`Traxgen`: Ground-Truth Trajectory Generation for AI Agent Evaluation

Maria Emilia Mazzolenis
The Vanguard Group
Charlotte, NC, USA
`emilia_mazzolenis@vanguard.com`

Ruirui Zhang
The Vanguard Group
Charlotte, NC, USA
`ruirui_zhang@vanguard.com`

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 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 with human-validated references and achieve average median speedups of over 17,000× compared to LLM-based methods. To probe LLM reasoning, we compared 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 underscore LLM planning limitations and establish `Traxgen` as a scalable, resource-efficient tool for evaluating planning-intensive AI agents in compliance-critical domains.

1 Introduction

Modern AI agents are expected to go beyond generating plausible responses; they are often required to 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. Trajectories can reveal whether an agent reasons effectively, selects appropriate tools, and respects 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 evaluation. To address this gap, we present an automated framework for generating and evaluating agent trajectories, enabling consistent benchmarking in 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, and platform-compatible formats, achieving orders-of-magnitude speedups over LLM-based generation with improved accuracy [1].

[1]Experimentation code is available here. Package available on PyPI.

- **An empirical study** of six LLMs' trajectory planning under diverse prompts, input formats, and inference strategies, evaluated on tasks spanning nine domains, three workflow complexities, and multi-agent, multi-intent scenarios.

2 Related Work

2.1 Evaluation Strategies for Agents

Evaluating multi-agent dialogue systems is challenging, requiring assessment of message quality, outcome correctness, and overall agent effectiveness. 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 evaluation. 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 capturing reasoning steps and intermediate tool calls [Qin et al., 2023]. ToolSandbox introduces Milestones and Minefields, events that must or must not occur, to track critical events in workflows [Lu et al., 2024].

Trajectory evaluation is essential for understanding agent performance in multi-step interactions. For instance, Agent WARPP [Mazzolenis and Zhang, 2025] relied on trajectory-level metrics such as execution fidelity, parameter fidelity, and tool accuracy to assess workflow adherence in multi-agent systems. 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 with proprietary workflows, trajectories often depend on user inputs and intermediate tool outputs, making them costly and slow to obtain at scale. This motivates the need for an automated framework that generates trajectories for systematic analysis and benchmarking.

2.2 Trajectory Ground Truth Generation

Evaluating AI agents in complex, tool-augmented tasks requires high-quality ground truth trajectories. Existing generation methods are often labor-intensive or error-prone and generally fall into two paradigms: human-in-the-loop LLM generation or fully automated LLM-driven approaches.

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., 2024] employs human annotators who incrementally create complex, branching scenarios from simpler cases, validated via LLM 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] integrates human-written examples and LLM-generated dialogues, with an emphasis on human curation.

Automated LLM-driven approaches reduce human involvement. APIGen-MT [Prabhakar et al., 2025] creates an LLM-reviewed blueprint for intent and API use, then collects trajectories by simulating human-agent interactions. ToolLLM [Qin et al., 2023] generates trajectories from instructions, tools, and execution examples with a DFS-based decision tree guided by LLM reasoning. While scalable and more cost-effective than ReAct-generated trajectories [Yao et al., 2022]), these methods can yield incomplete or incorrect trajectories as they rely on model's predictions for termination.

3 Traxgen

Unlike prior stochastic or hybrid approaches, we introduce a fully deterministic trajectory generation paradigm. `Traxgen` (MIT-license) transforms high-level workflow specifications and user profiles into trajectories specifying which agents invoke which tools, in what order, with all parameters set. Trajectories can vary across users based on conditional logic, tool availability, and user attributes.

`Traxgen` is designed for domains where compliance, auditability, and repeatability are critical, and where flexible or stochastic evaluation could introduce ambiguity or legal risk. By generating fully specified trajectories, `Traxgen` ensures agents are assessed against explicitly defined correct paths. This is particularly important in industries such as finance or healthcare, where exact adherence to workflows is required. Instructions on installation and usage are provided in 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 `Traxgen` workflow is a structured specification encoding a sequence of tool-based operations 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: Groups of steps that can execute in any order, allowing flexible sequencing and multiple valid trajectories via permutations. Although permutations grow factorially, groups rarely exceed 3–4 steps, as fine-grained actions are usually bundled into high-level operations.

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 section A.2 for all action definitions).

Example workflows appear in Appendix sections A.5 through A.13.

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 four trajectory formats (App. A.15) for interoperability with existing frameworks: (1) **Tool Only**, which lists only tool names without arguments; (2) **Google Style**, used in Google's Vertex AI evaluation service; (3) **LangChain Tool Style**, compatible with the LangChain evaluation ecosystem; and (4) `Traxgen` **Style**, which captures agent names and tool calls with arguments.

3.3 System Architecture

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

(1) Workflow Interpretation. Each workflow is parsed into an intermediate planner object that formalizes all valid tool sequences. Workflows are represented as (T, C, S), where T is the set of tools, C are conditional rules, and S are soft ordering blocks. Conditional rules evaluate boolean expressions over user data with operators $\{==, \neq, >, <, \geq, \leq, \in, \notin, \text{contains}, \text{not contains}\}$, triggering dynamic changes such as skipping tools, truncating sequences, overriding trajectories, or adjusting parameters.

(2) Trajectory Planning. `Traxgen` constructs a directed acyclic graph $G = (V, E)$ with tools as vertices and edges enforcing hard ordering. The process includes:

- **Node insertion:** Insert all candidate tools T from the workflow into the graph.

- **Conditional pruning:** Remove nodes whose execution is skipped or truncated according to conditional rules C, along with their incident edges.
- **Edge wiring:** Reconnect remaining nodes into a linear chain, enforcing precedences.
- **Cycle check:** Assert the graph remains acyclic to catch contradictory constraints.
- **Soft ordering:** For each soft-ordering block $S = \{s_1, \dots, s_k\}$, generate all intra-block permutations $\mathcal{P} = \prod_{i=1}^{k} \mathrm{Perm}(s_i)$, respecting hard constraints and pruning invalid sequences.

(3) Multi-Agent Composition. Given agents $A = [a_1, \dots, a_n]$, each agent's valid trajectories $\mathcal{T}_i$ are computed, and multi-agent sequences are formed by concatenating the agent trajectories in order.

For each user profile, multi-agent trajectories are generated in all requested formats. Optional visualizations of the dependency graph support debugging and analysis. A validation layer detects malformed workflows, invalid profiles, missing parameters, and unsupported APIs, ensuring correct and reproducible trajectory generation.

Algorithm 1 Traxgen Trajectory Generation

Require: User profiles U, workflows W, agents A, formats F, optional visualization flag v
Ensure: Trajectories $\mathcal{T}$

for all agent $a \in A$ **do**
    $(T, C, S) \leftarrow$ parse_workflow$(W[a])$
end for
for all user $u \in U$ **do**
    for all agent $a \in A$ **do**
        $G \leftarrow \mathrm{DAG}(T, C, S)$ ▷ Apply conditionals, ordering, permutations
    end for
    $T_u \leftarrow$ concatenate$(\{G_a\}_{a \in A})$
    for all $f \in F$ **do**
        $T_{u,f} \leftarrow$ format(T_u, f)
    end for
    if v **then**
        visualize$(\{G_a\}, T_u, A)$
    end if
    $\mathcal{T}[u] \leftarrow \{T_{u,f}\}_{f \in F}$
end for
return $\mathcal{T}$

4 Experimentation

4.1 Data Construction

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

Stage I: Workflow design. Workflows are manually defined using a control-flow language (e.g., `skip`, `end_after`, `override_trajectory`) to specify tool sequences, parameter bindings, and policy constraints. Three workflows were created for each of the three complexity tiers (see §4.2).

Stage II: User profile generation. For each workflow, we create a pool of diverse user 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 deterministic trajectory. Two blinded annotators validate whether each trajectory follows the workflow's policy logic and aligns with user data, using structured guidelines on tool order, parameter correctness, conditionals, and agent boundaries. A trajectory is marked valid only if all constraints are satisfied. While `Traxgen`-generated drafts seeded the process, all gold trajectories were independently reviewed by human annotators to avoid self-alignment.

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.

4.2 Data Distribution and Complexity Levels

Workflow Complexity. We categorize workflows as: simple (linear with few conditionals), intermediate (branching with optional soft ordering), and complex (nested conditionals, multiple soft orderings, heavy contextual dependence).

Data Distribution. To balance effort and coverage, we sample 100 profiles per complex intent, 75 per intermediate, and 50 per simple, capturing the greater diversity and error surface of complex workflows while maintaining metric stability. In total, the dataset includes 675 task instances and 71 tools, with 10% involving multi-intent cases.

4.3 General Experimentation Setup

We evaluate models on generating agent trajectories conditioned on user intent, profile, and workflow. Experiments vary prompting (Vanilla, ReAct [Yao et al., 2022], ReAct few-shot), input format (natural language vs. JSON), and workflow complexity. To benchmark Traxgen, we include multiple LLMs as baselines, since recent literature predominantly relies on LLMs for trajectory synthesis, and outputs from all approaches are compared to human-validated reference trajectories.

4.4 Evaluation Metrics

To handle multiple predicted and gold trajectories from 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 similarity metric. We then evaluate the aligned pairs.

Let $\mathcal{G}$ and $\mathcal{P}$ denote the sets of ground-truth and predicted trajectories, each a sequence of $(\text{tool}, \text{params})$ steps. We define metrics as follows:

Exact Match and Count Agreement We compute *Exact Match* as $\mathbf{1}(\mathcal{P} = \mathcal{G})$, and *Count Agreement* as $\frac{|\mathcal{P}|}{|\mathcal{G}|} \times 100\%$, capturing over- or under-prediction in the number of predicted trajectories.

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 (PRF) based on multiset overlap (ignoring order). Standard PRF metrics are reported separately for tools and parameter triplets.

Contiguous Overlap Length (CMR) 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 count toward Count Agreement, ensuring trajectory quality is evaluated independently of prediction quantity.

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

Table 2: Average runtime (seconds) per trajectory by model (Sec. 6) across workflow complexities.

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, comparing outputs to validated references using the metrics in 4.4. As a control, LLM baselines are prompted with either (a) the original JSON workflows or (b) equivalent natural-language descriptions, isolating the effect of structured input. The same six LLMs evaluated in Section 6 are used here for consistency, with full benchmarking results reported 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.

Across twelve LLM configurations (six models, each with JSON or natural-language workflow inputs), `Traxgen` outperforms on all 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 controlled experiments isolating the planning stage of tool use. The benchmark abstracts execution, focusing on the model's ability to generate policy-compliant trajectories from user instructions and structured workflows. Each task requires reasoning over user data and multi-step workflows, including selecting tools, binding parameters, and handling conditionals, in a single pass. We evaluate six diverse LLMs spanning architectures, openness, and scale: open models `DeepSeek-Chat-v3-0324` [Liu et al., 2024, DeepSeek AI, 2025], `Mistral-7B-Instruct` [Jiang et al., 2023, Mistral AI, 2023], `Llama-4-Maverick` [Touvron et al., 2023, Meta AI, 2025]; and proprietary ones `Gemini-2.0-Flash-001` [Team et al., 2024, Google DeepMind, 2025], `Claude-3.7-Sonnet` [Anthropic, 2025], `GPT-4.1` [Achiam et al., 2023, OpenAI, 2025]. Our setup follows plan-first evaluation protocols [Zheng et al., 2024], enabling assessment without interactive noise.

6.1 Experiment-Specific Setup

We conduct three controlled studies, each isolating a factor affecting trajectory-planning quality: *workflow representation*, *prompt design*, and *inference-time search*. Using the same nine workflows and evaluation metrics ensures observed differences are due to the factor under study.

Study 1: Input Representation (Natural Language vs. JSON). Trajectory planning often uses structured task representations (e.g., graphs, trees, JSON). To isolate the effect of structure, we compare model performance on (a) natural language workflow descriptions and (b) equivalent structured JSON (used in `Traxgen`) across the three complexity levels.

Study 2: Prompt Engineering. Prompting strategies shape model behavior in constrained reasoning tasks. We tested three designs: Vanilla (minimal instruction-only), ReAct (interleaved reasoning and actions, [Yao et al., 2022]), and ReAct + few-shot (ReAct augmented with a worked example matched to workflow complexity). Llama-4 Maverick (open) and Sonnet 3.7 (proprietary) were tested to balance coverage and depth.

Study 3: Direct Generation vs. Guided Search. We examine inference strategy using ToolLLM's DFSDT, a depth-first search decision-tree algorithm that augments LLMs with backtracking and branch exploration [Qin et al., 2023]. We replace live APIs with static simulations for deterministic, side-effect-free execution. The same LLM generates both ReAct-style direct trajectories (*Direct*) and DFSDT-guided trajectories (*Guided*), enabling a clean comparison of pure in-context planning versus search-augmented planning. We limited this study to 50 user trajectories per domain to capture trends while controlling for DFSDT's longer runtime.

6.2 Results

Trajectory Quality Evaluation LLM-generated trajectories often required cleaning before comparison to the ground truth. We used a Python script to standardize outputs, addressing issues such as markdown fences, bracket mismatches, and null literals. DeepSeek tended to hallucinate, returning unstructured code snippets. Across models, the most common tool errors were incorrect tool order, omitted steps, and extra steps. Parameter errors were mostly missing inputs; value mismatches were rare. Errors were concentrated in complex workflows, fewer in intermediate, and minimal in simple workflows. Detailed metrics appear in Appendix Table 9.

Model Comparison Model performance on complex workflows shows a stratification by model class and format, with differences confirmed using the Kruskal–Wallis test followed by pairwise Mann–Whitney U tests (Holm-corrected, $p < 0.05$). For both JSON and natural language, Gemini and Sonnet significantly outperform others across nearly all metrics. Sonnet has strong tool and parameter-level accuracy on complex workflows, while Gemini shows comparable or better performance on intermediate. 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 significantly across most metrics, particularly on complex workflows. These findings suggest that Gemini and Sonnet may be best suited for handling high-complexity, multi-step tasks in both formats.

Complexity Comparison. LLM performance declines as workflow complexity increases. Most models perform well on simple workflows, with top models achieving near-perfect scores. Performance becomes more variable on intermediate workflows, particularly for lower-performing models, and degrades further on complex workflows. Even the best models show noticeable drops in exact-match and CMR metrics on complex tasks, highlighting the growing challenge of higher-complexity workflows.

Input Representation Comparison Across models, we find that JSON and natural language inputs yield significantly different performance in most cases (Mann–Whitney U test with $p < 0.05$), indicating that input format choice has a robust effect. For intermediate workflow, JSON inputs consistently outperformed all other options across every model and metric. In contrast, simple workflow showed minimal sensitivity to input choice; performance differences were negligible and varied idiosyncratically by model. The most striking effects emerged in complex workflow, where input had a substantial impact: while JSON remained optimal for the most capable models (such as GPT-4.1 and Claude Sonnet), natural language yielded improvements for mid-tier and open-source models.

Prompt Engineering Method Comparison Prompt style impacts performance differently across workflow complexity and model type (Appendix Table 11). For simple workflows, all prompts achieved near-perfect exact-match and parameter F1, with slight gains for ReAct. For intermediate workflows, Vanilla prompts gave Llama-4 the highest exact-match in natural language, while Sonnet favored ReAct, highlighting model- and domain-specific sensitivity. In complex workflows, ReAct consistently led in Tool F1. Few-shot prompting did not reliably outperform simpler prompts, suggesting examples do not always aid constrained reasoning. Statistical testing shows that prompt engineering differences are not consistent: Sonnet with natural language benefits strongly from few-shot prompting, while Llama-4 often shows no significant difference or even favors vanilla prompts.

Model	Format	Exact-Match (%)	Count (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex Workflow									
Mistral	JSON	0.0 ± 0.0	69.8 ± 1.7	0.525 ± 0.017	0.414 ± 0.015	36.7 ± 1.5	29.4 ± 1.2	33.5 ± 1.5	18.3 ± 1.3
Deepseek	JSON	5.5 ± 1.1	73.9 ± 1.7	0.706 ± 0.015	0.659 ± 0.016	48.1 ± 1.5	46.6 ± 1.5	32.4 ± 1.8	27.2 ± 1.8
Gemini	JSON	11.5 ± 1.6	84.2 ± 1.8	0.759 ± 0.017	0.762 ± 0.017	67.1 ± 1.7	66.3 ± 1.7	57.1 ± 2.0	56.4 ± 2.0
Sonnet	JSON	38.5 ± 2.4	69.8 ± 1.7	0.975 ± 0.003	0.977 ± 0.003	93.6 ± 0.8	91.9 ± 0.8	92.9 ± 0.9	91.2 ± 0.9
Llama4	JSON	15.2 ± 1.8	100.3 ± 1.9	0.877 ± 0.006	0.870 ± 0.007	66.2 ± 1.3	63.9 ± 1.3	60.8 ± 1.5	58.5 ± 1.5
Gpt4.1	JSON	26.0 ± 2.2	70.4 ± 1.7	0.940 ± 0.005	0.938 ± 0.006	76.1 ± 1.2	75.2 ± 1.2	73.8 ± 1.4	73.1 ± 1.4
Mistral	NL	0.2 ± 0.2	69.8 ± 1.7	0.505 ± 0.016	0.432 ± 0.015	30.6 ± 1.2	24.0 ± 1.0	26.4 ± 1.2	12.1 ± 0.9
Deepseek	NL	13.5 ± 1.7	74.1 ± 1.7	0.775 ± 0.012	0.718 ± 0.015	58.1 ± 1.6	55.4 ± 1.6	43.3 ± 2.0	39.9 ± 2.1
Gemini	NL	23.8 ± 2.1	87.1 ± 1.3	0.914 ± 0.006	0.918 ± 0.007	77.6 ± 1.1	76.8 ± 1.2	65.1 ± 1.8	65.1 ± 1.8
Sonnet	NL	16.5 ± 1.9	70.1 ± 1.7	0.954 ± 0.004	0.962 ± 0.003	84.0 ± 1.0	82.7 ± 1.0	75.4 ± 1.5	74.9 ± 1.5
Llama4	NL	14.8 ± 1.8	84.9 ± 1.4	0.920 ± 0.006	0.924 ± 0.006	75.6 ± 1.2	73.5 ± 1.3	69.1 ± 1.6	67.3 ± 1.6
Gpt4.1	NL	16.5 ± 1.9	71.0 ± 1.7	0.930 ± 0.004	0.929 ± 0.004	72.6 ± 1.3	70.1 ± 1.3	65.5 ± 1.7	64.2 ± 1.6
Trajectory	PACKAGE	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
Intermediate Workflow									
Mistral	JSON	2.7 ± 1.1	67.3 ± 1.6	0.658 ± 0.019	0.566 ± 0.022	53.8 ± 1.9	46.5 ± 1.9	50.2 ± 2.1	34.0 ± 2.4
Deepseek	JSON	49.8 ± 3.3	81.8 ± 1.6	0.814 ± 0.019	0.743 ± 0.023	83.6 ± 2.0	75.3 ± 2.2	76.6 ± 2.7	60.8 ± 3.2
Gemini	JSON	76.9 ± 2.8	100.0 ± 0.0	0.972 ± 0.005	0.905 ± 0.014	98.5 ± 0.5	94.3 ± 1.0	98.5 ± 0.5	94.3 ± 1.0
Sonnet	JSON	59.6 ± 3.3	85.1 ± 1.7	0.968 ± 0.006	0.955 ± 0.010	96.3 ± 0.8	96.3 ± 0.8	94.2 ± 1.3	94.2 ± 1.3
Llama4	JSON	43.1 ± 3.3	107.6 ± 3.9	0.919 ± 0.006	0.912 ± 0.009	92.9 ± 1.2	92.4 ± 1.2	92.5 ± 1.3	92.0 ± 1.3
Gpt4.1	JSON	63.6 ± 3.2	81.8 ± 1.6	0.994 ± 0.003	0.988 ± 0.006	99.1 ± 0.4	99.1 ± 0.4	99.1 ± 0.4	99.1 ± 0.4
Mistral	NL	6.7 ± 1.7	67.1 ± 1.7	0.376 ± 0.026	0.325 ± 0.025	28.4 ± 2.2	22.3 ± 2.1	22.2 ± 2.2	14.7 ± 2.0
Deepseek	NL	3.6 ± 1.2	68.0 ± 1.6	0.452 ± 0.025	0.421 ± 0.027	33.5 ± 2.0	32.9 ± 2.0	9.3 ± 1.6	8.5 ± 1.6
Gemini	NL	64.0 ± 3.2	83.3 ± 1.6	0.662 ± 0.031	0.657 ± 0.031	65.7 ± 3.1	65.7 ± 3.1	65.4 ± 3.1	65.4 ± 3.1
Sonnet	NL	35.6 ± 3.2	75.8 ± 1.9	0.600 ± 0.030	0.563 ± 0.031	57.2 ± 3.0	57.2 ± 3.0	54.9 ± 3.1	54.9 ± 3.1
Llama4	NL	44.4 ± 3.3	85.6 ± 1.5	0.640 ± 0.030	0.627 ± 0.030	65.8 ± 3.1	65.5 ± 3.1	65.5 ± 3.1	65.5 ± 3.1
Gpt4.1	NL	44.9 ± 3.3	69.8 ± 2.0	0.662 ± 0.031	0.658 ± 0.031	65.9 ± 3.1	65.9 ± 3.1	65.7 ± 3.1	65.7 ± 3.1
Trajectory	PACKAGE	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
Simple Workflow									
Mistral	JSON	23.3 ± 3.5	99.3 ± 0.7	0.738 ± 0.027	0.574 ± 0.031	66.5 ± 2.9	49.8 ± 3.1	60.0 ± 3.4	37.3 ± 3.6
Deepseek	JSON	30.0 ± 3.8	99.3 ± 0.7	0.881 ± 0.016	0.912 ± 0.016	81.2 ± 2.1	75.3 ± 2.0	50.0 ± 4.1	30.2 ± 3.7
Gemini	JSON	68.7 ± 3.8	100.0 ± 0.0	0.955 ± 0.006	0.998 ± 0.002	92.0 ± 1.0	92.0 ± 1.0	69.0 ± 3.8	69.0 ± 3.8
Sonnet	JSON	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	JSON	96.0 ± 1.6	104.0 ± 1.6	0.999 ± 0.001	0.999 ± 0.001	99.7 ± 0.3	99.7 ± 0.3	99.7 ± 0.3	99.7 ± 0.3
Gpt4.1	JSON	96.7 ± 1.5	100.0 ± 0.0	0.992 ± 0.003	0.991 ± 0.004	98.5 ± 0.7	98.5 ± 0.7	98.0 ± 0.9	98.0 ± 0.9
Mistral	NL	32.0 ± 3.8	105.3 ± 6.0	0.700 ± 0.029	0.566 ± 0.033	63.0 ± 3.2	50.7 ± 3.3	56.2 ± 3.7	40.8 ± 3.7
Deepseek	NL	28.0 ± 3.7	99.3 ± 0.7	0.825 ± 0.017	0.870 ± 0.020	74.3 ± 2.0	68.0 ± 2.3	29.5 ± 3.7	28.8 ± 3.7
Gemini	NL	44.7 ± 4.1	100.0 ± 0.0	0.874 ± 0.012	0.948 ± 0.011	80.5 ± 1.7	79.8 ± 1.8	44.7 ± 4.1	44.7 ± 4.1
Sonnet	NL	99.3 ± 0.7	100.0 ± 0.0	0.999 ± 0.001	1.000 ± 0.000	99.8 ± 0.2	99.8 ± 0.2	99.3 ± 0.7	99.3 ± 0.7
Llama4	NL	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	NL	96.0 ± 1.6	100.0 ± 0.0	0.994 ± 0.002	1.000 ± 0.000	99.0 ± 0.4	99.0 ± 0.4	96.0 ± 1.6	96.0 ± 1.6
Trajectory	PACKAGE	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

Table 3: Performance by model (Sec. 6), input format, and workflow complexity (mean ± SE, NL=Natural Language)

Direct Generation and Guided Search Comparison DFSDT (Appendix Table 10) underperform direct generation across all complexity levels (Table 3). One consistent pattern is that DFSDT-generated trajectories often skip required steps defined in the workflow, 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 AI agents. `Traxgen` aligns with manually validated ground truth and outperforms LLM baselines by orders of magnitude in accuracy and efficiency, requiring no external model inference. `Traxgen` removes inference-time randomness, enabling stable, repeatable comparisons across workflows and agents. Unlike prompting-based methods, sensitive to phrasing and sampling, it provides a consistent reference for validation. Ablation studies highlight the importance of input structure: JSON schemas consistently outperform natural language, while ReAct-style prompting offers only marginal, inconsistent gains. These results suggest that architectural improvements may be more impactful than further prompt tuning. Overall, `Traxgen` provides a reliable foundation for evaluating AI agents in planning-intensive tasks.

8 Limitations

While `Traxgen` enables reproducible evaluation of agent trajectories, it has not yet been validated on real-world workflows, which often involve complex interdependencies, multimodal inputs, and non-idempotent behaviors. Extending `Traxgen` to open-ended or multimodal workflow remains an exciting future direction and natural progression toward broader applicability. Moreover, enumerating all permutations of soft-order blocks can grow factorially, limiting scalability, and the framework does not adapt to novel or ambiguous inputs without pre-specified logic. Additionally, our LLM benchmarking is constrained by available models and prompt designs, which may not generalize to newer architectures or strategies. Finally, while `Traxgen` can support reliability in regulated domains, human oversight remains essential to mitigate risks such as automation errors or misuse.

References

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

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

Anthropic. Claude 3.7 sonnet announcement, 2025. URL `https://www.anthropic.com/news/claude-3-7-sonnet`.

David Castillo-Bolado, Joseph Davidson, Finlay Gray, and Marek Rosa. Beyond prompts: Dynamic conversational benchmarking of large language models. *Advances in Neural Information Processing Systems*, 37:42528–42565, 2024.

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

DeepSeek AI. Deepseek-chat-v3-0324 release, 2025. URL `https://api-docs.deepseek.com/news/news250325`.

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

Google Cloud. Introducing agent evaluation in vertex ai gen ai evaluation service, 2024. URL `https://cloud.google.com/blog/products/ai-machine-learning/introducing-agent-evaluation-in-vertex-ai-gen-ai-evaluation-service`.

Google DeepMind. Gemini 2.0 flash-001 model documentation, 2025. URL `https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash`.

Crina Grosan and Ajith Abraham. Rule-based expert systems. In *Intelligent systems: A modern approach*, pages 149–185. Springer, 2011.

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

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

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

Labelbox. How to train and evaluate ai agents and trajectories with labelbox, 2025. URL `https://labelbox.com/blog/how-to-train-and-evaluate-ai-agents-and-trajectories-with-labelbox/#enhance-agent-trajectory`.

LangChain. Evaluation concepts, 2024. URL `https://docs.smith.langchain.com/evaluation/concepts#evaluators`.

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

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

Maria Emilia Mazzolenis and Ruirui Zhang. Agent warpp: Workflow adherence via runtime parallel personalization. *arXiv preprint arXiv:2507.19543*, 2025.

Meta AI. Introducing llama-4 maverick and scout, 2025. URL `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`.

Mistral AI. Mistral-7b-instruct model card, 2023. URL `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1`.

OpenAI. Openai agents sdk, 2025. URL `https://openai.github.io/openai-agents-python/`.

OpenAI. Introducing gpt-4.1, 2025. URL `https://openai.com/index/gpt-4-1/`.

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

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

Nick C Russell, Arthur HM Ter Hofstede, Wil MP Van Der Aalst, and Nataliya A Mulyar. Workflow control-flow patterns: A revised view. 2006.

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

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

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

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

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

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

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

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 conditional actions

Logic Construct	Definition
`skip`	Skips the execution of one or more steps when a specified condition is met.
`end_after`	Terminates the workflow 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	Metric	Value
Complex	Package	Exact-Match (%)	100.0 ± 0.0
		Count (%)	100.0 ± 0.0
		Tool F1	1.0 ± 0.0
		Param F1	1.0 ± 0.0
		CO % tools	100.0 ± 0.0
		CO % params	100.0 ± 0.0
		Prefix % tools	100.0 ± 0.0
		Prefix % params	100.0 ± 0.0
Intermediate	Package	Exact-Match (%)	100.0 ± 0.0
		Count (%)	100.0 ± 0.0
		Tool F1	1.0 ± 0.0
		Param F1	1.0 ± 0.0
		CO % tools	100.0 ± 0.0
		CO % params	100.0 ± 0.0
		Prefix % tools	100.0 ± 0.0
		Prefix % params	100.0 ± 0.0
Simple	Package	Exact-Match (%)	100.0 ± 0.0
		Count (%)	100.0 ± 0.0
		Tool F1	1.0 ± 0.0
		Param F1	1.0 ± 0.0
		CO % tools	100.0 ± 0.0
		CO % params	100.0 ± 0.0
		Prefix % tools	100.0 ± 0.0
		Prefix % params	100.0 ± 0.0

Table 5: Package evaluation results across all workflow 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.Model versions are described in Section 6.

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. Model versions are described in Section 6.

Routine	DeepSeek	Gemini	GPT4.1	Llama4	Mistral	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 workflow complexities. Model versions are described in Section 6.

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
	NL	deepseek	114	109	2	9	3	0	0	0	103	0	0	3	0	0	0
	NL	gemini	150	150	0	0	0	0	0	0	150	0	0	0	0	0	0
	NL	gpt4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	NL	Llama4	10	10	0	0	0	0	0	0	10	0	0	0	0	0	0
	NL	mistral	85	83	2	3	0	0	0	21	5	0	12	2	1	2	9
	NL	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
	NL	deepseek	164	157	3	10	4	0	0	0	147	3	2	1	1	0	3
	NL	gemini	225	225	0	0	0	0	0	0	225	0	0	0	0	0	0
	NL	gpt4.1	45	45	0	0	0	0	0	0	0	0	0	0	0	0	0
	NL	Llama4	155	155	0	1	0	0	0	0	155	0	3	12	0	0	0
	NL	mistral	114	113	1	4	0	0	0	13	15	0	6	2	0	12	9
	NL	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
	NL	deepseek	302	297	1	15	4	0	0	0	289	0	0	3	0	0	1
	NL	gemini	400	400	0	2	0	0	0	0	400	0	0	33	0	0	0
	NL	gpt4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	NL	Llama4	160	159	1	1	0	0	0	0	158	0	0	7	0	0	0
	NL	mistral	145	144	1	10	0	0	8	15	12	2	8	27	0	24	48
	NL	sonnet	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Table 9: LLM Output Cleaning Metrics by Workflow Type, Workflow Format, and Model (NL = Natural Language). Model versions are described in Section 6.

Model	Format	Exact-Match (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex workflow								
Sonnet	NL	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	NL	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	NL	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	NL	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	NL	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	NL	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	NL	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	NL	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	NL	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: DFSDT Experiment Results. Model versions are described in Section 6.

Model	Format	Prompt Type	Exact-Match (%)	Count (%)	Tool F1	Param F1	CMR % tools	CMR % params	Prefix % tools	Prefix % params
Complex Workflow										
Llama4	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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	JSON	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	JSON	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	JSON	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	NL	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	NL	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	NL	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: LLM Performance across Model, Format, Prompt Tyle and Workflow Complexity (NL = Natural Language). Model versions are described in Section 6.

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 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.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 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 product's availability by calling
  `return_product_availability(product_id = user_provided_info['product_id'],
                               availability = inventory_info[user_provided_i
                               ↪  nfo['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 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 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 user’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 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 doesn’t 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 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 employee’s 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 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: 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
↪  customer’s 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 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

```
  "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_m
↪  inutes'])
↪  ->
[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_di
      ↪  fference"]}]},
    {"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 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 to ensure they follow
the defined workflow logic and are consistent with the provided customer data.

Each annotation task includes:
- A workflow (a step-by-step recipe that tells the agent which tools to use, in what order, and under what conditions, to complete a task)
- 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 workflow based on the given user data and fully satisfies the task requirements.

Notes: Some trajectories will have 'soft ordering' defined. Soft ordering refers to groups of steps that can execute in any order. When soft ordering is present, more than one trajectory should be created and you will need to approve multiple possible trajectories that differ only in the order of those steps.

---
## 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. If 'soft ordering' is present, the right number of trajectories are generated and contain the right tool calls in the correct order.
2. Conditional logic (`skip`, `end_after`, `override_trajectory`) is triggered appropriately based
on customer data.
3. No extra tool calls are included
4. Tool parameters are fully and correctly filled using customer data and workflow-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 workflow or allowed by policy.
- In multi-intent workflows, an agent calls tools outside its scope.
---
## 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.|
| `wrong_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"]
}
```
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper clearly states the research problem: generating deterministic trajectories for multi-agent and tool-use evaluation in task-oriented dialogue systems.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a Limitations section discussing that `Traxgen` has only been evaluated on a restricted set of workflows and does not yet handle complex, multimodal, or non-idempotent real-world scenarios. We note that factorial growth in soft-order permutations can limit scalability, the framework cannot adapt to novel or ambiguous inputs without pre-specified logic, and LLM benchmarking is constrained by available models and prompt designs. We also highlight that human oversight remains essential to mitigate risks in high-stakes or regulated domains.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: This paper does not present new theoretical results or formal proofs. Instead, it focuses on introducing a framework for deterministic trajectory generation and evaluating it empirically across multiple domains. While we include definitions and structured formalisms for workflows and trajectories, these are methodological rather than theoretical theorems, and thus no assumptions or formal proofs are required.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides detailed descriptions of the datasets, task domains, workflows, and evaluation setup, ensuring that the experimental results can be reproduced. The methodology section specifies the trajectory generation process, agent configurations, and evaluation metrics, while the appendix offers additional implementation details. Furthermore, experimentation code is publicly available, which will allow researchers to directly reproduce our results and extend them to new domains.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper provides both anonymized code and dataset access in the supplemental material, with clear instructions on installation, environment setup, and commands to reproduce the main results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The paper specifies the experimental setting by describing the workflows, trajectory generation process, model usage, baseline creation and evaluation setup.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The paper reports statistical significance for its main experimental comparisons. Specifically, we tested for distributional assumptions (Shapiro–Wilk for normality, Levene's test for homogeneity of variance) and, since these assumptions were not satisfied, we applied the non-parametric Kruskal–Wallis test across model and format groups. When significant differences were found, we conducted post-hoc pairwise Mann–Whitney U tests with Holm correction. Results in the text explicitly state when performance differences are statistically significant (e.g., between JSON and natural language input, or across prompting methods), ensuring that our main claims are supported by significance testing.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The paper does not report compute resources explicitly. In practice, the experiments were run via API calls to hosted LLM services (OpenAI / OpenRouter), so no local CPU/GPU specifications or runtime measurements are applicable. The only local compute involved was a standard personal laptop CPU for issuing requests, which did not impact the experimental outcomes.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The research conforms with the NeurIPS Code of Ethics. The paper presents a synthetic evaluation toolkit and benchmark for trajectory generation without collecting or exposing sensitive personal data. No human subjects are involved, and no ethical risks such as bias amplification, privacy breaches, or misuse of restricted data are introduced. The work focuses on deterministic synthetic data and benchmarking, which aligns with responsible AI development practices.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses potential impacts in multiple places. The positive impact comes from Experiment 1 results, where `Traxgen` enables fast, deterministic generation of ground-truth trajectories without passing user data to LLMs, reducing compute costs and improving reproducibility, which can enhance reliability and transparency in high-stakes domains such as finance and healthcare. The negative impacts are discussed in the Limitations section, highlighting risks such as automation errors or misuse of the system, emphasizing the need for human oversight.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: No release of high-risk data or models.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes; the authors of this paper are also the creators of `Traxgen`, and any external LLMs used are properly cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces new assets in the form of `Traxgen` (a Python toolkit) and synthetic datasets for trajectory generation benchmarking. Both are described in detail within the paper, including their structure, functionality, and intended usage. At submission time, the assets are shared in anonymized form, consistent with the guidelines.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing or human-subject studies.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper clearly describes the use of LLMs in the experimental setup. Specifically, we benchmark multiple large language models (e.g., GPT-4.1, Gemini, Llama, Mistral, Sonnet) on structured and natural-language workflows to evaluate trajectory generation quality. However, the core method we propose (`Traxgen`) does not rely on LLMs for its operation; LLMs are used only as baselines and for comparative evaluation.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.