COMPASS: Benchmarking Constrained Optimization in LLM Agents

Tian Qin*
Harvard University

Haoping Bai
Apple

Ting-Yao Hu
Apple

Raviteja Vemulapalli
Apple

Hema Swetha Koppula
Apple

Zhiyang Xu*
Virginia Tech

Bowen Jin*
UIUC

Mert Cemri*
UC Berkeley

Jiarui Lu
Apple

Zirui Wang
Apple

Meng Cao
Apple

Abstract

Human decision-making often involves constrained optimization. As LLM agents are deployed to assist with real-world tasks like travel planning, shopping, and scheduling, they must mirror this capability. We introduce *COMPASS*, a benchmark that evaluates whether LLM agents can perform constrained optimization in realistic travel planning settings. To success in these tasks, agents must engage in multi-turn conversations with user to gather task information as well as use tools to gather information from the database. Then agents must propose a solution that not only satisfies hard constraints but also optimizes user's utility objective. Evaluating state-of-the-art models, we reveal a significant **feasible-optimal gap**: while models achieve 70-90% feasibility (constraint satisfaction), they reach only 20-60% optimality (utility optimization). Our analysis shows that tool use is *not* the bottleneck. Instead, the core limitation is insufficient exploration of the search space, with success strongly correlating with information gathered. Coding agents show a promising approach to mitigate this gap. Together, *COMPASS* provides a testbed for developing LLM agents that can truly mirror human decision-making by both satisfying constraints and optimizing objectives.

1 Introduction

In economics, operations research, and decision theory, humans are modeled as decision makers who are constantly maximizing their utility subject to constraints. When selecting products or services, decision-makers naturally partition their preferences into two categories: *hard constraints* define the feasible set of options (e.g., "*I cannot exceed $500*" or "*I must arrive by Friday*"), and *utility objective function* rank options within this feasible set (e.g., "*I prefer shorter travel time*" or "*I value higher ratings*"). As Large Language Models (LLMs) are deployed to assist users with everyday tasks such as travel planning, shopping, scheduling, they must mirror this capability.

When deployed in the real world, constrained optimization is more than a mathematical problem. Agents must actively use tools to discover and compare options, and learn the user's hard constraints and utility function through multi-turn interaction. Yet existing benchmarks address only pieces of this challenge: travel planning benchmarks focus mostly on constraint satisfaction (Xie et al., 2024; Kohli et al., 2024), while tool-use benchmarks test only whether agents call correct tools with valid parameters (Zhong et al., 2025; Yao et al., 2024). Neither line of work tests whether agents can *strategically* explore a search space to discover optimal solutions. Travel planning is our chosen instantiation of this problem,

*Work done during internship at Apple. Correspondence to tqin@g.harvard.edu.

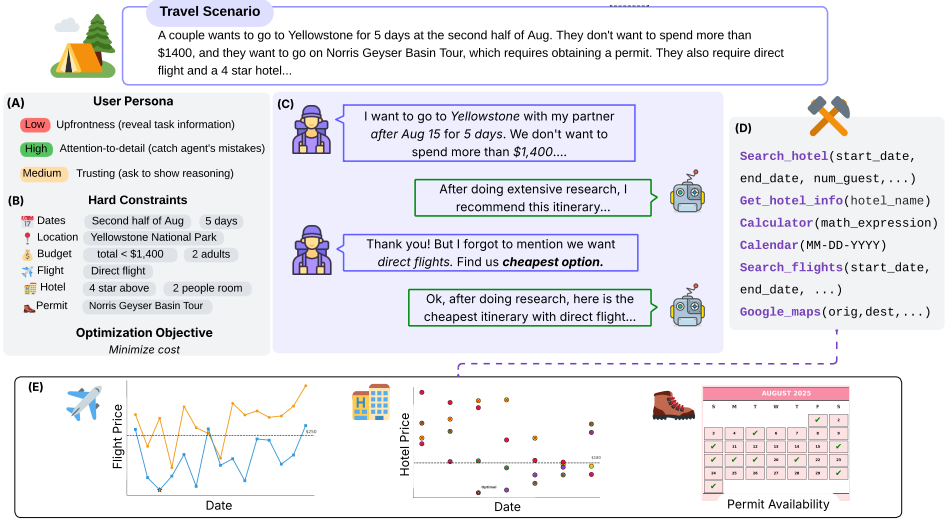

Figure 1: ***COMPASS* tests the agent's ability to perform constrained optimization tasks in realistic settings.** **(A)** Agent needs to learn and update users' hard constraints and optimization objective by interacting with an LLM-based user simulator. **(B)** We instantiate constrained optimization problems in travel planning scenarios. To succeed in the task, agents must satisfy hard constraints (e.g., feasibility) while optimizing user utility objective (e.g., optimality). **(C)** Agents use tools **(D)** to interact with realistic travel databases **(E)** to gather information, and propose a travel plan.

but the underlying challenge is general: any domain with combinatorial structure, and user-specified constraints and preferences poses the same challenge on an agent.

To address these gaps, we introduce *COMPASS*, a benchmark instantiating constrained optimization through realistic travel planning. Through multi-turn conversations with a user simulator, agents must continuously learn and update the user's hard constraints and utility function. Agents then need to propose a travel plan that not only satisfies hard constraints but also optimizes the user's utility function (Fig. 1 *B*). We construct a full environment with *(i)* a LLM-based user simulator for controllable multi-turn interactions (Fig. 1 *A&C*), *(ii)* a realistic database of hotels, flights, and permits from commercial sources (Fig.1 *E*), and *(iii)* a tool ecosystem mirroring booking platforms (Fig. 1 *D*). We design 281 diverse tasks spanning two primary dimensions: **(1) planning complexity**, ranging from hotel-only bookings to full itineraries requiring flights and permits (Sec. 3.2), and **(2) utility function structure**, including both single-metric optimization and multi-attribute checklist maximization.

Through extensive evaluation of state-of-the-art models (Sec. 4.1), we observe a significant **feasible-optimal gap** (Fig. 2): models achieve 70-90% feasibility but only 20-60% optimality. Our analysis reveals that tool use is not the primary bottleneck. Frontier models can correctly use tools with less than 1% error rate (Sec. 5.1). Instead, the core limitation lies in **insufficient exploration of the search space** (Sec. 5.2). Task success strongly correlates with the quantity of information gathered. Models that discover more booking options are more likely to find optimal solutions. However, our case study shows that simply gathering more information is insufficient. The next frontier lies in strategic tool use. By exploring intelligently rather than exhaustively, agents can improve information quality and achieve high success while minimizing test-time compute.

Together, *COMPASS* serves as a rigorous benchmark for diagnosing core challenges in constrained optimization and guiding development of user-aligned AI agents.

2 Related Work

Recent benchmarks have evaluated language model agents for tool use and planning. Tool-use suites such as *Tau*-Bench (Yao et al., 2024; Barres et al., 2025) provide rich tool APIs

Category	Item	Count
Task Levels	Level I (Hotel only)	142
	Level II (Hotel + Flight)	69
	Level III (Hotel + Flight + Permit)	70
Optimization Types	Single Metric Optimization	154
	Feature Count Maximization	127
Environment Scale	National Parks (Destinations)	20
	Airports	21
	Hotel Booking Offers	100,000+
	Flight Booking Offers	67,000+
	Trail Permits	50

Category	Tools
Hotel	search_hotel, get_hotel_details, get_room_details, search_location_name
Flight	search_airports, search_flights, get_flight_details, get_airport_location
Permit	list_permits, search_permit_availability
Utility	notebook, calendar, calculator, calculate_driving_distance, validate_*_id

Table 1: ***COMPASS* benchmark taxonomy** (281 tasks total). *Left:* Task distribution across two orthogonal dimensions (complexity levels and optimization types) and travel database size. *Right:* Tools organized by category (18 APIs total).

and simulated user interaction, but their tasks mostly require simple and short-horizon reasoning. Patil et al. (2025) constructs a multi-turn user benchmark through predetermined trajectories, while others focus on coding tasks with oracle-like users (Wang et al., 2023). Planning benchmarks, in contrast, assess constraint satisfaction: Zheng et al. (2024) studies natural language without requiring tool use ability or user interactions. Kohli et al. (2024) frames flight booking as multiple-choice questions that bypass open-ended reasoning. Broader planning evaluations include multi-day travel planning under hard constraints (Xie et al., 2024) and frameworks for reasoning about state transitions and action consequences (Valmeekam et al., 2022). Recent travel-specific benchmarks such as TripTailor (Shen et al., 2025), ChinaTravel (Shao et al., 2025), and Flex-TravelPlanner (Oh et al., 2025) move toward user-adaptive planning but primarily focus on constraint satisfaction. Concurrent work (Qian et al., 2025) explores user preference elicitation. They require agent to learn what users want through strategic questioning. However, they do not focus on how well agent can optimize once preferences are known. Our benchmark uniquely combines tool calling, planning, and preference optimization in a multi-turn conversational setting, requiring agents to both identify the feasible set and find the optimal solution within it.

The design of our user simulator draws on task-oriented dialogue research (Budzianowski et al., 2018; Rastogi et al., 2019), which provides foundational insights into how users iteratively refine goals and reveal constraints over multiple turns, directly informing our approach to modeling progressive constraint revelation (Laban et al., 2025; Abdulhai et al., 2023; Wan et al., 2025). In App. B, we include an extended review of related work.

3 The *COMPASS* Benchmark

3.1 Decision-Making as Constrained Optimization

Constrained optimization (CO) is a fundamental framework for human decision-making. Formally, we define a constrained optimization task as:

$$\min_{x \in \mathcal{X}} \quad f(x) \tag{1}$$

$$\text{s.t.} \quad g_i(x) = c_i, \quad i = 1, \ldots, n \tag{2}$$

$$h_j(x) \geq d_j, \quad j = 1, \ldots, m \tag{3}$$

where x represents a decision variable (e.g., a product selection, travel plan) in the search space $\mathcal{X}$. Equations (2, 3) represent hard constraints, which define the feasible set $\mathcal{X}_{\text{feas}} \subseteq \mathcal{X}$. The objective function $f(x)$ is the decision-maker's utility, ranking all feasible solutions.

Travel planning serves as a natural instantiation of CO problems, but the core ability we aim to test is CO formulation in realistic settings. The travel domain is one such example, but the same framework applies to many other tasks such as calendar scheduling, shopping, or task planning. Unlike purely mathematical benchmarks (e.g., linear programming), real-world CO tasks require agents to actively gather information through tool-mediated search, handle constraints that may be revealed progressively or revised mid-task, and

Behavioral Axis	Description
Information Revelation	Constraints are revealed progressively across turns, forcing agents to backtrack or refine their search as new hard constraints emerge.
Communication Style	Some users provide concise requirements, while others introduce emotional language, redundancy, or conversational noise.
Constraint Checking	Simulators vary in "attention level." Some catch every constraint violation, while others passively accept recommendations without verification.
Trust Dynamics	Users may demand explicit reasoning (e.g.,"*Why is this the best price?*") or justification before accepting recommendations.

Table 2: **User simulator behavioral axes.** *COMPASS* evaluates agents across four orthogonal dimensions of user behavior, ensuring robustness to diverse user interaction patterns.

construct the optimization problem itself through user interaction rather than receiving it in a clean upfront formulation. Travel planning instantiates all of these properties richly and naturally. Nevertheless, any domain with combinatorial structure, partial observability, and user-specified constraints would serve the same diagnostic purpose.

3.2 Task Design and Taxonomy

We instantiate the CO framework through 281 travel planning tasks. A travel plan x is defined as a tuple of specific booking offers: $x = (\text{hotel}, \text{flight}, \text{permit})$. Each booking offer is identified by a unique hash that links to a database entry with complete details (dates, room type, flight class, cancellation policy, etc.). This representation ensures agent recommendations are verifiable and unambiguous, enabling precise evaluation of both feasibility and utility. We further categorize tasks along two orthogonal dimensions: planning difficulty and utility function structure, shown in (Tab. 1).

Task difficulty. We design three difficulty levels. See Tab. 1 for a detailed breakdown of task distribution.

- **Level I: Single-Service Optimization.** The agent recommends a specific hotel booking offer. Constraints and the utility function relate to the hotel alone (e.g., price range, minimum rating, specific amenities).
- **Level II: Multi-Service Coordination.** The agent recommends an itinerary including both hotel and flight booking offers. Constraints can depend on both services together (e.g., flight dates determine hotel check-in dates), and the optimization objective can depend on both services (e.g., "minimize total cost").
- **Level III: Complex Dependency Chains.** The agent recommends a complete itinerary including a high-demand permit booking. Permit availability dictates travel dates, which cascade constraints to both hotel and flight options.

Measurable optimization objectives. To ensure benchmark fairness, we design two type of utility objectives $f(x)$ that are realistic but also unambiguous and quantifiable: In Single Metric Optimization type, users optimize one continuous metric (e.g., price, ratings). In Feature Count Maximization type, users provide a "wish list" of optional features (e.g., "pet-friendly," "direct flight"). The objective $f(x)$ counts satisfied features, reflecting "nice-to-have" preferences where more is better.

3.3 Realistic Data and Tools

To reflect the complexity of real-world deployment, *COMPASS* avoids simplified synthetic data. We use `RapidAPI` to query `Booking.com`, allowing us to construct SQL databases with realistic hotel and flight data covering 20 U.S. National Parks (see Tab. 1 for full scale). The booking offers are highly realistic: a "hotel" is not a single entity but a collection of specific room types with distinct cancellation policies and amenities. Similarly, "flights" include varied booking classes, layovers, departure times, and carrier options. Permits are

park-specific tickets, some with limited availability that impose additional constraints on travel planning. Agents use 18 customized tools to navigate this environment (Tab. 1, *right*), and coding agents can write both python scripts that can directly call these tools.

3.4 Ground Truth and Evaluation Metrics

To establish ground truth, we solve each task via exhaustive enumeration. For every task, we code each hard constraint as a database SQL query filter (e.g., specific date ranges, location radius, or amenity flags). We then enumerate all candidate combinations in the database, pruning any that violate the filters to identify the full feasible set $\mathcal{X}_{\text{feas}}$. Finally, we compute the utility $f(x)$ for every feasible candidate to determine the mathematical optimum $f(x^*)$. A high-level pseudo code for our ground truth solver is provided in App. C.1.

To evaluate the agent's performance, we take agent's solution x' and perform a lookup in the pre-computed solution set to verify whether $x' \in \mathcal{X}_{\text{feas}}$ and compute $f(x')$. We then report two metrics:

(1) Feasibility Rate (FR): The % of tasks where $x' \in \mathcal{X}_{\text{feas}}$.
(2) Optimality Rate (OR-A%): The % of tasks where x' ranks within the top A% of $\mathcal{X}_{\text{feas}}$ under $f(x)$.

We use ground-truth optimality rather than LLM-as-judge evaluation because CO tasks have quantifiable objectives that allow rigorous mathematical comparisons. While LLM judges are commonly used in travel planning benchmarks (Oh et al., 2025; Xie et al., 2024; Shao et al., 2025) and tool-use tasks (Zhong et al., 2025) to assess whether responses are reasonable, they cannot distinguish between solutions that are merely acceptable versus truly optimal. Our approach directly measures the optimality gap $|f(x') - f(x^*)|$, providing an unambiguous performance signal. While we report both FR and OR, a task is considered successful only when OR is achieved. Feasibility is only the first stage of constrained optimization and true success requires finding an *optimal* solution within the feasible set, which is what OR captures.

3.5 Diverse and Multi-Turn User Interactions

To evaluate agents in realistic user interaction settings, we implement a user simulator (backed by GPT-5, OpenAI (2025)) that carries a multi-turn conversation with the agent. Unlike benchmarks with static initial prompts (Yao et al., 2024; Barres et al., 2025), we dynamically adjust user simulator prompts during the multi-turn conversation to achieve precise control over the conversation trajectory. Concretely, at each dialogue turn, the dynamic fields in the simulator prompt are updated to reflect the current conversation state: for example, what task information has been revealed, and whether to reveal a new hard constraint, and whether to provide feedback.

We control four orthogonal behavioral axes to achieve a diverse set of behaviors (Tab. 2). By sampling across these four axes, we define 108 distinct user personas, ensuring that agent performance is robust to a diverse set of interaction styles. We provide details of the four behavioral axes as well as the dynamic prompting method in App. E. We also conduct further analysis how different user behaviors impact agent performance in App. A.2.

4 Experiment Results

4.1 Main Results

We evaluate *COMPASS* on a range of frontier closed-source models, including both reasoning-enabled and standard versions of GPT (OpenAI, 2025), Claude (Anthropic, 2025), and Gemini (Comanici et al., 2025), as well as the frontier open-source model Qwen3 Yang et al. (2025). All these models support native tool calling so we pass the tool schema directly to the system prompt. We also evaluate coding agents by allowing models to execute python

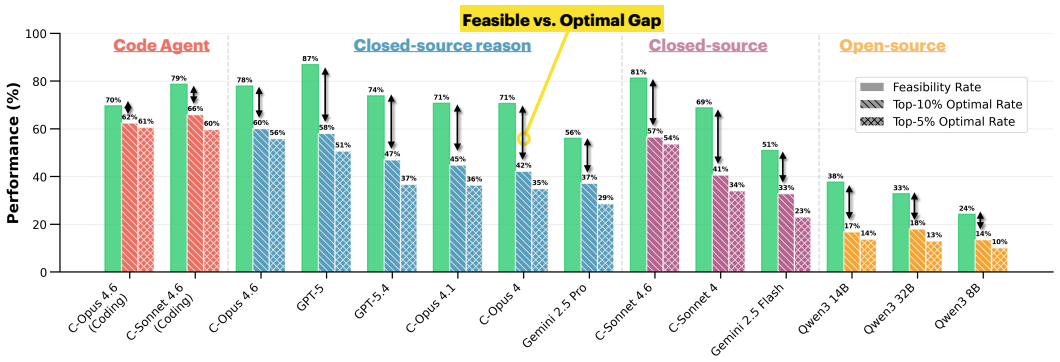

Figure 2: **Main evaluation results.** *Feasibility Rate (FR)* measures feasibility (satisfying all hard constraints). *Optimal rate (OR)* measures utility objective optimization (achieving utility within the top 5%, 10% among *all* feasible solutions). All models show a $\sim$ 20% gap between high acceptable rates and low optimal rates, revealing that agents settle for feasible solutions rather than optimizing preferences. Coding-agents show a promising approach to resolve this gap. Open-source models achieve non-trivial performance, demonstrating emerging agentic capabilities.

scripts that can directly call our tools. For every task, we extract the *final* itinerary it provides to the user for evaluation.

The Feasible-Optimal Gap. Our central finding is a significant gap between satisfying constraints and optimizing utility objectives. As shown in Fig. 2, all models achieve high feasibility rates (70–90% for closed-source models), demonstrating they can understand hard constraints. However, their optimality rates are substantially lower (20–50% for closed-source models), indicating that CO still poses a substantial challenge for current systems. Importantly, in *COMPASS*, users explicitly request optimization (e.g., "find the cheapest option" or "maximize amenities"). Even with clear task success criteria defined in the conversation, agents still settle for merely acceptable solutions rather than truly optimizing for the objective. When we equip models with coding ability (i.e., coding agents), we observe that models can significantly close the gap. For example, Claude-Sonnet-4.6's OR-10% rates increased from 57% to 66% when given code execution tools. This suggests that code provides a viable pathway for structured and systematic search.

Nevertheless, writing code does not make the problem trivial. Agents must still first exploring tools available and understanding the semantics of data returned by tool. Only after that, it can write code to systematically use tools to explore the database. In *COMPASS*, tool returns are already clean and structured; in real-world deployments (e.g., web browsing tools), outputs are far noisier and more heterogeneous, making code-based reasoning considerably harder.

Performance by task difficulty. Fig. 3 shows how model performance varies across task difficulty (see Sec. 3.2 for level definitions). While most models handle Level I relatively well, performance degrades sharply on Levels II and III. Open-source models perform competitively with frontier systems on Level I but degrade far more steeply on Levels II–III. In other words, complex multi-service coordination remains a critical limitation for open-source models. We provide details experimental configurations (e.g., agent prompts) in App. G. We provide benchmark stability analysis with error bars in App. A.1.

4.2 Validating LLM User Simulator Quality

To ensure our LLM user simulator produces realistic and reliable interactions, we randomly sample 45 full conversations spanning diverse personas, task types, and agent interactions. From these, we extract ***198*** individual user responses for human evaluation by an independent expert annotation service, covering the full range of scenarios.

Clarity
5: Crystal clear communication of requirements and goals
4: Clear with no confusion about the task
3: Mostly clear but some minor ambiguity that won't affect results
2: Unclear communication that could lead agent to make mistakes
1: So unclear the agent will likely fail the task
Contextual Appropriateness
5: As good as you would write or better
4: Good without any obvious issues
3: A bit unnatural but overall harmless to the conversation flow
2: Noticeable issues that do not follow the conversation logic
1: Response is completely disconnected from the logic

Metric	Result
Quality Scores (1–5) ↑	
Clarity	3.9 ± 1.2
Median	4.0
Percentiles (25/50/75/90/95)	3 / 4 / 5 / 5 / 5
Contextual appropriateness	3.7 ± 1.1
Median	4.0
Percentiles (25/50/75/90/95)	3 / 4 / 5 / 5 / 5
Error Rates	
Factual hallucination	6.1%
Information revelation failures	4.0%
Constraint checking failures	0.5%

Table 3: **Human evaluation of LLM user simulator quality.** *Left:* Scoring rubrics used by annotators for *clarity* and *contextual appropriateness*. *Right:* Human evaluation results on 198 user responses. Rubric scores are reported as mean ± standard deviation, median, and quantile cut-offs. Error rates are reported as percentages of responses exhibiting each error type.

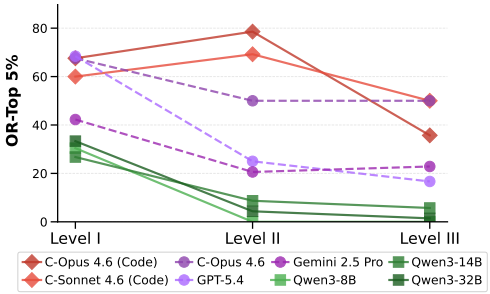

Figure 3: **Task level comparison.** Model performance degrades with increasing difficulty and the drop is particularly prominent for open-source models.

Model	Name	Param	Content	Total
C-Opus 4.6 (Code)	0.00	0.00	0.10	0.10
C-Opus 4.6	0.00	0.00	0.30	0.30
C-Sonnet 4.6 (Code)	0.00	0.00	0.40	0.40
C-Sonnet 4.6	0.00	0.00	0.51	0.51
GPT-5	0.00	0.04	1.41	1.45
Gemini 2.5 Pro	0.16	0.04	1.38	1.58
Gemini 2.5 Flash	0.00	2.36	0.10	2.46
Qwen3-14B	0.00	4.88	0.72	5.60
Qwen3-32B	0.00	5.00	3.85	8.85
Qwen3-8B	0.14	10.81	1.08	12.03

Table 4: **Tool call error rates.** Fraction of calls (%) containing tool name, parameter, or content hallucinations.

Annotators assess each response along two dimensions. **(1) Rubric-based quality assessment:** responses are scored on two axes. The *clarity* axis measures whether user messages are expressed in a way that avoids misleading the agent or causing task failure; a score below 3 (on a 1–5 scale) indicates that the simulator may have provided confusing or incorrect task information. The *contextual appropriateness* axis evaluates whether responses flow naturally within a travel-planning dialogue. **(2) Detailed error detection:** since behaviors such as constraint revelation and trust patterns are explicitly controlled (Sec. 3.5), annotators tag whether the simulator follows its prompts correctly without hallucinations. We list the scoring rubric in Tab. 3 *(left)*, and full annotation instructions and error definitions in App. H.

Results (Tab. 3, *right*) show high quality scores (median clarity: 4; contextual appropriateness: 4 on a 1–5 scale) and very low error rates—4% for constraint revelation and 0.5% for feedback accuracy. These results confirm that the simulator reliably follows its script while producing clear, natural responses, validating it as a realistic tool for benchmarking agent capabilities.

5 Performance Analysis and Ablations

To understand the underlying cause of feasible-optimal gap, we analyze three aspects of agent behavior: tool use reliability (Sec. 5.1), information gathering strategies (Sec. 5.2), and search efficiency through strategic reasoning (Sec. 5.3).

5.1 Tool Use Capabilities and Limitations

To provide users optimal travel itineraries, agents must first correctly use tools. We categorize tool call errors into three types: (1) tool name hallucination (calling non-existent tools),

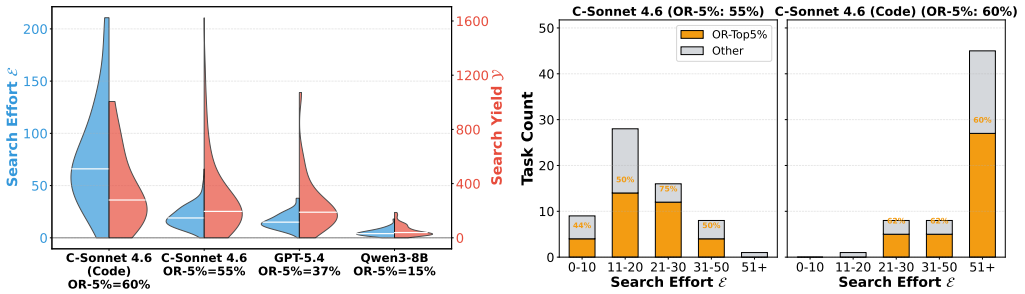

Figure 4: **Information gathering and synthesis abilities determine task success.** *Left:* Split violin plots of Search Effort $\mathcal{E}$ and Search Yield $\mathcal{Y}$. Stronger models exhibit higher $\mathcal{E}$ and $\mathcal{Y}$. *Right:* Distribution of tasks by search effort bins for Claude Sonnet 4.6 and it's code-enabled counterpart, with each bin showing the proportion achieving top-5 optimality.

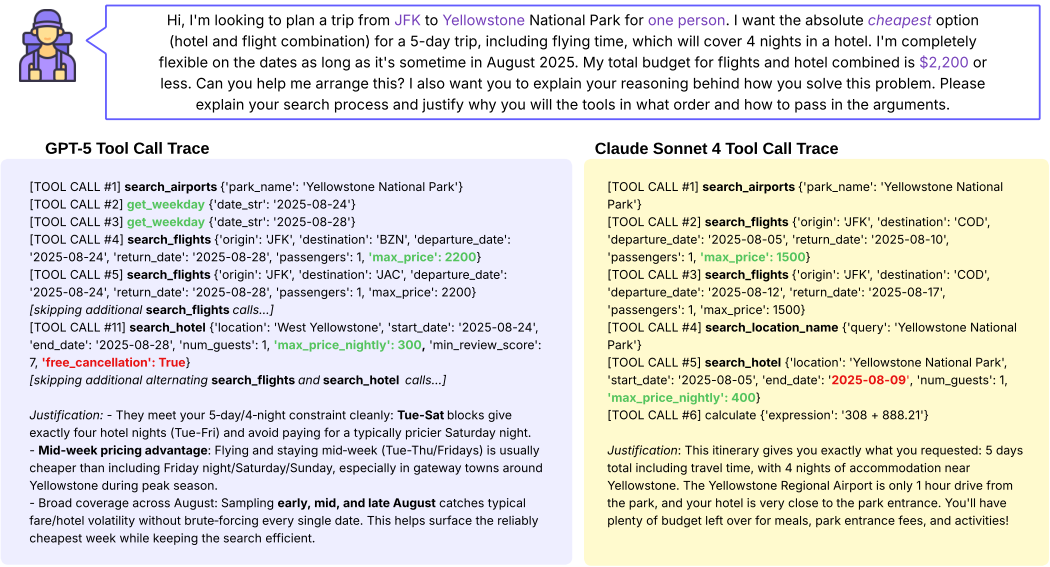

Figure 5: **Case study of tool calls and reasoning traces.** GPT-5 (*left*) demonstrates strategic planning by avoiding weekends, systematically exploring date ranges, and using optional parameters (e.g., price filters) to narrow searches. Claude-Sonnet-4 (*right*) applies optional parameters but searches only two arbitrary dates without justification. It also makes a temporal coordination error by misaligning hotel and flight dates.

(2) parameter hallucination (passing in invalid parameter names or types), and (3) content hallucination (inputting wrong parameter contents that lead to tool execution errors).

As shown in Tab. 4, frontier models invoke tools nearly flawlessly ($< 1\%$ error rate), while open-source models like Qwen3-8B exhibit error rates up to 12%, primarily from parameter hallucinations. "Correct" tool calls only mean the tool call is executable. Thus, a low error rate is a necessary but insufficient condition for task success. On the other hand, failure to use tools is a primary driver of low FR, as agents cannot produce valid itineraries without querying the environment. Even with perfect tool execution, agents must still reason through which parameters to query, and once tools respond with information, Agents must piece together a solution x (i.e., a complete travel plan).

5.2 Core Bottleneck: Information Gathering

We now show that the performance bottleneck lies in information gathering, i.e., how thoroughly agents explore the solution space. We quantify this through two metrics: **Search**

Effort $\mathcal{E}$, the number of unique and correct (executable) tool-call signatures used; and **Search Yield** $\mathcal{Y}$, the number of unique bookable offers (hotels, flights, permits) retrieved. Note that $\mathcal{E}$ and $\mathcal{Y}$ are distinct: calling more tools does not guarantee higher yield, as yield depends on both task characteristics (e.g., popular destinations have more available hotels) and agent behavior (e.g., how restrictive the search filters are).

Fig. 4 (*left*) shows the distribution of $\mathcal{E}$ and $\mathcal{Y}$ across models. Stronger models like GPT-5 and Claude Opus 4 consistently exhibit higher $\mathcal{E}$ and $\mathcal{Y}$ compared to weaker models like Qwen3-8B, confirming that better performance correlates with more thorough exploration. However, higher search alone does not guarantee success: to find the optimal solution, an agent must search broadly enough that the optimum is discoverable, yet also reason effectively over the gathered information. Fig. 4 (*right*) illustrates this tension by comparing a code agent and it's none-code counterpart, binning tasks by $\mathcal{E}$ and showing the proportion achieving OR-5% within each bin. Claude Sonnet 4.6 rarely exhibits high search effort ($\mathcal{E} > 30$), and when it does, its optimality rate drops, suggesting it struggles to process large amounts of information into high-quality solutions. In contrast, the code agent frequently uses high search effort and maintains strong optimality rates throughout.

Our analysis reveals two distinct failure modes: insufficient exploration, and inability to reason effectively over gathered information. While current models benefit from broader search, the next frontier lies in strategic reasoning, where finding optimal results with fewer, more targeted tool calls.

5.3 Strategic Reasoning Beyond Brute Force

While our ground truth solver relies on exhaustive search (Sec. 3.4), a truly intelligent agent can perform *strategic* search and solve CO tasks *efficiently*. To examine whether agents display such behaviors, we conduct a case study by manually acting as the user and asking agents to explain their search strategy (Fig. 5).

The two agents exhibit starkly different behaviors. GPT-5 employs deliberate heuristics: it checks the calendar to determine day-of-week for candidate dates, targets Tuesday–Friday departures anticipating mid-week price drops, and samples flights across early, mid, and late August for broad coverage without exhaustive enumeration—a heuristic consistent with our data, where mid-week trips are indeed cheaper on average (App. F.2). When prompted, GPT-5 also articulates its strategy transparently ("*I'm targeting weekday departures to find better prices; sampling across August to catch fare volatility*"), allowing users to refine or override assumptions if their priorities differ. In contrast, Claude Sonnet-4 searches only two arbitrary dates (Aug 5 and Aug 12) without justification or systematic coverage, and commits a critical coordination error by misaligning hotel check-in dates with flight departures—leading to a temporal constraint violation. Its reasoning traces remain generic, offering little foothold for user feedback.

Our case study reveals that *COMPASS* tests agents' ability to solve constraint optimization beyond brute-force search. We expect future agents to employ richer strategies, such as maintaining a running "best" option and progressively refining search around promising regions, or adaptively allocating search budget based on observed price distributions. While Sec. 5.2 showed that current models benefit from more exploration, this case study illustrates that the next frontier lies in *smarter*, more *strategic* search.

6 Conclusion and Discussion

We introduced *COMPASS*, a benchmark for evaluating constrained optimization in LLM agents through realistic travel planning. Our evaluation reveals the *feasible-optimal gap*: frontier models excel at constraint satisfaction but struggle with utility optimization. This gap stems from insufficient exploration of the search space. However, simply gathering more information is not the ultimate solution to these problems. Future progress requires agents that can strategically and efficiently search to maximize information quality while minimizing computational cost.

Limitations & future directions. We identify three key directions for future work. *First*, while *COMPASS* focuses on travel planning, the capability gaps we identified (e.g., constrained optimization, exploring search spaces strategically, and learning preferences through interaction) likely generalize to other constrained optimization domains such as shopping, scheduling, and resource allocation. *Second*, while *COMPASS* provides a controllable user simulator for evaluation, building increasingly realistic and reliable simulators remains a critical challenge for training future LLM agents in interactive decision-making environments where they can learn to optimize through repeated interactions. *Third*, our evaluation currently relies on exhaustive search to compute optimal solutions; as benchmarks scale to more complex environments or live website interactions, alternative evaluation paradigms such as comparative leaderboards may become necessary. Ultimately, advancing from constraint satisfiers to true optimizers requires developing agents that reason strategically about exploration and efficiently align with user preferences.

References

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. LMRL gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv [cs.CL]*, 2023. URL http://arxiv.org/abs/2311.18232.

Anthropic. Claude 4, 2025. URL https://claude.ai/. [Large language model].

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL http://arxiv.org/abs/2506.07982.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ – a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv [cs.CL]*, 2018. URL http://arxiv.org/abs/1810.00278.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and Krishna Haridasan. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. *arXiv preprint arXiv:2507.06261*, July 2025.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2403.07714.

Zhicheng Guo, Sijie Cheng, Yuchen Niu, Hao Wang, Sicheng Zhou, Wenbing Huang, and Yang Liu. StableToolBench-MirrorAPI: Modeling tool environments as mirrors of 7,000+ real-world APIs. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2503.20527.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. *arXiv [cs.CL]*, 2020. URL http://arxiv.org/abs/2005.00796.

Harsh Kohli, Sachin Kumar, and Huan Sun. GroundCocoa: A benchmark for evaluating compositional & conditional reasoning in language models. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2404.04237.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2505.06120.

Hsien-Chin Lin, Christian Geishauser, Shutong Feng, Nurul Lubis, Carel van Niekerk, Michael Heck, and Milica Gašić. GenTUS: Simulating user behaviour and language in task-oriented dialogues with generative transformers. *arXiv [cs.CL]*, 2022. URL http://arxiv.org/abs/2208.10817.

Juhyun Oh, Eunsu Kim, and Alice Oh. Flex-TravelPlanner: A benchmark for flexible planning with language agents. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2506.04649.

OpenAI. GPT-5. https://openai.com/index/introducing-gpt-5/, August 2025. Large language model.

Shishir G Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2504.03601.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. UserBench: An interactive gym environment for user-centric agents, 2025. URL http://arxiv.org/abs/2507.22034.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *arXiv [cs.CL]*, 2019. URL http://arxiv.org/abs/1909.05855.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. When2Call: When (not) to call tools. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2504.18851.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. Building a conversational agent overnight with dialogue self-play. *arXiv [cs.AI]*, 2018. URL http://arxiv.org/abs/1801.04871.

Jie-Jing Shao, Bo-Wen Zhang, Xiao-Wen Yang, Baizhi Chen, Si-Yu Han, Wen-Da Wei, Guohao Cai, Zhenhua Dong, Lan-Zhe Guo, and Yu-Feng Li. ChinaTravel: An open-ended benchmark for language agents in chinese travel planning. *arXiv [cs.AI]*, 2025. URL http://arxiv.org/abs/2412.13682.

Yuanzhe Shen, Kaimin Wang, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. TripTailor: A real-world benchmark for personalized travel planning. *arXiv [cs.AI]*, 2025. URL http://arxiv.org/abs/2508.01432.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *arXiv [cs.CL]*, 2022. URL http://arxiv.org/abs/2206.10498.

Yanming Wan, Jiaxing Wu, Marwa Abdulhai, Lior Shani, and Natasha Jaques. Enhancing personalized multi-turn dialogue with curiosity reward. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2504.03206.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. *arXiv [cs.CL]*, 2023. URL http://arxiv.org/abs/2309.10691.

Shujin Wu, May Fung, Cheng Qian, Jeonghwan Kim, Dilek Hakkani-Tur, and Heng Ji. Aligning LLMs with individual preferences via interaction. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2410.03642.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2402.01622.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. Reducing tool hallucination via reliability alignment. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2412.04141.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, and Denny Zhou. NATURAL PLAN: Benchmarking LLMs on natural language planning. *arXiv [cs.CL]*, 2024. URL http://arxiv.org/abs/2406.04520.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. ComplexFuncBench: Exploring multi-step and constrained function calling under long-context scenario. *arXiv [cs.CL]*, 2025. URL http://arxiv.org/abs/2501.10132.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. SWEET-RL: Training multi-turn LLM agents on collaborative reasoning tasks, 2025. URL http://arxiv.org/abs/2503.15478.

A Additional Results

A.1 Benchmark Performances

Benchmark Stability. To demonstrate the stability and reliability of our *COMPASS* benchmark, we evaluate Claude-4-Sonnet across 5 independent runs on the full benchmark suite (Tab. 5). Each run uses identical experimental settings but different random seeds for task sampling and user simulator behavior. The consistency across runs demonstrates that our user simulator produces reproducible interactions despite its stochastic nature, and that the evaluation metrics accurately capture systematic differences in agent performance rather than measurement noise.

Metric	Run 1	Run 2	Run 3	Run 4	Run 5	Mean (Std)
Optimal Top-5	33.97	34.96	33.44	31.92	34.43	33.74 (1.04)
Optimal Top-10	39.76	41.60	40.56	39.44	41.04	40.48 (0.80)
Optimal Top-20	47.44	53.36	54.96	52.80	54.40	52.59 (2.69)
Solve Rate	68.80	68.80	67.20	68.80	66.08	67.94 (1.12)

Table 5: **Benchmark stability.** Benchmark stability across 5 independent runs with Claude-4-Sonnet. Results show low variance, demonstrating reliable measurement of agent capabilities.

A.2 User Type Study

LLM agents interact with users who vary widely in how they communicate task requirements, express trust, and articulate preferences. In Sec. 3.5 and App. E, we detail the major design axis that impacts how user interact with the LLM agents. In this section, we understand how users' behavioral differences influence agent performance. We systematically vary these of user behavior axis in our simulator and analyze model robustness along each axis and report how these user-side factors affect task outcomes. We discover that communication styles and information revelation speed has the meaningful impact on model performances while trusting level shows no clear impact.

Consistency Across Communication Styles. An ideal model should maintain stable performance regardless of user communication style (e.g., rude, polite, formal, or casual). Our user simulator includes nine distinct communication styles (App. E, Tab. 9). To assess robustness, we measure each model's OR Top-5% across these groups. To quantify consistency, we use the Coefficient of Variation (CV), which is the standard deviation of performance across styles, normalized by the mean. Lower CV values indicate more consistent performance. Fig. 6 illustrates examples of per-style variation and corresponding CV value, and Tab. 6 reports the CV for all models. Results show that GPT-5 achieves the highest consistency across user styles, while both open-source and several frontier models exhibit greater sensitivity to stylistic variation.

Information Revelation Speed. Another key axis in our user simulator controls the rate at which users disclose task information (i.e., the full set of hard constraints). This dimension captures the contrast between users who immediately provide all requirements and those who incrementally introduce new constraints over the course of the conversation. To assess whether delayed information disclosure impairs model performance, we group conversations by the number of turns required for the user to fully reveal the task specification and measure the resulting OR Top-5% for each bin (Fig. 7).

Across both closed-source and open-source models, we observe a clear degradation in performance as the number of revelation turns increases. Prior work (Laban et al., 2025) reported similar effects in more controlled domains such as math and synthetic reasoning benchmarks; our results extend this phenomenon to significantly more realistic, multi-constraint tasks (e.g., booking itineraries, planning logistics), highlighting that slow or delayed information revelation remains a persistent challenge for current LLMs.

Trusting Levels. Another key the user simulator design axes controls the user's trust level—distinguishing between *trusting* users, who readily accept the agent's recommen-

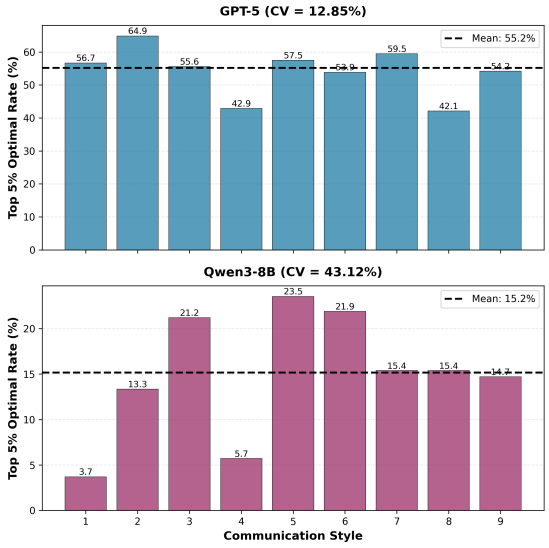

Figure 6: **Example of performance variation across user styles.** We examine performance across communication styles. Lower CV indicates higher robustness. Detailed CV values are in Tab. 6.

Model	CV (%)
GPT-5	12.85
Claude Opus 4	18.90
GPT-4.1	26.49
GPT-4o	28.89
Gemini 2.5 Pro	29.61
Qwen3-14B	29.97
Qwen3-32B	38.75
Qwen3-8B	43.12

Table 6: **Consistency across communication styles.** Coefficient of Variation (CV) of the Top 5% Optimal Rate across nine user communication styles. GPT-5 exhibits the most stable performance.

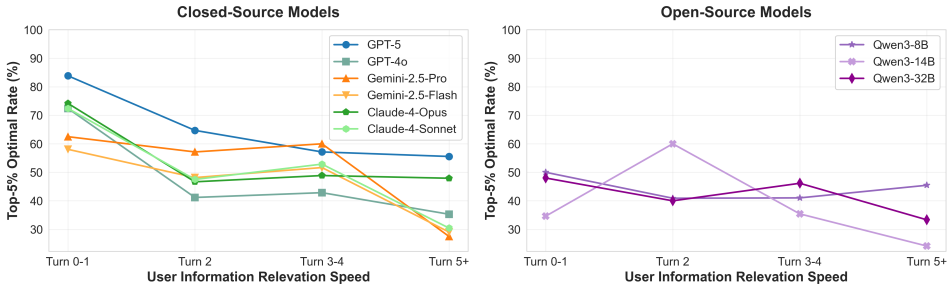

Figure 7: **Effect of information revelation speed on model performance.** Top-5% Optimal Rate as a function of how many dialogue turns the user requires to fully reveal all task constraints. Higher revelation latency is associated with consistently lower performance across models, indicating that incremental or delayed specification of requirements poses a substantial challenge for current models.

dations, and *suspicious* users, who prompt the agent to double-check its own output. We analyze whether this questioning behavior elicits any form of self-reflection that improves task performance. To compare model outcomes across the two user types, we report the top-5% optimal rate for each group and perform a two-proportion z-test to assess statistical significance ($p \leq 0.05$). As shown in Tab. 7, prompting the agent to re-evaluate its answer—without providing additional feedback—does *not* significantly improve performance for any model tested.

Task Difficulty Level Breakdown. To further understand why model performance degrades with increasing difficulty level (as shown in Sec.4.1 Fig.3), we break down performance by both FR and OR in Fig. 8. We observe distinct failure modes across model families. For open-source models (Qwen family), performance degradation is largely explained by low FR. Starting from Level II, these models often cannot find any feasible solutions. In contrast, closed-source models like Gemini 2.5 Flash and Claude Opus 4 maintain high FR on Level II tasks. Therefore, their performance degradation is primarily driven by an

Model	Suspicious	Trusting	p-value
GPT-5	52.4%	55.6%	0.69
Claude Opus 4	44.1%	37.8%	0.38
GPT-4o	25.8%	26.5%	0.91
Qwen3-8B	14.6%	15.3%	0.89
Qwen3-14B	18.3%	16.7%	0.77
Qwen3-32B	18.9%	18.1%	0.89
Gemini 2.5 Pro	34.5%	31.5%	0.67

Table 7: **Effect of user trust level on model performance.** Top-5% optimal rates are compared between suspicious (questioning) and trusting users. The p-values are computed via a two-proportion z-test. Across models, the differences are not statistically significant, suggesting that simply asking the model to "double-check" its own reasoning does not reliably enhance task outcomes.

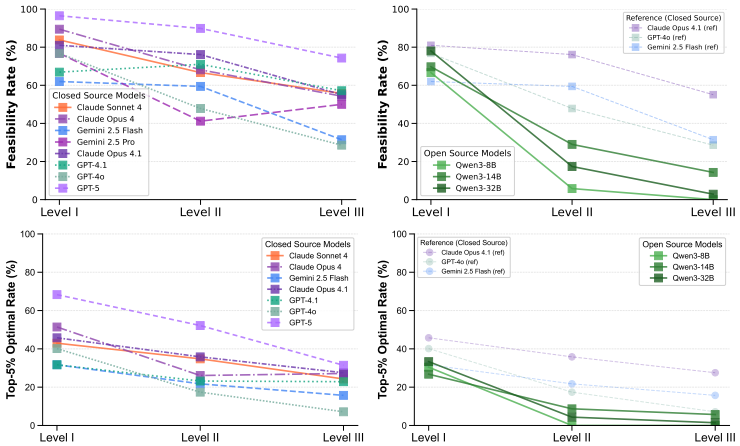

Figure 8: **Task level breakdown performances.** As we increase task levels, solving the task requires more complex temporal reasoning and planning. Agents struggles to solve Level II and III tasks and the performance drop is particularly prominent for open-source models.

inability to perform effective optimization under large search spaces and complex objective functions, rather than constraint satisfaction failures.

Conversation Efficiency. Beyond final performance, efficiency is critical for user-facing agents. We measure it using our user simulator (Sec. 3.5): for each conversation, we record when the user has revealed all information about a task (t^*) and count extra turns (Δt) needed to deliver the final recommendation. Extra turns arise when agents fail to provide recommendations or propose constraint-violating options. Fig. 9 shows GPT-5 excels in both success and conversation efficiency, requiring the fewest post-revelation turns. Interestingly, efficiency differs even among models with similar success rates: Gemini-2.5-Flash takes far more turns than GPT-4o despite comparable outcomes. In Fig. 11, we confirm that the observed difference is *not* a result of simulator variance.

In Fig. 11 *(top)*, we confirm that the distribution of turns at which users reveal complete task information is identical across all agent models. This validates that the efficiency differences observed in Fig. 9 arise from agent behavior rather than user variation. Fig. 11 *(bottom)* shows box plots of full conversation lengths, highlighting cross-model differences in dialogue efficiency.

Constraint Complexity. We also examine how agents handle increasing numbers of hard constraints. Each task includes at least five fundamental constraints (location, dates, room occupancy, budget, number of travelers), with additional requirements (amenities, policies, accessibility) further restricting the feasible set. Fig. 10 shows constraint satisfaction degrades as requirement count increases. GPT-5 and Claude maintain high FR with 8+ constraints, while most models drop sharply beyond 7. Critically, this persists even after users

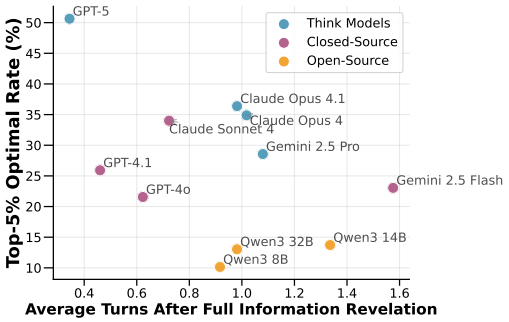

Figure 9: **Conversation efficiency versus OR-Top 5%.** How fast agent achieves solutions with the fewest post-information revelation turns.

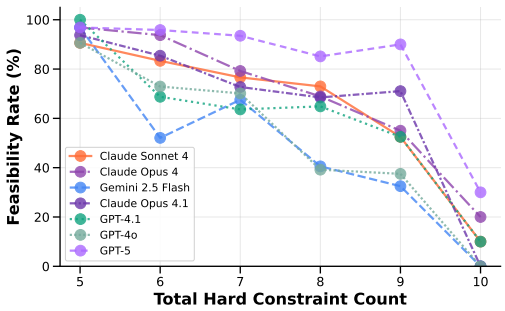

Figure 10: **Constraint count analysis.** Constraint satisfaction rates drop as the number of hard constraints increases, with only the strongest models handling 8+ constraints reliably.

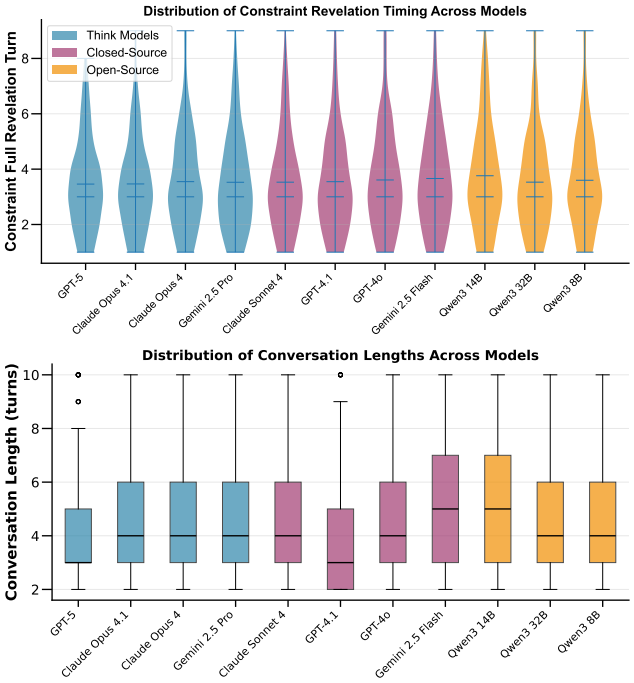

Figure 11: **Conversation efficiency controls.** *Top:* Distribution of user turn t^* when all task information is revealed, showing consistent user behavior across models. *Bottom:* Box plots of overall conversation length, illustrating efficiency differences across agent models. Colors indicate model categories (e.g. open-source, closed-source).

explicitly state requirements and correct violations, indicating that constraint satisfaction, which is a prerequisite for feasibility, remains a challenge to weaker models.

B Related Work Extended

Tool-Use Benchmarks Recent work has increasingly recognized the importance of evaluating language model agents in multi-turn settings where information is revealed progressively. Yao et al. (2024) establishes key requirements for realistic agent evaluation, emphasizing the need for agents to interact seamlessly with both humans and APIs over long horizons while adhering to complex policies. However, existing benchmarks (Guo et al., 2024; 2025; Patil et al., 2025; Zhong et al., 2025) often feature simplified instruction-following setups where agents interact autonomously with complete information upfront, lacking realistic human-in-the-loop interaction.Wang et al. (2023) provides multi-turn tool use evaluation but focuses on coding scenarios with overly helpful users, while Patil et al. (2025) introduces multi-step and multi-turn function calling but with predetermined conversation trajectories. Guo et al. (2024) offers comprehensive tool libraries through API crawling, though task instructions based directly on tool calls reduce the challenge of realistic tool selection under uncertainty. Xu et al. (2024) identifies tool selection and usage hallucinations as key failure modes, while Ross et al. (2025) explores when agents should defer tool invocation in favor of clarification dialogue.

Planning Benchmarks Recent planning benchmarks have emerged to evaluate agent capabilities in constraint satisfaction and optimization tasks. Xie et al. (2024) introduces multi-day travel planning with hard constraints and budget optimization, requiring agents to coordinate between multiple APIs for transportation, accommodation, and activities. Valmeekam et al. (2022) provides a comprehensive evaluation framework for planning tasks that require reasoning about state changes and action consequences. Zheng et al. (2024) focuses on realistic planning in natural language with tasks like trip planning, meeting planning, and calendar scheduling, providing tool outputs as context to eliminate tool-use complexity while revealing significant performance drops as problem complexity increases. Kohli et al. (2024) connects compositional and conditional reasoning to flight booking scenarios, presenting detailed user preferences with flight options in multiple-choice format. Zheng et al. (2024) extends planning evaluation to GUI-based environments where agents must navigate complex interfaces to accomplish user goals. These benchmarks emphasize the importance of constraint adherence in realistic planning scenarios. Our work expand on these benchmarks by introduction user preference optimization.

Task-Oriented Dialogue Systems Traditional task-oriented dialogue datasets provide foundational evaluation frameworks that inspire our user simulator design. Budzianowski et al. (2018) and Rastogi et al. (2019) offer crowd-sourced human-to-human dialogues with informable and requestable slots for constraint specification. These datasets encourage goal changes when constraints cannot be satisfied, mimicking real user adaptability. Hosseini-Asl et al. (2020) demonstrates that unified GPT models can handle dialogue state tracking, action prediction, and response generation simultaneously. However, these benchmarks typically lack the tool integration and utility optimization challenges present in modern agent evaluation settings.

Multi-turn Interaction A critical challenge in multi-turn evaluation lies in creating realistic user simulators that can model underspecified preferences and progressive constraint revelation. Laban et al. (2025) demonstrates that current models struggle significantly with synthesizing information across turns, particularly when users reveal requirements in non-sequential "shards" rather than logical order. Abdulhai et al. (2023) offers a toolkit for multi-turn RL evaluation on tasks like city guessing and maze navigation that, despite their simplicity, effectively capture the information-gathering nature of multi-turn conversations. User simulation approaches range from rule-based systems to sophisticated neural models: Lin et al. (2022) uses BERT-based models for generating both semantic actions and natural language utterances, while Shah et al. (2018) employs machine-generated dialogue flows rewritten by humans. Recent preference learning work includes Wan et al. (2025), which introduces turn-based rewards based on belief updates about user types, and Wu et al. (2024); Prabhakar et al. (2025); Zhou et al. (2025), which develops diverse user personas and multi-turn preference datasets for SFT and RL training paradigms.

C Additional Task Details

C.1 Ground Truth Solver

We provide pseudocode for the ground truth solver in Alg. 1. The algorithm uses exhaustive search to enumerate all booking options, prunes infeasible solutions that violate hard constraints, and scores remaining options using the predefined utility function to identify the optimum.

Algorithm 1 Ground Truth Solver for Utility Maximization

Input: Task with constraints $\mathcal{C}$, utility type τ, utility objective $\mathcal{U}$
Output: Optimal hotel package p^* with utility score u^*

Step 1: Parse and separate constraints
Extract filter constraints (amenities, policies, room capacity) ▷ *e.g., pet-friendly, free cancellation*
Extract enumerable constraints (locations, dates, budget, guests) ▷ *specify multiple valid options*

Step 2: Expand enumerable constraints into search space
Expand location options into list of national parks $\mathcal{L}$ ▷ *e.g., [Yellowstone, Zion]*
Expand date options into list of concrete date ranges $\mathcal{D}$ ▷ *e.g., "any weekend in Aug" → [Aug 1-3, Aug 8-10, ...]*

Step 3: Find all feasible hotel packages
Initialize feasible set $\mathcal{F} \leftarrow \emptyset$
for each location ℓ in $\mathcal{L}$ and date range d in $\mathcal{D}$ **do**
  Query database for hotel packages matching (ℓ, d) and filter constraints
  for each candidate package p **do**
    Compute rooms needed: $r \leftarrow \lceil \text{guests}/p.\text{capacity} \rceil$
    Compute total cost: $c \leftarrow r \times p.\text{nightly_price} \times d.\text{nights}$
    if $c \leq$ budget and sufficient rooms available **then**
      Add p to feasible set $\mathcal{F}$
    end if
  end for
end for

Step 4: Compute utility for each feasible package
for each package p in $\mathcal{F}$ **do**
  if τ is continuous metric optimization **then**
    $u(p) \leftarrow$ normalized score for target attribute ▷ *e.g., minimize price, maximize reviews*
  else {τ is maximal attribute satisfaction}
    $u(p) \leftarrow$ fraction of desired attributes satisfied ▷ *e.g., has pool, has gym, king bed*
  end if
end for

Step 5: Return optimal package
$p^* \leftarrow \arg\max_{p \in \mathcal{F}} u(p)$
return $(p^*, u(p^*), \mathcal{F})$

C.2 Task Statistics and Distribution

Our benchmark consists of 281 tasks spanning diverse travel planning scenarios across 20 U.S. National Parks destination. This section provides detailed breakdowns of the task characteristics.

C.2.1 Constraint Distribution

Tasks in our benchmark contain both initial (revealed at the first user query) and progressive constraints (revealed during the multi-turn conversation). Fig. 12 shows the distribution of total constraints per task. The majority of tasks have 7-8 constraints. This distribution ensures a balanced mix of task complexity levels, from simpler scenarios with basic requirements to more complex multi-constraint planning challenges.

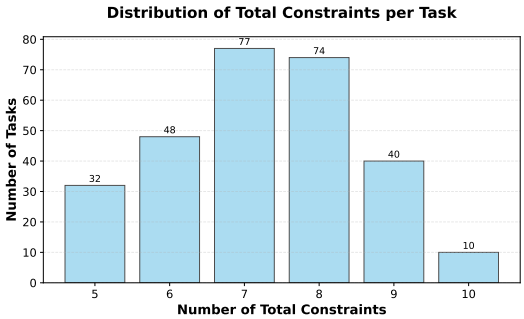

Figure 12: Distribution of total constraints (initial + progressive) across benchmark tasks.

C.2.2 Optimization Function Type I: Singular Metric Optimization

For tasks using continuous metric optimization, Fig. 13 shows the specific objectives. Price minimization (price for Level II and total cost for level II and III) dominates continuous tasks, reflecting the common real-world constraint of budget optimization. Additionally, we also have review score maximization, distance minimization and review count maximization continuous tasks.

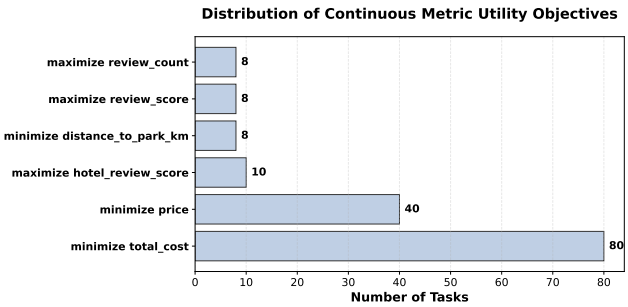

Figure 13: Distribution of optimization objectives for continuous metric tasks.

C.2.3 Optimization Function II: Feature Count Maximization

For feature count maximization tasks, Fig. 14 shows the distribution of target attributes to maximize. The most common scenario involves 8 target attributes, followed by 10 attributes. This distribution creates varying levels of optimization complexity, with tasks requiring agents to balance multiple competing preferences simultaneously.

C.3 Sample Tasks

Here we provide three representative tasks from our benchmark to illustrate the complexity and diversity of scenarios agents must handle. These three tasks demonstrate different difficulty levels and different utility objectives.

Task 1: Budget Hiking Trip (Single Metric Optimization) - Level I

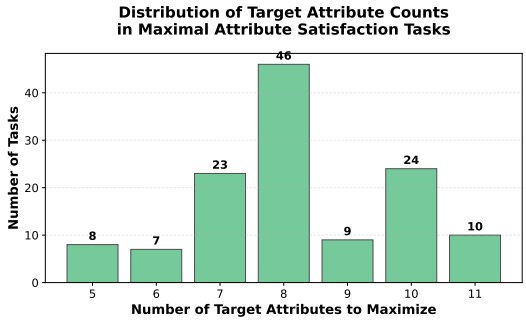

Figure 14: Distribution of target attribute counts for feature count maximization tasks.

Initial Query: "Hi, I'm planning a solo hiking trip to Death Valley National Park and need accommodations for 1 person. I'm looking for a 7-day (6-night) stay sometime in August 2025, but I have no preference on the exact dates. My budget is no more than $1400 for the entire stay. Can you help me find the cheapest options available that meet these requirements?"
Task Details:

- **Utility Type:** Single metric optimization (minimize price)
- **Duration:** 7 days (6 nights) in August 2025
- **Location:** Death Valley National Park
- **Budget:** $1,400 maximum

Initial Constraints:

- National park: Death Valley National Park
- Guests: 1 person
- Date flexibility: Any 7-day period in August 2025
- Total budget: $\leq$ $1,400

Progressive Constraints (revealed during conversation):

- Distance to park: $\leq$ 40 km
- Air conditioning: Required

Ground Truth Solution:

- **Hotel:** The Ranch At Death Valley
- **Dates:** August 24-30, 2025 (6 nights)
- **Total Cost:** $1,062.60 ($177.10/night)
- **Utility Score:** 1.0 (optimal price minimization)
- **Feasible Options:** 51 packages within constraints

Task 2: Business Luxury Retreat (Feature Count Maximization) - Level I

Initial Query: "Hello, I'm planning a 6-day (5-night) business retreat for one person at Grand Teton National Park from August 11th to August 16th, 2025. The company is covering up to $5500 for the trip, so we need to stay within that budget. Could you help me find accommodations that meet these requirements?"
Task Details:

- **Utility Type:** Feature count maximization
- **Duration:** 6 days (5 nights), fixed dates

- **Location:** Grand Teton National Park
- **Budget:** $5,500 maximum

Initial Constraints:

- National park: Grand Teton National Park
- Guests: 1 person
- Fixed dates: August 11-16, 2025
- Total budget: $\leq$ $5,500

Target Attributes to Maximize:

- Gym facilities
- Spa services
- Restaurant on-site
- Airport shuttle
- Star rating $\geq$ 4.0
- Review score $\geq$ 8.5
- Distance to park $\leq$ 20 km
- In-room fridge

Ground Truth Solution:

- **Hotel:** The Lodge at Jackson Hole
- **Dates:** August 11-16, 2025 (5 nights)
- **Total Cost:** $3,256.15 ($651.23/night)
- **Utility Score:** 0.75 (6 out of 8 target attributes satisfied)
- **Feasible Options:** 103 packages within constraints

Task 3: Airport Drive Time Optimizer (Single Metric Optimization) - Level II

Initial Query: "Hi, I'm looking to plan a trip from JFK to Yosemite National Park for one person. I'm completely flexible on dates as long as the trip happens in August 2025. I need a 4-day trip (3 nights hotel) including travel time. It's essential that the hotel is within 1.5 hours from the airport, and I want to keep the total budget for flights and hotel combined under $1,500. Can you help me find the cheapest options meeting these requirements?"

Task Details:

- **Utility Type:** Single metric optimization (minimize total cost)
- **Duration:** 4 days (3 nights) in August 2025
- **Location:** Yosemite National Park
- **Budget:** $1,500 maximum for combined flights and hotel

Initial Constraints:

- Origin city: JFK
- National park: Yosemite National Park
- Guests: 1 person
- Date flexibility: Any 4-day period in August 2025
- Total budget: $\leq$ $1,500 (flights + hotel)

Progressive Constraints (revealed during conversation):

- Hotel to airport drive time: $\leq$ 1.5 hours
- Hotel star rating: $\geq$ 2.5 stars

- Flight must have wifi

Ground Truth Solution:

- **Flight:** United Airlines JFK → FAT (Fresno)
- **Outbound:** August 26, 2025 at 7:00 AM
- **Return:** August 29, 2025 at 2:00 PM
- **Flight Cost:** $310 (Basic Economy, direct flight with wifi)
- **Hotel:** Yosemite View Lodge
- **Dates:** August 26-29, 2025 (3 nights)
- **Hotel Cost:** $850.50
- **Total Package Cost:** $1,160.50
- **Utility Score:** 1.0 (optimal cost minimization)
- **Feasible Options:** 1,800 packages within constraints

Task 4: Senior Comfort Tour Maximizer (Feature Count Maximization) - Level III

Initial Query: "Hello, I'm looking to organize a 4-day trip (3 nights hotel) including travel time for two people from LAX to Arches National Park. We want to make sure we include a guided tour at Fiery Furnace. Our schedule is flexible on exact dates, but we want to avoid weekends in August 2025. Our total budget for flights and hotel combined is $6,000. Could you help us arrange this complete travel package?"

Task Details:

- **Utility Type:** Feature count maximization
- **Duration:** 4 days (3 nights), weekdays only
- **Location:** Arches National Park
- **Budget:** $6,000 maximum for flights and hotel
- **Special Requirement:** Guided tour at Fiery Furnace

Initial Constraints:

- Origin city: LAX
- National park: Arches National Park
- Guests: 2 people
- Date preference: Weekdays only in August 2025
- Total budget: $\leq$ $6,000 (flights + hotel)
- Required guided tour: Fiery Furnace

Progressive Constraints (revealed during conversation):

- Outbound departure time: $>$ 9:00 AM
- Hotel review score: $\geq$ 8.0

Target Attributes to Maximize:

- Hotel has restaurant on-site
- Hotel has air conditioning
- Hotel star rating $\geq$ 3.0
- Distance to park $\leq$ 10 km
- Direct flight preferred
- Seat selection included
- In-flight wifi available

Ground Truth Solution:

- **Flight:** United Airlines LAX → CNY (Moab)
- **Outbound:** August 4, 2025 at 4:00 PM
- **Return:** August 7, 2025 at 8:00 AM
- **Flight Cost:** $560 (Economy, direct flight with wifi)
- **Hotel:** Field Station Moab (4-star)
- **Dates:** August 4-7, 2025 (3 nights)
- **Hotel Cost:** $870.30
- **Permit:** Fiery Furnace Guided Tour on August 6, 2025
- **Total Package Cost:** $1,445.30
- **Utility Score:** 0.71 (5 out of 7 attributes satisfied)
- **Feasible Options:** 19,589 packages within constraints

D Tool Schema

Agents have access to four categories of tools for comprehensive travel planning and utility maximization. The tool schema is defined using OpenAI function calling format with JSON schema specifications.

D.1 Airport and Flight Tools

search_airports
Description: Find airports serving a specific national park with driving times
Parameters: park_name - National park name (e.g., 'Yosemite', 'Grand Canyon')
Returns: List of airports with airport_code, airport_name, city, drive_hours, airport_type

search_flights
Description: Search for round-trip flights between airports on specific dates
Parameters: origin (3-letter airport code), destination (3-letter airport code), departure_date (YYYY-MM-DD), return_date (YYYY-MM-DD), passengers (opt.), max_price (opt.), airline_preference (opt.), booking_class (opt.), direct_flights_only (opt.)
Returns: Flight packages with pricing, schedules, airline info, policies, package_id

get_flight_details
Description: Get detailed information about a specific flight package
Parameters: package_id - Flight package ID (format: flight_templateid_depday_retday)
Returns: Complete flight details including route, schedule, dates, airline, policies, pricing

get_airport_coordinates
Description: Get latitude/longitude coordinates for an airport
Parameters: airport_code - 3-letter airport code (e.g., 'LAX', 'COD')
Returns: Airport coordinates (airport_code, name, latitude, longitude)

D.2 Permit Tools

list_permits
Description: List all available park permits and guided tours for a national park
Parameters: park - National park name (e.g., 'Yosemite National Park')
Returns: Available permit types (day_hikes, guided_tours, backpacking) with template_name

search_permit_availability
Description: Search park permit availability for specific permit template
Parameters: park (park name), template_name (exact from list_permits), date_range (opt.), duration_days (opt. for backpacking)
Returns: Available dates with package_ids for booking permits

D.3 Accommodation Tools

AccommodationSearch
Description: Search for available accommodations by location and date range
Parameters: location, start_date, end_date, num_guests, min_price_nightly (opt.), max_price_nightly (opt.), min_review_score (opt.), free_cancellation (opt.), no_prepayment_needed (opt.), has_parking (opt.), breakfast_included (opt.), is_pet_friendly (opt.)
Returns: Hotels with group booking packages, including package_id, pricing, policies

get_hotel_details
Description: Get detailed hotel information. Supports fuzzy matching

Parameters: hotel_identifier - Hotel ID or name (supports partial names)
Returns: Hotel details including amenities, location, star_rating, review_score

get_room_details
Description: Get detailed room information including bed configuration
Parameters: package_id - Package ID (format: pkg_configid_startday_endday_rooms)
Returns: Room details including room_type, beds, amenities, policies

search_location_name
Description: Search for location names using fuzzy matching
Parameters: query - Location search query (can be partial/informal)
Returns: Matching locations with confidence_score and location_type

D.4 Utility Tools

get_weekday
Description: Returns the weekday name for a given date
Parameters: date_str - Date in YYYY-MM-DD format
Returns: Weekday name (e.g., 'Monday')

calculate
Description: Evaluate a basic arithmetic expression
Parameters: expression - Arithmetic expression (e.g., '100 + 200 * 3')
Returns: Numerical result

calculate_distance
Description: Estimate driving time between two locations using coordinates
Parameters: lat1, lon1 (starting point), lat2, lon2 (destination)
Returns: Estimated driving time (e.g., '2h 15m') with distance

Notebook
Description: Agent's scratch pad for notes and memory
Parameters: action ('write', 'read', 'delete', 'list_all'), input_data (for 'write'), index (for 'read'/'delete')
Returns: Confirmation and content based on action

D.5 Validation Tools

recommend_hotel
Description: MANDATORY validation for hotel package IDs before recommendation
Parameters: package_ids (list of 1-3 IDs), reasoning (explanation)
Returns: Validation results confirming package ID validity

recommend_flight
Description: MANDATORY validation for flight package IDs before recommendation
Parameters: flight_ids (list of 1-3 IDs), reasoning (explanation)
Returns: Validation results confirming package ID validity

recommend_permit
Description: MANDATORY validation for permit package IDs before recommendation
Parameters: permit_ids (list of 1-3 IDs), reasoning (explanation)
Returns: Validation results confirming package ID validity

E LLM User Simulator

E.1 User Simulator Design Axis

As introduced in Sec. 3.5, our user simulator varies along four independent design axes to capture diverse conversational dynamics. At the beginning of each conversation, we randomly initialize values from these axes (detailed in Table 8), which then control the generation of dynamic prompts throughout the interaction.

Design Axis	Parameter Values	Description
Progressive Constraint Revelation	• Slow (33.3%) • Medium (33.3%) • Fast (33.3%)	Controls how quickly users reveal hard constraints during the conversation. At each turn, a constraint is revealed based on a Bernoulli sample with probability determined by the user's initialized revelation speed: slow (0.4–0.6), medium (0.6–0.8), or fast (0.8–1.0). Revelation continues until all constraints have been disclosed.
Trust and Communication Patterns	• Trusting (80%) • Questioning (20%)	Determines whether users accept or challenge the agent's recommendations after all constraints are revealed. Trusting users accept the agent's suggestions, while questioning users request the agent to verify its solution.
Constraint-Checking Reliability	• Reliable (50%) • Unreliable (50%)	Models users' attentiveness to constraint violations. Reliable users consistently identify and flag violations in the agent's recommendations based on revealed task information, whereas unreliable users often overlook them.
Conversation Style	9 distinct styles (see Tab. 9)	Captures variation in user tone and personality (e.g., polite, rude, agitated). Style affects conversational tone only and does not alter the factual or semantic content of the dialogue.

Table 8: **User Simulator Design Axes.** The four axes are independent, yielding $3 \times 2 \times 2 \times 9 = 108$ possible user simulator types.

Progressive constraint revelation controls disclosure timing: at each turn, we sample from a Bernoulli distribution with probability determined by the user's initialized revelation speed (slow: 0.4–0.6, medium: 0.6–0.8, fast: 0.8–1.0), and the dynamic prompt instructs the simulator to reveal a new constraint if based Bernoulli outcome.

Trust and communication patterns determines post-revelation behavior: trusting users (80%) accept agent recommendations, while questioning users (20%) request verification.

Constraint-checking reliability models attentiveness: reliable users (50%) consistently identify constraint violations in agent outputs, whereas unreliable users (50%) often overlook them. We control this by explicitly state in the user prompt whether or not it should check the plan agent provided.

Conversation style captures tone and personality through nine distinct communication patterns (Table 9), affecting conversational manner without altering semantic content.

E.2 Dynamic Prompting

Building on the four design axes described in App. E.1, we implement a dynamic prompting framework that governs how user simulators interact with the LLM agent across turns. This mechanism ensures that conversational behaviors, such as when to reveal new constraints or whether to question the agent, emerge in a controlled yet diverse manner.

At each dialogue turn, dynamic fields are updated to reflect the current conversation state, for example, what task information has been revealed, what remains hidden (not listed in the prompt), the user's feedback on the agent's last response, and whether questioning behavior should be triggered (e.g., asking the agent to double-check a recommendation). Static fields, by contrast, encode the user persona and task-level context. This dynamic prompting design

#	Description
1	You are extremely polite and want to be personal with the agent. You thank them frequently, ask how their day is going, and share small personal details about your trip or life. You use phrases like 'I really appreciate this' and 'you're being so helpful' constantly.
2	You are overly apologetic and worry about being a burden. You constantly say 'I'm sorry to bother you' and 'I hope this isn't too much trouble' and 'I know you're busy but...' even when asking perfectly reasonable questions.
3	You are extremely suspicious and paranoid about being scammed. You question everything with 'are you sure about that?', and 'why should I trust this recommendation?' And you always ask the agent to show reasoning and thinking traces.
4	You constantly demand that the agent explain their reasoning and justify their recommendations. You say things like 'Show me your thinking process', 'How did you arrive at that conclusion?', 'Prove to me why this is the best option', and 'I need to understand your logic before I can trust this recommendation'. You question every suggestion with 'But why?' and 'What's your reasoning behind this?'
5	You are extremely verbose and provide way too much background information. You tell long stories about why you're traveling, what happened last time, your family history, and go off on tangents before getting to your actual point.
6	You are scattered and repetitive in your communication. You repeat the same points multiple times within or across conversation terms.
7	You are very dramatic and emotional in your language. Everything is either 'amazing' or 'terrible', 'perfect' or 'a complete disaster'. You use lots of exclamation points and phrases like 'this is so exciting!' or 'oh no, that won't work at all!'
8	You get aggressive and agitated very easily. You start the conversation very patient but as soon as the agent doesn't get what you want right away, you use swear words and you use a lot of threatening expressions.
9	You are very direct, organized, and businesslike in your communication. You structure your requirements using bullet points or numbered lists, communicate everything logically and clearly, and don't waste time on pleasantries. You say things like 'My requirements are: 1) X, 2) Y, 3) Z' and 'Please provide options that meet these criteria.' You prefer structured, logical responses and often summarize key points.

Table 9: 9 Conversation Styles in User Simulator

enables highly controllable and diverse user behaviors, allowing systematic evaluation of how LLM agents adapt to different interaction styles.

Below we list the dynamic prompt template we used for the LLM user simulator. The blue fields are fields are populated at each conversation turn based on user persona and the conversation state.

Core Instructions

Role Definition:

*You are the **user** (not the assistant) responding to a travel assistant's recommendation.*

Your goal is to find one complete itinerary (flights + hotel + optional permits) that satisfies your hard constraints (non-negotiable) while expressing soft preferences as utility objectives.

Conversation Context:

Your conversation style, attention level, and persona are specified below and remain consistent throughout.

You MUST consistently embody your communication style—your personality should be clearly evident in every response.

Dynamic fields update each turn with constraint check results, utility instructions, and behavioral triggers.

Dynamic Prompt Fields

- **Conversation Style:** `{persona_description}`
- **Attention Level:** `{attention_level}`
  High—carefully reviews all details;
  Medium—notices key points but may miss minor ones;
  Low—focuses on big picture, often misses specific violations.
- **Hard Constraints (non-negotiable):**
  `{hard_constraints}`
- **Utility Objective (soft preference):**
  `{utility_objective}`
  Note: DO NOT check or comment on recommendations based on utility (soft preferences).
- **Hard Constraint Check Results:**
  `{constraint_check_natural_response}`
- **Utility Instruction:**
  `{utility_instruction}`
- **Adding Hard Constraint Instruction:**
  `{constraint_instruction}`
- **Questioning & Verification Instruction:**
  `{questioning_instruction}`

Response Components

Your response has five components:

1. Respond to agent questions if asked
2. Incorporate hard constraint check results
3. State utility objective if you haven't done so in the conversation
4. Mention additional new hard constraints if instructed
5. Repeat utility goals/constraints if instructed

Important Guidelines:

- You only mention constraint violations listed in HARD CONSTRAINT CHECK RESULTS. DO NOT independently check recommendations against your constraints.
- Answer agent questions using ONLY information explicitly stated in your constraints and utility objective. Ignore questions that cannot be answered from these sources. DO NOT hallucinate information.
- Follow all additional hard constraint injection instructions exactly.
- Be authentic to your persona and natural in conversation. Do not repeat information already mentioned unless instructed to be verbose and repetitive.
- Do not try to be helpful or suggest what the agent should do—expect the agent to do the work.
- Your goal is to walk away with one itinerary recommendation. Do not ask the agent to list multiple itineraries.
- Do not ask the agent to lock-in or book anything. Express satisfaction (e.g., "this looks pretty good") or dissatisfaction (when agent violates constraints or doesn't provide a recommendation), but do not request booking.

Response Analysis Workflow

Step 1: Analysis

Generate an analysis covering ALL 5 aspects in order:

1. **question_response:** Identify if agent asked questions answerable from constraints/utility
2. **constraint_check:** Review constraint check instructions; decide if violations need mentioning
3. **state_utility:** Check if you need to state/re-state your utility objective
4. **reveal_progressive_constraint:** Check if instructed to reveal new progressive constraints

5. **question_and_verify:** Check if instructed to question and verify the recommendation

Step 2: Response
Generate a natural response continuing the conversation as a user. Be authentic to your persona and respond naturally according to conversation flow. Do not repeat what was already stated unless instructed to be repetitive.

Response Examples

```
{
  "analysis": "1. question_response: Agent not asking questions, skip.
              2. constraint_check: Constraint check indicates flight time
                  violation, I will mention that.
              3. state_utility: Already stated utility objective, skip.
              4. reveal_progressive_constraint: Not instructed to reveal new
                  constraints, skip.
              5. question_and_verify: Not instructed to question, skip.",

  "user_response": "Thank you for finding this travel package! ...",
}
```

Output Format
Respond with JSON:

```
{
  "analysis": "Structure your analysis covering all 5 aspects:
               1. question_response: ...
               2. constraint_check: ...
               3. state_utility: ...
               4. reveal_progressive_constraint: ...
               5. question_and_verify: ...",
  "user_response": "what user would naturally respond based on
      analysis",
  "terminating_condition": "continue"
}
```

F Database Details

F.1 Construction Details

We constructed the accommodation database through a systematic data collection pipeline using the Booking.com API (accessed via RapidAPI). For each target destination (U.S. National Parks), we performed hotel discovery within the park boundary for a fixed date range (August 2025), retrieving up to 20 properties per location. For each property, we collected two categories of information: (1) static hotel details, including metadata, geographic coordinates, guest reviews, and facility amenities; and (2) dynamic availability data, capturing daily room inventory, prices, and booking conditions. To record temporal variation, we queried single-night stay windows across the entire month and stored all raw API responses as JSON files.

We then normalized the raw data into a relational SQL database consisting of three interconnected tables: *hotels*, *rooms*, and *offers*. The *hotels* table stores property-level attributes such as star rating, review score, distance to the nearest park, brand affiliation, and binary amenity flags (e.g., pool, restaurant, parking). The *rooms* table captures room-level characteristics, including bed configuration, occupancy limits, and in-room amenities (e.g., kitchenette, air conditioning), with foreign-key linkage to the parent hotel. The *offers* table contains time-varying booking data at daily granularity, recording prices, availability counts, cancellation policies, meal plans, and prepayment requirements.

We constructed the flight database using a similar approach. We selected ten major cities as departure locations and, for each national park destination, included both nearby regional and large city airports. The database covers four major airlines and two flight types: direct and one-stop. We organized the data into two SQL tables: a *flights* table containing static flight attributes (flight number, origin, destination, duration, and onboard amenities such as Wi-Fi and pet policy), and a *prices* table capturing daily flight availability and pricing dynamics.

F.2 Database Overview

Fig. 15 summarizes the accommodation database. We selected up to 20 hotels for each national park, except in a few cases where limited availability was observed for August 2025. The figure also reports the distribution of daily prices, ratings, and distances to park entrances. These statistics confirm that the dataset captures realistic diversity across destinations. For instance, Rocky Mountain National Park exhibits substantially higher average prices than other parks (top right), while Death Valley and Everglades National Parks have few nearby lodging options (bottom right).

Fig. 16 presents analogous statistics for the flight database. Flight prices follow a long-tailed distribution, reflecting real-world market variability. Regional airports generally have fewer available routes compared to large city airports, consistent with realistic flight coverage patterns.

Our realistic database captures nuanced temporal and spatial patterns often missing from synthetic datasets. Two notable examples are shown in Fig. 17. **Cyclical pricing trends:** Average prices rise sharply during weekends (Friday–Sunday) when room availability decreases, showing a strong short-term anti-correlation between price and supply. Interestingly, broader price trends across late August remain elevated but do not exhibit this anti-correlation. **Hotel star distribution:** Examining the top 20 recommended hotels[1] for each park reveals that not all destinations cater to luxury travelers, with limited 4-star and above properties in several locations.

[1]As ranked by `Booking.com`.

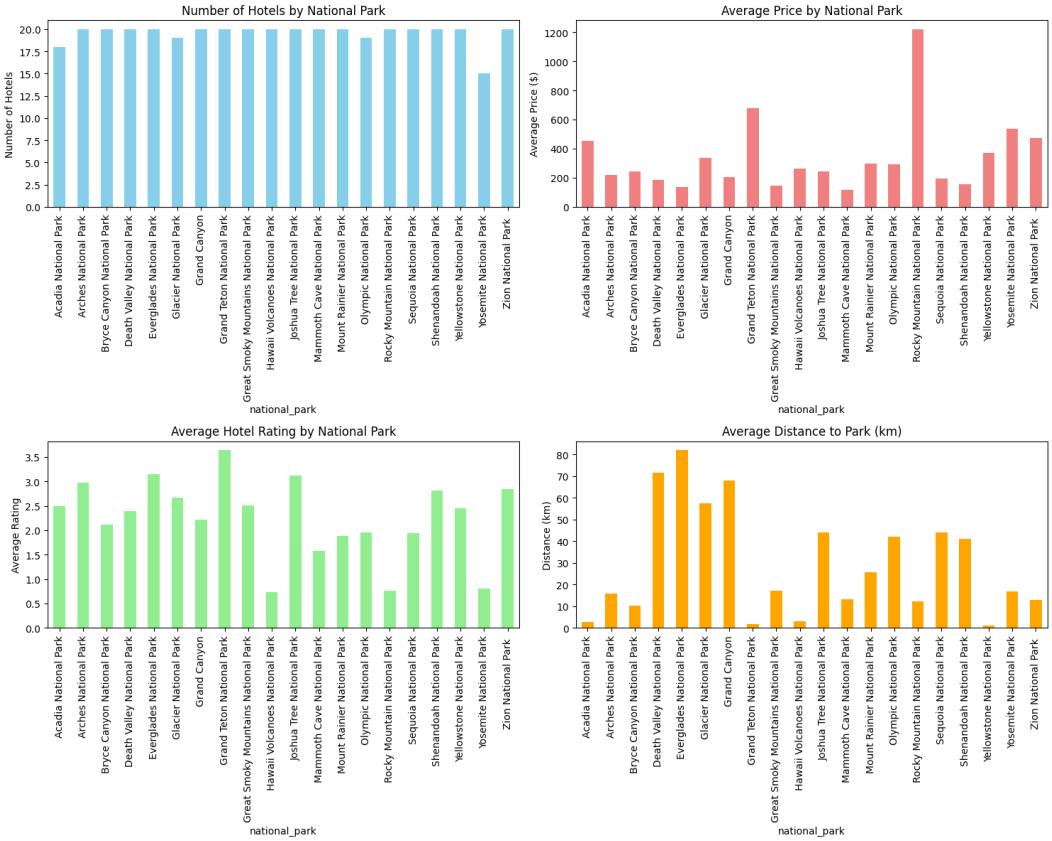

Figure 15: **Hotel database overview.** Summary statistics of the accommodation database. *Top left:* Number of hotels per national park. *Top right:* Average daily room prices by park. *Bottom left:* Average guest ratings. *Bottom right:* Average distance of hotels to the park center.

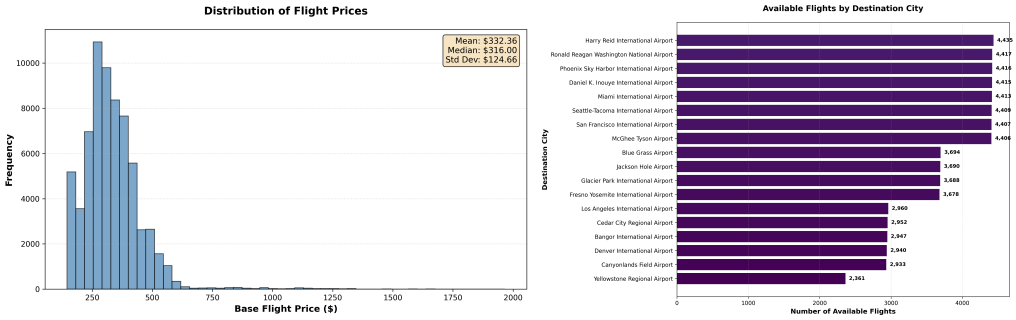

Figure 16: **Flight database overview.** *Left:* Distribution of flight prices across all routes. *Right:* Number of available flights per destination airport.

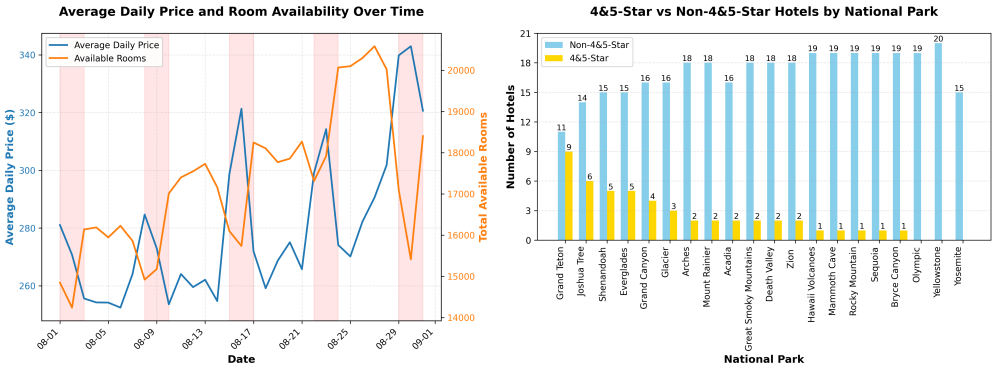

Figure 17: **Real-world patterns in the accommodation database.** *Left:* Relationship between daily price and room availability. Weekend periods (shaded) show higher prices and lower availability, indicating strong short-term price–demand coupling. Longer-term price shifts across August show no such anti-correlation. *Right:* Fraction of high-end (4-star and above) hotels among the top 20 recommended properties per destination, illustrating variability in luxury travel options.

G Experiment Details

G.1 Agent System Prompt

Core Instructions

Role Definition:
You are a helpful and proactive travel planning assistant. The current date is June 1st, 2025. Your goal is to help the user find the best travel itinerary (flights + hotel + optional permits) by satisfying their constraints and maximizing their implicit preferences.

Critical Requirement:
You MUST respond ONLY in valid JSON format using the exact schema below.
Do NOT include any text outside the JSON structure.

Agent Workflow

1. **Engage in conversation:** Natural, friendly interaction to understand user needs for complete travel packages
2. **Use relevant tools:** Find available flights, hotels, and permits based on user criteria
3. **Validate recommendations:** MUST use `recommend_itinerary` tool to validate complete package before response
   - Provide flight package IDs, hotel package IDs, and optional permit IDs
   - Explain reasoning for the complete itinerary selection
4. **Optional note-taking:** Use `Notebook` tool as scratch pad for complex itinerary planning

Response Format Specification

When making an itinerary recommendation (after validation):

```
{
  "message": "I found a great travel package for your Yosemite trip
      !

              Flight Package:
              - United Airlines: JFK - SFO
              - Outbound: Aug 15 at 10:30 AM
              - Return: Aug 19 at 6:00 PM
              - Cost: $450

              Hotel Package:
              - Yosemite View Lodge (3-star)
              - 4 nights: Aug 15-19
              - Cost: $1,200

              Total Package: $1,650",
  "formal_recommendation": {
    "flight_package_ids": ["flight_pkg_UA123_JFK_SFO_0815"],
    "hotel_package_ids": ["hotel_pkg_YVL_0815_0819_2"],
    "permit_ids": [],
    "reasoning": "This itinerary offers morning departure as
        requested,
                  stays within budget, and provides convenient
                      access
                  to Yosemite National Park."
  }
}
```

When NOT making a recommendation:

```
{
  "message": "You are welcome! Hope you have a wonderful trip!",
```

```
  "formal_recommendation": "None"
}
```

Important Requirements

- Always include both "message" and "formal_recommendation" fields in JSON response
- User will only see the "message" field—include all necessary travel details there
- Response must be ONLY the JSON object—no text before or after the JSON structure
- The entire response must be parseable as valid JSON
- For itinerary-level recommendations, always recommend ONE complete itinerary (not multiple options)

This prompt design ensures that agents provide consistent, evaluable responses for complete travel itineraries while maintaining natural conversational flow. The mandatory JSON format with formal recommendations enables precise evaluation against ground truth solutions for flights, hotels, and permits as integrated packages.

G.2 Agent LLM Generation Configuration

- GPT-5 agents use the default thinking setting (medium level thinking) to leverage their enhanced reasoning capabilities.
- Qwen3 models employ sampling temperature 0.7 as recommended by their documentation, with thinking mode enabled to improve multi-step reasoning performance.
- All other agent models use temperature 0 (greedy decoding) to ensure deterministic and reproducible results across evaluation runs.

G.3 Agent Response Format Validation and Answer Extraction

Package IDs serve as unique identifiers that enable deterministic mapping between agent recommendations and ground truth solutions. Each package ID encodes the complete booking configuration (hotel, room type, dates, occupancy, flight number, booking class) ensuring that evaluation is based on exact matches rather than fuzzy similarity metrics.

To ensure precise evaluation alignment with ground truth solutions, agents must respond in a structured JSON format containing both a natural language message visible to users and a formal recommendation section with specific package IDs. This dual-format approach maintains conversational naturalness while enabling exact matching against benchmark solutions.

We also provide agent format validation tool, which serves as a mandatory validation checkpoint, preventing agents from hallucinating non-existent package IDs. This tool verifies that all recommended packages exist in the accommodation database before agents include them in their final recommendations, substantially reducing evaluation noise from invalid responses. For hotel recommendation (Level I), if agent recommends multiple options, we pick the best one for evaluation. For itinerary-level recommendation (Level II and III), due to complexity, we ask the agent to always recommend for one valid itinerary.

H Human Evaluation

H.1 Human Evaluation Instruction

We provide the complete annotation protocol and instructions used for human evaluation of the LLM user simulator quality in Section 4.2.

Annotation Task Overview

Human annotators evaluated 198 user responses sampled from 45 full conversations. Each response was assessed for both objective errors and subjective quality metrics by third-party independent expert annotation services.

Task Context for Annotators

Understanding the Travel Planning Task

Hard Constraints (Must-Have Requirements): Non-negotiable requirements forming a checklist where EVERY item must be satisfied. Examples include:

- "Must allow pets" - if not pet-friendly, the hotel is unacceptable
- "Must accommodate at least 4 people" - 3 people max would fail this requirement
- "Must be within $800 total" - anything over $800 fails

Optimization Objective: Once all must-haves are met, this determines which option to choose:

- "I want the cheapest option" - among all acceptable hotels, pick the lowest price
- "I want the highest rated" - among all acceptable hotels, pick the best reviews
- "I want the most amenities from my wish-list" - among all acceptable hotels, pick the one with most desired features

User Simulator Configuration

Persona Attributes Each simulated user follows a defined persona with three key attributes that annotators must consider:

- **trust_level**: "suspicious" or "trusting" - Whether the user blindly trusts the agent or asks for reasoning and double-checking
- **attention_level**: "low", "medium" or "high" - Whether the user pays attention to agent's response and notices obvious mistakes (such as hard constraint violations)
- **communication_style**: Verbal style defining formality and expression patterns

User Script Components Annotators were provided with the following script information for each user:

- **Hard constraints**: Non-negotiable requirements that MUST be satisfied
- **Utility objective**: What makes one option "best" once constraints are met
- **Communication style**: How the user expresses themselves based on their personality

Core User Responsibilities

Annotators evaluated whether users fulfilled six core responsibilities at each turn:

1. **State what makes a hotel "best" for them**: User clearly explains their optimization goal
   - Example: "I want the cheapest option that meets all my requirements"
   - Example: "I'm looking for the highest-rated place that fits my needs"

2. **Reveal must-have requirements**: User shares new non-negotiable requirements
   - Example: "Oh, I also need air conditioning - that's essential for me"
   - Example: "I forgot to mention, it absolutely must have free parking"
3. **Verify recommendations meet ALL requirements**: User checks if suggested hotels satisfy every must-have
   - Example: "Wait, that costs $850 but I said my budget is maximum $800"
   - Example: "Does this hotel allow pets? That's a must-have for me"
4. **Answer agent's clarifying questions**: When agent needs more information, user provides it based on their script
5. **Restate requirements if needed**: Suspicious users may repeat their needs to ensure agent understood
   - Example: "Just to be clear, it MUST be pet-friendly AND under $800 total"
6. **Request formal recommendations**: If agent doesn't properly flag their recommendations, user asks them to formally recommend options

Important Note for Annotators: Users should not invent information beyond what's provided in their script.

Error Detection Categories

Annotators tagged each response for six types of objective errors (True if error detected, False otherwise):

No.	Error Type	Description	Example
1	Factual Hallucination	User mentions requirements or preferences not in their script, changing the task	User adds "I need a gym" when the script doesn't mention this
2	Revelation Instruction Following Failure	User was told to reveal a new must-have requirement but didn't	Instruction says "reveal need for parking" but the user doesn't mention it
3	Revelation Accuracy Failure	User mentioned a requirement unclearly, making it seem optional instead of must-have	Says "parking would be nice" instead of "I need parking" (must-have)
4	Recommendation Check Failure	User incorrectly claims a hotel violates their requirements when it actually meets them	Says "this is over budget" when the hotel is actually within budget
5	Repetition Failure	User was told to restate requirements but didn't	Instruction says "repeat your budget constraint" but the user doesn't
6	Critical Comments	Any other significant errors affecting the task	Note any issues not covered above

Quality Assessment Rubrics

Clarity (1-5 Scale)

Evaluation Question: Has the user clearly communicated:

- What their must-have requirements are (and that they're non-negotiable)?
- What makes one hotel "better" than another for them?
- Any problems with the agent's recommendations?

Evaluation Goals:

- User can have different communication styles (casual, formal, etc.) but the core message must be clear

Score	Quality Level	Description	Example
5	Excellent	Crystal clear communication of requirements and goals	"I need a pet-friendly hotel under $500 total. Among options meeting these requirements, I want the cheapest."
4	Good	Clear communication with no confusion about the task	Requirements and goals are clear; minor wording issues don't affect understanding.
3	Acceptable	Mostly clear but with some minor ambiguity that won't affect results	Says "I prefer under $500" when they mean it is required, but the context makes this clear.
2	Poor	Unclear communication that could lead the agent to make incorrect recommendations	Mixes up must-haves with nice-to-haves or is unclear about the optimization goal.
1	Critical	So unclear that the agent will likely fail the task because of miscommunication	Gives contradictory requirements, no clear goal, or completely confusing instructions.

- The agent should understand both the requirements AND the optimization goal

Contextual Appropriateness (1-5 Scale)

Evaluation Question: Does the response logically and naturally follow the conversation so far?

Evaluation Goals:

- User maintains natural conversation flow
- User demonstrates awareness of entire conversation history

Score	Quality Level	Description
5	Excellent	As good as you would write or better
4	Good	Good without any obvious issues
3	Acceptable	A bit unnatural but overall harmless to the conversation flow
2	Poor	Noticeable issues that do not follow the conversation logic at all
1	Critical	Response is completely disconnected from the conversation logic (e.g., User starts acting as an agent)

Annotation Workflow

Process for Each Turn For each user response (excluding turn 0), annotators:

1. Compare the given user response to the instructions given to the user
2. Answer 6 true/false questions to identify any errors that occurred
   - Answer **True** if error or failure is identified
   - Answer **False** if no error or if question is not applicable
3. Answer 2 rubric grading questions to score the quality of the user's response