OXFORD JOURNAL
OF STUDENT
SCHOLARSHIP

Agentic AI for Conversational Flight Search: Design and Evaluation of a Hybrid React and GPT-4o Framework

Arjun Gupta
arjun@bsbkmail.com

ABSTRACT

An affordable flight search remains a fragmented and cognitively demanding task. Travelers navigate several airline and online travel agency sites, compare fares that change by the minute, and re-enter the same personal details repeatedly. Rule-based chatbots have addressed this only partially, because they demand structured input and handle ambiguity poorly. This paper presents an agentic flight-search system that accepts natural-language queries, extracts and validates travel parameters, retrieves live fares, and returns ranked itineraries with explicit decision cues. The system pairs a React front end with a Node.js and Express back end. GPT-4o is invoked only for intent classification, entity extraction, clarification phrasing, and result summarization; all control flow, validation and ranking remain deterministic application code. The control design is model-agnostic; GPT-4o was chosen because it supports constrained structured outputs. Live fares are obtained through the Google Flights engine of SerpAPI. The system was evaluated on 30 scripted conversations spanning six input categories, with five trials per category, and on a usability study with ten participants. Twenty-eight of the 30 trials succeeded (93.3%). Vague queries were resolved in a mean of 2.0 turns through a single consolidated clarification, and both correction pathways preserved all unamended fields. Both observed failures and one flagged misclassification arose in language-model stages; the deterministic core contributed no errors. Participants rated the system 4.24 of 5 on average, with ease of use highest (4.43) and explanation of recommendations lowest (3.70). The results indicate that a hybrid design, in which a language model handles interpretation and ordinary code handles control, can deliver reliable, low-effort flight search while keeping failures localizable. Recognition of unrecognized parameter values is identified as the principal reliability gap.

Keywords: *Agentic AI, Conversational Agents, Flight Search, Large Language Models, Human–Computer Interaction, Usability, LLM Reliability*

1. INTRODUCTION

Modern web and mobile services notify users of appointments, sort their mail, and host general-purpose assistants. Yet the ordinary task of finding a reasonably priced flight remains laborious. A typical search still involves opening several airline and online travel agency tabs, comparing fares that change by the minute, checking layover durations by hand, and entering the same passenger details on each site in turn. The result is a form of cognitive overload produced not by any single difficult step but by the accumulation of repeated entry, inconsistent presentation, and uncertainty about which option is genuinely best.

Conventional rule-based chatbots and form-driven booking engines address this only in part. They generally require structured input, respond poorly to ambiguity, and force the user to restart when information is incomplete. Agentic artificial intelligence offers a different model: systems that reason about a goal, plan a sequence of actions, invoke external tools, and adapt to changing context. Acharya et al. [1] characterize agentic AI as autonomous intelligence directed at complex goals with limited human supervision and survey the algorithmic foundations on which such systems are built.

Two limitations recur in existing conversational travel systems. The first is brittleness under noisy input: real users write incomplete, contradictory, or informally phrased requests, and a system that cannot accumulate partial information across turns will either fail or interrogate the user field by field. The second is opacity: when reasoning, control flow, and ranking are all delegated to a language model, a failure is difficult to attribute to a particular stage, which makes the system hard to improve and hard to trust.

Usability research indicates that this second problem is not merely an engineering inconvenience. Wali et al. [2] observe that intelligent task-automation agents, although technically feasible, are not widely adopted due to usability problems and argue that such systems must be designed according to explicit usability principles to achieve acceptable user satisfaction. A capable system that users cannot predict or correct is, in practice, an unused one.

The research question addressed here is, therefore, how an agentic flight-search system can be engineered to be simultaneously robust to noisy real-world input and straightforward for a non-expert user to operate.

This paper presents a complete working system whose design deliberately separates language understanding from control. GPT-4o interprets what the user has written; ordinary application code decides what happens next. The system accepts everyday English, accumulates missing information across turns without requiring the user to restate what is already known, validates parameters deterministically before any external call, retrieves live fares, and surfaces three independent decision signals: Best Value, Newest Aircraft and Recommended for You. Figure 1 contrasts this interaction model with conventional multi-site search.

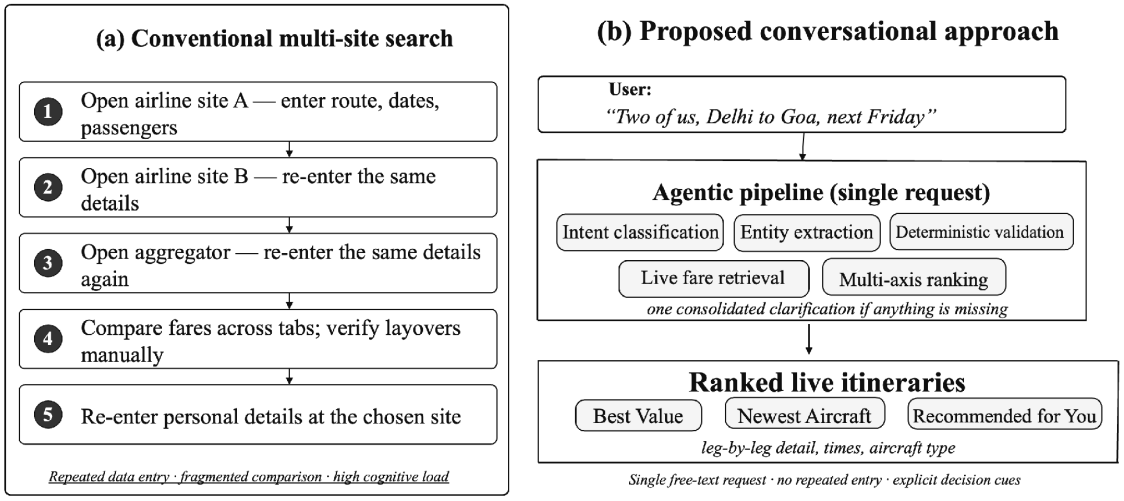


Figure 1. Conventional multi-site flight search (a) compared with the proposed conversational approach (b). The conventional path requires repeated data entry across independent sites; the proposed path resolves a single free-text request through one pipeline, requesting any missing information in a single consolidated clarification.

The principal contributions of this work are the following.

1) A hybrid agentic architecture that keeps orchestration, validation and ranking outside the language model so that system behavior is inspectable and failures can be localized to a specific stage. The architecture is model-agnostic; GPT-4o is one admissible interpreter, not a prerequisite of the control design.

2) A consolidated clarification loop that gathers every missing or invalid field into a single natural-language prompt, roughly halving the turns required for incomplete queries relative to sequential field-by-field questioning.

3) Two correction pathways, conversational field amendment and message-level editing with full state rewind, allowing a user to fix an earlier mistake without restarting the dialogue.

4) An evaluation protocol covering six realistic input categories, together with a usability study and a comparison against manual search, a commercial travel chatbot and a general-purpose assistant.

5) Empirical evidence that 93.3% of scripted trials succeed and that every observed error occurred at a language-model stage boundary rather than in the deterministic core.

The design targets several groups for whom form-driven booking interfaces are a particular obstacle. Users unaccustomed to complex websites, including many older adults, face a process that reduces to a single plain-English request, with any missing detail requested conversationally. Non-native English speakers and users unfamiliar with travel jargon can write informally, since city names, including older or renamed forms, are normalised to airport codes automatically. Finally, users prone to input errors are protected by deterministic validation, which rejects malformed dates, past dates, impossible passenger counts and invalid cabin classes before any search is issued.

The organisation of this paper is as follows. Section 2 reviews related work and states the literature gap. Section 3 describes the system architecture and the multistage processing pipeline. Section 4 gives implementation details and the rationale for selecting GPT-4o. Section 5 sets out the evaluation protocol, including ethics, recruitment, questionnaire items and scoring. Section 6 reports the results. Section 7 discusses strengths, limitations and future work, and Section 8 concludes.

2. RELATED WORK

2.1. Agentic AI and Tool-Using Language Models

Agentic AI is distinguished from conventional generative systems by autonomy of action rather than by fluency of output. Acharya et al. [1] and Wali et al. [2] survey the field and identify goal-directed behavior, contextual adaptation, and autonomous decision-making as its defining characteristics. Sapkota et al. [4] distinguish AI agents, which execute bounded tasks under supervision, from agentic AI, which coordinates multiple steps toward a goal. The system described here sits toward the bounded end of that spectrum: a single task domain permits stronger reliability guarantees than open-ended autonomy.

Tool-using language models supply the technical lineage. Yao et al. [3] interleave reasoning traces with tool calls (ReAct); Schick et al. [5] show that models can learn to invoke APIs (Toolformer); Wu et al. [11] generalise the pattern to multi-agent conversation (AutoGen). Those systems demonstrate that tool use is feasible, but they keep planning and error recovery inside the model or an agent runtime. This paper takes the opposite stance: the model is confined to four language tasks, and application code owns orchestration. Section 6.5 tests whether that confinement changes where failures occur.

2.2. Conversational Agents in Travel

Empirical work supports the feasibility of agentic travel assistants. Rishwanth et al. [6] present an agentic flight reservation system that automates search and booking through natural-language dialogue. Their system orchestrates a large language model to interpret free-form queries, extract structured booking parameters and interact with backend REST services, and uses contextual reasoning loops rather than rigid dialogue trees to obtain clarification. Across 100 booking scenarios they report a task success rate of 91%. Nethravathi et al. [7] extend the pattern to multi-model orchestration for live flight and hotel itinerary generation.

The present work differs in two respects. First, its scope is search rather than booking, which removes payment and reservation state from the trust boundary. Second, orchestration is not delegated to an agent framework: control flow, validation and ranking are implemented directly, and the language model is confined to four bounded language tasks.

2.3. LLM Reliability

Fluent generation is not reliable generation. Ji et al. [12] survey hallucination in natural language generation and distinguish intrinsic errors (inconsistency with the source) from extrinsic invention. In flight search both matter: an extractor that silently drops an unrecognised cabin class is an intrinsic contract failure; a summariser that relabels an airport is an extrinsic hallucination that reaches the user. Amershi et al. [13] advise scoping services when uncertain and making correction cheap; Weisz et al. [14] argue that generative applications must be designed for imperfection. If planning, retrieval, ranking and wording share one model call, a wrong itinerary cannot be attributed to a stage. If only wording and extraction live in the model, it can. That is the reliability argument tested here.

2.4. Usability, Accessibility and Personalization

Wali et al. [2] conclude that task-automation agents have outpaced adoption because usability was not treated as a first-class constraint. Langevin et al. [8] propose conversational-agent heuristics; Dubiel et al. [9] show that search-task design changes what an interactive evaluation can claim. Luger and Sellen [15] documented a gap between what users expect of conversational agents and what those agents can do. Three of Amershi et al.'s [13] human–AI guidelines—make clear what the system can do, support efficient correction, scope services when in doubt—map onto the consolidated clarification, the two correction pathways, and the hard refusal of booking intents.

Accessibility sharpens the same requirements. Czaja and Lee [16] show that older adults are disproportionately penalised by dense forms and unexplained errors. Conversational entry reduces that penalty only if missing fields are requested in plain language and an earlier mistake does not force a restart. The design is motivated by that population; the evaluation in Section 5.2 does not include it. Personalization uses a coarse, inspectable preference store—weighted counts over airline, stops, aircraft tier, price band and time of day—updated only from explicit approval or disapproval [17]. No recommendation is emitted until the score is strictly positive. Section 6.4 shows that users still wanted the why, not only the badge.

2.5. Literature Gap

The literature establishes that tool-using agents can complete travel tasks [6, 7], that language-model errors are hard to localise when control is also model-owned [12, 14], and that adoption depends on correction, clarification and explanation [2, 8, 13, 15]. What is missing is a working flight-search system that treats those three observations as one design constraint and then measures where failures land. That is the gap addressed here. Table 1 locates the design relative to rule-based and fully agentic systems.

Table 1. Rule-based and form-driven systems compared with agentic systems.

Criterion	Rule-based or form-driven system	Agentic system
Input mechanism	Structured entry through fixed form fields	Free-text natural language interpreted by a language model
Flexibility	Constrained to the options the form exposes	Adapts to the phrasing and completeness of the request
Handling of incomplete input	The user must locate and correct each field manually	Missing fields are detected and requested in a single clarification
Data integration	Static or manually refreshed sources	Live retrieval through external APIs at query time
Interaction cost	Repeated navigation and re-entry across sites	One conversational thread with retained context
Extensibility	Hard-coded workflows	Modular, tool-driven; new capabilities added as new tools
Failure attribution	Errors are usually field-level and visible in the form	Fully model-orchestrated agents bury errors in one trace; the hybrid design localises them to language-model stages

3. PROPOSED SYSTEM ARCHITECTURE

The system is a single-page React application served by a Node.js and Express back end. A user types a natural-language request; the backend classifies the intent, extracts entities via GPT-4o, validates them deterministically, retrieves live itineraries, ranks them along three independent axes, and returns both structured flight cards and a concise natural-language summary. No booking or payment function is implemented. The system is search-only by design, which keeps the evaluation focused on information-seeking performance and removes an entire class of transactional risk from a prototype.

The organising principle of the design is a strict division of responsibility: the language model interprets, and application code decides. Figure 2 shows the resulting three-layer structure.

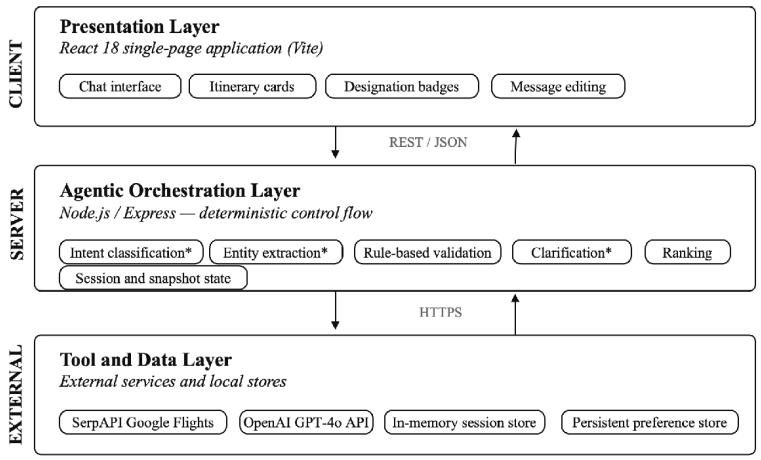


Figure 2. Three-layer system architecture. Only the four starred stages invoke GPT-4o; branching, validation, state management and ranking are ordinary application code, and all external credentials remain server-side.

3.2. Presentation Layer

The presentation layer is a React-18 single-page application built with Vite, comprising a landing page and the main chat window. The interface is chat-first rather than form-first: a scrollable message history, a free-text input, and itinerary cards embedded directly in the conversation. Each card presents the itinerary leg by leg, giving departure and arrival times, airport codes, airline, flight number and aircraft type, with layovers shown as labelled dividers between consecutive legs and any applicable designation badges displayed in distinct colours. A typing indicator and a disabled input convey that a response is pending.

Beyond ordinary message exchange, the layer supports message editing: selecting an earlier user message discards the conversation from that point onward and regenerates it. The layer contains no business logic and communicates with the back end solely through REST JSON, so that no credentials or tool access are ever exposed to the client.

3.3. Agentic Orchestration Layer

The orchestration layer coordinates every conversational turn. Each turn follows a fixed sequence implemented in application code: snapshot the current entity state; classify the incoming message against the full conversation history; if the intent is a flight search, extract entities and merge them into session state; validate deterministically, and either issue a single consolidated clarification or pass the request to the tool layer; score the returned itineraries on three independent axes; and generate a natural-language summary. GPT-4o is invoked at four points in this sequence and nowhere else.

State is maintained in two forms. Per-session state comprises the message history, the current entity set, and a stack of per-turn entity snapshots, held in an in-memory store keyed by a session identifier. The snapshot stack is what makes message editing exact rather than approximate: when a user edits an earlier

message, the back end rewinds both message history and entity state to the point immediately before that turn, so that information introduced by now-discarded turns is genuinely forgotten rather than silently persisting into the reconstructed conversation. Because GPT-4o is itself stateless, the applicable conversation history is supplied on every call, and all durable state remains under the control of the back end.

3.4. Tool and Data Layer

The agent has access to two external services and two local stores, all mediated by the back end. The Google Flights engine of SerpAPI is the sole source of flight data. The interaction protocol consists of HTTPS GET requests returning JSON, parameterised by IATA airport codes, ISO-8601 dates, passenger count, cabin-class code and currency. One-way and round-trip searches are distinguished explicitly, and round-trip retrieval follows the two-step token protocol of Google Flights: an initial call returns outbound options each bearing a departure token, and a second call exchanges that token for the corresponding return leg, which the back end merges with the outbound into a single round-trip itinerary. A three-stage fallback chain improves robustness on thin routes: the search is first issued with the requested cabin class, retried without the class constraint if no results are returned, and finally falls back to general web-search snippets when no structured inventory exists.

GPT-4o is accessed through the OpenAI Chat Completions API for the four language functions. The in-memory session store holds per-session conversational state and is intentionally ephemeral. The preference store is persistent: it records weighted counts over itinerary attribute buckets, namely airline, number of stops, aircraft tier, price range and departure time of day, as a human-readable JSON file that survives session end and back-end restart, and it drives the Recommended for You designation on subsequent searches.

3.5. Multistage Processing Pipeline

Figure 3 shows the sequence executed for every user request. GPT-4o first determines whether the message constitutes a flight-search request. Rather than decomposing complex requests into sub-tasks, the system operates on a deliberately limited domain and rejects non-search intents, such as booking, cancellation or modification, with a hard redirect. Classification considers the whole conversation rather than the latest message alone, so that terse follow-ups, such as a bare date supplied in answer to a clarifying question, are recognised as continuations of an existing search rather than as new requests.

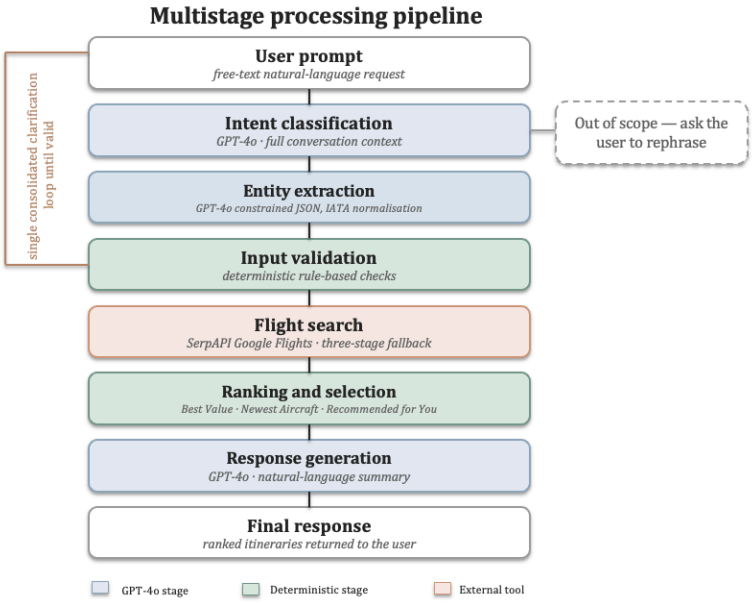


Figure 3. Multistage processing pipeline.

The second stage extracts the structured search parameters: origin, destination, departure date, optional return date, passenger count and optional travel class. This is implemented as constrained structured-output generation rather than conventional named-entity recognition. The model receives the full conversation together with the current date and returns a fixed JSON structure. Two constraints are enforced in the prompt. City names are normalised to IATA airport codes, so that Delhi becomes DEL and Bengaluru becomes BLR. Inference of unstated values is prohibited: any field the user has not explicitly supplied is set to null. Session entity state is updated whenever new values are extracted, which allows information to accumulate across turns.

Only origin, destination and departure date are strictly required. The other three fields are handled distinctly at search time: an absent return date denotes a one-way search, an unspecified passenger count defaults to a single adult, and an unspecified travel class defaults to economy, with that assumption disclosed explicitly in the response so that the user may request another class.

Validation is rule-based and implemented independently of the language model. It confirms that required fields are present, that dates are well-formed and correctly ordered, that the passenger count lies between one and nine, and that the cabin class is one of the four permitted values. Relative time expressions such as “tomorrow” or “next Friday” are resolved during extraction against the injected current date, but their validity is checked deterministically afterwards. Where validation fails, every detected problem is aggregated and GPT-4o generates one coherent clarification message covering all of them. Control returns to the user, and the cycle of classification, extraction and validation repeats until a consistent parameter set is obtained. This gate is what guarantees that the tool layer receives only clean requests.

Once validated, the query is issued against the Google Flights engine using the three-stage fallback described in Section 3.4. For round-trip queries the system performs bounded parallel execution: up to six outbound options are enriched concurrently, each exchanging its departure token for the matching return option in a separate call, with the results merged into unified round-trip itineraries. Aggregation then proceeds in code. Itineraries are scored on three independent axes: lowest price yields the Best Value designation; a heuristic aircraft-modernity score, under which a Boeing 787 or Airbus A350 ranks above an older narrow-body or regional type, yields Newest Aircraft, awarded to the itinerary with the highest average score across its legs; and alignment with learned preference weights yields Recommended for You.

The preference mechanism is feedback-driven. Users may register approval or disapproval of any displayed itinerary, and each judgement adjusts the weighted counts over that itinerary's attributes. On later searches every candidate is scored against these weights and the highest-scoring option is recommended, but only if its score is strictly positive; where no feedback history exists, no recommendation is shown rather than an arbitrary one. Because the three designations are computed independently, they may fall on a single itinerary or on three different ones, giving complementary rather than redundant decision signals.

Finally, GPT-4o synthesises the ranked results into a concise natural-language summary, delivered alongside the individual itinerary cards. Conversational context is retained throughout, so a user may change a date or a cabin class and re-enter the same pipeline without losing previously established parameters.

4. IMPLEMENTATION

4.1. Technology Stack

The back end is written in Node.js (v18 or later) using the Express framework, with the native Fetch API for all outbound HTTP requests. The front end is a React-18 single-page application bundled with Vite, using React Router for navigation and react-markdown to render model-generated text. Language reasoning uses GPT-4o through the OpenAI Chat Completions API. Flight data is retrieved through SerpAPI, using the Google Flights engine for structured fare search and the general Google engine for the web-snippet fallback. Automated tests use the built-in Node test runner, with no additional testing framework.

No agent-orchestration framework such as LangChain or LangGraph is used. All orchestration is implemented directly in application code. This is a deliberate choice rather than an omission: it keeps the control flow fully inspectable, which is a precondition for the failure-attribution analysis reported in Section 6.5.

4.2. System Configuration

The following parameters are recorded for reproducibility. Deterministic reasoning calls, namely intent classification and entity extraction, run at temperature 0; user-facing generation, namely clarification messages and result summaries, runs at temperature 0.4. Prompting uses role-scoped system prompts for each function; injection of the current ISO date to anchor relative-date resolution; explicit anti-fabrication rules prohibiting inferred or defaulted values; constrained output contracts, namely a single-label response for classification and a fixed JSON schema for extraction, with defensive stripping of markdown code fences before parsing; and provision of the full conversation context on every call, since the model is stateless.

Operational limits are a maximum of six concurrent return-leg lookups per round-trip search, a fallback chain limited to three stages, a 60-second timeout with up to five automatic retries per OpenAI call, a validation bound of one to nine passengers, and a fixed four-value cabin-class vocabulary comprising economy, premium economy, business and first. API keys are held server-side in environment variables and are never transmitted to the client.

4.3. Why GPT-4o

The control path does not depend on GPT-4o. Orchestration, validation, ranking, state rewind and the preference store are ordinary application code. Any chat-completion model that can emit a constrained label and a constrained JSON object can occupy the four language slots.

GPT-4o was selected for those slots only. OpenAI's structured-output evaluation reported perfect schema adherence for gpt-4o-2024-08-06 with strict structured outputs, against well below 50% for gpt-4-0613 under prompting alone [18, 19]. Temperature-0 classification is a one-token contract; Section 6.6 records a mean of one completion token per such call. A single Chat Completions interface also served both deterministic and user-facing slots without a second vendor [19].

No multi-model ablation was run. Repeating the 30-trial protocol on GPT-4o mini, GPT-4 Turbo or an open-weight constrained decoder remains future work (Section 7). The claim supported here is narrower: once a model is confined to language tasks, remaining failures are language-model failures and can be localised.

5. EVALUATION METHODOLOGY

5.1. Functional Evaluation Protocol

Thirty scripted conversations were conducted across six input categories, with five trials per category. A turn is defined as one user message together with the assistant's reply.

A trial succeeds if and only if (i) the displayed itineraries match the intended origin, destination, date, passenger count and cabin class; (ii) those results appear within six turns; (iii) on correction trials the amended field changes and every other field does not; and (iv) no fabricated airport or city name is shown to the user. Edit-based corrections count turns in the final conversation state, so a successful edit is one turn. Trials that recover from an initial misclassification are successes but flagged.

The chat session was reset before every trial, and learned preferences were cleared before evaluation began. Given five trials per category, the reported rates are functional verification across representative phrasings, not statistical estimates. No confidence intervals are reported.

5.2. Usability Study Protocol

Ten participants each completed a standardized task: find flights for two passengers from Delhi to Mumbai on 21 October in business class, then amend the request to economy. The task was supplied in writing; no assistance was given. Time ran from the first keystroke to the amended results. Participants then rated the ten statements in Table 4 on a five-point Likert scale (1 = strongly disagree, 5 = strongly agree).

5.2.1. Ethics, Consent, Recruitment and Data Handling

The usability evaluation was a low-risk, non-clinical task plus an anonymous questionnaire. No health, financial or contact data were collected. Under the author's institutional policy for student project evaluations of this type, the activity was determined not to require formal IRB review: adult volunteers, no identifiable data, no deception, no more than everyday risk. No ethics-approval number was issued, and none is claimed.

Participants were a convenience sample of campus and personal contacts, aged 18 or older, regular internet users, completing a 15-minute English session with no payment. An information sheet stated purpose, task, duration, third-party fare retrieval, and the right to withdraw without reason. Verbal informed consent was obtained first. Ratings were stored under codes P01–P10 with no names, emails, device identifiers or IP addresses, on a local spreadsheet accessible only to the author, to be deleted two years after publication. Older adults were not recruited [9, 10, 16].

5.2.2. Items, Scoring and Analysis

The ten items were written for this study and are not a standardised instrument. Q1–Q3 measure ease of use, Q4–Q6 cognitive load (reverse-worded), Q7–Q8 result quality and explanation, and Q9–Q10 perceived accessibility and reuse intent. Reverse-worded items are scored as 6 minus the raw rating. Construct scores are item means; the overall score is the mean of 100 responses. Dispersion is the sample SD. No inferential tests, factor analysis or Cronbach's alpha are reported at n = 10. All participants were aged 18–30 and regular internet users.

5.3. Baseline Comparison Protocol

Three baselines were exercised on the same standardized task: manual search on Google Flights, the Myra chatbot of MakeMyTrip, and the free tier of ChatGPT. Each was scored on interactions required to reach live fares, whether those fares were live or estimated, and how the amendment was handled. As a single-task comparison, it is illustrative rather than statistically grounded.

5.4. Instrumentation

Token consumption was instrumented at the API client across six conversations spanning the distinct execution paths, comprising 28 logged GPT-4o calls and approximately 24,000 tokens.

6. RESULTS

6.1. Overall Task Success

The system completed 28 of 30 trials successfully, a task success rate of 93.3%. Table 2 summarises results by category, and Figure 4 presents the success rate and the mean turn count together.

Table 2. Functional evaluation results by input category (five trials per category).

Category	Representative prompts	Mean turns	Success
Vague or missing information	"Find me a flight"; "flights to Goa"; "two tickets please"	2.0	5/5 (100%)
Contradictory	Return date before departure; past date; 15 passengers; 30 February; "luxury" class	2.2	4/5 (80%)
Small or obscure airports	Kannur; Jharsuguda; Pakyong; Kalaburagi; Pasighat; Salem; Aurangabad	1.0	4/5 (80%)
Corrections	Class, date, passenger count or origin changed after an initial search	1.8	5/5 (100%)
Complete, single turn	"Two of us, DEL to BOM, business class, August 20"	1.0	5/5 (100%)
Complete, multiple turns	Route, then date, then class across three messages	3.0	5/5 (100%)
Overall	**30 trials across six categories**	—	**28/30 (93.3%)**

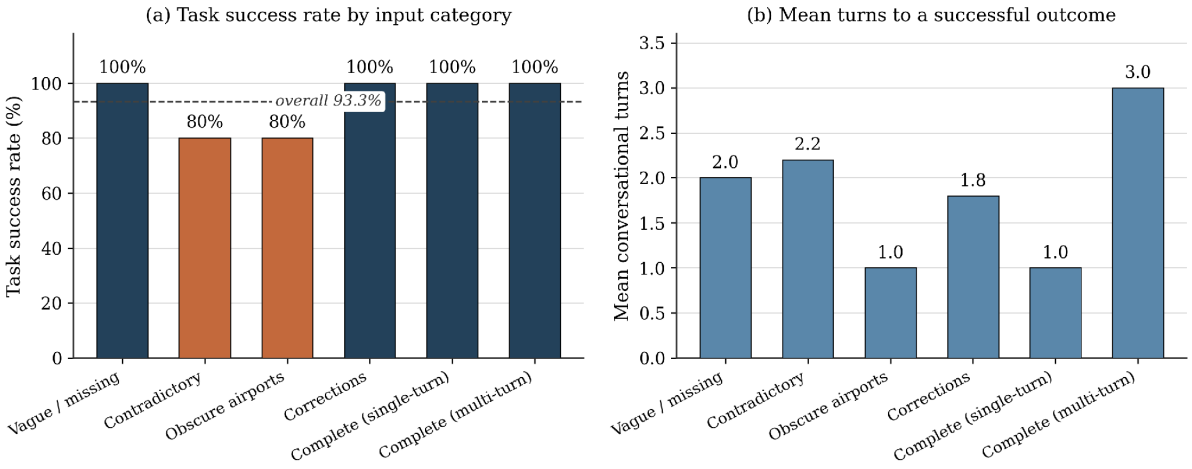


Figure 4. Task success rate (a) and mean conversational turns to a successful outcome (b), by input category. Categories containing a failure are shown in a contrasting color.

6.2. Behaviour by Input Category

All five vague queries were resolved in exactly two turns: one consolidated clarification requesting every missing field, followed by a single user reply. A sequential baseline that requested one field per turn would require approximately four turns for the same queries, implying a reduction of roughly half. One trial, “two tickets please”, was initially misclassified as an out-of-scope transactional request and received the booking refusal. The conversation nonetheless recovered within the same two-turn budget, because the refusal message itself solicits the required search fields; the trial is counted as a success but flagged.

In the contradictory category, four of five invalid inputs were caught before any search was performed: a return date preceding departure, a past date, an out-of-range passenger count, and an impossible calendar date. The last of these, 30 February, was treated as unusable rather than silently normalised to a nearby valid date. In every case all violations present were aggregated into a single clarification; in the passenger-count trial, the missing date and the exceeded passenger limit were raised together. Previously supplied valid entities were retained, so the user needed to re-supply only the offending field.

The fifth trial in that category revealed a genuine failure. An unrecognised cabin-class term, “luxury”, was silently mapped to null by the constrained extractor, which then triggered the economy default, and the generated disclosure inaccurately stated that no class had been specified. Because unrecognised values are absorbed during extraction, the deterministic cabin-class validator is unreachable through conversational input. This is an unintended interaction between the anti-fabrication constraint and the validation layer, and it is the single most consequential defect identified in the evaluation. The failed trial completed in one turn precisely because the invalid value was never challenged; correct behaviour would have added a clarification turn.

All five small-airport queries resolved to the correct IATA code, including a renamed city, Kalaburagi to GBI, formerly Gulbarga, and an ambiguous name, Aurangabad to IXU, where the alternative city of that name has no commercial airport. Two thin routes, Pakyong and Kalaburagi, returned no structured

inventory, and the web-snippet fallback was correctly triggered with an honest disclosure that no results were available. This distinguishes the extraction accuracy of the system, five of five, from the inventory coverage of the data source, three of five with structured results.

One trial in that category nonetheless failed, and it failed at the presentation layer rather than the retrieval layer. Pasighat was correctly extracted as IXT and searched accurately, but the response summary mislabelled the destination as "Leh", the city served by IXL, a hallucinated name for an airport code that lies outside the model's reliable knowledge. The fallback web snippets then drifted toward results for Leh, reinforcing the mislabel. The search was correct; the user was told otherwise. The trial is scored as a failure because the misidentification reaches the user despite correct underlying execution.

Corrections succeeded through both pathways, conversational amendment of a single field and direct message editing with state rewind, in every case retaining the unamended fields. The edit pathway recorded the lowest interaction cost in the evaluation, one turn, reflecting that editing replaces the original message rather than extending the conversation. Complete queries confirmed that turn count tracks information delivery exactly: single-message queries containing all parameters resolved in 1.0 turns and scripted three-message queries in exactly 3.0 turns, with no wasted clarification in either case. The multi-turn set also confirmed correct handling of international routes and abbreviated city names such as "Delhi to Syd".

6.3. Baseline Comparison

Table 3 and Figure 5 report the single-task comparison against the three baselines.

Table 3. Baseline comparison on a standardized task: two passengers, Delhi to Mumbai, 21 October, business class, subsequently amended to economy.

System	Interactions	Interaction notes	Live fares	Correction handling
Proposed system	2	Consolidated clarification; three independent badges; search	Yes	One-turn edit
MakeMyTrip (Myra)	3 t	Structured cards with a recommendation; booking-oriented	Yes	Error, then retry
ChatGPT (free tier)	Not reached	Fare estimates only; later described its output as live while still estimating	No	Partial, on estimates
Google Flights (manual)	9	Request decomposed across form fields; ranked results, no personalisation	Yes	Dropdown re-edit

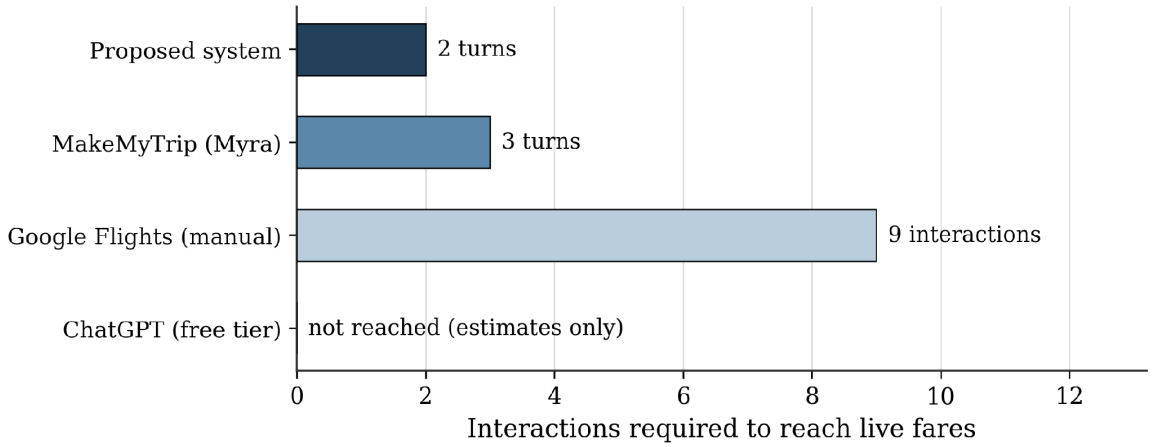


Figure 5. Interactions required to reach live fares on the standardised task. The general-purpose assistant did not reach live fares at any interaction count and is shown for completeness only.

The proposed system reached live results in the fewest interactions and handled the amendment most cheaply. The comparison should be read with two qualifications. First, it rests on a single task. Second, the systems are not strictly comparable in scope: the commercial chatbot supports booking and the manual interface exposes filtering capabilities that the proposed system does not implement, so a lower interaction count reflects a narrower task as well as a more efficient interaction model.

6.4. Usability and Cognitive Load

All ten participants completed the standardised task successfully. Mean time to completion was 106.4 seconds (SD 16.1, range 90 to 140), and every participant completed the task in exactly two conversational turns, independently reproducing the two-turn figure obtained in Section 6.3.

Table 4 reports per-item results and Figure 6 presents the item means with dispersion. The mean across all ten items was 4.24 on the five-point scale (pooled SD 0.87 across the 100 individual responses). Ease of use was the strongest construct at 4.43, with the two highest individual items being the absence of any need to learn the interface before use (4.70, SD 0.48) and the ease of expressing the request (4.60, SD 0.52). Two items reached unanimous agreement: every participant gave the maximum rating for having sufficient information to choose a flight and for willingness to use the system again (SD 0). This is a ceiling effect and should be interpreted cautiously given the small and demographically narrow sample.

Table 4. Usability questionnaire items and descriptive statistics (n = 10). Items Q4–Q6 are reverse-worded; reported means are reverse-scored. Construct means are item averages.

Item	Construct	Statement	Mean	SD
Q1	**Ease of use**	**I found it easy to express my flight request in everyday language.**	4.60	0.52

Q2	**Ease of use**	**I did not need to learn the interface before I could use the system.**	4.70	0.48
Q3	**Ease of use**	**I found it easy to change a detail of my request.**	4.00	—
Q4*	**Cognitive load**	**Entering my request required a lot of effort. (reverse-scored)**	3.60	—
Q5*	**Cognitive load**	**I felt overwhelmed by the returned options. (reverse-scored)**	3.80	—
Q6*	**Cognitive load**	**Comparing the returned options required a lot of effort. (reverse-scored)**	3.80	—
Q7	**Results**	**The results gave me enough information to choose a flight.**	5.00	0.00
Q8	**Results**	**I understood why a particular option had been recommended.**	3.70	0.67
Q9	**Access / intent**	**People with little prior familiarity with booking sites could use this system.**	4.20	1.32
Q10	**Access / intent**	**I would be willing to use this system again.**	5.00	0.00
	Ease of use (Q1–Q3)	**Construct mean**	**4.43**	—
	Cognitive load (Q4–Q6)	**Construct mean (reverse-scored)**	**3.73**	—
	All items	**Overall mean, 100 responses**		

Note. SDs marked "—" were not separately reported beyond the construct-level clustering in the text. Q1–Q3 average to 4.43; Q4–Q6 to 3.73; all ten items to 4.24.

Two areas rated lower. Cognitive load was the weakest construct at 3.73, its three items clustered between 3.60 and 3.80. Understanding why a particular option had been recommended received the lowest rating of any item that was not reverse-worded (3.70, SD 0.67). The badges communicate which option is favoured more successfully than why it is favoured. Perceived accessibility for users unfamiliar with booking sites rated 4.20 with the widest dispersion (SD 1.32).

These results are indicative rather than conclusive. The study used a single system with no control condition; with ten participants from one age band, the findings characterise technology-proficient young adults and cannot be generalised to the wider population the system is intended to serve.

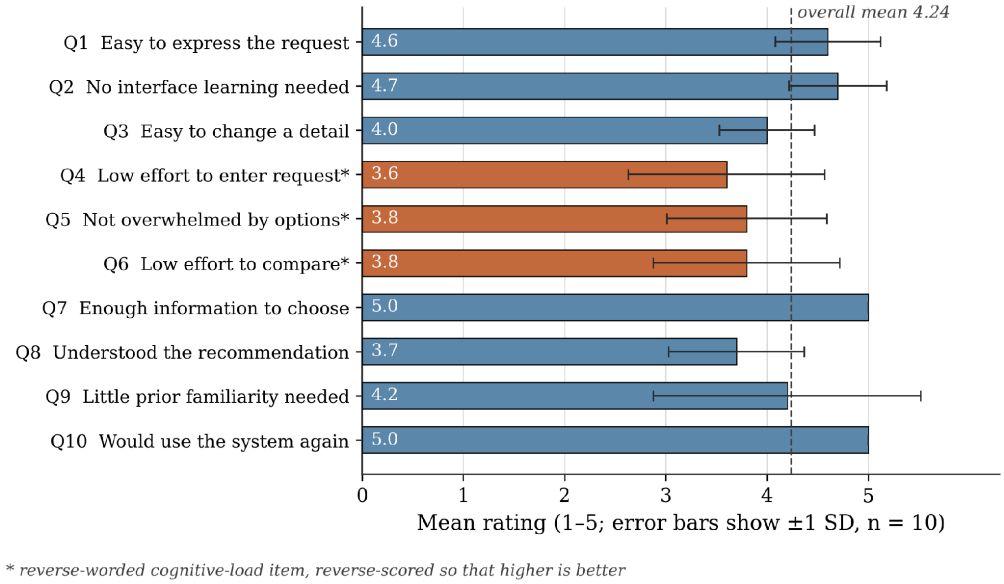


Figure 6. Mean usability rating by questionnaire item, with error bars showing one standard deviation (n = 10). Reverse-worded cognitive-load items are shaded differently and are plotted reverse-scored.

6.5. Error Attribution

Across the two failures and the one flagged misclassification, every observed error occurred in a language-model stage: intent misrouting at classification, silent absorption of an unrecognised value at extraction, and label hallucination at summarisation. None originated in the deterministic core of validation, orchestration or ranking. Figure 7 shows the distribution.

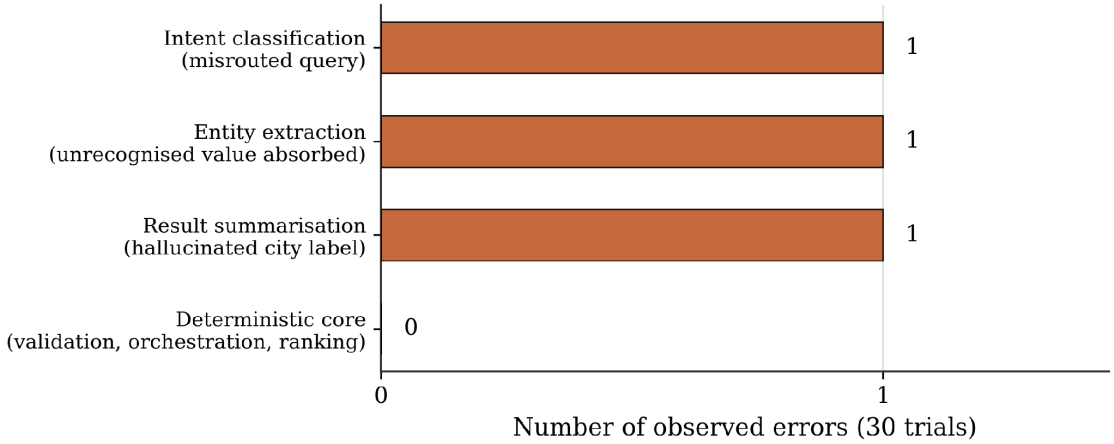


Figure 7. Distribution of observed errors by pipeline stage across 30 trials. All three instances arose at language-model stages; the deterministic core contributed none.

This result most directly supports the design argument of Section 3. Future reliability work should distinguish “unstated” from “unrecognised” during extraction, and should pass extracted city names

forward to the summariser rather than allowing it to re-derive them from airport codes. With only three error instances the pattern is suggestive rather than established.

6.6. Token Consumption

Instrumentation across six conversations recorded 28 GPT-4o calls and approximately 24,000 tokens. Prompt tokens accounted for roughly 95% of consumption. Result summarisation dominated, consuming about 67% of the total across only 7 of the 28 calls, driven by the serialised flight payload. Intent classification averaged a single completion token per call. Mean consumption was approximately 4,000 tokens per conversation. Figure 8 shows the breakdown.

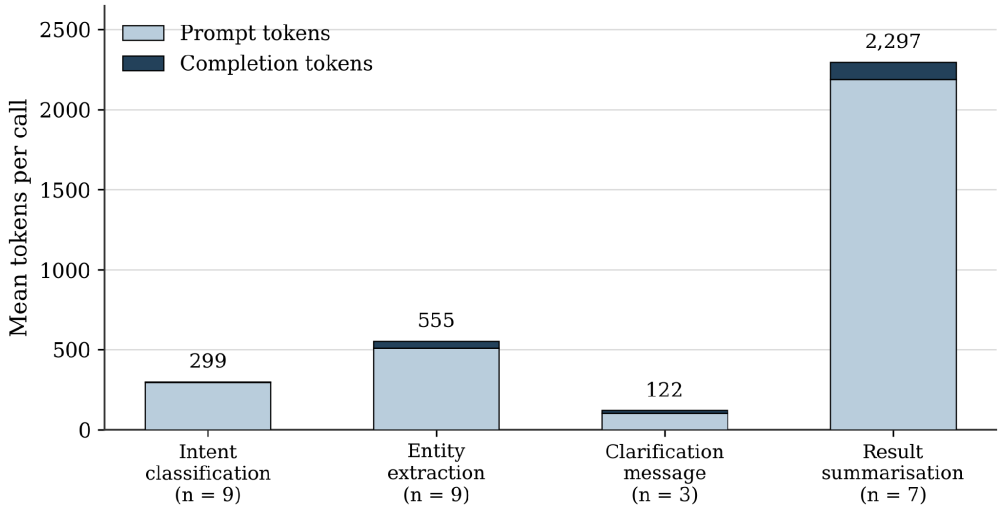


Figure 8. Mean prompt and completion tokens per GPT-4o call by reasoning function, measured over six instrumented conversations.

7. DISCUSSION

The central finding is not the headline success rate but where the failures were located. Confining the language model to four bounded language tasks concentrated failure modes at those four boundaries and left the remainder of the pipeline clean. Validation rejected every malformed input it was able to see, ranking produced no anomalies, and state rewind preserved unamended fields in all correction trials. The one validation failure occurred because an upstream language-model constraint prevented the invalid value from ever reaching it.

The consolidated clarification produced the clearest interaction benefit. Resolving every vague query in exactly two turns, against an estimated four for sequential questioning, halves the exchange count where users are most likely to abandon a conversational interface. The usability study reproduced the two-turn figure independently.

Several limitations constrain what these results support. Five trials per category and 30 in total suffice for functional verification, not statistical estimation. The usability study had ten participants from one age

band and no control condition; two items with zero variance indicate a ceiling effect. The baseline comparison rests on one task across systems of unequal scope.

A further limit is the single-model evaluation. GPT-4o was chosen for structured-output reliability (Section 4.3), not because the framework requires it. No ablation on other models was run. The architecture predicts that swapping the interpreter should change the language-stage error rate and leave the deterministic core untouched; that prediction is untested.

The system itself is bounded. Booking and payment are refused. Multi-city itineraries and flexible-date search are unsupported. A single flight-data provider means inventory gaps appear as system limits. Preference learning uses coarse buckets and needs explicit feedback. Older and less technology-proficient users, who most motivated the design, were not in the evaluation.

Three directions follow. First, distinguish unrecognised values from unstated ones at extraction so the validator becomes reachable. Second, ground the summariser in extracted place names. Third, surface the reasoning behind each designation. An evaluation with older adults, larger trial counts, a control condition, a multi-model ablation, and multi-city support would follow.

8. CONCLUSION

This paper presented an agentic conversational flight-search system built on a deliberate division of responsibility: a language model interprets natural language, and ordinary application code performs all orchestration, validation and ranking. GPT-4o occupies the language slots because it supports constrained structured outputs; the control design does not depend on that choice. The system accepts free-form English, accumulates parameters across turns, requests everything missing in a single consolidated clarification, retrieves live fares, and presents ranked itineraries with three independent decision signals.

Across 30 scripted conversations spanning six input categories the system succeeded on 28, a rate of 93.3%. Vague queries resolved in a mean of 2.0 turns, both correction pathways preserved unamended fields, and every observed error occurred at a language-model stage while the deterministic core contributed none. Ten participants rated the system 4.24 of 5 overall, with ease of use strongest and explanation of recommendations weakest.

The results are drawn from a small, single-model evaluation and should be read accordingly. What they do support is a design argument: keeping control flow outside the language model makes an agentic system’s failures attributable, and attributable failures are the ones that can be fixed. The specific defect identified here, silent absorption of unrecognised parameter values during extraction, is exactly the kind of finding that this architecture makes visible and that a fully model-orchestrated system would likely have obscured.

CONFLICTS OF INTEREST

The author declares no conflicts of interest regarding the publication of this paper.

DATA AVAILABILITY

The evaluation transcripts, anonymised questionnaire responses and token-usage logs supporting the findings of this study are available from the author on reasonable request. No identifiable participant data were collected.

REFERENCES

[1] Acharya, D.B., Kuppan, K. and Divya, B. (2025) Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey. IEEE Access, 13, 18912–18936. https://doi.org/10.1109/ACCESS.2025.3532853

[2] Wali, A., Mahamad, S. and Sulaiman, S. (2023) Task Automation Intelligent Agents: A Review. Future Internet, 15, 196. https://doi.org/10.3390/fi15060196

[3] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. and Cao, Y. (2023) ReAct: Synergizing Reasoning and Acting in Language Models. International Conference on Learning Representations (ICLR).

[4] Sapkota, R., Roumeliotis, K.I. and Karkee, M. (2025) AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. arXiv:2505.10468.

[5] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N. and Scialom, T. (2023) Toolformer: Language Models Can Teach Themselves to Use Tools. Advances in Neural Information Processing Systems (NeurIPS).

[6] Rishwanth, S., Yashwanth, L. and Shiny, R.M. (2025) Agentic AI for Conversational Flight Reservation Systems. Proceedings of the 2025 5th International Conference on Mobile Networks and Wireless Communications, Tumkur, 1–5. https://doi.org/10.1109/ICMNWC66779.2025.11354415

[7] Nethravathi, B., Gupta, P., Shree, C.P.L., et al. (2026) A Multi-Model Orchestrated Agent for Live Flight and Hotel Itinerary Generation. Discover Artificial Intelligence, 6, 423. https://doi.org/10.1007/s44163-026-01090-w

[8] Langevin, R., Lordon, R.J., Avrahami, T., Cowan, B.R., Hirsch, T. and Hsieh, G. (2021) Heuristic Evaluation of Conversational Agents. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.

[9] Dubiel, M., Halvey, M., Azzopardi, L. and Daronnat, S. (2020) Interactive Evaluation of Conversational Agents: Reflections on the Impact of Search Task Design. Proceedings of the 2020 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR).

[10] Prabhudesai, R. (2026) Evaluating Multimodal Agentic AI in Reducing Cognitive Load for High-Frequency Travelers.

[11] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D. and Wang, C. (2023) AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

[12] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A. and Fung, P. (2023) Survey of Hallucination in Natural Language Generation. ACM Computing Surveys, 55(12), 1–38. https://doi.org/10.1145/3571730

[13] Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P.N., Inkpen, K., Teevan, J., Kikin-Gil, R. and Horvitz, E. (2019) Guidelines for Human-AI Interaction. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3290605.3300233

[14] Weisz, J.D., He, J., Muller, M., Hoefer, G., Miles, R. and Geyer, W. (2024) Design Principles for Generative AI Applications. Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3613904.3642466

[15] Luger, E. and Sellen, A. (2016) "Like Having a Really Bad PA": The Gulf between User Expectation and Experience of Conversational Agents. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 5286–5297. https://doi.org/10.1145/2858036.2858288

[16] Czaja, S.J. and Lee, C.C. (2007) The Impact of Aging on Access to Technology. Universal Access in the Information Society, 5, 341–349. https://doi.org/10.1007/s10209-006-0060-x

[17] Resnick, P. and Varian, H.R. (1997) Recommender Systems. Communications of the ACM, 40(3), 56–58. https://doi.org/10.1145/245108.245121

[18] OpenAI (2024) Introducing Structured Outputs in the API. https://openai.com/index/introducing-structured-outputs-in-the-api/

[19] OpenAI (2024) Hello GPT-4o. https://openai.com/index/hello-gpt-4o/