Traxgen: Ground-Truth Trajectory Generation for AI Agent Evaluation

Anonymous ACL submission

Abstract

As AI agents take on complex, goal-driven workflows, response-level evaluation becomes insufficient. Trajectory-level evaluation offers deeper insight but typically relies on high-quality reference trajectories that are costly to curate or prone to LLM sampling noise. We introduce Traxgen, a Python toolkit that constructs gold-standard trajectories via directed acyclic graphs (DAGs) built from structured workflow specifications and user data. Traxgen generates deterministic trajectories that align perfectly with human-validated references and achieve average median speedups of over 17,000× compared to LLM-based methods. To probe LLM reasoning, we compared multiple models across three workflow complexities (simple, intermediate, complex), two input formats (natural language vs. JSON), and three prompt styles (vanilla, ReAct, and ReAct-few-shot). While LLM performance varied, Traxgen outperformed every configuration in both accuracy and efficiency. Our results shed light on LLM planning limitations and establish Traxgen as a more scalable, resource-efficient alternative for reproducible evaluation of planning-intensive AI agents.

1 Introduction

Modern AI agents are increasingly expected to go beyond generating plausible responses; they must execute structured, goal-driven workflows that are auditable, policy-aligned, and robust to model or prompt changes. As these systems grow more complex, traditional response-level evaluation becomes insufficient (Yehudai et al., 2025). Instead, evaluation must consider the trajectory—the ordered sequence of tool calls or decisions an agent makes to complete a task. These trajectories expose whether an agent is reasoning effectively, choosing appropriate tools, and respecting task-specific constraints.

Recent frameworks have introduced support for trajectory-level benchmarking, typically by comparing an agent's behavior to a ground truth trajectory (LangChain, 2024; Google Cloud, 2024). However, these evaluations rely on or benefit from the availability of high-quality reference trajectories, which are often manually constructed. While LLMs have also been explored as a means to generate ground truth trajectories (Yao et al., 2024; Zhang et al., 2025), the effects of model size and workflow complexity on their performance are still poorly understood. Moreover, there are no standardized tools for generating high-quality reference trajectories, limiting reproducibility and rigorous evaluation. To address this gap, we present an automated framework for generating and evaluating agent trajectories, enabling more consistent benchmarking in both single- and multi-agent settings. Our contributions are as follows:

- **A Python toolkit** for ground truth trajectory generation in single- and multi-agent settings, supporting conditional logic, synthetic data generation, and integration with popular platforms. Our approach achieves speedups of several orders of magnitude over LLM-based trajectory generation while also seeing improved performance[1].
- **An empirical study** evaluating the trajectory planning capabilities of six diverse LLMs across varying prompt styles, input formats, and inference strategies. We evaluate LLMs performance on a curated suite of tasks spanning nine domains and three levels of workflow complexity, and compare direct generation against a search-based planning baseline.

Experimentation and code is available here.

2 Related Work

2.1 Evaluation Strategies for Agents

Evaluating multi-agent dialogue systems remains a complex challenge, requiring the assessment of in-

[1] See the package on PyPI

dividual message quality, outcome correctness, and the overall effectiveness of the agents. A common approach uses LLMs as judges to rate responses based on metrics like helpfulness, relevance, and coherence (Zheng et al., 2023; Gu et al., 2024). However, such approaches emphasize surface-level dialogue quality and often overlook agents' internal reasoning or coordination dynamics (Son et al., 2024; Feuer et al., 2024). To address limitations, recent work looks beyond conversation-level metrics. τ-Bench compares final database states with annotated ground truth goals to measure tool-use reliability across trials (Yao et al., 2024). LTM Benchmark evaluates agents' ability to retain and apply long-term memory in dynamic user interactions (Castillo-Bolado et al., 2024). CURATe explores agents' ability to personalize recommendations using safety-critical user data across sessions (Alberts et al., 2024).

Another emerging direction focuses on trajectory-level evaluations. Recent work has explored capturing tool choices, reasoning, and key decisions in agent workflows. MetaTool, for instance, examines tool selection under ambiguity (Huang et al., 2023). ToolLLaMA provides datasets that capture reasoning steps and intermediate tool calls(Qin et al., 2023b), though it lacks support for collaborative settings. ToolSandbox introduces Milestones and Minefields, events that must or must not occur, to track critical events in agent workflows (Lu et al., 2024b).

Trajectory evaluation is essential for understanding agent performance in multi-step interactions. While frameworks such as the OpenAI Agents SDK (OpenAI, 2025) and platforms like Langchain (LangChain, 2024), Vertex AI (Google Cloud, 2024), and Labelbox (Labelbox, 2025) offer agent tracing and evaluation tools, they typically assume and/or benefit from ground truth trajectories. In real-world systems driven by proprietary workflows, such ground truths are rarely available, exposing a critical shortfall in existing methodologies. There is a pressing need for an automated framework capable of generating trajectories that accurately capture internal reasoning and the collaborative dynamics of multi-agent interactions.

2.2 Trajectory Ground Truth Generation

Evaluating LLM agents in complex, tool-augmented tasks requires high-quality ground truth trajectories. However, existing generation methods are either labor-intensive or prone to errors. Existing approaches to trajectory generation broadly fall into two paradigms: human-in-the-loop LLM generation and fully automated LLM-driven methods.

In the human-in-the-loop paradigm, MetaTool Benchmark (Huang et al., 2023) utilizes human experts to label user queries based on tool necessity, supplemented by LLM-driven verification and manual review of ambiguous outputs. Similarly, ToolSandbox (Lu et al., 2024a) employs human annotators who incrementally create complex, branching scenarios from simpler cases, which are then validated through LLM-based consistency checks. DataSciBench (Zhang et al., 2025) initially generates responses using LLMs and subsequently relies on human experts to resolve inconsistencies. τ-Bench (Yao et al., 2024) similarly integrates human-written examples and LLM-generated dialogues, with an emphasis on human curation.

Automated LLM-driven approaches aim to minimize human involvement. APIBench-MT (Prabhakar et al., 2025) first creates an LLM-reviewed blueprint for intent and API use, then uses it to collect trajectories via simulating human-agent interactions. ToolLLM (Qin et al., 2023a) generates trajectories using LLMs based on defined instructions, tools, and execution examples. ToolLLM adopts a Depth-First Search-based Decision Tree algorithm guided by LLM reasoning to construct trajectories iteratively. Despite their scalability and cost-effectiveness (better than ReAct generated trajectories(Yao et al., 2023)), these methods often suffer from incomplete or incorrect trajectories due to reliance on the model's capability to correctly predict termination conditions.

3 Traxgen

In contrast to prior stochastic or hybrid approaches, we introduce a fully deterministic trajectory generation paradigm. Our toolkit (MIT-license) transforms high-level workflow specifications and customer profiles into trajectories specifying which agents should invoke which tools, in what order, with all required parameters and values resolved. These trajectories serve as the gold-standard blueprint for execution and can be distinct across users based on conditional logic, tool availability, and customer attributes. Instructions on how to install and run it are provided in the Appendix A.1.

3.1 Required Inputs

3.1.1 Workflow

Inspired by symbolic AI planning (Chen et al., 2024), workflow modeling (Russell et al., 2006), and rule-based expert systems (Grosan and Abraham, 2011), a workflow in `Traxgen` is a structured specification that encodes a sequence of tool-based operations required to accomplish a task. Workflows are JSON objects with three key components:

Steps: An ordered list of tool calls defining the actions in the workflow. Each step includes a tool name and parameter templates indicating where to source values from user-provided or system data. The list enumerates all possible tool invocations for the workflow.

Soft Ordering: A set of lists indicating groups of steps that can execute in any order. This introduces flexible sequencing, generating multiple valid trajectories by permuting the relative order of these steps. For example, a group of two steps produces two permutations (2!). Multiple groups multiply the number of generated trajectories accordingly.

Conditionals: Logic blocks that dynamically influence the trajectory based on user data, external JSON inputs, or tool outputs. Conditionals specify actions such as `skip`, `end_after`, and `override_params` targeting specific steps, enabling pruning, early termination, or parameter overrides in the trajectory generation (See Appendix Table 4 for all action definitions).

Examples of workflows can be found in the Appendix starting on section A.5.

3.1.2 User Data

`Traxgen` workflows operate with user-specific data that drives conditional branching and parameter binding. User data is provided as JSON objects including fields such as (a) agent sequence (a list of workflows to be executed), (b) customer_id or other domain-specific identifiers, and (c) user_provided_info as the subset of information that a client LLM provides to the agent during interaction. An example customer data can be found in the Appendix section A.14.

3.2 Supported Trajectory Formats

`Traxgen` supports multiple trajectory formats (see Appendix section A.15), enabling interoperability with existing frameworks and tools:

Tool Only: Minimalistic format listing only the sequence of tool calls.

Google Style: Format supported by Google's Vertex AI evaluation service.

LangChain Tool Style: Format compatible with LangChain tool evaluation ecosystem.

`Traxgen` Style: Format capturing the agent name as well as the tool calls with associated arguments in tool call format.

3.3 System Architecture

The toolkit comprises four modular stages, represented in Algorithm 1:

(1) Workflow Interpretation. Each JSON workflow is parsed into an intermediate planner object that formalizes all possible valid tool sequences, given the specified logic. The planner applies: conditional pruning based on user attributes, parameter overrides, reordering respecting soft/hard constraints.

The logic system supports branching, replanning, and early termination.

(2) Trajectory Planning. `Traxgen` builds a directed acyclic graph whose nodes are the remaining tool steps and whose edges encode mandatory precedences. The process unfolds as follows:

Node insertion: All candidate steps (from the workflow's ordered list) become nodes in an initially empty graph.

Conditional pruning: Nodes flagged by `skip` or past an `end_after` target are removed, along with their incident edges.

Edge wiring: The pruned list of steps is reconnected into a linear chain, creating one edge from each step to its successor, enforcing hard ordering.

Cycle check: We assert the graph remains acyclic, catching contradictory constraints.

Soft ordering: For each soft-ordering block (all of whose members survived pruning), we generate all intra-block permutations and splice them back into the DAG's fixed inter-block structure.

This yields a DAG backbone that guarantees correctness under hard constraints, onto which soft-block permutations layer to produce all valid trajectories (see Algorithm 1).

(3) Output Realization. For each customer profile, a fully grounded agent-level trajectory is generated and returned in all requested formats.

(4) Visualization and Auditing. To support transparency and debugging, the toolkit provides visualizations of the pruned dependency graph. Multi-agent workflows are color-coded to highlight agent-specific behaviors.

3.4 Robustness and Validation

We implement a validation layer that enforces syntactic and semantic correctness at each stage. Errors such as malformed workflows, invalid customer profiles, missing tool parameters, or unsupported API flags are detected early.

Algorithm 1 `Traxgen` Trajectory Generation

Require: Customers C, workflows W, formats F, visualize flag v
Ensure: Trajectories $\mathcal{T}$

for all customer $c \in C$ **do**
  $A \leftarrow c$.agent_sequence
  if $|A| > 1$ **then**
    $(\tau, \pi) \leftarrow$ GEN_MULTI_AGENT(A, c, W)
  else
    $\pi \leftarrow$ PARSE_WORKFLOW$(W[A[0]], c)$
    ▷ prune unreachable nodes, apply value overrides
    APPLY_CONDITIONAL_ACTIONS(π)
    ▷ build pruned DAG including relevant tool calls
    ADD_TOOLS_TO_GRAPH(π)
    ▷ generate valid paths (respect soft-blocks, deduplicate)
    $\tau \leftarrow \pi$.GENERATE_VALID_TRAJECTORIES()
    replicate π for each τ_i
  end if
  for all (τ_i, π_i) **do**
    for all $f \in F$ **do**
      $T_{c,f}$ += FORMAT(π_i, τ_i, f)
    end for
    if v **then**
      VISUALIZE(π_i, τ_i, A)
    end if
  end for
  $\mathcal{T}[c] \leftarrow$ merge $T_{c,*}$ if multi-agent else $T_{c,*}$
end for
return $\mathcal{T}$

4 Experimentation

4.1 Data Construction

We generate data for nine customer-service workflows using a structured three-stage process:

Stage I: Workflow design. We manually define structured JSON workflows, specifying the sequence of tool calls, parameter bindings, and policy constraints using a compact control-flow language (e.g., `skip`, `end_after`, `override_trajectory`). Three workflows were generated for each of the three complexity tiers (see §4.3).

Stage II: Customer profile generation. For each workflow, we create a pool of diverse customer profiles in JSON form, populated via templated sampling supported by `Traxgen`. Profiles include relevant user-specific information (e.g., address, product ID, leave dates) required to instantiate tool parameters.

Stage III: Trajectory Annotation and Verification. We use `TraxGen` to compile each workflow–profile pair into a fully grounded, deterministic trajectory. Two annotators, blinded to the generation source, validate whether each output trajectory strictly adheres to the policy logic defined in the routine and is consistent with the corresponding customer data. Annotators were provided with structured scoring guidelines to assess tool order, parameter correctness, conditional execution, and agent boundaries. A trajectory is marked as valid only if it fully satisfies all policy constraints. Detailed annotation instructions and error tag definitions are provided in Appendix A.16.

4.2 Key Characteristics

Deterministic Trajectory-Based Evaluation We differ from prior tool-use benchmarks (Qin et al., 2023a; Yao et al., 2024) by abstracting away open-ended creativity and nuanced interpretation from the evaluation process. Rather than relying on live API calls or stochastic user goals with binary success/failure outcomes, we implement a reproducible, rule-based evaluation framework focused on trajectory conformance.

Each task is constructed with a fixed user intent and a fully specified customer profile, ensuring that there exists a predetermined set of correct trajectories consistent with domain policy. This design enables exact-match comparison between model outputs and gold reference paths, evaluating performance not just on final outcomes but on whether models follow the correct sequence of actions throughout the entire process. The focus on trajectory conformance rather than end-state success directly mirrors enterprise workflow requirements, where compliance, auditability, and traceability are non-negotiable for production deployment.

Multi-Intent and Multi-Agent Tasks To simulate longer-horizon interactions, we also include a subset of tasks that require planning across multiple linked intents (e.g., `BookFlight` followed by `CancelFlight`). These tasks are modeled as modular, multi-agent trajectories, where each sub-intent is handled by an individual policy workflow. This structure supports evaluation of inter-agent coordination and policy handoff.

4.3 Data Distribution and Complexity Levels

Workflow Complexity We categorize workflow into three levels of complexity: simple (linear or near-linear flows with minimal conditionals), intermediate (moderate branching and optional soft ordering), and complex (nested conditionals, soft orderings across multiple tool sets, and strong reliance on contextual variables).

Data Distribution To balance annotation effort and task coverage, we sample 100 customer profiles per complex intent, 75 per intermediate intent, and 50 per simple intent. This distribution reflects the increased diversity and error surface in complex workflows, while ensuring robust metric stability across all tiers. In total, we include 775 task instances and 71 unique tools, with over 10% comprising multi-intent cases.

Intent	Complexity	Domain	# Test Cases	# APIs
`checkOrderStatus`	Simple	E-Commerce	50	3
`checkProductAvailability`	Simple	E-Commerce	50	5
`resendEmailReceipt`	Simple	E-Commerce	50	4
`submitTimeOffRequest`	Intermediate	HR	75	8
`updateAddress`	Intermediate	HR	75	7
`accountSuspensionRequest`	Intermediate	HR	75	7
`bookFlight`	Complex	Travel	100	12
`cancelFlight`	Complex	Travel	100	12
`flightDisruption`	Complex	Travel	100	13

Table 1: Intents categorized by complexity, domain, number of test cases, and number of APIs.

4.4 General Experimentation Setup

Across all experiments, we task models with generating agent trajectories conditioned on a user intent, customer profile, and workflow. We evaluate a range of prompting strategies (vanilla, ReAct, ReAct with few-shot), input representations (natural language vs. structured JSON), and workflow complexity levels. Both our custom generation package `Traxgen` and multiple LLMs are tested under these conditions. Outputs are compared against the human-validated reference trajectories.

4.5 Evaluation Metrics

To handle multiple predicted and gold trajectories—due to soft ordering or multi-output models—we align each prediction to its best-matching ground-truth trajectory using the Hungarian algorithm (Kuhn, 1955), maximizing a chosen similarity metric. We then evaluate the aligned pairs using the metrics below.

Let $\mathcal{G}$ and $\mathcal{P}$ be the sets of ground-truth and predicted trajectories (each a sequence of $(\text{tool}, \text{params})$ steps).

Exact Match and Count Agreement We compute *Exact Match* as $\mathbf{1}(\mathcal{P} = \mathcal{G})$, a binary indicator of set equality (ignoring order), and *Count Agreement* as $\left(\frac{|\mathcal{P}|}{|\mathcal{G}|}\right) \times 100\%$, capturing over- or under-prediction in number of trajectories predicted.

Tool- and Parameter-Level PRF We flatten each matched trajectory pair into a multiset of tools $\mathcal{T} = [t_1, t_2, \dots]$ and a multiset of parameter triplets $\mathcal{P} = [(t, k, v)_j]$, where each t is a tool, k a parameter key, and v its value. We compute precision, recall, and F1 based on multiset overlap (ignoring order): true positives (TP), false positives (FP), and false negatives (FN) are counted by comparing predicted elements against ground truth. Standard PRF metrics are reported separately for tools and parameter triplets.

Contiguous Overlap Length (CO) Measures the longest substring C shared between $\mathcal{G}$ and $\mathcal{P}$:

$$C = \max\{k : \mathcal{G}_{i+\ell} = \mathcal{P}_{j+\ell} \text{ for } \ell = 0, \dots, k-1\}.$$

We report the percentage of $\mathcal{G}$ recovered in a single uninterrupted chunk as $100 \times \frac{C}{|\mathcal{G}|}$.

Prefix Length. Captures the longest common prefix L between $\mathcal{G}$ and $\mathcal{P}$:

$$L = \max\{k : \mathcal{G}_i = \mathcal{P}_i \text{ for all } i = 1, \dots, k\}.$$

We report the normalized percentage as $\text{PrefixScore}(\mathcal{G}, \mathcal{P}) = 100 \times \frac{L}{|\mathcal{G}|}$.

Unmatched ground-truth trajectories are excluded from PRF and length calculations but contribute to the Count Agreement metric. This separation ensures trajectory-level quality is evaluated independently from prediction quantity.

5 Experiment 1: `Traxgen` Evaluation

5.1 Experiment-Specific Setup

We assess `Traxgen`'s ability to generate accurate trajectories from structured workflows and user profiles. We evaluated `Traxgen` on the same inputs and compared its outputs to the validated references using the metrics in 4.5. As a control, we include LLM baselines prompted with either (a) the

Routine Complexity	DeepSeek	Gemini	GPT4.1	Llama4	Mistral	Sonnet	Package
Complex workflow	28.82	5.01	4.48	14.26	8.70	7.43	0.00048337
Intermediate workflow	16.78	2.87	3.52	7.45	5.06	4.81	0.00017534
Simple workflow	9.30	1.53	2.08	3.28	3.22	3.60	0.00009979

Table 2: Average runtime (seconds) per trajectory by model across routine complexities.

original JSON workflows or (b) equivalent natural-language descriptions, isolating the impact of structured input. A full analysis of LLM performance appears in Section 6.

5.2 Results

`Traxgen` achieves 100% alignment with the gold trajectories across all evaluation metrics, validating its ability to deterministically and accurately capture conditional workflow logic (see Appendix Table 5). This confirms its suitability as a ground-truth generator for downstream benchmarking.

Compared to twelve LLM configurations (six models each run with both JSON-structured and natural-language workflow inputs under a uniform prompting strategy) `Traxgen` consistently outperforms across all evaluation metrics. While the full LLM benchmark is deferred to Section 6, we note here that `Traxgen`'s performance is not only more accurate but also significantly more efficient. `Traxgen` eliminates the need for token-based inference, achieving median speedups of 30,000× on simple workflow and over 17,000× across all complexity levels (see Table 2). Moreover, unlike LLMs, which process an average of 750–3,400 tokens per example (see Appendix tables 6, 7), `Traxgen` executes near-instantaneously and incurs minimal compute and energy costs. Our method lowers environmental impact and enhances reproducibility, offering a more sustainable and efficient solution for large-scale benchmarking.

6 Experiment 2: LLM Benchmarking

To assess in-context planning, we design a suite of controlled experiments that isolate the planning stage of tool use. The benchmark abstracts away execution, focusing on the model's ability to generate policy-compliant trajectories from user instructions and structured workflows. Each task requires reasoning over customer data and multi-step workflows—selecting tools, binding parameters, and handling conditionals—in a single forward pass. To ensure broad coverage, we evaluate six diverse LLMs spanning architectures, openness, and scale: open models `DeepSeek-Chat-v3-0324`, `Mistral-7B-Instruct`, `LLaMA-4-Maverick`, and proprietary ones `Gemini-2.0-Flash-001`, `Claude-3.7-Sonnet`, and `GPT-4.1`. Our setup follows plan-first evaluation protocols (Zheng et al., 2024), enabling deterministic assessment of planning quality without interactive noise.

6.1 Experiment-Specific Setup

We perform three controlled studies, each isolating a different variable that can affect trajectory-planning quality: *representation of the workflow*, *prompt engineering*, and *inference-time search*. The same nine workflows and evaluation metrics are used throughout, so any performance change can be attributed to the factor under study.

Study 1: Input Representation (Natural Language vs. JSON. Trajectory planning often involves structured task representations (e.g., graphs, trees, JSON). However, it remains unclear how much of an LLM's success stems from the structure itself versus the model's understanding of task semantics. To isolate this factor, we compared each model's performance when given (a) the natural language description of the workflow, and (b) the equivalent structured JSON representation (used in `Traxgen`) across the three complexity levels. All other prompt elements were held constant.

Study 2: Prompt-Engineering Strategies Prompting strategies influence model behavior, especially in constrained reasoning tasks. We tested three prompt designs: Vanilla prompt, a minimal instruction-only setup with no reasoning steps; ReAct-style prompt, which interleaves reasoning (thought) and action steps; and ReAct + few-shot, which follows the same format as ReAct but is augmented with a worked example matched to the routine's complexity. This sub-experiment used two representative models—Llama-4 Maverick (open) and Sonnet 3.7 (proprietary)—to strike a balance between coverage and depth.

Study 3: Direct Generation *vs.* Guided Search

A third variable in our experimental setup is the inference strategy. Recent work on ToolLLM introduced DFSDT, a depth-first search–based decision-tree algorithm that augments an LLM with explicit backtracking and branch exploration (Qin et al., 2023b). We adapt DFSDT by replacing live APIs with static, simulated tool functions, enabling deterministic and side-effect-free execution within each task. The same underlying LLM is used to generate both ReAct-style direct trajectories (*Direct*) and search-guided trajectories via DFSDT, enabling a clean comparison of (i) pure in-context planning versus (ii) planning with external tool-based feedback. To strike a balance between evaluation cost and insight depth, we limited this sub-experiment to 50 customers per domain. This subset was sufficient to capture meaningful trends in performance while controlling for DFSDT's longer execution time and additional system complexity.

6.2 Results

Trajectory Quality Evaluation The raw trajectories generated by the LLMs often required additional cleaning before they could be directly used or compared to the ground truth. To address this, we developed a Python script to standardize and clean the outputs. Common issues included the presence of markdown fences surrounding the code, bracket mismatches, and null literals. Notably, DeepSeek showed a higher tendency to hallucinate, frequently returning plain code snippets without proper structure. Detailed cleaning metrics and error frequencies are reported in the appendix table 9.

Model Comparison Model performance on complex workflows shows a stratification by model class and format. For both JSON and natural language, Gemini and Sonnet outperform other models across nearly all metrics. Sonnet demonstrates strong tool and parameter-level accuracy on complex workflow, while Gemini shows comparable or better performance on intermediate workflows. LLaMA4 and GPT-4.1 follow closely, with strong F1 and prefix scores but lower exact match and CMR. In contrast, Mistral and DeepSeek trail behind across most metrics, particularly on complex workflows. These findings suggest that Gemini and Sonnet are best suited for handling high-complexity, multi-step tasks in both formats.

Complexity Comparison LLM performance varies across different level of complexities. Figure 1 shows how all models except Mistral performed relatively well based on F1 score for tool and parameters in simple complexity tasks. However, models show inconsistent performance in intermediate tasks illustrated by larger variance, and tend to degrade over complexity in JSON prompt formatting.

Prompt Formatting Comparison For intermediate workflow, JSON formatting consistently outperformed all other options across every model and metric. In contrast, simple workflow showed minimal sensitivity to formatting choice—performance differences were negligible and varied idiosyncratically by model. The most striking effects emerged in complex workflow, where formatting had a substantial impact: while JSON remained optimal for the most capable models (such as GPT-4.1 and Claude Sonnet), Python formatting yielded dramatic improvements for mid-tier and open-source models (including Deepseek, Gemini, and Llama4)

Prompt Engineering Method Comparison Results indicate that prompt style influences performance differently depending on routine complexity and model type (see Appendix Table 11). For simple workflow, all prompt types achieved near-perfect exact-match and parameter F1 scores, with slight gains observed in the ReAct format. For intermediate workflow, the vanilla prompt surprisingly yielded the highest exact-match scores for Llama-4 in natural language format, while Sonnet favored ReAct prompts, suggesting model- and domain-

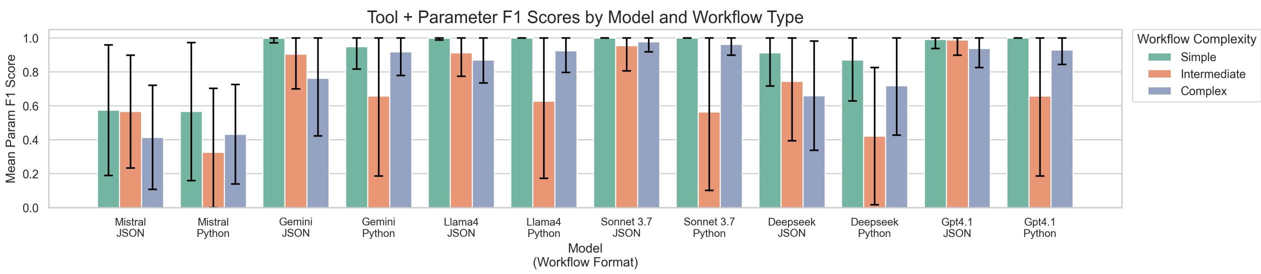

Figure 1: Mean F1 scores for tool and parameter extraction across models and workflow formats, stratified by workflow complexity.

Model	Format	Exact-Match (%)	Count (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex workflow									
Mistral	J	0.0 ± 0.0	69.8 ± 34.4	0.525 ± 0.335	0.414 ± 0.307	36.7 ± 29.2	29.4 ± 24.9	33.5 ± 29.6	18.3 ± 26.6
Deepseek	J	5.5 ± 22.8	73.9 ± 34.7	0.706 ± 0.291	0.659 ± 0.322	48.1 ± 30.6	46.6 ± 30.3	32.4 ± 35.4	27.2 ± 35.7
Gemini	J	11.5 ± 31.9	84.2 ± 36.6	0.759 ± 0.333	0.762 ± 0.340	67.1 ± 34.2	66.3 ± 34.0	57.1 ± 40.8	56.4 ± 40.4
Sonnet	J	38.5 ± 48.7	69.8 ± 34.2	0.975 ± 0.059	0.977 ± 0.059	93.6 ± 15.8	91.9 ± 16.9	92.9 ± 18.0	91.2 ± 18.9
Llama4	J	15.2 ± 36.0	100.3 ± 37.8	0.877 ± 0.117	0.870 ± 0.135	66.2 ± 25.8	63.9 ± 25.9	60.8 ± 30.8	58.5 ± 30.6
Gpt4.1	J	26.0 ± 43.9	70.4 ± 34.5	0.940 ± 0.098	0.938 ± 0.112	76.1 ± 25.0	75.2 ± 25.0	73.8 ± 28.1	73.1 ± 27.9
Mistral	P	0.2 ± 5.0	69.8 ± 34.3	0.505 ± 0.311	0.432 ± 0.293	30.6 ± 24.7	24.0 ± 19.6	26.4 ± 24.9	12.1 ± 19.0
Deepseek	P	13.5 ± 34.2	74.1 ± 34.1	0.775 ± 0.235	0.718 ± 0.291	58.1 ± 31.3	55.4 ± 31.8	43.3 ± 41.0	39.9 ± 41.6
Gemini	P	23.8 ± 42.6	87.1 ± 25.3	0.914 ± 0.129	0.918 ± 0.139	77.6 ± 22.9	76.8 ± 23.5	65.1 ± 36.3	65.1 ± 36.3
Sonnet	P	16.5 ± 37.2	70.1 ± 34.2	0.954 ± 0.071	0.962 ± 0.064	84.0 ± 19.1	82.7 ± 20.4	75.4 ± 30.1	74.9 ± 30.1
Llama4	P	14.8 ± 35.5	84.9 ± 28.5	0.920 ± 0.123	0.924 ± 0.127	75.6 ± 25.0	73.5 ± 25.8	69.1 ± 32.3	67.3 ± 32.7
Gpt4.1	P	16.5 ± 37.2	71.0 ± 33.8	0.930 ± 0.088	0.929 ± 0.086	72.6 ± 26.4	70.1 ± 27.0	65.5 ± 33.3	64.2 ± 32.6
Intermediate workflow									
Mistral	J	2.7 ± 16.1	67.3 ± 23.8	0.658 ± 0.290	0.566 ± 0.333	53.8 ± 28.9	46.5 ± 28.4	50.2 ± 31.1	34.0 ± 35.3
Deepseek	J	49.8 ± 50.1	81.8 ± 24.6	0.814 ± 0.291	0.743 ± 0.349	83.6 ± 29.3	75.3 ± 32.4	76.6 ± 40.7	60.8 ± 48.0
Gemini	J	76.9 ± 42.2	100.0 ± 0.0	0.972 ± 0.081	0.905 ± 0.205	98.5 ± 7.1	94.3 ± 14.3	98.5 ± 7.1	94.3 ± 14.3
Sonnet	J	59.6 ± 49.2	85.1 ± 24.8	0.968 ± 0.094	0.955 ± 0.149	96.3 ± 12.7	96.3 ± 12.7	94.2 ± 20.2	94.2 ± 20.2
Llama4	J	43.1 ± 49.6	107.6 ± 58.1	0.919 ± 0.086	0.912 ± 0.138	92.9 ± 17.7	92.4 ± 18.1	92.5 ± 18.8	92.0 ± 19.1
Gpt4.1	J	63.6 ± 48.2	81.8 ± 24.1	0.994 ± 0.047	0.988 ± 0.089	99.1 ± 6.6	99.1 ± 6.6	99.1 ± 6.6	99.1 ± 6.6
Mistral	P	6.7 ± 25.0	67.1 ± 26.0	0.376 ± 0.394	0.325 ± 0.378	28.4 ± 33.4	22.3 ± 30.9	22.2 ± 33.5	14.7 ± 30.7
Deepseek	P	3.6 ± 18.6	68.0 ± 24.5	0.452 ± 0.378	0.421 ± 0.405	33.5 ± 30.4	32.9 ± 30.3	9.3 ± 24.7	8.5 ± 23.7
Gemini	P	64.0 ± 48.1	83.3 ± 23.6	0.662 ± 0.470	0.657 ± 0.471	65.7 ± 46.9	65.7 ± 46.9	65.4 ± 47.1	65.4 ± 47.1
Sonnet	P	35.6 ± 48.0	75.8 ± 29.2	0.600 ± 0.449	0.563 ± 0.462	57.2 ± 45.2	57.2 ± 45.2	54.9 ± 46.4	54.9 ± 46.4
Llama4	P	44.4 ± 49.8	85.6 ± 22.7	0.640 ± 0.449	0.627 ± 0.454	65.8 ± 46.5	65.5 ± 47.0	65.5 ± 47.0	65.5 ± 47.0
Gpt4.1	P	44.9 ± 49.8	69.8 ± 29.8	0.662 ± 0.471	0.658 ± 0.472	65.9 ± 47.1	65.9 ± 47.1	65.7 ± 47.2	65.7 ± 47.2
Simple workflow									
Mistral	J	23.3 ± 42.4	99.3 ± 8.2	0.738 ± 0.325	0.574 ± 0.385	66.5 ± 35.5	49.8 ± 38.3	60.0 ± 42.0	37.3 ± 43.8
Deepseek	J	30.0 ± 46.0	99.3 ± 8.2	0.881 ± 0.191	0.912 ± 0.195	81.2 ± 25.4	75.3 ± 24.4	50.0 ± 50.2	30.2 ± 45.9
Gemini	J	68.7 ± 46.5	100.0 ± 0.0	0.955 ± 0.068	0.998 ± 0.027	92.0 ± 12.1	92.0 ± 12.1	69.0 ± 46.2	69.0 ± 46.2
Sonnet	J	100.0 ± 0.0	100.0 ± 0.0	1.000 ± 0.000	1.000 ± 0.000	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Llama4	J	96.0 ± 19.7	104.0 ± 19.7	0.999 ± 0.009	0.999 ± 0.012	99.7 ± 4.1	99.7 ± 4.1	99.7 ± 4.1	99.7 ± 4.1
Gpt4.1	J	96.7 ± 18.0	100.0 ± 0.0	0.992 ± 0.042	0.991 ± 0.054	98.5 ± 8.3	98.5 ± 8.3	98.0 ± 11.4	98.0 ± 11.4
Mistral	P	32.0 ± 46.8	105.3 ± 74.0	0.700 ± 0.359	0.566 ± 0.407	63.0 ± 39.2	50.7 ± 40.1	56.2 ± 45.1	40.8 ± 45.1
Deepseek	P	28.0 ± 45.1	99.3 ± 8.2	0.825 ± 0.214	0.870 ± 0.242	74.3 ± 24.2	68.0 ± 28.7	29.5 ± 45.6	28.8 ± 44.9
Gemini	P	44.7 ± 49.9	100.0 ± 0.0	0.874 ± 0.147	0.948 ± 0.132	80.5 ± 20.5	79.8 ± 21.6	44.7 ± 49.9	44.7 ± 49.9
Sonnet	P	99.3 ± 8.2	100.0 ± 0.0	0.999 ± 0.012	1.000 ± 0.000	99.8 ± 2.0	99.8 ± 2.0	99.3 ± 8.2	99.3 ± 8.2
Llama4	P	100.0 ± 0.0	100.0 ± 0.0	1.000 ± 0.000	1.000 ± 0.000	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Gpt4.1	P	96.0 ± 19.7	100.0 ± 0.0	0.994 ± 0.028	1.000 ± 0.000	99.0 ± 4.9	99.0 ± 4.9	96.0 ± 19.7	96.0 ± 19.7

Table 3: Performance across simple, intermediate, complex workflows. Format: J=JSON, P=Natural Language.

specific prompt sensitivity. In complex workflows, ReAct consistently outperforms the other methods in terms of Tool F1. Notably, few-shot prompting did not consistently outperform simpler prompt designs, indicating that adding examples may not universally benefit constrained reasoning tasks.

Direct Generation and Guided Search Comparison Appendix Table 10 shows that the DFSDT approach underperforms direct generation across all complexity levels. One consistent pattern is that DFSDT-generated trajectories often skip required steps defined in the routine, leading to low exact-match and step-level F_1 scores. A likely contributor is the way in which DFSDT determines when a plan is complete—potentially stopping before all mandatory steps in the policy have been executed. This highlights a limitation of search-based planning without explicit end-condition supervision.

7 Discussion

We introduced `Traxgen`, a deterministic trajectory generation framework for reproducible, scalable benchmarking of tool-augmented LLM agents. The toolkit aligns perfectly with manually validated ground truth and outperforms LLM-based baselines by orders of magnitude in both accuracy and efficiency. Crucially, `Traxgen` ensures full data sovereignty by requiring no external model inference during generation. Beyond performance, `Traxgen` reframes planning evaluation by removing inference-time randomness, enabling stable, repeatable comparisons across workflows and agents. Unlike prompting-based methods, which are sensitive to phrasing and sampling, it offers a consistent reference point for empirical validation.

Our ablation studies show that input structure plays a critical role in LLM planning: JSON schemas consistently outperform natural language, and ReAct-style prompting yields only marginal, inconsistent gains. These trends suggest that architectural improvements—such as schema-constrained decoders—may be more impactful than further prompt tuning. Ultimately, `Traxgen` provides a reliable foundation for evaluating AI agents in planning-intensive settings, where reproducibility, accuracy, and transparency are essential.

8 Limitations

While Traxgen enables reproducible, deterministic evaluation of agent trajectories, it has not yet been validated on real-world enterprise workflows, which often involve complex interdependencies, multimodal inputs (e.g., images, logs), and behaviors like retries or non-idempotent calls. Deterministic enumeration of soft-order permutations can also cause factorial growth, limiting scalability for large workflows; we cap block sizes to ensure tractability, but broader use may require sampling or summarization. A risk, however, is that Traxgen's rigidity also reduces flexibility: unlike generative agents, it cannot adapt to novel or ambiguous inputs without pre-specified logic. Finally, our LLM benchmarking (T2) is limited by model access and prompt design assumptions, which may not reflect newer architectures or alternative strategies. While these limitations impact deployment, Traxgen still provides a robust platform for experimental evaluation.

References

Lize Alberts, Benjamin Ellis, Andrei Lupu, and Jakob Foerster. 2024. Curate: Benchmarking personalised alignment of conversational ai assistants. *arXiv preprint arXiv:2410.21159*.

David Castillo-Bolado, Joseph Davidson, Finlay Gray, and Marek Rosa. 2024. Beyond prompts: Dynamic conversational benchmarking of large language models. *arXiv preprint arXiv:2409.20222*.

Dillon Z. Chen, Pulkit Verma, Siddharth Srivastava, Michael Katz, and Sylvie Thiébaux. 2024. Ai planning: A primer and survey (preliminary report). *Preprint*, arXiv:2412.05528.

Benjamin Feuer, Micah Goldblum, Teresa Datta, Sanjana Nambiar, Raz Besaleli, Samuel Dooley, Max Cembalest, and John P Dickerson. 2024. Style outweighs substance: Failure modes of llm judges in alignment benchmarking. *arXiv preprint arXiv:2409.15268*.

Google Cloud. 2024. Introducing agent evaluation in vertex ai gen ai evaluation service. Accessed: 2025-04-13.

Crina Grosan and Ajith Abraham. 2011. *Rule-Based Expert Systems*, pages 149–185. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, and 1 others. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and 1 others. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Labelbox. 2025. How to train and evaluate ai agents and trajectories with labelbox. Accessed: 2025-04-13.

LangChain. 2024. Evaluation concepts. Accessed: 2025-04-13.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2024a. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2024b. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.

OpenAI. 2025. Openai agents sdk. Accessed: 2025-04-13.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Preprint*, arXiv:2504.03601.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023a. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

N.C. Russell, A.H.M. Hofstede, ter, W.M.P. Aalst, van der, and N.A. Mulyar. 2006. *Workflow control-flow patterns : a revised view*. BPM reports. BPMcenter.org.

Guijin Son, Hyunwoo Ko, Hoyoung Lee, Yewon Kim, and Seunghyeok Hong. 2024. Llm-as-a-judge & reward model: What they can and cannot do. *arXiv preprint arXiv:2409.11239*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. Taubench: A benchmark for tool-agent-user interaction in real-world domains. *Preprint*, arXiv:2406.12045.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025. Datascibench: An llm agent benchmark for data science. *Preprint*, arXiv:2502.13897.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2024. Natural plan: Benchmarking llms on natural language planning. *Preprint*, arXiv:2406.04520.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Appendix Contents

A.1 Generating Trajectories with Traxgen

```
pip install traxgen

from traxgen import generate_trajectories

customer_data = json.load(open("test_data/customer_data/simple_routine.json"))

workflow_data = {
    "check_order_status": json.load(open("simple/check_order_status.json")),
    "resend_email_receipt": json.load(open("simple/resend_email_receipt.json")),
   "check_product_availability": json.load(open("simple/check_product_availability.json")),
}

output = generate_trajectories(
        customer_data=customer_data,
        routine_data=routine_data,
        id_field='customer_id',
        trajectory_format= ['google'],
        output_path = 'output/simple_routines',
        output_mode = return_format,
        enable_visualization=False)
```

A.2 Traxgen supported workflows conditional actions

Logic Construct	Definition
`skip`	Skips the execution of one or more steps when a specified condition is met.
`end_after`	Terminates the routine immediately after the specified step if the condition is met.
`override_trajectory`	Replaces the default step sequence with a new list of steps, enabling a custom path.
`all_of`	A composite condition that is satisfied only if **all** subconditions are true. Used within an `if` clause.
`any_of`	A composite condition that is satisfied if **any** subcondition is true. Used within an `if` clause.

Table 4: Definitions of conditional actions supported in Traxgen JSON workflows.

A.3 `Traxgen` Evaluation Results

Routine	Model	Exact-Match (%)	Count (%)	Tool F1	Param F1	CO % tools	CO % params	Prefix % tools	Prefix % params
Complex	Package	100.0 ± 0.0	100.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Intermediate	Package	100.0 ± 0.0	100.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Simple	Package	100.0 ± 0.0	100.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0

Table 5: Package evaluation results across all routine complexities. All metrics are reported as mean ± standard deviation across evaluation splits.

A.4 Main LLM Experiment Results

Workflow	DeepSeek	Gemini	GPT-4.1	LLaMA 4	Mistral	Sonnet 3.7
Complex	2703.58	3371.62	2429.05	2872.32	3528.30	2921.56
Intermediate	1445.40	1707.81	1307.93	1388.18	1722.44	1536.91
Simple	868.06	984.62	786.44	791.65	1123.87	977.76

Table 6: Average total token usage per workflow complexity using structured JSON workflow instructions.

Routine	DeepSeek	Gemini	GPT-4.1	LLaMA 4	Mistral	Sonnet
Complex	2615.45	3366.92	2425.30	2621.34	3279.63	2818.16
Intermediate	1041.62	1357.16	1001.81	1034.91	1448.38	1224.04
Simple	869.46	941.53	771.91	801.27	1133.48	953.47

Table 7: Average total token usage per workflow complexity using natural language workflow instructions.

Routine	Py DeepSeek	Py Gemini	Py GPT4.1	Py Llama4	Py Mistral	Py Sonnet
Complex workflow	24.90	5.59	4.63	10.64	9.05	7.79
Intermediate workflow	10.59	2.55	3.12	5.05	7.08	5.32
Simple workflow	8.60	1.42	1.79	3.25	3.97	3.61

Table 8: Average runtime (seconds) per trajectory by Natural Language -based models across routine complexities.

Table 9: LLM Output Cleaning Metrics by Workflow Type, Workflow Format, and Model

Workflow	Format	Model	Initial Fail	Recovered	Unrecovered	Bracket Mismatch	Hallucinated Code	Incorrect Format	Invalid Commas	Junk Btw. Brackets	Markdown Fences	Mismatched Quotes	Missing Commas	Null Literals	Single Quotes	Ellipses	[] Expr in quotes
simple	json	deepseek	105	102	2	4	1	0	0	0	94	0	0	1	0	0	0
	json	gemini	150	150	0	0	0	0	0	0	150	0	0	0	0	0	0
	json	gpt4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	json	llama4	17	17	0	0	0	0	0	0	17	0	0	0	0	0	0
	json	mistral	65	64	1	2	0	0	0	5	1	0	7	0	1	0	15
	json	sonnet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	py	deepseek	114	109	2	9	3	0	0	0	103	0	0	3	0	0	0
	py	gemini	150	150	0	0	0	0	0	0	150	0	0	0	0	0	0
	py	gpt4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	py	llama4	10	10	0	0	0	0	0	0	10	0	0	0	0	0	0
	py	mistral	85	83	2	3	0	0	0	21	5	0	12	2	1	2	9
	py	sonnet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
intermediate	json	deepseek	130	123	6	15	1	0	0	0	102	2	0	1	0	0	0
	json	gemini	225	225	0	0	0	0	0	0	225	0	0	0	0	0	0
	json	gpt4.1	37	37	0	0	0	0	0	0	0	0	0	0	0	0	0
	json	llama4	71	71	0	0	0	0	0	0	71	0	0	0	0	0	0
	json	mistral	58	58	0	3	0	0	0	14	5	0	8	4	2	5	4
	json	sonnet	7	7	0	0	0	0	0	0	0	0	1	0	0	0	0
	py	deepseek	164	157	3	10	4	0	0	0	147	3	2	1	1	0	3
	py	gemini	225	225	0	0	0	0	0	0	225	0	0	0	0	0	0
	py	gpt4.1	45	45	0	0	0	0	0	0	0	0	0	0	0	0	0
	py	llama4	155	155	0	1	0	0	0	0	155	0	3	12	0	0	0
	py	mistral	114	113	1	4	0	0	0	13	15	0	6	2	0	12	9
	py	sonnet	8	8	0	0	0	0	0	0	0	0	5	0	0	0	0
complex	json	deepseek	289	264	19	18	6	1	0	1	246	2	0	16	0	1	0
	json	gemini	400	400	0	0	0	0	0	0	400	0	0	6	0	0	0
	json	gpt4.1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	json	llama4	126	125	1	1	0	0	0	0	124	0	0	13	0	0	0
	json	mistral	111	109	2	14	0	2	5	9	2	1	5	9	1	7	56
	json	sonnet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	py	deepseek	302	297	1	15	4	0	0	0	289	0	0	3	0	0	1
	py	gemini	400	400	0	2	0	0	0	0	400	0	0	33	0	0	0
	py	gpt4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	py	llama4	160	159	1	1	0	0	0	0	158	0	0	7	0	0	0
	py	mistral	145	144	1	10	0	0	8	15	12	2	8	27	0	24	48
	py	sonnet	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Model	Format	Exact-Match (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex workflow								
Sonnet	P	0.0 ± 0.0	0.354 ± 0.419	0.228 ± 0.263	26.3 ± 35.9	16.7 ± 21.5	24.4 ± 36.4	15.0 ± 21.6
Llama4	P	0.0 ± 0.0	0.279 ± 0.253	0.227 ± 0.258	17.6 ± 20.6	14.9 ± 18.3	12.1 ± 20.1	10.7 ± 18.0
Gpt4.1	P	0.0 ± 0.0	0.516 ± 0.321	0.400 ± 0.191	33.3 ± 31.6	23.6 ± 18.7	25.3 ± 35.5	16.2 ± 21.2
Intermediate workflow								
Sonnet	P	0.0 ± 0.0	0.478 ± 0.254	0.306 ± 0.198	38.7 ± 30.9	23.7 ± 17.1	31.0 ± 34.8	19.5 ± 19.0
Llama4	P	0.0 ± 0.0	0.531 ± 0.222	0.314 ± 0.205	41.6 ± 30.0	27.3 ± 14.8	34.4 ± 33.1	22.3 ± 17.6
Gpt4.1	P	0.0 ± 0.0	0.556 ± 0.159	0.353 ± 0.176	44.1 ± 24.8	26.7 ± 14.5	33.1 ± 31.0	22.0 ± 17.2
Simple workflow								
Sonnet	P	8.0 ± 27.2	0.949 ± 0.121	0.110 ± 0.289	94.9 ± 12.0	38.7 ± 18.1	94.9 ± 12.1	38.7 ± 18.1
Llama4	P	14.7 ± 35.5	0.840 ± 0.178	0.438 ± 0.452	82.2 ± 23.4	55.6 ± 27.2	67.1 ± 35.8	43.8 ± 28.7
Gpt4.1	P	8.7 ± 28.2	0.892 ± 0.141	0.307 ± 0.429	86.9 ± 16.3	45.1 ± 21.2	75.3 ± 37.1	34.0 ± 24.6

Table 10: Model performance across complex, intermediate, and simple workflow.

Model	Format	Prompt Type	Exact-Match (%)	Count (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex workflow										
Llama4	J	react few shot	10.8 ± 31.0	91.2 ± 41.7	0.834 ± 0.183	0.831 ± 0.195	71.3 ± 23.4	68.8 ± 23.7	69.5 ± 25.8	66.8 ± 26.2
Llama4	J	react	15.2 ± 36.0	100.3 ± 37.8	0.877 ± 0.117	0.870 ± 0.135	66.2 ± 25.8	63.9 ± 25.9	60.8 ± 30.8	58.5 ± 30.6
Llama4	J	vanilla	12.5 ± 33.1	91.7 ± 40.7	0.817 ± 0.167	0.812 ± 0.181	65.4 ± 25.5	64.6 ± 26.0	58.9 ± 31.9	58.5 ± 32.0
Llama4	P	react few shot	20.0 ± 40.1	86.3 ± 31.0	0.898 ± 0.150	0.896 ± 0.154	83.3 ± 21.2	81.8 ± 21.4	80.7 ± 25.5	79.3 ± 25.3
Llama4	P	react	14.8 ± 35.5	84.9 ± 28.5	0.920 ± 0.123	0.924 ± 0.127	75.6 ± 25.0	73.5 ± 25.8	69.1 ± 32.3	67.3 ± 32.7
Llama4	P	vanilla	15.5 ± 36.2	82.7 ± 27.3	0.882 ± 0.155	0.887 ± 0.162	74.4 ± 25.1	73.7 ± 25.6	68.4 ± 32.2	68.1 ± 32.3
Sonnet	J	react few shot	41.8 ± 49.4	80.5 ± 30.0	0.944 ± 0.138	0.945 ± 0.136	96.4 ± 10.2	94.5 ± 11.9	96.4 ± 10.2	94.5 ± 11.9
Sonnet	J	react	38.5 ± 48.7	69.8 ± 34.2	0.975 ± 0.059	0.977 ± 0.059	93.6 ± 15.8	91.9 ± 16.9	92.9 ± 18.0	91.2 ± 18.9
Sonnet	J	vanilla	50.5 ± 50.1	80.4 ± 30.5	0.919 ± 0.186	0.919 ± 0.188	90.1 ± 22.0	88.8 ± 22.4	87.6 ± 26.4	86.5 ± 26.5
Sonnet	P	react few shot	19.5 ± 39.7	76.1 ± 32.1	0.925 ± 0.142	0.926 ± 0.139	88.3 ± 17.9	86.7 ± 18.3	85.9 ± 22.4	84.4 ± 22.5
Sonnet	P	react	16.5 ± 37.2	70.1 ± 34.2	0.954 ± 0.071	0.962 ± 0.064	84.0 ± 19.1	82.7 ± 20.4	75.4 ± 30.1	74.9 ± 30.1
Sonnet	P	vanilla	0.0 ± 0.0	69.6 ± 34.3	0.049 ± 0.071	0.031 ± 0.050	4.2 ± 7.1	1.0 ± 4.4	0.0 ± 0.0	0.0 ± 0.0
Intermediate workflow										
Llama4	J	react few shot	62.2 ± 48.6	96.4 ± 13.7	0.937 ± 0.132	0.911 ± 0.180	95.6 ± 15.2	94.4 ± 16.6	94.6 ± 18.6	92.9 ± 21.0
Llama4	J	react	43.1 ± 49.6	107.6 ± 58.1	0.919 ± 0.086	0.912 ± 0.138	92.9 ± 17.7	92.4 ± 18.1	92.5 ± 18.8	92.0 ± 19.1
Llama4	J	vanilla	39.1 ± 48.9	100.2 ± 21.4	0.917 ± 0.110	0.903 ± 0.170	87.4 ± 22.4	86.3 ± 23.3	87.1 ± 22.9	85.6 ± 24.6
Llama4	P	react few shot	56.0 ± 49.7	83.3 ± 23.6	0.652 ± 0.456	0.629 ± 0.469	64.6 ± 46.0	62.6 ± 46.7	63.7 ± 47.1	62.0 ± 47.2
Llama4	P	react	44.4 ± 49.8	85.6 ± 22.7	0.640 ± 0.449	0.627 ± 0.454	65.8 ± 46.5	65.5 ± 47.0	65.5 ± 47.0	65.5 ± 47.0
Llama4	P	vanilla	90.7 ± 29.2	100.0 ± 0.0	0.985 ± 0.064	0.971 ± 0.113	98.1 ± 8.7	96.8 ± 12.1	96.9 ± 15.1	95.2 ± 18.6
Sonnet	J	react few shot	26.2 ± 44.1	68.4 ± 24.2	0.827 ± 0.371	0.826 ± 0.374	82.2 ± 37.3	82.2 ± 37.3	81.6 ± 38.3	81.6 ± 38.3
Sonnet	J	react	59.6 ± 49.2	85.1 ± 24.8	0.968 ± 0.094	0.955 ± 0.149	96.3 ± 12.7	96.3 ± 12.7	94.2 ± 20.2	94.2 ± 20.2
Sonnet	J	vanilla	66.2 ± 47.4	90.4 ± 19.7	0.970 ± 0.075	0.964 ± 0.106	98.7 ± 7.5	98.7 ± 7.5	98.1 ± 11.4	98.1 ± 11.4
Sonnet	P	react few shot	45.3 ± 49.9	73.8 ± 25.0	0.636 ± 0.470	0.635 ± 0.476	62.8 ± 46.8	62.6 ± 47.0	59.1 ± 48.8	59.1 ± 48.8
Sonnet	P	react	35.6 ± 48.0	75.8 ± 29.2	0.600 ± 0.449	0.563 ± 0.462	57.2 ± 45.2	57.2 ± 45.2	54.9 ± 46.4	54.9 ± 46.4
Sonnet	P	vanilla	0.0 ± 0.0	66.4 ± 24.0	0.078 ± 0.098	0.035 ± 0.074	7.8 ± 9.8	1.5 ± 6.2	0.0 ± 0.0	0.0 ± 0.0
Simple workflow										
Llama4	J	react few shot	67.3 ± 47.1	100.0 ± 0.0	0.955 ± 0.065	0.948 ± 0.076	91.2 ± 13.3	91.2 ± 13.3	91.2 ± 13.3	91.2 ± 13.3
Llama4	J	react	96.0 ± 19.7	104.0 ± 19.7	0.999 ± 0.009	0.999 ± 0.012	99.7 ± 4.1	99.7 ± 4.1	99.7 ± 4.1	99.7 ± 4.1
Llama4	J	vanilla	76.0 ± 42.9	112.0 ± 38.3	0.967 ± 0.080	0.958 ± 0.110	94.5 ± 11.9	94.5 ± 11.9	93.8 ± 15.0	93.8 ± 15.0
Llama4	P	react few shot	60.0 ± 49.2	102.0 ± 14.0	0.946 ± 0.070	0.935 ± 0.085	90.5 ± 12.2	90.5 ± 12.2	90.5 ± 12.2	90.5 ± 12.2
Llama4	P	react	100.0 ± 0.0	100.0 ± 0.0	1.000 ± 0.000	1.000 ± 0.000	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Llama4	P	vanilla	99.3 ± 8.2	100.0 ± 0.0	1.000 ± 0.000	0.996 ± 0.054	100.0 ± 0.0	99.5 ± 6.1	100.0 ± 0.0	99.5 ± 6.1
Sonnet	J	react few shot	99.3 ± 8.2	100.0 ± 0.0	0.999 ± 0.012	0.999 ± 0.016	99.8 ± 2.0	99.8 ± 2.0	99.8 ± 2.0	99.8 ± 2.0
Sonnet	J	react	100.0 ± 0.0	100.0 ± 0.0	1.000 ± 0.000	1.000 ± 0.000	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Sonnet	J	vanilla	97.3 ± 16.2	100.0 ± 0.0	0.996 ± 0.023	1.000 ± 0.000	99.3 ± 4.0	99.3 ± 4.0	97.3 ± 16.2	97.3 ± 16.2
Sonnet	P	react few shot	100.0 ± 0.0	100.0 ± 0.0	1.000 ± 0.000	1.000 ± 0.000	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Sonnet	P	react	99.3 ± 8.2	100.0 ± 0.0	0.999 ± 0.012	1.000 ± 0.000	99.8 ± 2.0	99.8 ± 2.0	99.3 ± 8.2	99.3 ± 8.2
Sonnet	P	vanilla	93.3 ± 25.0	100.0 ± 0.0	0.990 ± 0.039	1.000 ± 0.000	98.3 ± 6.3	98.3 ± 6.3	93.3 ± 25.0	93.3 ± 25.0

Table 11: Model performance across complex, intermediate, and simple workflow.

A.5 Simple Workflow - Check Order Status

JSON Format

```
{
    "agent": "check_order_status",
    "steps": [
        "ask_for_order_id() -> [order_id]",
        "get_order_status(order_id = user_provided_info['order_id']) -> [status]",
        "return_order_status(order_status = order_status)",
        "close_case(order_id = user_provided_info['order_id'])"
    ],
    "soft_ordering": [],
    "conditionals": []
}
```

Natural Language (PY) Format

```
- Ask the user for their order ID using ask_for_order_id().
- Look up the order status by calling get_order_status(order_id = user_provided_
    info['order_id']).
- Inform the user of their current order status with return_order_status(order_
    status = order_status).
- Finally, mark the request as complete by calling close_case(order_id = user_
    provided_info['order_id']).
```

A.6 Simple Workflow - Check Product Availability

JSON Format

```
{
  "agent": "check_product_availability",
  "steps": [
    "ask_for_product_id() -> [product_id]",
    "check_inventory(product_id = user_provided_info['product_id']) -> [availability]",
    "return_product_availability(product_id = user_provided_info['product_id'],
    availability = inventory_info[user_provided_info['product_id']]['availability'])",
    "close_case(customer_id = customer_id)"
  ],
  "soft_ordering": [],
  "conditionals": []
}
```

Natural Language (PY) Format

```
- Ask the user for the product ID by calling `ask_for_product_id()`.
- Check inventory by invoking `check_inventory(product_id = user_provided_info['
    product_id'])`, which returns availability.
- Return the products availability by calling
  `return_product_availability(product_id = user_provided_info['product_id'],
                                     availability = inventory_info[user_provided_info['
                                          product_id']]['availability'])`.
- Finally, wrap up the interaction with `close_case(customer_id = customer_id)`.
```

A.7 Simple Workflow - Resend Email Request

JSON Format


```
{
  "agent": "resend_email_receipt",
  "steps": [
    "ask_for_order_id() -> [order_id]",
    "check_order_exists(order_id = user_provided_info['order_id']) -> [exists]",
    "send_email_receipt(order_id = user_provided_info['order_id'])",
    "escalate_to_support(order_id = user_provided_info['order_id'])",
    "complete_case(customer_id = customer_id)"

  ],
  "soft_ordering": [],
  "conditionals": [
    {
      "if": [
        {
          "field": "user_provided_info['order_id']",
          "operator": "==",
          "compare_to": "order_id"
        }
      ],
      "then": [{"action": "skip", "target": "escalate_to_support"}],
      "else": [{"action": "skip", "target": "send_email_receipt"}]
    }
  ]
}
```


Natural Language (PY) Format

```
- Begin by asking the user for their order ID using ask_for_order_id().
- Check if the order exists by calling check_order_exists(order_id = user_
    provided_info['order_id']).
     - If the "user_provided_info['order_id']" matches the number in 'order_id',
         proceed to send the receipt via email using send_email_receipt(order_id =
          user_provided_info['order_id']).
     - If they do not match match, escalate the issue to support using escalate_to
         _support(order_id = user_provided_info['order_id']).
- Finally, mark the case as complete by calling complete_case(customer_id =
    customer_id).
```

A.8 Intermediate Workflow - Account Suspension Request

JSON Format

```
{
  "agent": "account_suspension_request",
  "steps": [
    "ask_suspension_type() -> [suspension_type]",
    "ask_suspension_reason() -> [reason]",
    "get_user_status(employee_id = employee_id) -> [status]",
    "notify_already_suspended(employee_id = employee_id)",
    "ask_reactivation_date() -> [reactivation_date]",
    "suspend_account(employee_id = employee_id, type = user_provided_info['suspension_type'],
    reason = user_provided_info['suspension_reason'])",
    "send_suspension_confirmation(employee_id = employee_id)",
    "close_case(suspension_id = suspension['suspension_id'])"
  ],
  "soft_ordering": [
    ["ask_suspension_type", "ask_suspension_reason"]
  ],
  "conditionals": [
    {
      "if": [
        {
          "field": "suspension['suspension_status']",
          "operator": "==",
          "value": "suspended"
        }
      ],
      "then": [
        {
          "action": "end_after",
          "target": "notify_already_suspended"
        }
      ],
      "else": [
        {
          "action": "skip",
          "target": "notify_already_suspended"
        }
      ]
    },
    {
      "if": [
        {
          "field": "user_provided_info['suspension_type']",
          "operator": "!=",
          "value": "temporary"
        }
      ],
      "then": [
        {
          "action": "skip",
          "target": "ask_reactivation_date"
        }],
      "else": [
        {
          "action": "override_params",
          "target": "suspend_account",
          "params": {
            "employee_id": "employee_id",
            "type": "user_provided_info['suspension_type']",
            "reason": "user_provided_info['suspension_reason']",
            "reactivation_date": "user_provided_info['reactivation_date']"
          }}
      ]
    }
  ]
}
```

Natural Language (PY) Format

```
1. Ask the user which type of suspension they need (temporary or permanent) by
    calling `ask_suspension_type()`.

2. Ask the user to explain their reason for suspension by calling `ask_suspension
    _reason()`.
   *(Steps 1 and 2 can happen in any order, but both must be completed before
      moving forward.)*

3. Retrieve the  u s e r s  current suspension status by calling `get_user_status(
    employee_id = employee_id)`.

4. If the suspension['suspension_status'] is already "suspended":
   - Call `notify_already_suspended(employee_id = employee_id)` to inform the
      user.
   - End the process here.

5. If the suspension type is **temporary**:
   - Ask for the desired reactivation date by calling `ask_reactivation_date()`.

6. Call `suspend_account(...)` with the following parameters:
   - `employee_id = employee_id`
   - `type = user_provided_info['suspension_type']`
   - `reason = user_provided_info['suspension_reason']`
   - If the suspension is temporary, also include `reactivation_date = user_
      provided_info['reactivation_date']`.

7. Send a confirmation message by calling `send_suspension_confirmation(employee_
    id = employee_id)`.

8. Close the case by calling `close_case(suspension_id = suspension['suspension_
    id'])`.
```

A.9 Intermediate Workflow - Submit Time Off Request

JSON Format

```
{
    "agent": "submit_time_off_request",
    "steps": [
        "ask_for_pto_dates() -> [start_date, end_date]",
        "get_pto_balance(employee_id = employee_id) -> [pto_balance]",
        "inform_employee_balance_low()",
        "check_conflicts(start_date = user_provided_info['start_date'],
        end_date = user_provided_info['end_date'], pto_balance = vacation['pto_balance'])
        -> [conflict_status]",
        "inform_employee_conflict()",
     "submit_leave_request(employee_id = employee_id, start_date = user_provided_info['start_date'],
        end_date = user_provided_info['end_date'])
        -> [leave_request_id]",
      "notify_manager(manager_id = manager_id, leave_request_id = vacation['leave_request_id']) ->
        [manager_notification_status]",
     "send_confirmation(employee_id = employee_id, leave_request_id = vacation['leave_request_id']) ->
        [confirmation_status]",
        "close_case(leave_request_id = vacation['leave_request_id'])"
    ],
    "soft_ordering": [["ask_for_pto_dates", "get_pto_balance" ]],
    "conditionals": [
        {
            "if": [
                {
                    "field": "vacation['pto_balance']",
                    "operator": "<",
                    "value": 1
                }
            ],
            "then": [
                {
                    "action": "end_after",
                    "target": "inform_employee_balance_low"
                }
            ],
            "else": [
                {
                    "action": "skip",
                    "target": "inform_employee_balance_low"
                }
            ]
        },
        {
            "if": [
                {
                    "field": "conflict_status",
                    "operator": "==",
                    "value": true
                }
            ],
            "then": [
                {
                    "action": "end_after",
                    "target": "inform_employee_conflict"
                }
            ],
            "else": [
                {
                    "action": "skip",
                    "target": "inform_employee_conflict"
                }]}]}
```

Natural Language (PY) Format

```
- Begin by asking the user for their desired time off dates using ask_for_pto_
    dates(). This returns start_date and end_date.
- Retrieve the employee's current PTO balance using get_pto_balance(employee_id =
     employee_id).
    - If vacation['pto_balance'] is less than 1, inform the employee their
        balance is too low using inform_employee_balance_low(), then end the
        trajectory.
- Check for any scheduling conflicts by calling check_conflicts(start_date = user
    _provided_info['start_date'], end_date = user_provided_info['end_date'], pto_
    balance = vacation['pto_balance']).
    - If conflict_status is true, notify the employee about the conflict using
        inform_employee_conflict(), then end the trajectory.
- If there are no issues, submit the leave request using submit_leave_request(
    employee_id = employee_id, start_date = user_provided_info['start_date'], end
    _date = user_provided_info['end_date']). This returns a leave_request_id.
- Notify the employee's manager about the request using notify_manager(manager_id
     = manager_id, leave_request_id = vacation['leave_request_id']).
- Send a confirmation to the employee with send_confirmation(employee_id =
    employee_id, leave_request_id = vacation['leave_request_id']).
- Finally, close the case using close_case(leave_request_id = vacation['leave_
    request_id']).

Note on Soft Ordering: You can either call ask_for_pto_dates() first and then get
    _pto_balance(), or do it the other way around; the order of those two
    functions doesnt matter.
```

A.10 Intermediate Workflow - Update Address

JSON Format

```
{
    "agent": "update_address",
    "steps": [
        "get_employment_details(employee_id = employee_id) -> [employment_type, employee_status]",
        "validate_address(address = user_provided_info['address']) -> [validation_status]",
        "escalate_to_hr(employee_id = employee_id)",
     "update_employee_address(employee_id = employee_id, address = user_provided_info['address']) ->
        [notification_status]",
        "notify_payroll(employee_id = employee_id) -> [notification_status]",
        "check_contact_info(employee_id = employee_id) -> [has_contact_info]",
     "update_contact_info(employee_id = employee_id, new_phone = user_provided_info['new_phone']) ->
        [phone_update_status]",
        "complete_case(employee_id = employee_id)"
    ],
    "soft_ordering": [],
    "conditionals": [
        {
            "if": [
                {
                    "field": "validation_status",
                    "operator": "==",
                    "value": "invalid"
                }
            ],
            "then": [
                {
                    "action": "end_after",
                    "target": "escalate_to_hr"
                }
            ],
            "else": [
                {
                    "action": "skip",
                    "target": "escalate_to_hr"
                }
            ]
        },{
            "if": [
                {
                    "field": "employment_type",
                    "operator": "not in",
                    "value": [
                        "Full Time"
                    ]
                }
            ],
            "then": [
                {
                    "action": "skip",
                    "target": "notify_payroll"
                }
            ]
        },{
            "if": [
                {
                    "field": "has_contact_info",
                    "operator": "==",
                    "value": false
                }
            ],
            "then": [
                {
                    "action": "skip",
                    "target": "update_contact_info"
                }]}]}
```

Natural Language (PY) Format

```
- Start by retrieving the user's employment details using get_employment_details(
    employee_id = employee_id), which returns employment_type and employee_status
    .
- Validate the new address using validate_address(address = user_provided_info['
    address']).
    - If validation_status is "invalid", escalate the issue to HR by calling
        escalate_to_hr(employee_id = employee_id), then end the trajectory.
- If the address is valid, update the employees address using update_employee_
    address(employee_id = employee_id, address = user_provided_info['address']).
- If the employee's employment_type is "Full Time", notify the payroll team using
     notify_payroll(employee_id = employee_id). Otherwise, skip this step.
- Check if the employee has contact information by calling check_contact_info(
    employee_id = employee_id), which returns has_contact_info.
    - If has_contact_info is false, skip updating the contact info.
    - Otherwise, update the phone number using update_contact_info(employee_id =
        employee_id, new_phone = user_provided_info['new_phone']).
- Finally, mark the case as complete using complete_case(employee_id = employee_
    id).
```

A.11 Complex Workflow - Book Flight

JSON Format

```
{
  "agent": "book_flight",
  "steps": [
    "ask_for_basic_flight_details() -> [origin, destination, departure_date, return_date]",
    "get_customer_preferences(customer_id = customer_id) -> [cabin_preference, seat_preference]",
   "get_customer_frequent_traveler_status(customer_id = customer_id) -> frequent_traveler_status",
    "search_regular_flights(origin = user_provided_info['origin'],
    destination = user_provided_info['destination'], departure_date =
    user_provided_info['departure_date'],
    return_date = user_provided_info['return_date'], cabin_preference =
    user_provided_info['cabin_preference'], seat_preference =
    user_provided_info['seat_preference']) ->
    [flight_number]",
    "search_priority_flights(origin = user_provided_info['origin'], destination =
    user_provided_info['destination'], departure_date = user_provided_info['departure_date'],
    return_date = user_provided_info['return_date'], cabin_preference =
   user_provided_info['cabin_preference'], seat_preference = user_provided_info['seat_preference'])
    ->[flight_number]",
    "get_passport_visa_info(customer_id = customer_id)",
    "check_visa_requirements(customer_id = customer_id,
    destination = user_provided_info['destination']) -> [visa_status]",
    "get_customer_payment_method(customer_id = customer_id) -> [payment_method]",
    "create_booking(flight_number = user_provided_info['flight_number']) -> [booking_id]",
   "create_booking_with_points(flight_number = user_provided_info['flight_number']) -> [booking_id]",
    "add_special_services(booking_id = booking_info['booking_id'],
    service_type = traveler_info['special_assistance'])",
   "notify_airport_ground_team(customer_id = customer_id, booking_id = booking_info['booking_id'],
    service_type =
    traveler_info['special_assistance'])",
    "complete_case(customer_id = customer_id)"],
  "soft_ordering": [],
  "conditionals": [{
      "if": [
        {"field": "traveler_info['frequent_traveler_status']", "operator": "==", "value": null}],
      "then": [{ "action": "skip", "target": "search_priority_flights" }],
    "else": [{ "action": "skip", "target": ["search_regular_flights", "get_passport_visa_info"] }]},{
      "if": [{
          "field": "payment_method['payment_type']",
          "operator": "==",
          "value": "Points" }],
      "then": [{ "action": "skip", "target": "create_booking" }],
      "else": [{ "action": "skip", "target": "create_booking_with_points" }]},{
      "if": [{
          "all_of": [
            {
              "field": "traveler_info['frequent_traveler_status']",
              "operator": "in",
              "value": ["Gold", "Platinum"]
            },{
              "field": "traveler_info['special_assistance']",
              "operator": "!=",
              "value": null}]}],
      "then": [],
      "else": [{ "action": "skip", "target": "notify_airport_ground_team"}]},
    {
      "if": [
        {"field": "traveler_info['special_assistance']",
          "operator": "==",
          "value": null}],
      "then": [{ "action": "skip", "target": "add_special_services" }]},{
    "if": [{"field": "traveler_info['is_blacklisted']",
          "operator": "==",
          "value": true}],
      "then": [{ "action": "end_after", "target": "check_visa_requirements" }]}]}
```

Natural Language (PY) Format

```
## Step 1: Ask for Basic Flight Details
- Call the ask_for_basic_flight_details() function to ask the customer for:
    Origin, Destination, Departure date, and Return date.
## Step 2: Retrieve Customer Preferences
- Call `get_customer_preferences(customer_id = customer_id)` to check if the
    customer has preferences for the flight booking.
## Step 3: Check Frequent Traveler Status
- Call `get_customer_frequent_traveler_status(customer_id = customer_id)` to
    determine if the customer is a frequent traveler.
  - **If frequent traveler status is None**:
    - Proceed to Step 4 (Search Regular Flights).
  - **If frequent traveler status is not None**:
    - Skip Step 4 and Step 6.
    - Proceed to Step 5 (Search Priority Flights).
## Step 4: Search Regular Flights (Only if not a frequent traveler)
- Call `search_regular_flights(origin = user_provided_info['origin'], destination
     = user_provided_info['destination'], departure_date = user_provided_info['
    departure_date'], return_date = user_provided_info['return_date'], cabin_
    preference = user_provided_info['cabin_preference'], seat_preference = user_
    provided_info['seat_preference'])`.
- Proceed to Step 6.
## Step 5: Search Priority Flights (Only if frequent traveler)
- Call `search_priority_flights(origin = user_provided_info['origin'],
    destination = user_provided_info['destination'], departure_date = user_
    provided_info['departure_date'], return_date = user_provided_info['return_
    date'], cabin_preference = user_provided_info['cabin_preference'], seat_
    preference = user_provided_info['seat_preference'])`.
- Proceed to Step 7.
## Step 6: Check Passport and Visa Requirements (Only for non-frequent travelers)
- Call `get_passport_visa_info(customer_id = customer_id)` to retrieve passport
    and visa information.
- Then call `check_visa_requirements(customer_id = customer_id, destination =
    user_provided_info['destination'])` to determine if a visa is required.
  - **If the customer is blacklisted**: End the flow after this step and notify
      the customer accordingly.
  - **Otherwise**: Inform the customer about the visa requirement status.
- Proceed to Step 7.
## Step 6: Retrieve Passport and Visa Information
Call get_passport_visa_info(customer_id = customer_id) to retrieve passport and
    visa information.
## Step 7: Check Visa Requirements
Call check_visa_requirements(customer_id = customer_id, destination = user_
    provided_info['destination']) to determine if a visa is required.
If the customer is blacklisted (traveler_info['is_blacklisted'] is true): End the
     flow after this step and notify the customer accordingly.
## Step 8: Retrieve Payment Method and Create Booking
- Call `get_customer_payment_method(customer_id = customer_id)` to get the
    customers payment method.
  - **If the payment method is 'Points'**: Call `create_booking_with_points(
      flight_number = user_provided_info['flight_number'])`.
  - **Otherwise**: Call `create_booking(flight_number = user_provided_info['
      flight_number'])`.
- Proceed to Step 9.
## Step 9: Add Special Services
- **If the customer has listed any special assistance needs**: Call `add_special_
    services(booking_id = booking_info['booking_id'], service_type = traveler_
    info['special_assistance'])",`.
- Proceed to Step 10.
## Step 10: Notify Airport Ground Team
- **If the customer is Gold or Platinum frequent traveler AND has special
    assistance needs**:
  - Call `notify_airport_ground_team(customer_id = customer_id, booking_id =
      booking_info['booking_id'], service_type = traveler_info['special_
      assistance'])`.
## Step 11: Final Confirmation and Case Completion
- Share the booking ID and confirmation details with the customer.
- Call `complete_case(customer_id = customer_id)` to finalize the process.
- Thank the customer: "Thank you for booking with us. Have a pleasant journey!"
```

A.12 Complex Workflow - Cancel Flight

JSON Format

```
{
  "agent": "cancel_flight",
  "steps": [
   "get_customer_loyalty_info(customer_id = customer_id) -> [frequent_flyer_status, loyalty_points]",
    "get_booking_details(customer_id = customer_id) -> [booking_id, booking_date,
    payment_method, total_paid, is_refundable, purchased_insurance, booking_channel]",
    "check_cancellation_policy(booking_id = booking_info['booking_id']) -> [is_refundable]",
    "calculate_cancellation_fee(booking_id = booking_info['booking_id']) -> [cancellation_fee]",
    "waive_cancellation_fee(loyalty_points = traveler_info['loyalty_points'], booking_id =
    booking_info['booking_id']) -> [fee_waived]",
    "offer_alternate_flight_options(customer_id = customer_id, original_booking_id =
    booking_info['booking_id']) -> [flight_options]",
    "process_flight_change(old_booking_id = booking_info['booking_id'])",
    "cancel_flight(booking_id = booking_info['booking_id'])",
   "get_customer_payment_method(customer_id = customer_id, booking_id = booking_info['booking_id']) ->
    [payment_method]",
    "process_refund(booking_id = booking_info['booking_id'], payment_method =
    payment_method['payment_type'])",
    "issue_travel_credit(customer_id = customer_id, amount = booking_info['total_paid'])",
    "complete_case(customer_id = customer_id)"
  ],
  "soft_ordering": [
    ["get_customer_loyalty_info", "get_booking_details"],
      ["check_cancellation_policy", "calculate_cancellation_fee"]
  ],
  "conditionals": [
     {"if": [{
          "field": "user_provided_info['change_flight']",
          "operator": "==",
          "value": true
        }],
      "then": [{ "action": "skip", "target": ["cancel_flight", "get_customer_payment_method",
      "process_refund", "issue_travel_credit"] }
      ],
      "else": [{ "action": "skip", "target": ["process_flight_change"] }]},
    {"if": [
        {
          "any_of": [
            { "field":"booking_info['is_refundable']",
                "operator":"==",
                "value": true },
            { "field":"booking_info['purchased_insurance']",
                "operator":"==",
                "value": true }
          ]}],
      "then": [{ "action":"skip", "target":"issue_travel_credit" }
      ],
      "else": [{ "action":"skip", "target":"process_refund" }]},
    {"if": [{
          "field": "traveler_info['loyalty_points']",
          "operator": ">=",
          "value": 10000}],
      "then": [{ "action": "override_trajectory", "target": ["get_customer_loyalty_info",
      "get_booking_details", "waive_cancellation_fee", "cancel_flight", "process_refund",
      "complete_case"]}],
      "else": [{ "action": "skip", "target": ["waive_cancellation_fee"] }]}]}
```

Natural Language (PY) Format

```
## Step 1: Retrieve Customer Loyalty Information
- Call `get_customer_loyalty_info(customer_id = customer_id)` to retrieve:
  - **Frequent flyer status**
  - **Loyalty points**
## Step 2: Retrieve Booking Details
- Call `get_booking_details(customer_id = customer_id)` to retrieve:
  - **Booking ID**, booking date, payment method, total paid
  - **Is refundable**, purchased insurance, booking channel
## Step 3: Shortcut for High Loyalty Customers
- If `traveler_info['loyalty_points'] >= 10000`:
  - **Override the trajectory**: perform only:
    1. `get_customer_loyalty_info`
    2. `get_booking_details`
    3. `waive_cancellation_fee`
    4. `cancel_flight`
    5. `process_refund`
    6. `complete_case`
  - **Skip** all other steps (Steps 4, 5, 7, 9, 11).
  - Then return from the routine.
## Step 4: Check Cancellation Policy
- Call `check_cancellation_policy(booking_id = booking_info['booking_id'])` to
    determine if the booking is refundable.
  - **Note**: Can be done before or after Step 5 per soft ordering.
## Step 5: Calculate Cancellation Fee
- Call `calculate_cancellation_fee(booking_id = booking_info['booking_id'])` to
    retrieve the fee amount.
- If `traveler_info['loyalty_points'] < 10000`, **skip** Step 6 and proceed to
    Step 7.
## Step 6: Waive Cancellation Fee
- Call `waive_cancellation_fee(loyalty_points = traveler_info['loyalty_points'],
    booking_id = booking_info['booking_id'])` to waive the fee.
  - **Only executed if** `traveler_info['loyalty_points'] >= 10000`. Otherwise
      skipped.
## Step 7: Offer Flight Change Option
- Call `offer_alternate_flight_options(customer_id = customer_id, original_
    booking_id = booking_info['booking_id'])` to offer alternatives.
- If `user_provided_info['change_flight'] == True`:
  - Call `process_flight_change(old_booking_id = booking_info['booking_id'])`.
  - **Skip** the following:
    - Step 8: `cancel_flight`
    - Step 9: `get_customer_payment_method`
    - Step 10: `process_refund`
    - Step 11: `issue_travel_credit`
  - Then return from the routine.
- Else:
  - Continue to Step 8.
## Step 8: Cancel Flight
- Call `cancel_flight(booking_id = booking_info['booking_id'])` to finalize
    cancellation.
## Step 9: Retrieve Payment Method
- Call `get_customer_payment_method(customer_id = customer_id, booking_id =
    booking_info['booking_id'])` to determine the original payment type.
## Step 10: Process Refund
- If `booking_info['is_refundable'] == True` **or** `booking_info['purchased_
    insurance'] == True`:
  - Call `process_refund(booking_id = booking_info['booking_id'], payment_method
      = payment_method['payment_type'])`.
  - **Skip** Step 11.
- Else:
  - **Skip** this step (Step 10) and proceed to Step 11.
## Step 11: Issue Travel Credit
- Call `issue_travel_credit(customer_id = customer_id, amount = booking_info['
    total_paid'])` to issue credit.
  - **Only executed if** booking is non refundable and no insurance. Otherwise
      skipped.
## Step 12: Complete the Case
- Call `complete_case(customer_id = customer_id)` to mark the process as complete
    .
**Note on Soft Ordering:**
- You may call `get_customer_loyalty_info` before or after `get_booking_details`.
- You may call `check_cancellation_policy` before or after `calculate_
    cancellation_fee`.
```

A.13 Complex Workflow - Flight Disruption

JSON Format


```
{"agent": "handle_flight_disruption",
  "steps": ["get_booking_details(customer_id=customer_id) -> [booking_id, origin, destination]",
    "check_flight_status(flight_number=
    booking_info['flight_number'], flight_date=booking_info['flight_date'])
    -> [status, estimated_delay_minutes, delay_reason]",
   "notify_customer_disruption(customer_id=customer_id, flight_number=booking_info['flight_number'],
   status = flight_info['status'], delay_reason=flight_info['delay_reason'], estimated_delay_minutes =
    flight_info['estimated_delay_minutes'])",
    "ask_rebooking_preference(customer_id=customer_id) -> [wants_rebook]",
   "search_alternate_flights(origin=booking_info['origin'], destination=booking_info['destination'],
    flight_date=booking_info['flight_date'],
    cabin_class=booking_info['cabin_class']) -> [alternate_flights]",
    "offer_flight_options_to_customer(customer_id=customer_id, flights=
    search_results['alternate_flights']) ->[selected_flight_id]",
    "create_rebooking(original_booking_id=booking_info['booking_id'], new_flight_id=
    user_provided_info['selected_flight_id']) -> [new_booking_id, fare_difference]",
   "process_fare_difference(customer_id=customer_id, fare_difference=search_results['fare_difference'])",
"check_overnight_need(estimated_delay_minutes=flight_info['estimated_delay_minutes']) ->
[needs_overnight_accommodation]",
    "arrange_accommodation(customer_id=customer_id) -> [hotel_booking_id]",
   "arrange_transport(customer_id=customer_id, hotel_booking_id=search_results['hotel_booking_id'])",
    "issue_meal_vouchers(customer_id=customer_id, delay=flight_info['estimated_delay_minutes']) ->
    [voucher_codes]",
    "offer_compensation(customer_id=customer_id, delay_reason=flight_info['delay_reason']) ->
    [compensation_details]",
    "complete_case(customer_id=customer_id)"],
  "soft_ordering": [["arrange_accommodation", "arrange_transport"]],
  "conditionals": [{
      "if": [{"field": "flight_info['status']", "operator": "==", "value": "On Time"}],
      "then": [{"action": "override_params", "target": "notify_customer_disruption", "params": {
              "customer_id": "customer_id",
              "flight_number": "booking_info['flight_number']",
              "status": "flight_info['status']"}},
        { "action": "end_after", "target": "notify_customer_disruption" }}},
        {"if": [{
          "field": "flight_info['status']",
          "operator": "==",
          "value": "Cancelled"}],
      "then": [{"action": "override_params", "target": "notify_customer_disruption", "params": {
              "customer_id": "customer_id",
              "flight_number": "booking_info['flight_number']",
              "status": "flight_info['status']",
              "delay_reason": "flight_info['delay_reason']"}}]},
    {"if": [{"all_of": [
            {"field": "flight_info['status']", "operator": "==", "value": "Cancelled"},
            {"field": "flight_info['delay_reason']", "operator": "in", "value": ["Mechanical",
            "Crew Issue"]}]}],
      "then": [{ "action": "override_trajectory",
         "target": ["get_booking_details", "offer_flight_options_to_customer", "create_rebooking",
       "arrange_accommodation", "arrange_transport", "offer_compensation", "update_loyalty_points",
          "complete_case"]}]},
         {"if": [{"field": "user_provided_info['wants_rebook']","operator": "==","value": false}],
      "then": [{"action": "skip","target": ["search_alternate_flights",
      "offer_flight_options_to_customer","create_rebooking","process_fare_difference"]}]},
    {"if": [{"field": "flight_info['estimated_delay_minutes']","operator": "<","value": 360}],
      "then": [{ "action": "skip", "target":["arrange_accommodation", "arrange_transport",
      "issue_meal_vouchers"]}]},
   {"if": [{"all_of": [{"field": "traveler_info['frequent_traveler_status']","operator": "in","value":
          ["Gold", "Platinum", "Diamond"]},{"field": "flight_info['delay_reason']",
          "operator": "!=","value": "Weather"}]}],
    "then": [{"action": "override_params", "target": "offer_compensation","params": { "customer_id":
      "customer_id", "delay_reason": "flight_info['delay_reason']","extra_miles":
      "booking_info['compensation_allowed']"}}]},{
      "if": [{ "field": "flight_info['delay_reason']","operator": "==", "value": "Weather"}],
      "then": [{"action": "skip","target":["offer_compensation"]}]}]}
```

Natural Language (PY) Format

```
Step 1: Retrieve Booking Details
- Call get_booking_details(customer_id=customer_id) and capture booking_id and
    origin & destination
Step 2: Check Flight Status
- Call check_flight_status(flight_number=booking_info['flight_number'], flight_
    date=booking_info['flight_date']) and capture: status ( On  Time ,
       Delayed  ,  Cancelled  ), estimated_delay_minutes, delay_reason (if
    cancelled)
Step 3: Notify the Customer of the Disruption
- Call notify_customer_disruption() with the following parameters based on the
    value of flight_info['status']".
- If flight_info['status']" is On Time, use parameters: customer_id=customer_id,
    flight_number=booking_info['flight_number'], status=flight_info['status'])
    and end the flow here.
- If flight_info['status'] is Cancelled, use parameters: customer_id=customer_id,
     flight_number=booking_info['flight_number'], status = flight_info['status'],
     delay_reason=flight_info['delay_reason']
- If flight_info['status'] is Delayed, use parameters: customer_id=customer_id,
    flight_number=booking_info['flight_number'], status = flight_info['status'],
    delay_reason=flight_info['delay_reason'], estimated_delay_minutes = flight_
    info['estimated_delay_minutes']
Step 4: Ask Rebooking Preference
- Call ask_rebooking_preference(customer_id=customer_id) and capture wants_rebook
    . - If user_provided_info['wants_rebook'] == false, skip Steps 5 8 .
Step 5: Search for Alternate Flights
- Call search_alternate_flights(origin=booking_info['origin'], destination=
    booking_info['destination'], flight_date=booking_info['flight_date'], cabin_
    class=booking_info['cabin_class'],) and capture alternate_flights
Step 6: Offer Flight Options
- Call offer_flight_options_to_customer(customer_id=customer_id, flights=search_
    results['alternate_flights']) and capture selected_flight_id
Step 7: Create the New Booking
- Call create_rebooking(original_booking_id=booking_info['booking_id'], new_
    flight_id=user_provided_info['selected_flight_id']) and capture new_booking_
    id and fare_difference
Step 8: Process Any Fare Difference
- Call process_fare_difference(customer_id=customer_id, fare_difference=search_
    results['fare_difference']).
Step 9: Check Overnight Accommodation Need
- Call check_overnight_need( estimated_delay_minutes=flight_info['estimated_delay
    _minutes']) and capture needs_overnight_accommodation
Steps 10 & 11: Arrange Hotel and Transport
- Only if flight_info['estimated_delay_minutes'] is over 360, call arrange_
    accommodation(customer_id=customer_id) and capture hotel_booking_id
- Call arrange_transport(customer_id=customer_id, hotel_booking_id=search_results
    ['hotel_booking_id']).
- (These two steps may execute in either order.)
Step 12: Issue Meal Vouchers
- If flight_info['estimated_delay_minutes'] under 360, skip this step.
- Otherwise, call issue_meal_vouchers(customer_id=customer_id, delay=flight_info
    ['estimated_delay_minutes']) and capture voucher_codes
Step 13: Offer Compensation
- Call offer_compensation(customer_id=customer_id, delay_reason=flight_info['
    delay_reason'],) and capture compensation_details.
- If traveler_info['frequent_traveler_status'] in ["Gold", "Platinum", "Diamond
    "], include extra_miles = booking_info['compensation_allowed'] in the
    parameters to become offer_compensation( customer_id=customer_id, delay_
    reason=flight_info['delay_reason'], extra_miles = booking_info['compensation_
    allowed'])
- If flight_info['status'] == "Cancelled" and flight_info['delay_reason'] in ["
    Mechanical", "Crew Issue"], override the trajectory to execute in order with
    the parameters defined above:
    1. get_booking_details()
    2. offer_flight_options_to_customer()
    3. create_rebooking()
    4. arrange_accommodation()
    5. arrange_transport()
    6. offer_compensation()
    7. update_loyalty_points()
    8. complete_case()
Step 14: Complete the Case
- Call complete_case(customer_id=customer_id).
```

A.14 User Data Example

User Data Example Provided to `Traxgen`


```
{
   "agent_sequence": [
     "submit_time_off_request"
   ],
   "employee_id": 2709079,
   "manager_id": 7215773,
   "conflict_status": false,
   "employment_type": "Full Time",
   "has_contact_info": false,
   "suspension": {
     "suspension_id": 601790,
     "suspension_status": "not suspended"
   },
   "vacation": {
     "leave_request_id": 191059,
     "pto_balance": 9
   },
   "validation_status": "valid",
   "user_provided_info": {
     "address": "12 Grimmauld Place, London, UK",
     "end_date": "2025-06-27",
     "new_phone": 6512227804,
     "reactivation_date": "2025-06-03",
     "start_date": "2025-06-12",
     "suspension_reason": "Leave of Absence",
     "suspension_type": "temporary"
   }
 }
```

A.15 Traxgen Trajectory Format

Traxgen Style

```
 [
  [
    "agent: assistant",
    "tool: ask_for_order_id()",
    "tool: get_order_status(order_id=63920)",
    "tool: return_order_status(order_status=Delivered)",
    "tool: close_case(order_id=63920)"
  ]
]
```

Google Style

```
 [[
{'tool_name': 'ask_for_order_id', 'tool_input': {}},
{'tool_name': 'get_order_status', 'tool_input': {'order_id': 63920}},
{'tool_name': 'return_order_status', 'tool_input': {'order_status': 'Delivered'}},
{'tool_name': 'close_case', 'tool_input': {'order_id': 63920}}
]]
```

Langchain Style

```
[
  [
    {
      "role": "assistant",
      "tool_calls": [
        { "name": "ask_for_order_id",    "arguments": {} }
      ]
    },
    {
      "role": "assistant",
      "tool_calls": [
        { "name": "get_order_status",    "arguments": { "order_id": 63920 } }
      ]
    },
    {
      "role": "assistant",
      "tool_calls": [
        { "name": "return_order_status", "arguments": { "order_status": "Delivered" } }
      ]
    },
    {
      "role": "assistant",
      "tool_calls": [
        { "name": "close_case",          "arguments": { "order_id": 63920 } }
      ]
    }
  ]
]
```

Tool-Only Style

```
 ['ask_for_order_id', 'get_order_status', 'return_order_status', 'close_case']
```

A.16 Annotator Instructions

Annotator Instructions

````
"""
# Trajectory Annotation Instructions

## Objective

You will review tool-call trajectories generated by our `TraxGen-py` toolkit to ensure they follow
the defined **routine logic** and are consistent with the provided **customer data**.

Each annotation task includes:
- A **routine** (structured JSON workflow)
- A **customer profile** (database-like JSON input)
- A **generated trajectory** (tool calls + parameters)

Your goal is to determine whether the generated trajectory **adheres to the policy** defined
in the routine and fully satisfies the task requirements.
---
## When to Mark as `Pass`
Mark the trajectory as `Pass` if all of the following conditions are met:

1. **All required tool calls** are present in the correct order (allowing for soft ordering if applicable
2. **Conditional logic** (`skip`, `end_after`, `override_trajectory`) is triggered appropriately based
on customer data.
3. **No extra tool calls** are included, unless explicitly allowed by the routine.
4. **Tool parameters** are fully and correctly filled using customer data and routine-defined rules.
5. In multi-agent workflows, each agent only calls tools defined in its assigned sub-intent.
---
## When to Mark as `Fail`

Mark the trajectory as `Fail` if any of the following issues are present:

- A required tool is **missing**.
- Tools are called in the **wrong order**, violating hard constraints.
- A conditional rule is **misapplied** (e.g., skipped when it should not be).
- A tool has **incorrect or missing parameters**.
- **Extra tools** are called that are not defined in the routine or allowed by policy.
- In multi-intent workflows, an agent calls tools outside its scope (**agent boundary violation**).
---
## Common Error Tags

If a trajectory is marked as `Fail`, please include one or more of the following tags:
| Tag               | Description                                                   |
|-------------------|---------------------------------------------------------------|
| `missing_tool`    | A required tool was not called.                               |
| `wrong_order`     | Tools were called in the incorrect order.                     |
| `wrong_condition` | A condition (e.g., `skip`, `end_after`) was applied wrongly.|
| `bad_param`       | Tool parameters were missing or incorrect.                    |
| `extra_tool`      | Unnecessary or invalid tool calls were included.              |
| `agent_violation` | A tool was used by the wrong agent in a multi-intent task.   |
---
##  Output Format

Each task should be annotated using this format:

```json
{
  "customer_id": "1802531",
  "annotator_id": "A1",
  "result": "fail",
  "tags": ["missing_tool", "bad_param"],
  "comments": "Missing confirmation step; booking ID param was null in 'GetFlightInfo'."
}
````
````

A.17 Acknowledgment

AI assistance is used in this paper.