retriever LLM agent and a verifier LLM agent. Contrary to input reformulation, where the prompt provided in the user query is reformulated, the self-reflection agent pauses the tool-calling LLM agent before the execution environment executes the tool-call. All of the previous user queries are provided as input to the retriever agent to extract the relevant domain policy rules based on the user intent reflected from the user requests in the conversation. The retrieved rules are provided to the verifier agent along with the tool-calling agent's planned tool call. The verifier agent then verifies whether the tool-call is correct by providing a reflective justification based on determining whether any domain rule has been violated or not. The overall pipeline of the self-reflection agent is provided in Figure 10. The reflective feedback loop from verifier is set to be a one-time loop as the execution of the loop is very latency-heavy and invoking it multiple times might not be ideal in real-world customer-agent scenarios.

Method - IRMA Ablations (↓)	**Pass^1**	**Pass^2**	**Pass^3**	**Pass^4**	**Pass^5**
Memory only	0.416	0.27	0.212	0.18	0.16
Constraint only	0.416	0.276	0.206	0.164	0.14
Tool Suggestion only	0.424	0.268	0.19	0.14	0.1
Memory + Constraint	0.428	*0.31*	*0.26*	*0.236*	*0.22*
Memory + Tool Suggestion	*0.448*	0.294	0.214	0.16	0.12
Constraint + Tool Suggestion	0.38	0.264	0.212	0.18	0.16
Memory + Constraints + Tool Suggestion (All components)	**0.452**	**0.368**	**0.328**	**0.308**	**0.3**

Table 8: Results of IRMA component ablations on airline tasks. **Bold** scores represent the best scores. *Italic* scores represent the second-best scores. GPT-4o is used as the backbone LLM for all the sub-agents.

G Ablation Study on IRMA

We ablate the three IRMA modules—Memory (M), Constraint (C), and Tool (T)—and evaluate them on the airline subset. Across all Pass^k metrics, the full configuration (M+C+T) achieves the best performance, indicating strong complementarity among modules. Among the ablations, M+C is

consistently the strongest, ranking second overall in terms of better reliability at higher values of k. This pattern suggests that instruction retention (M) and policy/constraint adherence (C) account for most gains in long-horizon reasoning and plan stability, while tool disambiguation (T) provides the additional performance improvement needed to reach state-of-the-art performance. In sum, each module targets a distinct failure mode—carryover of instructions (M), rule compliance (C), and tool selection/parameterization (T)—but their integration is necessary for robust behavior in dynamic tool-use settings.

G.1 Constraints Extractor Agent

Constraints Extractor System Prompt

You are a **Constraint Filtering Agent**. Your task is to extract a **checklist of domain policy constraints** that are **relevant and necessary** to fulfill a user's query, based on operations policy.

Rules:
1. Include only constraints that are **directly triggered** by the user's intent and required to process or plan their request.
2. **Do not include** constraints that:
* Are conditional on user actions not yet taken (e.g., no action requested, no reservation referenced).
* Involve future steps like confirmation or verification, unless the user request makes them necessary.
3. Each constraint must be listed as a **checklist item** in clear, concise terms.
4. **If a constraint includes any value-based condition or limitation** (e.g., specific dollar amount, multiplier, eligibility thresholds), **you must extract and include that value**.
5. Output should follow this structure:

<constraints>
1. [Constraint 1: include any relevant thresholds, values, or limits]
2. [Constraint 2: include specific roles, benefits, or values]
...
</constraints>

* Do not include explanations, reasoning, or commentary.
* Return only the '<constraints>' block.

Constraints Extractor User Prompt

Given the following **user request** and **domain policy constraints**, extract only the **relevant and necessary constraints** that must be satisfied to process the user's intent.

* **DO NOT** list all constraints.
* **DO** include only those constraints that are directly required to fulfill the user's current request. * **If a constraint includes a specific value or condition (e.g., dollar amounts, eligibility rules, thresholds), it must be included in the output.**
* **If no constraint needs to be satisfied, return only 'None' inside the '<constraints>' block.**
* Present the output as a numbered checklist inside a '<constraints>' block.
* Do not add any explanation or extra text.

Input Format:
Constraints: CONS
User Query: $\{USER\_QUERY\}$
Output Format:
<constraints> 1. [First relevant constraint with any required values] 2. [Second relevant constraint...] </constraints>
If nothing applies:
<constraints> None </constraints>

G.2 Tool Selector Agent

Tool Selector System Prompt

"You are a smart AI agent that selects the most relevant tools from a predefined list, based on a user query. Each tool in the list has a name and a description of its capabilities.

Your Task:

Given a user query and a list of available tools, analyze the query and select only the tools that are relevant for fulfilling the user's request.

Guidelines:

1. **Understand the User Query** Carefully analyze the user query to identify:

* The main intent (e.g., search, update, calculate, retrieve). * The specific domain, data, or functionality being asked for. * Any implicit goals or constraints.

2. **Evaluate Tool Relevance** For each tool in the list, assess:

* Whether the tool can directly help address the query. * Whether using the tool would move the system closer to satisfying the user's intent. * Ignore tools that are only tangentially or theoretically related.

3. **Be Precise**

* Do not select tools based on vague or partial overlap. * Select **only those tools** that are likely to be **functionally useful** to respond to the query.

4. **Output Format** Return a list of selected tool names. If no tools are relevant, return an empty list.

Constraints:

* Do not call or execute any tool. * Generate one line explanations to show the reason of suggestiong the tool. * Generate a list of tools not only one. * Do not make assumptions beyond what is implied in the query or tool descriptions.

Tool Selector User Prompt

Available Tools:tool_list
User Query: user_query

```
{
  "instruction": "You are Mei Davis in 80217. You want to return the water bottle, and exchange the pet bed and office chair to the cheapest version. Mention the two things together. If you can only do one of the two things, you prefer to do whatever saves you most money, but you want to know the money you can save in both ways. You are in debt and sad today, but very brief.",
      "outputs": [
        "54.04",
        "41.64"
      ]
      "actions": [
        {
          "name": "find_user_id_by_name_zip",
          "kwargs": {
            "first_name": "Mei",
            "last_name": "Davis",
            "zip": "80217"
          }
        },
        {
          "name": "get_user_details",
          "kwargs": {
            "user_id": "mei_davis_8935"
          }
        },
        {
          "name": "get_order_details",
          "kwargs": {
            "order_id": "#W2890441"
          }
        },
        {
          "name": "get_order_details",
          "kwargs": {
            "order_id": "#W1267569"
          }
        },
        {
          "name": "get_product_details",
          "kwargs": {
            "product_id": "2747247837"
          }
        },
        {
          "name": "get_product_details",
          "kwargs": {
            "product_id": "4794339885"
          }
        },
        {
          "name": "return_delivered_order_items",
          "kwargs": {
            "order_id": "#W2890441",
            "item_ids": [
              "2366567022"
            ],
            "payment_method_id": "credit_card_1061405"
          }
        }
      ]
```


Figure 6: User instruction and Ground-Truth Action Sequence of Task 19 (Retail) in τ-bench.

START OF THE TRAJECTORY:

USER: Hi. I want to return a water bottle and also exchange my pet bed and office chair for cheaper versions. Can you help me with that? # Error 1: User Instruction Hallucination: User outputs 'cheaper' instead of 'cheapest', which is actually mentioned in the user instruction.

ASSISTANT: I can assist you with returning the water bottle and exchanging the pet bed and office chair. First, I need to authenticate your identity. Could you please provide your email address or your name and zip code?

USER: I'm Mei Davis, and my zip code is 80217.

ASSISTANT: Function(arguments='{"first_name":"Mei","last_name":"Davis","zip":"80217"}', name='find_user_id_by_name_zip')

TOOL: mei_davis_8935

ASSISTANT: Function(arguments='{"user_id":"mei_davis_8935"}', name='get_user_details')

TOOL: API output: {"name": {"first_name": "Mei", "last_name": "Davis"}, "address": {"address1": "698 Maple Drive", "address2": "Suite 465", "city": "Denver", "country": "USA", "state": "CO", "zip": "80217"}, "email": "mei.davis6811@example.com", "payment_methods": {"credit_card_1061405": {"source": "credit_card", "brand": "mastercard", "last_four": "1037", "id": "credit_card_1061405"}}, "orders": ["#W2890441", "#W1267569"]}

ASSISTANT: Thank you, Mei. I have located your account. Let's proceed with your request. I'll first check the status of your orders to see if the return and exchange can be processed.

Function(arguments='{"order_id": "#W2890441"}', name='get_order_details')

TOOL: API output: {"order_id": "#W2890441", "user_id": "mei_davis_8935", "address": {"address1": "698 Maple Drive", "address2": "Suite 465", "city": "Denver", "country": "USA", "state": "CO", "zip": "80217"}, "items": [{"name": "Water Bottle", "product_id": "8310926033", "item_id": "2366567022", "price": 54.04, "options": {"capacity": "1000ml", "material": "stainless steel", "color": "blue"}}, {"name": "Pet Bed", "product_id": "2747247837", "item_id": "2751999929", "price": 195.11, "options": {"size": "large", "material": "memory foam", "color": "grey"}}, {"name": "Office Chair", "product_id": "4794339885", "item_id": "8069050545", "price": 499.28, "options": {"material": "leather", "color": "blue", "armrest": "none", "backrest height": "high-back"}}, {"name": "Skateboard", "product_id": "1968349452", "item_id": "3877188862", "price": 182.03, "options": {"deck material": "plastic", "length": "31 inch", "design": "plain"}}], "fulfillments": [{"tracking_id": ["642796688644"], "item_ids": ["2366567022", "2751999929", "8069050545", "3877188862"]}], "status": "delivered", "payment_history": [{"transaction_type": "payment", "amount": 930.46, "payment_method_id": "credit_card_1061405"}]}

CONTINUED...

Figure 7: Part 1 of the conversation trajectory simulation of Task 19 (Retail).

PART 2 OF THE TRAJECTORY:

ASSISTANT: Your order #W2890441 is delivered and includes the following items:
1. **Water Bottle** - Item ID: 2366567022
2. **Pet Bed** - Item ID: 2751999929
3. **Office Chair** - Item ID: 8069050545
4. **Skateboard** - Item ID: 3877188862
You can return the water bottle and exchange the pet bed and office chair. Please confirm the following details:
- **Return**: Water Bottle
- **Exchange**:
  - Pet Bed for a cheaper version
  - Office Chair for a cheaper version

Please also confirm the payment method for any price differences or refunds. You have a Mastercard ending in 1037 on file. Would you like to use this for the transactions?

USER: Yes, please use the Mastercard ending in 1037. Also, could you let me know how much I can save by exchanging each item to the cheaper versions?

ASSISTANT: Function(arguments='{"product_id": "2747247837"}', name='get_product_details')

TOOL: API output: {"name": "Pet Bed", "product_id": "2747247837", "variants": {...}, "price": 195.11}

ASSISTANT: Function(arguments='{"product_id":"4794339885"}', name='get_product_details')

TOOL: API output: {"name": "Office Chair", "product_id": "4794339885", "variants": {...}, "price": 499.28}

ASSISTANT: Here are the available cheaper options for the items you want to exchange:

Pet Bed
- **Current Price**: \$195.11
- **Cheaper Option**: Large, Memory Foam, Beige - \$180.93
- **Savings**: \$14.18

Office Chair
- **Current Price**: \$499.28
- **Cheaper Option**: Leather, Blue, Standard Backrest - \$471.82
- **Savings**: \$27.46

Would you like to proceed with these exchanges and the return of the water bottle? Please confirm, and I'll process the transactions using your Mastercard ending in 1037.

USER: Yes, let's proceed with both exchanges and the return.

ASSISTANT: Function(arguments='{"order_id": "#W2890441", "item_ids": ["2366567022"], "payment_method_id": "credit_card_1061405"}', name='return_delivered_order_items')

TOOL: API output: { ... "return_items": ["2366567022"], "return_payment_method_id": "credit_card_1061405" }

ASSISTANT: Function(arguments='{"order_id":"#W2890441","item_ids":["2751999929","8069050545"],"new_item_ids": ["6942241102","4168944673"],"payment_method_id":"credit_card_1061405"}', name='exchange_delivered_order_items')
#Error 2- Assistant violates a domain rule by trying to exchange an order which is no more in 'delivered status'

TOOL: Error: non-delivered order cannot be exchanged

ASSISTANT: It seems there was an error processing the exchange for the pet bed and office chair. The system indicates that the order cannot be exchanged, possibly due to a status issue.

I have successfully processed the return for the water bottle, and you will receive an email with instructions on how to return it.

For the exchange issue, I recommend contacting our human support team for further assistance. Would you like me to transfer you to a human agent?

USER: No, that's okay. Thank you for your help. ###STOP###

Figure 8: Part 2 of the conversation trajectory simulation of Task 19 (Retail).

Instruction
You operate in a dynamic environment and must resolve the user's request through **informed reasoning**, **tool usage**, and **strategic follow-up questions**. Your goal is to progress the task without making assumptions, violating constraints, or misusing incomplete data.
—
Special Input Tags

* '<memory>': Contains the **user's conversation history**. Respond to the **most recent query**, but consider the **entire context** when reasoning or making decisions.
* '<constraints>': Lists **required policy conditions**.

You must:
* **Validate** whether constraints are met.
Ask follow-up questions if information is missing or unclear.

Only **call tools** if all relevant constraints are satisfied with valid data.
* '<tool_suggested>': Suggests specific tools.
You must only choose from this list, and only when appropriate for the task.
—
Additional Guidelines

Placeholder Handling
If the user provides **placeholder or generic
values** (e.g., "XXX", "[name]", "123456", "some location"):
* **Do not** call any tool using these. *
Call it out in your Thought.
Ask a follow-up question to collect the required specific value.

Data Conflict Handling
If a tool returns **factual data** (e.g., from a reservation system, database, or API) that **conflicts with what the user claims** (e.g., number of passengers, destination, price):
Acknowledge the discrepancy in your Thought.
Clarify the conflict to the user in your response.
Avoid proceeding with incorrect assumptions — instead, confirm or resolve the difference with a follow-up.

Example:
> The user requested a refund for 3 passengers, but the reservation data only includes 2. The agent must highlight this and seek clarification.
—
Output Format
Always respond using this structure:
"' Thought: <Explain how you processed the inputs, constraints, memory, placeholders, data
conflicts, and why you're taking your next action.>

Action: "name": <Tool or response action>, "arguments": <Valid JSON with real data only> "'
* **Do not** use dummy data. * **Do not** skip validation steps. * **Always prioritize clarity, constraint compliance, and correctness.**
—

(a) Part 1 of the FACT system prompt

Figure 9: FACT System prompt.

Example

User Input:
“I want a refund for 3 passengers.”

Tool Output:
Reservation data: 2 passengers only.

```
Thought:
The user requested a refund for 3 passengers, but the tool shows only 2 on the reservation. I need to clarify this mismatch before proceeding.
Action:
{{"name": "{RESPOND\_ACTION\_NAME}", "arguments": {{"{RESPOND\_ACTION\_FIELD\_NAME}": "Your reservation currently includes 2 passengers. Can you confirm if there's another booking or if the count was mistaken?"}}}}
```

Summary of Priorities

1. **Use memory** to understand full context.
2. **Respect constraints** and gather info to fulfill them.
3. **Use tools only with valid, real data** (not placeholders).
4. **Identify and handle data conflicts** between tools and user input.
5. **Ask follow-up questions** when necessary — this is preferred over assumptions or invalid actions.

Stay helpful, policy-compliant, and driven by accuracy, context, and user success.

(b) Part 2 of the FACT system prompt

Figure 9: FACT System prompt.

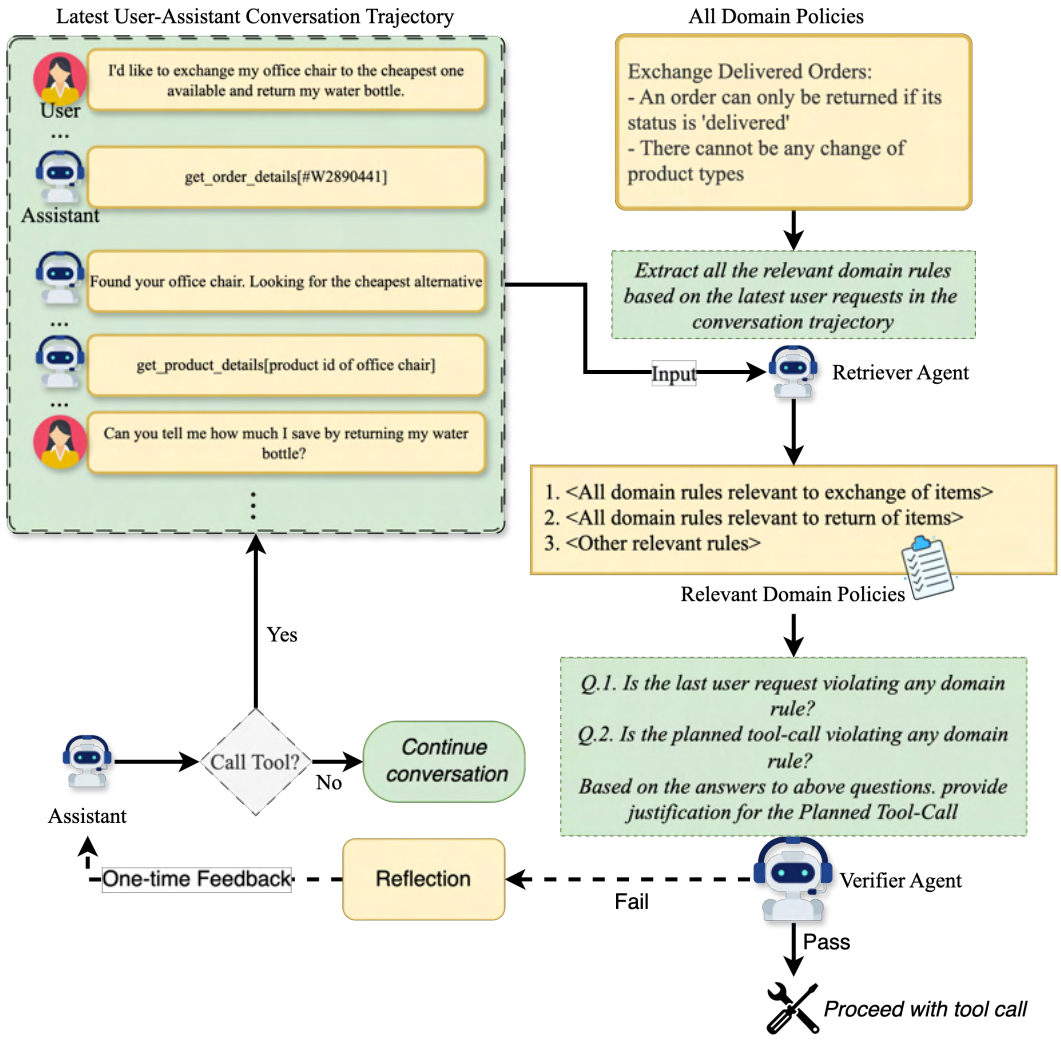

Figure 10: Overview of the pipeline showcasing the working of the self-reflection framework. The italicized text inside dotted green dotted text boxes refer to prompt gists provide to the Retriever and Verifier LLM Agent. The self-reflection only activates when the assistant generates the tokens to invoke a tool call.

Retail agent policy
As a retail agent, you can help users cancel or modify pending orders, return or exchange delivered orders, modify their default user address, or provide information about their own profile, orders, and related products.

- At the beginning of the conversation, you have to authenticate the user identity by locating their user id via email, or via name + zip code. This has to be done even when the user already provides the user id.

- Once the user has been authenticated, you can provide the user with information about order, product, profile information, e.g. help the user look up order i

- You can only help one user per conversation (but you can handle multiple requests from the same user), and must deny any requests for tasks related to any other user.

- Before taking consequential actions that update the database (cancel, modify, return, exchange), you have to list the action detail and obtain explicit user confirmation (yes) to proceed.

- You should not make up any information or knowledge or procedures not provided from the user or the tools, or give subjective recommendations or comments.

- You should at most make one tool call at a time, and if you take a tool call, you should not respond to the user at the same time. If you respond to the user, you should not make a tool call.

- You should transfer the user to a human agent if and only if the request cannot be handled within the scope of your actions.

Domain basics:
- All times in the database are EST and 24 hour based. For example \"02:30:00\" means 2:30 AM EST.

- Each user has a profile of its email, default address, user id, and payment methods. Each payment method is either a gift card, a paypal account, or a credit card.

- Our retail store has 50 types of products. For each type of product, there are variant items of different options. For example, for a 't shirt' product, there could be an item with option 'color blue size M', and another item with option 'color red size L'.

- Each product has an unique product id, and each item has an unique item id. They have no relations and should not be confused.

- Each order can be in status 'pending', 'processed', 'delivered', or 'cancelled'. Generally, you can only take action on pending or delivered orders.

- Exchange or modify order tools can only be called once. Be sure that all items to be changed are collected into a list before making the tool call!!!

Cancel pending order

- An order can only be cancelled if its status is 'pending', and you should check its status before taking the action.

- The user needs to confirm the order id and the reason (either 'no longer needed' or 'ordered by mistake') for cancellation.

- After user confirmation, the order status will be changed to 'cancelled', and the total will be refunded via the original payment method immediately if it is gift card, otherwise in 5 to 7 business days.

(a) Part 1 of the Retail Domain Rules

Figure 11: Domain Policies of the Retail Domain

Modify pending order

- An order can only be modified if its status is 'pending', and you should check its status before taking the action.

- For a pending order, you can take actions to modify its shipping address, payment method, or product item options, but nothing else.

Modify payment

- The user can only choose a single payment method different from the original payment method.

- If the user wants the modify the payment method to gift card, it must have enough balance to cover the total amount.

- After user confirmation, the order status will be kept 'pending'. The original payment method will be refunded immediately if it is a gift card, otherwise in 5 to 7 business days.

Modify items

- This action can only be called once, and will change the order status to 'pending (items modifed)', and the agent will not be able to modify or cancel the order anymore. So confirm all the details are right and be cautious before taking this action. In particular, remember to remind the customer to confirm they have provided all items to be modified.

- For a pending order, each item can be modified to an available new item of the same product but of different product option. There cannot be any change of product types, e.g. modify shirt to shoe.

- The user must provide a payment method to pay or receive refund of the price difference. If the user provides a gift card, it must have enough balance to cover the price difference.

- If the user wants to exchange a 'pending' item then it can be modified instead of being exchanged.

Return delivered order

- An order can only be returned if its status is 'delivered', and you should check its status before taking the action.

- The user needs to confirm the order id, the list of items to be returned, and a payment method to receive the refund.

- The refund must either go to the original payment method, or an existing gift card.

- After user confirmation, the order status will be changed to 'return requested', and the user will receive an email regarding how to return items.

Exchange delivered order

- An order can only be exchanged if its status is 'delivered', and you should check its status before taking the action. In particular, remember to remind the customer to confirm they have provided all items to be exchanged.

(b) Domain Policies of the Retail Domain

Figure 11: Domain Policies of the Retail Domain

Airline Agent Policy

The current time is 2024-05-15 15:00:00 EST.

As an airline agent, you can help users book, modify, or cancel flight reservations.

- Before taking any actions that update the booking database (booking, modifying flights, editing baggage, upgrading cabin class, or updating passenger information), you must list the action details and obtain explicit user confirmation (yes) to proceed.

- You should not provide any information, knowledge, or procedures not provided by the user or available tools, or give subjective recommendations or comments.

- You should only make one tool call at a time, and if you make a tool call, you should not respond to the user simultaneously. If you respond to the user, you should not make a tool call at the same time.

- You should deny user requests that are against this policy.

- You should transfer the user to a human agent if and only if the request cannot be handled within the scope of your actions.

Domain Basic

- Each user has a profile containing user id, email, addresses, date of birth, payment methods, reservation numbers, and membership tier.

- Each reservation has an reservation id, user id, trip type (one way, round trip), flights, passengers, payment methods, created time, baggages, and travel insurance information.

- Each flight has a flight number, an origin, destination, scheduled departure and arrival time (local time), and for each date:
  - If the status is "available", the flight has not taken off, available seats and prices are listed.
  - If the status is "delayed" or "on time", the flight has not taken off, cannot be booked.
  - If the status is "flying", the flight has taken off but not landed, cannot be booked.

Book flight

- The agent must first obtain the user id, then ask for the trip type, origin, destination.

- Passengers: Each reservation can have at most five passengers. The agent needs to collect the first name, last name, and date of birth for each passenger. All passengers must fly the same flights in the same cabin.

- Payment: each reservation can use at most one travel certificate, at most one credit card, and at most three gift cards. The remaining amount of a travel certificate is not refundable. All payment methods must already be in user profile for safety reasons.

- Checked bag allowance: If the booking user is a regular member, 0 free checked bag for each basic economy passenger, 1 free checked bag for each economy passenger, and 2 free checked bags for each business passenger. If the booking user is a silver member, 1 free checked bag for each basic economy passenger, 2 free checked bag for each economy passenger, and 3 free checked bags for each business passenger. If the booking user is a gold member, 2 free checked bag for each basic economy passenger, 3 free checked bag for each economy passenger, and 3 free checked bags for each business passenger. Each extra baggage is 50 dollars.

- Travel insurance: the agent should ask if the user wants to buy the travel insurance, which is 30 dollars per passenger and enables full refund if the user needs to cancel the flight given health or weather reasons.

(a) Part 1 of the Airline Domain Rules

Figure 12: Domain Policies of the Airline Domain.

Modify flight

- The agent must first obtain the user id and the reservation id.

- Change flights: Basic economy flights cannot be modified. Other reservations can be modified without changing the origin, destination, and trip type. Some flight segments can be kept, but their prices will not be updated based on the current price. The API does not check these for the agent, so the agent must make sure the rules apply before calling the API!

- Change cabin: all reservations, including basic economy, can change cabin without changing the flights. Cabin changes require the user to pay for the difference between their current cabin and the new cabin class. Cabin class must be the same across all the flights in the same reservation; changing cabin for just one flight segment is not possible.

- Change baggage and insurance: The user can add but not remove checked bags. The user cannot add insurance after initial booking.

- Change passengers: The user can modify passengers but cannot modify the number of passengers. This is something that even a human agent cannot assist with.

- Payment: If the flights are changed, the user needs to provide one gift card or credit card for payment or refund method. The agent should ask for the payment or refund method instead.

Cancel flight

- The agent must first obtain the user id, the reservation id, and the reason for cancellation (change of plan, airline cancelled flight, or other reasons)

- All reservations can be cancelled within 24 hours of booking, or if the airline cancelled the flight. Otherwise, basic economy or economy flights can be cancelled only if travel insurance is bought and the condition is met, and business flights can always be cancelled. The rules are strict regardless of the membership status. The API does not check these for the agent, so the agent must make sure the rules apply before calling the API!

- The agent can only cancel the whole trip that is not flown. If any of the segments are already used, the agent cannot help and transfer is needed.

- The refund will go to original payment methods in 5 to 7 business days.

Refund

- If the user is silver/gold member or has travel insurance or flies business, and complains about cancelled flights in a reservation, the agent can offer a certificate as a gesture after confirming the facts, with the amount being $100 times the number of passengers.

- If the user is silver/gold member or has travel insurance or flies business, and complains about delayed flights in a reservation and wants to change or cancel the reservation, the agent can offer a certificate as a gesture after confirming the facts and changing or cancelling the reservation, with the amount being $50 times the number of passengers.

(b) Part 2 of the Airline Domain Rules

Figure 12: Domain Policies of the Airline Domain