FlowAgent: Achieving Compliance and Flexibility for Workflow Agents

Anonymous ACL submission

Abstract

The integration of workflows with large language models (LLMs) enables LLM-based agents to execute predefined procedures, enhancing automation in real-world applications. Traditional *rule-based* methods tend to limit the inherent *flexibility* of LLMs, as their predefined execution paths restrict the models' action space, particularly when the unexpected, out-of-workflow (OOW) queries are encountered. Conversely, *prompt-based* methods allow LLMs to fully control the flow, which can lead to diminished enforcement of procedural *compliance*. To address these challenges, we introduce FlowAgent, a novel agent framework designed to maintain both compliance and flexibility. We propose the Procedure Description Language (PDL), which combines the adaptability of natural language with the precision of code to formulate workflows. Building on PDL, we develop a comprehensive framework that empowers LLMs to manage OOW queries effectively, while keeping the execution path under the supervision of a set of controllers. Additionally, we present a new evaluation methodology to rigorously assess an LLM agent's ability to handle OOW scenarios, going beyond routine flow compliance tested in existing benchmarks. Experiments on three datasets demonstrate that FlowAgent not only adheres to workflows but also effectively manages OOW queries, highlighting its dual strengths in compliance and flexibility. The code is available at `https://anonymous.4open.science/r/FlowAgent-DE68/`.

1 Introduction

With the enhanced understanding and reasoning capabilities of large language models (LLMs), pretrained LLMs are increasingly being utilized in dialogue systems (He et al., 2022; Bang et al., 2023). Compared with traditional chatbots, LLMs can interact more flexibly with users to address diverse needs, leveraging the vast amount of commonsense knowledge stored in their parameters (Yi et al., 2024). However, in real-world applications, we often expect chatbots to follow specific rules and procedures to perform certain tasks (e.g., guiding users to make an appointment for appropriate hospitals, departments, and doctors (Mosig et al., 2020; He et al., 2022)). The procedures that must be followed through dialogues are known as *workflows*. LLMs, acting as *workflow agents*, assist users via conversations and invoke relevant tools to fulfill requests (Xiao et al., 2024).

Existing research can be broadly classified into two categories: rule-based and prompt-based methods. **Rule-based** methods (Coze, 2024; Dify, 2024; Flowise, 2024) control the conversation between the agent and the user through deterministic programs, *modeling the progress of dialogue as state transitions within a graph composed of nodes representing different dialogue states*, as shown in the upper part of Fig. 1(a). In this approach, the LLM functions as a node within the graph and cannot control the entire conversation flow. As a result, this method provides high compliance but often at the expense of the LLM's inherent **flexibility**. As illustrated in the lower part of Fig. 1(a), introducing a new flexible feature within this system (e.g., allowing users to pause an appointment booking process to inquire about a condition before resuming) requires the addition of numerous transition egeds (dashed lines), significantly increasing complexity. In contrast, **prompt-based** methods leverage LLMs to autonomously manage dialogue by *representing workflows textually* (natural language, code or other structured data, Fig. 1(b)). While this method imparts soft control over LLM responses (workflow as part of prompt), LLMs' probabilistic nature often leads to **compliance** issues, like hallucinating incorrect information, which can have serious repercussions (e.g., notifying a user about a successful appointment booking when it hasn't occurred) (Zhang et al., 2023).

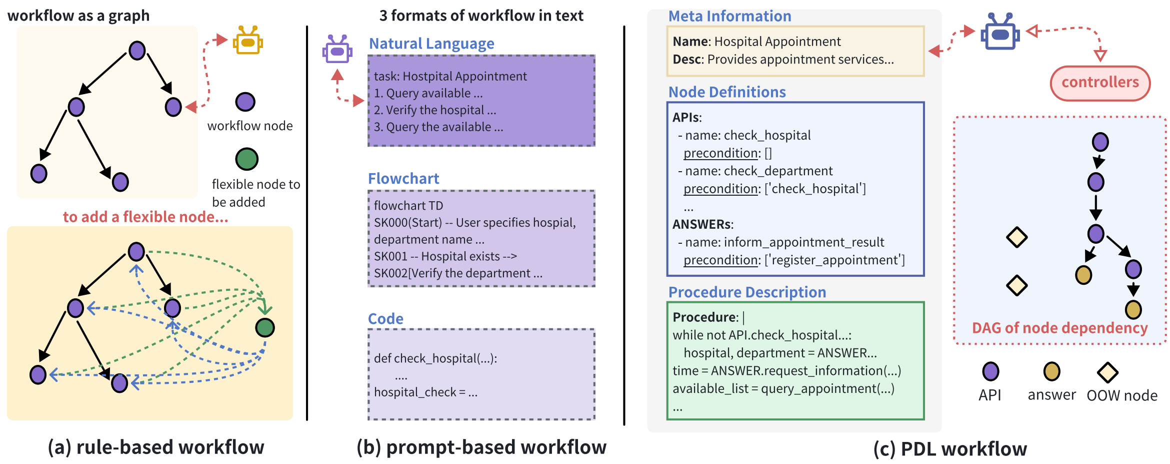


Figure 1: Comparison of different formats of workflow.

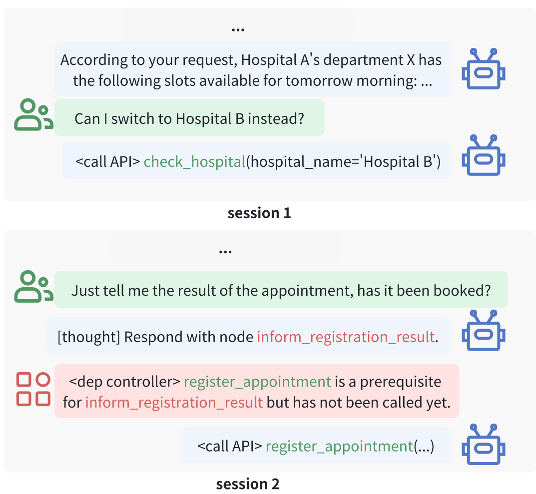


Figure 2: Two sample sessions of FLOWAGENT in the hospital appointment workflow.

This brings us to the critical question of our work: **How can we enhance LLM compliance with workflow tasks without diminishing their interaction flexibility?** This question arises from two primary challenges: 1) *How should we precisely represent workflows?* 2) *How can we effectively control LLM behavior?*

To address the first challenge, as shown in Fig. 1(c), we introduce *Procedure Description Language (PDL)*, which merges the fluidity of natural language with the precision of coding. PDL's flexible syntax allows for comprehensive node definitions, facilitating accurate workflow representations (see Sec. 4.1). To tackle the second challenge, we present the FLOWAGENT framework, which includes a set of controllers that manage agent behavior according to PDL-defined nodes. This system allows LLMs to make autonomous yet monitored and legally constrained decisions (see Sec. 4.2).

Fig. 2 illustrates two sessions in a hospital appointment setting. In session 1, when a user wishes to switch from Hospital A to Hospital B during the registration process, FLOWAGENT demonstrates *flexibility* by re-invoking the `check_hospital` API as per PDL directives. Conversely, in session 2, when the user prompts, "Just tell me the result of the appointment", the LLM might attempt to respond without executing the necessary booking API. However, the controllers in the FLOWAGENT framework prevent such an occurrence by ensuring that prerequisite conditions are met before informing the user of the booking result, highlighting the *compliance* offered by FLOWAGENT.

Our **contributions** are threefold:

1. We provide a systematic analysis of existing LLM-based workflow agents, focusing on compliance and flexibility. Based on this analysis, we propose the PDL syntax, combining natural language and code to flexibly describe node relationships and workflow procedures.
2. We introduce the FLOWAGENT framework, which aids in the execution of workflow agents. By crafting PDL-driven controllers, we dynamically balance compliance and flexibility. Experiments on three datasets demonstrate FLOWAGENT's balanced compliance and flexibility within and beyond pre-defined workflows.
3. We construct a comprehensive evaluation benchmark augmenting existing datasets to assess workflow agent performance in out-of-workflow (OOW)

scenarios.

2 Related Work

2.1 LLM-Driven Conversational Systems

The evolution of task-oriented dialogue (TOD) systems has transitioned from modular pipelines (Yi et al., 2024) to end-to-end LLM paradigms. While traditional systems suffered from error propagation across NLU, DST, and NLG modules (He et al., 2022; Su et al., 2021), modern approaches leverage LLMs for holistic dialogue management via workflow-guided interactions (Xiao et al., 2024; Wallace et al., 2024). This shift necessitates new evaluation metrics focusing on task success rates over modular accuracy (Arcadinho et al., 2024), motivating our framework's dual focus on procedural compliance and adaptive flexibility.

2.2 Agentic Workflow Architectures

The progression of LLMs has led to the development of LLM-based agents across various domains (Park et al., 2023; Tang et al., 2023; Qian et al., 2023). LLM-based agents enhance task execution through tool usage and dynamic planning (Yao et al., 2022; Schick et al., 2023; Wang et al., 2023; Zhu et al., 2024). We distinguish two paradigms: 1) *Workflow generation* creates procedures via LLM reasoning (Li et al., 2024; Xu et al., 2024; Liu et al., 2023; Chen et al., 2023; Valmeekam et al., 2022), and 2) *Workflow execution* operates within predefined structures (Xiao et al., 2024; Qiao et al., 2024).

Our research primarily focuses on the latter paradigm, treating workflows as predefined knowledge to build robust, user-centric agents. Within this context, two main approaches are adopted to integrate structured workflows with linear-text-processing language models: 1) *Rule-based Approach*: This method involves hard-coding workflow transition rules as fixed logic, defining the current node and state transitions explicitly in the program. 2) *Prompt-based Approach*: Here, workflows are represented in flexible formats such as natural language, code (or pseudocode), or flowchart syntax (Xiao et al., 2024; Zhu et al., 2024). Each method presents unique challenges: rule-based systems often lack flexibility, while prompt-based methods might deviate from intended procedures. Our solution aims to strike a balance between process control and adaptability, ensuring workflows are both structured and responsive to dynamic interactions.

3 Preliminary and Background

3.1 Workflow

A *workflow* defines a structured process designed to accomplish a specific task or goal within a particular scenario. For instance, in a hospital appointment booking scenario, a typical workflow involves steps such as querying the user for their preferred hospital, department, and time, retrieving available appointment slots using relevant tools, confirming the details with the user, and completing the booking. Formally, we can represent a workflow as a directed acyclic graph (DAG) denoted by $\mathcal{G}(\mathcal{V}, \mathcal{E})$ (Qiao et al., 2024; Zhang et al., 2024), where $\mathcal{V}$ represents the set of nodes, each corresponding to an atomic operation (e.g., querying the user, invoking an API, retrieving from a knowledge base), and $\mathcal{E}$ represents the directed edges that define the temporal and dependency relationships between these operations, effectively specifying the workflow's progression.

3.2 Workflow Agent

A *workflow agent* is designed to assist users in completing tasks by interacting with them and utilizing available tools. It can be conceptualized as an agent making sequential decisions within an environment composed of the user and the available tools. This interaction can be modeled as a Markov Decision Process (MDP), which provides a valuable framework for understanding the agent's decision-making process over time. In this framework, the agent transitions through a sequence of states (s), takes actions (a) based on the current state, and receives feedback (r) from the environment (user responses or tool-generated outputs). This process can be represented as $\{(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots, (s_{t-1}, a_{t-1}, r_{t-1})\}$. Consequently, the decision-making process of the workflow agent can be expressed as:

$$a_t \leftarrow \mathcal{A}(\mathcal{H}_{t-1}, \mathcal{G}), \tag{1}$$

where $\mathcal{H}_{t-1}$ encompasses all actions and observations up to time $t-1$, and $\mathcal{G}$ serves as the guide for the agent's actions.

Based on how the workflow is represented and integrated into the agent's decision-making process, workflow agents can be broadly classified into two categories: *rule-based agents* and *prompt-based agents*. Rule-based agents rely on explicitly programmed procedures to guide the workflow, while prompt-based agents utilize a single language model to autonomously manage the entire decision-making and dialogue process.

The first category, **rule-based agents**, implements the workflow procedure through explicit programming. Examples include Coze (Coze, 2024), Dify (Dify, 2024), and Flowise (Flowise, 2024). In these systems, the transitions between nodes are rigidly controlled by the program, with the LLM acting as a component within specific nodes to generate user responses, predict parameters for tool calls, or facilitate node transitions (e.g., classifying user intent). In this paradigm, the information accessible to the agent and its action space are limited, which can be expressed as:

$$a_t \leftarrow \mathcal{M}^v(\phi^v(\mathcal{H}_{t-1}), \psi^v(\mathcal{G})), \qquad (2)$$

where v is the current node, $\phi^v(\mathcal{H}_{t-1})$ is the selected information visible to v, $\psi^v(\mathcal{G})$ is a subgraph of $\mathcal{G}$ expanded from v, and $\mathcal{M}^v$ denotes the language model associated with node v.

The second category, **prompt-based agents** (Xiao et al., 2024; Zhu et al., 2024), represents the workflow as text $\mathcal{W}^{(f)}$ using a specific format f, and a single language model $\mathcal{M}$ autonomously manages the entire decision-making and dialogue process. This process can be represented as:

$$a_t \leftarrow \mathcal{M}(\mathcal{H}_{t-1}, \mathcal{G}^{(f)}), \qquad (3)$$

where $\mathcal{G}^{(f)}$ represents the graph structure implicitly encoded within $\mathcal{W}^{(f)}$.

4 Methodology

In this work, we introduce a novel procedural description language (PDL) designed to represent workflows, alongside FLOWAGENT, an execution framework that enhances the agent's behavioral control.

4.1 PDL Syntax

PDL consists of three primary components: 1) *Meta Information*: Basic workflow details such as name and description. 2) *Node Definitions*: Resources accessible to the agent, which include*API* nodes (for external tool calls) and *ANSWER* nodes (for user interaction). 2) *Procedure Description*: The procedural logic of the task, expressed in a mix of natural language and pseudocode.

For illustration, in the *Hospital Appointment* workflow, Fig.3 presents a segment of the *node definitions* [1]. Fig.4 illustrates a portion of the *procedure description*. Key features of PDL include: 1) *Precondition Specification*: Nodes include a *preconditions* attribute, defining dependencies between nodes. For example, `check_department` requires `check_hospital` as a prerequisite, ensuring hospital selection before department inquiry. 2) *Hybrid Representation*: The integration of natural language and code in the procedure description ensures a concise and yet flexible workflow representation, maintaining the clarity of NL with the accuracy of code.

```
APIs:
- name: check_hospital
    pre: []
- name: check_department
    pre: ['check_hospital']
- name: query_appointment
    pre: ['check_department']
- name: register_appointment
    pre: ['query_appointment']
- name: recommend_other_hospitals
    pre: ['register_appointment']

    ANSWERs:
- name: inform_appointment_result
    pre: ['register_appointment']
...
- name: answer_out_of_workflow_questions
- name: request_information
```

Figure 3: Example of Node Definations in PDL

4.2 FLOWAGENT Architecture

To enhance the compliance of workflow agents, we introduce FLOWAGENT, an execution framework tightly integrated with PDL. FLOWAGENT enforces a set of controllers that govern the agent's decision-making process, thereby promoting reliable action execution without sacrificing the LLM's autonomy.

Algorithm 1 outlines FLOWAGENT's overall execution. Each round begins with a user query (line 3), which the agent interprets to produce a response or a tool call (line 18), ultimately generating a user-facing response (line 21).

To ensure decision-making stability, FLOWAGENT incorporates two categories of controllers: *pre-decision* controllers ($\mathcal{C}_{\text{pre}} = \{c_i^{\text{pre}}\}_{i=1}^{C_{\text{pre}}}$) and *post-decision* controllers ($\mathcal{C}_{\text{post}} = \{c_j^{\text{post}}\}_{j=1}^{C_{\text{post}}}$). **Pre-decision controllers** proactively guide the agent's actions by evaluating the current state and providing feedback to the LLM (e.g., identifying unreachable nodes based on the dependency graph $\mathcal{G}^{(pdl)}$).

[1] For brevity, certain details have been omitted; see App.A.1 for the complete PDL specification.

	Datase	# Workflow	# Session	# Turn	# User Profile	# User Intentions	# OOW queries
session-level	SGD	26	442	11,594	390	1,593	811
	STAR	24	408	10,856	360	1,265	679
	In-house dataset	6	102	3,246	90	322	212
turn-level	SGD	26	338	5,016	-	834	496
	STAR	24	312	5,387	-	853	541
	In-house dataset	6	150	1,679	-	353	203

Table 1: Dataset Statistics

	SGD	ABCD	STAR	FLAP	FlowBench	In-house dataset
Workflow Format	-	NL	flowchart	NL	NL, code, flowchart	NL, code, flowchart, PDL
Multiple User Intentions	✔	✔	✔	✘	✔	✔
Incorporate User Persona	✘	✘	✘	✘	✔	✔
Automate dialogue construction	✘	✘	✘	✘	✔	✔
OOW Query Annotation	✘	✘	✘	✘	✘	✔

Table 2: Comparison of Contents Included in Different Datasets

```
while not API.check_hospital(hospital) or not API.check_department(hospital,
department):
    hospital, department = ANSWER.request_information('hospital', 'department')
time = ANSWER.request_information('time')
available_list = query_appointment(hospital, department, time)
try:
    # ... collect necessary information for registration
    result = API.register_appointment(hospital, ...)
    ANSWER.inform_appointment_result(result)
except:
    # if registration fails, recommend other hospitals
    available_list = API.recommend_other_hospitals(department, time)
    # ... try to register again
```

Figure 4: Example of Procedure Description in PDL

This feedback, denoted by $\mathcal{R}_{\text{pre}}$, serves as a form of *soft control*. However, LLMs can still generate unstable outputs even with pre-decision guidance. Therefore, **post-decision controllers** provide *hard constraints* by assessing the validity of proposed agent actions.

We designed modular controllers to adjust the behavior of the workflow agent across multiple dimensions, such as *enforcing node dependencies*, *constraining API call repetition*, and *limiting conversation length*. Below, using the workflow shown in Fig. 3 as an example, we briefly introduce the **node dependency controller**. It can operate in both pre- and post-decision modes. As a pre-decision controller ($c^{\text{pre}}_{\text{dep}}$), the controller analyzes the agent's current node and identifies inaccessible nodes by examining the dependency graph. For example, if the agent is at `check_hospital`, $c^{\text{pre}}_{\text{dep}}$ prevents the LLM from prematurely transitioning to `query_appointment` (soft control). As a post-decision controller ($c^{\text{post}}_{\text{dep}}$), the controller validates proposed node transitions. For instance, if the agent attempts to transition to `query_appointment` without completing `check_department`, the controller denies the request, providing feedback to the agent.

5 Evaluation and Data

5.1 Compliance Evaluation

We follow previous studies (Xiao et al., 2024; Chen et al., 2023) to conduct both turn-level and session-level assessments. In **turn-level evaluation**, there is a reference session (considered as ground truth) (Dai et al., 2022). For each turn in the reference session, the evaluation system provides the prefix of the session $\mathcal{H}_{t-1}$ to the bot for predicting the current $\hat{a}_t$. The judge compares $\hat{a}_t$ with a_t to determine if the bot's response for that turn is correct, and the average result across all turns yields the *Pass Rate*. To assess the agent's tool usage capability, for turns involving tool call-

Algorithm 1: FLOWAGENT Execution

Input: user $\mathcal{U}$, bot agent $\mathcal{A}^{(pdl)}$, system $\mathcal{S}$, workflow in PDL format $\mathcal{W}^{(pdl)}$, pre-decision controllers $\mathcal{C}_{\text{pre}} = \{c_i^{\text{pre}}\}_{i=1}^{C_{\text{pre}}}$, post-decision controllers $\mathcal{C}_{\text{post}} = \{c_j^{\text{post}}\}_{j=1}^{C_{\text{post}}}$, maximum attempts per turn $N_{\max}$
Output: conversation history $\mathcal{H}$

Initialize conversation history: $\mathcal{H} \leftarrow \emptyset$;
while *True* **do**
    $\mathcal{O}_\mathcal{U} \leftarrow \mathcal{U}(\mathcal{H})$;
    $\mathcal{H} \leftarrow \mathcal{H} \parallel \mathcal{O}_\mathcal{U}$;
    if $\mathcal{O}_\mathcal{U}$*.is_end* $=$ *True* **then**
        break ;
    for *turn_id* $\leftarrow 1$ ***to*** N_{max} **do**
        `// Traverse all pre-decision controllers`
        $\mathcal{R}_{\text{pre}} \leftarrow \emptyset$;
        foreach $c_i^{pre} \in \mathcal{C}_{pre}$ **do**
            $r_i \leftarrow c_i^{\text{pre}}.\text{process}(\mathcal{H}, \mathcal{W}^{(pdl)})$;
            $\mathcal{R}_{\text{pre}} \leftarrow \mathcal{R}_{\text{pre}} \parallel r_i$;
        $\mathcal{O}_\mathcal{A} \leftarrow \mathcal{A}^{(pdl)}(\mathcal{H}, \mathcal{W}^{(pdl)}, \mathcal{R}_{\text{pre}})$;
        `// Traverse all post-decision controllers`
        if_pass $\leftarrow$ True ;
        foreach $c_j^{post} \in \mathcal{C}_{post}$ **do**
            if $c_j^{post}.process(\mathcal{O}_\mathcal{A}) = False$ **then**
                if_pass $\leftarrow$ False ;
        if *if_pass* $=$ *True* **then**
            if $\mathcal{O}_\mathcal{A}$*.type* $=$ *tool_calling* **then**
                $\mathcal{O}_\mathcal{S} \leftarrow \mathcal{S}(\mathcal{O}_\mathcal{A})$;
                $\mathcal{H} \leftarrow \mathcal{H} \parallel \mathcal{O}_\mathcal{S}$;
            else if $\mathcal{O}_\mathcal{A}$*.type* $=$ *response_to_user* **then**
                $\mathcal{H} \leftarrow \mathcal{H} \parallel \mathcal{O}_\mathcal{A}$;
                break ;

ings, we evaluate the tool selection and parameter infilling performance of the agent in *Precision, Recall, and F1-score*.

For **session-level evaluation**, we simulate user interactions with the bot using an LLM, which serves to mimic real user behavior while minimizing human assessment costs. To ensure these simulated sessions accurately reflect real-world complexity, we define detailed user profiles comprising: (1) demographic information; (2) conversational style, capturing behavioral patterns; and (3) workflow-related user needs, detailing primary and secondary session objectives. An illustrative user profile is provided in App. A.2. For each generated session, we conduct a binary assessment to verify whether the user's primary workflow objectives are achieved, yielding the *Success Rate*. Additionally, by tracking the number of sub-tasks initiated and completed, we derive the *Task Progress* metric. Sessions are evaluated end-to-end using prompts consistent with those recommended by Xiao et al. (2024). Furthermore, we evaluate the LLM agent's performance in tool invocation with *Precision*, *Recall*, and *F1-score* metrics.

5.2 Flexibility Evaluation

Previous work (Zhong et al., 2018; Wu et al., 2019; Li et al., 2024) has primarily focused on evaluating whether bots can follow a specific procedure to complete a conversation, which partially emphasizes compliance while neglecting flexibility in handling user requests. Such incomprehensive evaluation may not reflect the capabilities of LLM agents under real-world scenarios, where an "imperfect" user might not adhere to the procedure and violates the sequential steps during multiple rounds of interactions. Consequently, to evaluate the performance of workflow agents in OOW scenarios, we have additionally developed a targeted evaluation method to assess flexibility.

Specifically, we categorize OOW scenarios into three types: (1) *intent switching*, where the user suddenly changes the original intent requests or requirements, including modification of API slots/-parameters and demand for cancellations; (2) *procedure jumping*, where the user does not follow the established workflow sequence to provide information and express confirmation, including skipping steps or jumping back; and (3) *irrelevant answering*, where the user deliberately avoids direct reply to questions raised by the agent, such as answers with topic shifts and rhetorical questions;

Based on these classifications, flexibility can be evaluated by examining the agent's performance in OOW scenarios using the metrics introduced in Sec. 5.1. At the turn-level, we insert OOW user interventions to assess the agent's immediate adaptive responses in these specific interactions. At the session-level, we assess the agent's overall performance in sessions that include OOW queries to measure its long-term flexibility.

5.3 Data

We constructed three test datasets based on existing datasets and business-related data: SGD (Rastogi et al., 2019), STAR (Mosig et al., 2020), and In-house. The data construction process is detailed in App. D.1. Statistics for these datasets are shown in Tab. 1, and differences from datasets used in other studies are highlighted in Tab. 2.

Specifically, our datasets include: (1) four types

Backbone Model	Method	In-house dataset			STAR			SGD		
		Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1
GPT-4o	ReAct$_{\text{NL}}$	62.50	80.33	63.16	40.17	78.33	76.96	**34.62**	82.44	**89.11**
	ReAct$_{\text{code}}$	57.26	75.20	75.86	38.27	75.10	55.32	29.23	76.67	82.32
	ReAct$_{\text{FC}}$	60.01	82.70	72.00	33.43	72.58	82.33	30.92	81.24	85.71
	FLOWAGENT	**67.72**	**85.12**	**80.60**	**42.78**	**80.42**	**84.00**	32.79	**84.21**	86.60
Qwen2-72B	ReAct$_{\text{NL}}$	40.51	80.01	78.90	16.67	59.34	82.12	13.46	67.94	84.42
	ReAct$_{\text{code}}$	32.78	65.58	75.20	10.42	56.70	63.63	15.76	59.84	72.55
	ReAct$_{\text{FC}}$	41.67	80.97	77.78	9.21	53.80	61.58	28.79	62.98	85.40
	FLOWAGENT	**44.32**	**82.22**	**84.21**	**18.42**	**61.42**	**86.86**	**30.84**	**69.91**	**88.02**

Table 3: Session-level Evaluation Results

Backbone Model	Method	In-house dataset			STAR			SGD		
		Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1
GPT-4o	ReAct$_{\text{NL}}$	18.03	72.20	75.42	4.55	43.59	81.58	3.31	49.42	74.12
	ReAct$_{\text{code}}$	16.23	57.27	73.68	2.08	40.74	70.21	2.92	54.23	64.57
	ReAct$_{\text{FC}}$	18.21	71.42	78.57	5.17	43.52	82.05	4.02	47.57	73.56
	FLOWAGENT	**32.01**	**75.20**	**81.57**	**10.21**	**52.31**	**85.32**	**7.16**	**56.64**	**77.83**
Qwen2-72B	ReAct$_{\text{NL}}$	16.76	69.41	72.27	6.25	48.30	82.92	5.01	47.00	82.83
	ReAct$_{\text{code}}$	0.00	60.41	71.62	2.02	45.31	70.80	2.08	45.35	70.79
	ReAct$_{\text{FC}}$	17.14	70.42	75.56	0.00	45.63	84.49	4.10	46.33	78.29
	FLOWAGENT	**30.20**	**75.70**	**80.01**	**8.72**	**50.28**	**86.72**	**8.25**	**49.30**	**89.88**

Table 4: Session-level Evaluation Results in OOW Scenarios

of workflows (see App. A); (2) user profiles required for session-level evaluation (see App. A.2); and (3) conversations needed for turn-level evaluation (see App. B.1).

6 Experiments

We raise the following research questions:
Q1: Compared with other models, does our proposed FLOWAGENT show improvements in compliance and flexibility?
Q2: In which way the proposed controllers exert constraints on the model to facilitate workflows with both compliance and flexibility?

6.1 Experimental Setup

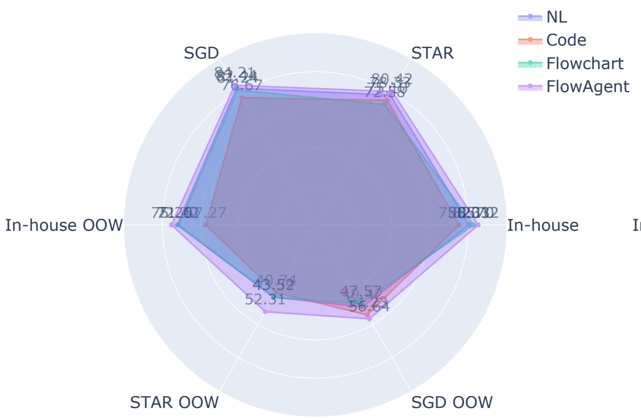
Baselines We selected ReAct (Yao et al., 2022) as a baseline method for comparison, which makes decisions in each round by utilizing a combination of *thought* and *action*, and treats the feedback from environment an *observation*. It belongs to the category of prompt-based methods introduced in Sec. 3.2. For representing the workflow, we chose three formats: natural language (NL), code, and FlowChart, denoted as ReAct$_{\text{NL}}$, ReAct$_{\text{code}}$, and ReAct$_{\text{FC}}$, respectively. To ensure a fair comparison, we reused the prompts from FlowBench (Xiao et al., 2024) in our experiments.

Implementation In session-level evaluation, `GPT-4o-mini` is used for user simulation. For the bot, we initially tested two representative model series, the GPT series (Achiam et al., 2023) and the Qwen series (Yang et al., 2024). Preliminary studies revealed that small models are not competent for complex workflow tasks. Therefore, in the present study, we choose `GPT-4o` and `Qwen2-72B` for demonstrations. During the evaluation process, we used `GPT-4-Turbo` for judgment. More implementation details can be seen in App. C.1.

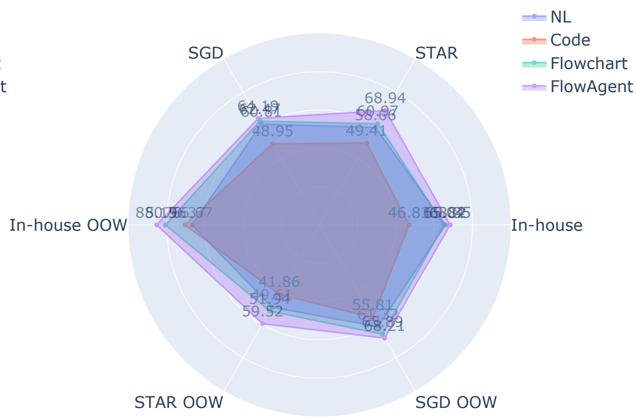
6.2 Session-level Experimental Results

A1.1: FLOWAGENT outperforms the other three baselines in terms of task compliance. We first compare the session-level performance of different methods in Tab. 3. The results indicate that FLOWAGENT outperforms the other three baselines in terms of task completion metrics *Success Rate*, *Task Progress*, and tool usage metrics like *Tool F1*.

A1.2: FLOWAGENT exhibits robustness towards OOW interventions with higher flexibility. Tab. 4 presents the performance of different



(a) *Task Progress* for GPT-4o in session-level evaluation.



(b) *Pass Rate* for Qwen2-72B in turn-level evaluation.

Figure 5: Visualization of the comparison of metrics for different models.

Method	In-house dataset			STAR			SGD		
	Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1	Success Rate	Task Progress	Tool F1
FLOWAGENT	**57.26**	**84.71**	76.13	**22.22**	70.44	**91.89**	**16.67**	**69.89**	**89.89**
-post	55.71	84.56	**76.70**	20.83	**72.57**	90.20	8.33	66.28	83.98
-post-pre	43.75	80.50	75.00	12.50	63.75	86.27	7.69	65.77	88.66

Table 5: Ablation Study Results

methods under OOW scenarios. A general performance decline is observed across all models on the three datasets. However, FLOWAGENT exhibits only a slight decline, achieving the best results across all datasets. Fig. 5(a) visualizes the *Task Progress* metric under different settings, highlighting FLOWAGENT's advantage in OOW scenarios, demonstrating strong flexibility.

6.3 Turn-level Experimental Results

A1.3: FLOWAGENT maintains the superior compliance and flexibility across datasets in turn-level evaluation. We present the turn-level experimental results of Qwen2-72B in Tab. 6. The results show that the FLOWAGENT framework achieves the best performance in both IW and OOW settings. What's more, Fig. 5(b) compares the *Success Rate* across different models and settings.

6.4 Ablation Studies

A2: Controllers play an indispensable role in enforcing steady progress of workflows with OOW interventions. We conducted ablation experiments on FLOWAGENT in OOW settings, with the results shown in Tab. 5. In the table, "-post" indicates the removal of the post-decision controllers $\mathcal{C}_{\text{post}}$ from the complete model, while "-post-pre" further removes the pre-decision controllers $\mathcal{C}_{\text{pre}}$. According to the experimental results, it is evident that removing either controller negatively impacts model performance, validating that controllers in FLOWAGENT enhance the model's compliance.

7 Conclusion

In this paper, we reviewed existing LLM-based workflow methods and compared their strengths and weaknesses in terms of compliance and flexibility. Aiming to enhance the compliance capability of LLMs without significantly compromising their flexibility, we proposed the PDL syntax to express workflows and used the FLOWAGENT framework to control agent behavior. For evaluating compliance and flexibility capabilities, we constructed datasets based on existing data and designed specific evaluation methods. Experiments on three datasets demonstrated that FLOWAGENT not only possesses strong compliance capabilities but also exhibits robust flexibility when handling out-of-workflow queries.

8 Limitations

We acknowledges two primary limitations:

Workflow Generation Our current research emphasizes enhancing LLM performance within manually constructed workflows using the PDL syntax. Consequently, the evaluation is limited to these artificially defined settings, lacking exploration of automated workflow generation (Qiao et al., 2024; Zhang et al., 2024). Future work should investigate dynamic workflow synthesis to adapt to varying and complex user demands without manual intervention.

Dialogue Diversity and Evaluation While this study evaluates agent performance in OOW scenarios using simulated user interactions, the real-world applicability relies on testing across a broader spectrum of authentic user demands.

References

OpenAI Josh Achiam, Steven Adler, and etc. 2023. Gpt-4 technical report. *ArXiv*.

Samuel Arcadinho, David Aparício, and Mariana Almeida. 2024. Automated test generation to evaluate tool-augmented llms as conversational ai agents. *ArXiv*, abs/2409.15934.

Namo Bang, Jeehyun Lee, and Myoung-Wan Koo. 2023. Task-optimized adapters for an end-to-end task-oriented dialogue system. *ArXiv*, abs/2305.02468.

Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2023. Autoagents: A framework for automatic agent generation. *ArXiv*, abs/2309.17288.

Coze. 2024. Coze platform. `https://www.coze.com`. Accessed: November 3, 2024.

Yinpei Dai, Wanwei He, Bowen Li, Yuchuan Wu, Zhen Cao, Zhongqi An, Jian Sun, and Yongbin Li. 2022. Cgodial: A large-scale benchmark for chinese goal-oriented dialog evaluation. In *Conference on Empirical Methods in Natural Language Processing*.

Dify. 2024. Dify repository. `https://github.com/langgenius/dify`. Accessed: November 3, 2024.

Flowise. 2024. Flowise repository. `https://github.com/FlowiseAI/Flowise`. Accessed: November 3, 2024.

Wanwei He, Yinpei Dai, Min Yang, Jian Sun, Fei Huang, Luo Si, and Yongbin Li. 2022. Unified dialog model pre-training for task-oriented dialog understanding and generation. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Autoflow: Automated workflow generation for large language model agents. *ArXiv*, abs/2407.12821.

B. Liu, Yuqian Jiang, Xiaohan Zhang, Qian Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+p: Empowering large language models with optimal planning proficiency. *ArXiv*, abs/2304.11477.

Johannes E. M. Mosig, Shikib Mehri, and Thomas Kober. 2020. Star: A schema-guided dialog dataset for transfer learning. *ArXiv*, abs/2010.11853.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Cheng 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. 2023. Chatdev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics*.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024. Benchmarking agentic workflow generation.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2019. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *AAAI Conference on Artificial Intelligence*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Yixuan Su, Lei Shu, Elman Mansimov, Arshit Gupta, Deng Cai, Yi-An Lai, and Yi Zhang. 2021. Multi-task pre-training for plug-and-play task-oriented dialogue system. In *Annual Meeting of the Association for Computational Linguistics*.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark B. Gerstein. 2023. Medagents: Large language models as collaborators for zero-shot medical reasoning. *ArXiv*, abs/2311.10537.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Neural Information Processing Systems*.

Eric Wallace, Kai Xiao, Reimar H. Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. The instruction hierarchy: Training llms to prioritize privileged instructions. *ArXiv*, abs/2404.13208.

Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. 2023. Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases. *arXiv preprint arXiv:2308.11761*.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. *ArXiv*, abs/1905.08743.

Rui Xiao, Wen-Cheng Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. 2024. Flowbench: Revisiting and benchmarking workflow-guided planning for llm-based agents. *ArXiv*, abs/2406.14884.

Shuyuan Xu, Zelong Li, Kai Mei, and Yongfeng Zhang. 2024. Aios compiler: Llm as interpreter for natural language programming and flow programming of ai agents. *ArXiv*, abs/2405.06907.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. 2024. Qwen2 technical report. *ArXiv*, abs/2407.10671.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *ArXiv*, abs/2402.18013.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bangbang Liu, Yuyu Luo, and Chenglin Wu. 2024. Aflow: Automating agentic workflow generation.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's song in the ai ocean: A survey on hallucination in large language models. *ArXiv*, abs/2309.01219.

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Global-locally self-attentive encoder for dialogue state tracking. In *Annual Meeting of the Association for Computational Linguistics*.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. 2024. Knowagent: Knowledge-augmented planning for llm-based agents. *ArXiv*, abs/2403.03101.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes)
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes)
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (yes)

Does this paper make theoretical contributions? (no)

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no)
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no)
- Proofs of all novel claims are included. (yes/-partial/no)
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no)
- Appropriate citations to theoretical tools used are given. (yes/partial/no)
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA)
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA)

Does this paper rely on one or more datasets? (yes)

If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets (yes)
- All novel datasets introduced in this paper are included in a data appendix. (NA)
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes)
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing. (NA)

Does this paper include computational experiments? (yes)

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (yes).
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes)
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes)
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (NA)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes)
- This paper states the number of algorithm runs used to compute each reported result. (yes)

- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes)

- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes)

- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (NA)

- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (NA)

Appendices

A Dataset Examples

A.1 PDL Example

Below is a PDL example in a real-world scenario. For formats of natural language, code and flowchat, see Xiao et al. (2024).

```
Name: 114 Hospital Appointment
Desc: Provides appointment services,
allowing users to query and recommend
hospitals and departments in Beijing.
Detailed_desc: Queries the availability
of appointment slots based on the user's
 specified hospital, department, and
time, and attempts to register; if no
slots are available at the specified
hospital, it will try to register at
other hospitals.

APIs:
  - name: check_hospital
    request: [hospital_name]
    response: [hospital_exists]
    precondition: []
  - name: check_department
    request: [department_name,
    hospital_name]
    response: [department_exists]
    precondition: [check_hospital]
  - name: query_appointment
    request: [hospital_name,
    department_name, appointment_time]
    response: [available_slots,
    available_list, specialist_count,
    general_count]
    precondition: [check_hospital,
    check_department]
  - name: recommend_other_hospitals
    desc: Searches for available slots
    at other hospitals for the specified
     department and time.
    request: [department_name,
    appointment_time]
    response: [available_slots,
    available_list]
    precondition: [check_department]
  - name: register_hospital
    request: [id_number,
    appointment_type, hospital_name,
    department_name, appointment_time]
    response: [appointment_status]
    precondition: [query_appointment]
  - name: register_other_hospital
    request: [id_number, hospital_name,
    doctor_name]
    response: [appointment_status]
    precondition: [
    recommend_other_hospitals]

ANSWERs:
  - name: hospital_not_found
    desc: Sorry, we currently cannot
    provide appointment services for
    this hospital. Please contact the
    hospital directly or consider other
    hospitals.
  - name: department_not_found
    desc: $hospital_name does not have
    the department you are looking for.
    I will transfer you to a customer
    service representative for further
    assistance. Please wait.
  - name: no_available_slots
    desc: We apologize, but there are no
     available slots for the department
    you want to register at any hospital
     on our platform. Please follow the
    WeChat public account "Beijing 114
    Appointment appointment" to register
     as per your needs. Thank you for
    calling, and have a nice day.
  - name: appointment_successful
    desc: Your appointment at
    $hospital_name $department_name for
    $appointment_time has been
    successful. A confirmation message
    will be sent to your phone number
    shortly. Is there anything else I
    can help you with?
  - name: appointment_failed
    desc: We apologize, but there are no
     available $appointment_type slots
    at $hospital_name $department_name
    for $appointment_time. Please follow
     the WeChat public account "Beijing
    114 Appointment appointment" to
    register as per your needs. Thank
    you for calling, and have a nice day
    .
  - name:
  other_hospital_appointment_successful
    desc: Your appointment at
    $recommend_other_hospitals-
    hospital_name with
    $recommend_other_hospitals-
    doctor_name for $appointment_time
    has been successful. A confirmation
    message will be sent to your phone
    number shortly. Is there anything
    else I can help you with?
  - name:
  other_hospital_appointment_failed
    desc: We apologize, but the ID
    information is incorrect, and we
    cannot proceed with the appointment.
     Please follow the WeChat public
    account "Beijing 114 Appointment
    appointment" to register as per your
     needs. Thank you for calling, and
    have a nice day.
  - name:
  answer_out_of_workflow_questions
  - name: request_information

Procedure: |
  [hospital_exists] = API.check_hospital
  ([hospital_name])
  if hospital_exists == false:
    ANSWER.hospital_not_found()
  elif hospital_exists == true:
    [department_exists] = API.
    check_department([department_name,
    hospital_name])
    if department_exists == false:
      ANSWER.department_not_found()
    elif department_exists == true:
      [available_slots, available_list,
```

```
        specialist_count, general_count] =
         API.query_appointment([
        hospital_name, department_name,
        appointment_time])
        if available_slots > 0:
          [appointment_status] = API.
          register_hospital([id_number,
          appointment_type, hospital_name,
           department_name,
          appointment_time])
          if appointment_status == "1":
            ANSWER.appointment_successful
            ()
          elif appointment_status == "0":
            ANSWER.appointment_failed()
        elif available_slots == 0:
          [available_slots, available_list
          ] = API.
          recommend_other_hospitals([
          department_name,
          appointment_time])
          if available_slots > 0:
            if appointment_willingness ==
            "true":
              [appointment_status] = API.
              register_other_hospital([
              id_number, hospital_name,
              doctor_name])
              if appointment_status ==
              "1":
                ANSWER.
                other_hospital_appointment_su
                ()
              elif appointment_status ==
              "0":
                ANSWER.
                pther_hospital_appointment_fa
                ()
          elif available_slots == 0:
            ANSWER.no_available_slots()
```

Listing 1: Example of PDL

A.2 User Profile Example

Below is an example of a used user profile. The "User Details" contains some randomly generated attributes; "Dialogue Style" specifies the user's conversational style; "User Needs" describes the user's requirements related to a specific workflow; "Interactive Pattern" further details the possible dialogue process for the user within that workflow.

```
**Persona**:
A 25-year-old bartender with three years
 of experience in the hospitality
industry. He is known for his honesty,
often giving customers sincere advice on
 their drink choices.

**User Details**:
- Name: Michael James Carter
- Sex: Male
- Age: 25
- Phone Number: 13812345678
- ID Number: 110105199801012345

**User Needs**:
- Michael needs to query available
appointment slots for specific hospitals
 and departments in Beijing.
- He may need to verify the existence of
 certain hospitals and departments.
- He wants to make an appointment for a
medical consultation at a preferred
hospital and department.
- If the preferred hospital or
department is not available, he may need
 recommendations for alternative
hospitals and departments.
- Michael may also need to know the
success or failure status of his
appointment registration.

**Dialogue Style**:
- Michael's dialogue style is likely to
be straightforward and sincere,
reflecting his honesty in his profession
 as a bartender.
- He may prefer clear and concise
information without unnecessary jargon.
- His tone is likely to be polite and
respectful but also direct, as he is
used to providing sincere advice to
customers.
- He may appreciate a friendly and
helpful attitude from the assistant.

**Interactive Pattern**:
- Michael might start by specifying the
hospital and department he is interested
 in.
- He is likely to ask for available
appointment slots for a specific time.
- If the hospital or department does not
 exist, he will appreciate being
notified promptly and clearly.
- If there are no available slots at his
 preferred hospital, he may ask for
recommendations for other hospitals.
- He will likely ask for the success
status of his appointment registration
and may need guidance on the next steps
if the registration fails.
- Michael may interact in a sequential
manner, addressing one query at a time,
and expecting clear and immediate
responses from the assistant.
```

Listing 2: Example of user profile

B Conversation Examples

B.1 Conversation Example on STAR Dataset

Below is a conversation generated on the STAR dataset (for turn-level evaluation). Note that in line 16, the user simulated by the LLM adopted an "intent-switching/detail-switching" OOW and modified the appointment time.

```
1 ...
2 BOT: Great, Alex. What's the name of the
   apartment you're interested in?
3 USER: Maple Apartments.
```

```
BOT: When would you like to view the Maple Apartments?
USER: I'd like to do it on Friday.
BOT: What time on Friday would you prefer for the viewing?
USER: How about 3 PM?
BOT: Have you paid the application fee?
USER: Yes, I have.
BOT: Would you like to add any custom message for the viewing?
USER: No, that's fine.
BOT: <Call API> book_apartment_viewing({'RenterName': 'Alex', 'Name': 'Maple Apartments', 'Day': 'Friday', 'StartTimeHour': '15', 'ApplicationFeePaid': 'Yes', 'Message': '', 'RequestType': 'CheckAvailability'})
SYSTEM: {'Status': 'Available'}
BOT: The viewing is available. Would you like to proceed with booking?
USER: Actually, I might need to change the time. Can we do 4 PM instead?
    (OOW type) intent_switching/detail-switching
BOT: <Call API> book_apartment_viewing({'RenterName': 'Alex', 'Name': 'Maple Apartments', 'Day': 'Friday', 'StartTimeHour': '16', 'ApplicationFeePaid': 'Yes', 'Message': '', 'RequestType': 'CheckAvailability'})
SYSTEM: {'Status': 'Available'}
BOT: The new time at 4 PM is also available. Shall I book it for you?
USER: Yes, please book it.
```

Listing 3: Example conversation on STAR

C Experimental Details

C.1 Implementation Details

For the GPT series, we specifically used the models `gpt-4o-2024-05-13`, `gpt-4o-mini-2024-07-18`, and `gpt-4-turbo-2024-04-09`. To ensure stable output results, we set the temperature to 0.2. For the Qwen2 series models, we utilized the vllm framework for inference, also setting the temperature to 0.2.

C.2 Prompts

Prompts for User Simulation Below are the prompts we used with LLM to simulate user behavior. The "User Profile" refers to Sec. A.2.

Note that for OOW simulation, we add an "additional constraints" field in the user profile to describe the user's current OOW intention. For example, "In this round, you can ask a question unrelated to the current topic" will be injected in the prompt if an "irrelevant answering" OOW intent is randomly chosen in the 5th round of dialogue. We dynamically adjust OOW queries during the conversation by controlling the injected instructions.

````
You are a real-life user that interact with an assistant of {{ assistant_description }} to achieve your specific objectives.

## User Profile
```
{{ user_profile }}
```

## History conversation
```
{{ history_conversation }}
```

## Specific requirements
1. Role Awareness: Remember you are playing the user role and speak in the first person.
2. Goal-Oriented: Keep the conversation focused on achieving your needs.
3. Style: Keep your response concise and real-life.
4. Engagement: Maintain an engaging and curious tone to facilitate effective dialogue.
5. Your output format should be:
```
Response: xxx (the response content)
```
6. Stop: End the conversation when the task is completed or when it becomes repetitive and no longer meaningful to continue. Set your response as "[END]" to stop the conversation.
````

Listing 4: Prompt for user simulation

Inference Prompt for FlowAgent Below is the inference prompt for our FlowAgent.

```
You are a bot designed to assist the user for a specific task described by the Procedure Description Language (PDL). Your goal is to engage in a friendly conversation with the user while helping them complete the task.

### Constraints
1. **Step Identification**: Throughout the conversation, you should determine the user's current step, (whether it is in the PDL or just general questions), and dynamically follow PDL:
    - If the user's query aligns with the PDL logic, proceed to the next step.
    - If the user ask irrelevant questions, generate a response that maintains a fluent and logical conversation.
2. **PDL Components**: The PDL includes several components:
    - meta information: `name, desc, desc_detail` are meta information about the PDL.
    - slots: `slots`s define the information you may need to collect
```

~~~~
    from user, or the values returned by
     the API.
    - reference answer: `answers` define
     the responses you should response
    to the user.
    - procedure: the final `procedure`
    string is a Pythonic language that
    defines the core logic of the
    procedure.
3. Notes:
    - You have to collect enough
    parameter values from the user
    before calling the apis.

### PDL
```PDL
{{ PDL }}
```

### Available APIs
{{ api_infos }}

### History Conversation
{{ conversation }}

### Current state
{{ current_state | trim }}

### Output Format
Your output format should be chosen from
 one of the two templates below.
1. If you need to interact with the user
 without calling an API (inquire slot
values or reply/answer):
```
Thought: xxx (description of your
thought process )
Response: xxx (the content you need to
inquire or reply)
```
2. If you need to call an API:
```
Thought: xxx (description of your
thought process )
Action: xxx (the function name to be
called, do not prefix "API_".)
Action Input: xxx (the parameters for
the function, must be in strictly valid
JSON format)
```
~~~~

Listing 5: Prompt for FLOWAGENT

Inference Prompt for ReAct For the baseline ReAct, we directly borrowed the prompt used in FlowBench (Xiao et al., 2024).

~~~~
You are a helpful assistant for the task
 of {{task_description}}.

### Specific requirements
1. You need to act as an assistant and
engage in a conversation with the user,
following the business process and API
information.
2. You have been provided with the
flowchart information for different
scenarios under a specific role.
3. You can only answer questions within
the scope of the given several workflow
processes. If the user asks a question
beyond these scopes, please apologize
and explain to the user in the response
part.
4. When asking for API input parameters,
 ensure that the provided parameter
values comply with the specified format
regarding both the correctness of the
format and the completeness of the
content. Do not assign values
arbitrarily. In instances where the
parameters do not meet the format
requirements, notify users to make the
adjustments until the requirements are
satisfied.
5. When the user has multiple requests
at the same time, please select one
appropriate request for processing first
 and inform the user that other requests
 will be resolved subsequently. If there
 is unfinished business in the previous
conversation, continue to provide the
necessary help and guidance to assist
them in completing the business process.
 When multiple APIs need to be called,
do so in separate rounds, with a maximum
 of one API call output per round. When
the user indicates that the business is
finished or says goodbye, respond
politely and end the conversation.

### Workflow information
```
{{workflow}}
```

### Tool information
{{toolbox}}

### Current time
{{current_time}}

### History conversation
{{history_conversation}}

### Output format
Your output format should be chosen from
 one of the two templates below:
1. If you need to interact with the user
:
```
Thought: xxx (description of your
thought process )
Response: xxx (the content you need to
inquire or reply)
```
2. If you need to call an API (only one
API call per time):
```
Thought: xxx (description of your
thought process )
Action: xxx (the function name to be
called, do not prefix "functions.")
Action Input: xxx (the parameters for
the function, must be in strictly valid
JSON format)
```
~~~~

Listing 6: Prompt for ReAct

Evaluation Prompts During the evaluation process, to ensure fairness in the results, we basically reused the prompts from FlowBench. However, for the final statistics, we only used binary results to mitigate the bias issue of the judge model (see the discussion in Sec. 5.1). Below are the prompts we used for turn-level evaluation.

```
Please serve as an impartial judge to
evaluate the response quality of the
assistant. Your evaluation should be
based on the following criteria:
(1) Correctness: Does the reply remain
consistent with the workflow knowledge
without any contradictions?
(2) Helpfulness: Has the user's request
been reasonably understood and addressed
, fulfilling the user 's needs within
the provided workflow scope?
(3) Humanness: Is the response coherent,
 clear, complete, and does it include
human acknowledgment?
Please compare the provided response
with the reference response and evaluate
 it based on the mentioned dimensions.
Then, aggregate these assessments to
assign an overall score.
A perfect score is 10 points, with 9-10
points indicating high quality, nearly
identical to the reference answer; 7-8
points indicating quality close to the
reference answer; 6-7 points being of
moderate quality; 4-5 points indicating
a lower quality response; and 2-3 points
 for a response with significant errors.
Finally, output a binary result to
determine if the predicted and reference
 responses are consistent (Yes or No).

Here is the knowledge related to the
workflow:
` ` `
{{ workflow_info }}
` ` `

Here is the previous conversation:
` ` `
{{ session }}
` ` `

Here is the true value response from the
 reference:
{{ reference_input }}

Here is the generated response from the
assistant:
{{ predicted_input }}


Please reply with the scores and
consistency judgment in the following
format:
` ` `
Correctness Score: xxx
Helpfulness Score: xxx
Humanness Score: xxx
Consistency: Yes/No
` ` `
```

Listing 7: Prompt for turn-level evaluation

D Additional Method Details

D.1 Data Construction

Based on existing datasets, we performed data transformation and construction to evaluate agent performance across the compliance and flexibility dimensions. Our data construction process consists of three stages: *workflow collection, workflow representation*, and *dialogue construction*.

Workflow Collection Our dataset comprises two existing datasets: SGD (Rastogi et al., 2019) and STAR (Mosig et al., 2020), as well as our own constructed dataset, In-house. The SGD dataset includes 26 task flows across 16 domains, while the STAR dataset covers 24 task flows across 13 domains. The In-house dataset, constructed manually based on real-world scenarios in business, contains 6 workflows and 16 tools across 6 domains.

Workflow Representation To compare the performance of our PDL syntax with other workflow formats, we converted each workflow under investigation into four formats: natural language, code, flowchart, and PDL. Referring to Xiao et al. (2024), we first converted the workflows from the original datasets into natural language. Then, we used a LLM to respectively transform them into code, flowchart, and PDL formats. The definitions of tools (a.k.a., APIs) follows the OpenAI function calling formats.[2] The entire workflow format conversion process was completed using GPT-4-Turbo.

Dialogue Construction For turn-level evaluation, we constructed diverse user intentions from tasks, using GPT-4o to directly construct reference sessions. We then parsed and annotated tool calls at the turn level. Regarding the construction of OOW scenarios, we strategically insert OOW queries into the reference session and record the OOW information.

For session-level evaluation, we selected user personas from Chan et al. (2024) that exhibits real-world diversity in response style and format. We

[2] https://platform.openai.com/docs/guides/function-calling

incorporated them into workflows to construct task-related user profiles. We employed three LLMs to respectively simulate the roles of user, agent, and system with the given user profiles, workflow descriptions, and tool definitions. We collected these simulated dialogues to form the session-level evaluation dataset. As for the OOW scenarios, we have simulated users generating OOW queries with a certain probability, prompting the agent to respond to these queries and continue the conversation. The example of generated conversation is shown in App. B.1

E Additional Experimental Results

E.1 Turn-level Evaluation Results

The table below presents the turn-level experimental results of Qwen2-72B. It's important to note that because Out-of-Workflow (OOW) turns typically involve fewer complex conditional judgments or API calls, the turn-level *Success Rate* for OOW turns can sometimes be higher than for In-Workflow (IW) turns. Additionally, since the turn-level evaluation for the OOW portion involves fewer API calls, directly calculating this metric may introduce significant variance. Therefore, we have left it blank in the table.

	Method	In-house dataset			STAR			SGD		
		Pass Rate	Tool F1	Parameter F1	Pass Rate	Tool F1	Parameter F1	Pass Rate	Tool F1	Parameter F1
IW	ReAct_{NL}	65.82	76.71	65.75	58.66	65.64	51.02	60.81	**68.02**	58.39
	$\text{ReAct}_{\text{code}}$	46.83	55.70	55.44	49.41	45.81	42.34	48.95	55.11	47.52
	ReAct_{FC}	65.04	71.58	67.70	60.97	65.19	50.29	62.47	65.40	55.17
	FLOWAGENT	**68.35**	**77.14**	**68.12**	**68.94**	**67.66**	**62.19**	**64.19**	67.65	**60.78**
OOW	ReAct_{NL}	66.67	71.42	-	49.61	60.33	-	61.32	47.76	-
	$\text{ReAct}_{\text{code}}$	45.35	45.71	-	41.86	57.89	-	55.81	36.50	-
	ReAct_{FC}	60.07	74.17	-	51.94	65.00	-	65.89	68.21	-
	FLOWAGENT	**71.67**	**80.55**	-	**59.52**	**70.74**	-	**68.21**	**70.74**	-

Table 6: Turn-level Evaluation Results of Qwen2-72B