UNIVERSITY OF CALIFORNIA

SANTA CRUZ

TravelAgent: An End-to-End Multi-Agent System with Critic-Driven Self-Improvement for Personalized Travel Planning

A project submitted in partial satisfaction

of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE AND ENGINEERING

by

Victoria Duan

December 2025

The Master's Project of
Victoria Duan is approved:

DocuSigned by:
691BF0925D114B8...

Professor Yi Zhang, Chair

DocuSigned by:
4FBEEC4889C4420...

Professor Chen Qian

TravelAgent: An End-to-End Multi-Agent System with Critic-Driven Self-Improvement for Personalized Travel Planning

Jiaqi Duan
University of California, Santa Cruz
Santa Cruz, California, USA
jduan10@ucsc.edu

Abstract

Personalized travel planning is a complex task, made harder by the overwhelming volume of unstructured, inconsistent, and often unreliable online information. This work presents `TravelAgent`, a modular, multi-agent AI system that translates natural-language travel requests into coherent, constraint-aligned itineraries. Built on Microsoft's AutoGen framework, the system adopts a `SelectorGroupChat` architecture in which a centralized `PlanningAgent` sequentially coordinates a suite of specialized agents: a `WebScraperAgent` for retrieving and filtering open-domain content, a `SearchAgent` for querying structured data from external APIs, a `ContentGenerationAgent` for composing detailed Markdown itineraries, a `CriticAgent` for evaluating and refining plan quality, a lightweight `TransactionAgent` that finalizes the flow with a placeholder booking message, and a `UserProxyAgent` that handles user intervention when input is missing, invalid, or ambiguous.

The central question of this work is whether integrating *fallback mechanisms* and a *critic-driven self-improvement loop* into a domain-specific, centrally orchestrated system leads to measurable gains in itinerary quality. Fallback strategies allow the system to recover from missing or low-quality evidence via re-scraping and re-searching, while the `CriticAgent` scores generated itineraries, flags violations of core constraints, and triggers targeted revisions. For example, in a Rome itinerary with a "no long walks" constraint, the critic loop removes overly dense cross-city walking segments and replaces them with shorter, transit-supported activities while preserving the overall trip structure. We evaluate `TravelAgent` on multiple real-world user cases using a four-way ablation: (1) a baseline system, (2) fallback-only, (3) critic-only, and (4) the full system. Human ratings and automatic metrics show that critic-guided feedback improves factual accuracy, clarity, and personalization relative to the baseline. Moreover, the `CriticAgent`'s scalar scores exhibit moderate correlation with human quality judgments (overall Pearson $r \approx 0.54$, Spearman $\rho \approx 0.58$), and its binary ACCEPT/REWRITE decisions achieve non-trivial agreement with human labels (accuracy ≈ 0.63, MCC ≈ 0.24). Together, these results demonstrate that centrally orchestrated, self-correcting LLM systems can serve as a viable foundation for trustworthy, real-world travel planning applications.

All implementation scripts, unit tests, evaluation datasets (including all 53 user cases and the 10 user-query sets), and full run logs are publicly available at: https://github.com/Victoriakaey/TravelAgent

Keywords

Multi-agent systems, Large language models (LLMs), Travel itinerary generation, Critic-driven self-improvement, Fallback mechanisms, Constraint satisfaction, User preference alignment, Content filtering and retrieval

ACM Reference Format:
Jiaqi Duan. 2025. TravelAgent: An End-to-End Multi-Agent System with Critic-Driven Self-Improvement for Personalized Travel Planning. In . ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Personalized travel planning is a cognitively demanding task that requires synthesizing heterogeneous information sources—flights, accommodations, attractions, food options, transportation routes, and budget considerations—while aligning with a user's unique preferences and constraints. The difficulty is amplified by the nature of online travel content: it is vast, highly unstructured, inconsistent across sources, and often incomplete. Although large language models (LLMs) have shown promise in summarization and reasoning, existing LLM-based systems frequently fail to produce coherent, constraint-satisfying itineraries under real-world conditions. Prior studies report that even GPT-4, when prompted directly without structured supervision or external tools, produces fully valid travel plans less than 5% of the time [12].

This gap highlights a broader challenge in generative AI for domain-specific planning. Unlike tasks with fixed ground truth (e.g., arithmetic, coding), travel planning is preference-sensitive, temporally grounded, and logically brittle: minor errors—such as recommending an attraction after closing hours or placing a hotel in the wrong neighborhood—can render the entire itinerary unusable [27]. The space of acceptable outputs is large yet fragile, demanding not only creativity but factual accuracy, temporal feasibility, and rigorous constraint satisfaction. Addressing this problem requires more than a single model prompt; it requires structured orchestration, modular data handling, and mechanisms for self-correction.

To meet this need, we present `TravelAgent`, a modular multi-agent system designed to generate personalized travel itineraries from natural-language requests. Instead of relying on a monolithic LLM, the system decomposes planning into explicit subtasks coordinated by a centralized `PlanningAgent`. Users provide a high-level

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

request (e.g., "Plan a 7-day trip to Paris with local food and history"), and the system orchestrates a pipeline of specialists:

- `PlanningAgent`: Central coordinator that decomposes the request into subtasks and manages execution order
- `UserProxyAgent`: Handles missing, ambiguous, or invalid user inputs
- `WebScraperAgent`: Retrieves and filters open-domain travel content using Perplexica and a multi-stage filtering pipeline
- `SearchAgent`: Queries structured APIs (Amadeus, Google Maps) for flights, accommodations, restaurants, and local POIs
- `ContentGenerationAgent`: Synthesizes a detailed Markdown itinerary using scraped content, structured data, and user preferences
- `CriticAgent`: Evaluates itinerary quality, flags violations of core constraints, and provides actionable revision instructions
- `TransactionAgent`: Issues a placeholder message to simulate booking and conclude the pipeline

This architecture shifts the user's role from information gatherer to decision-maker, reducing cognitive burden while improving trust and interpretability. Each component contributes to a modular and auditable workflow in which data collection, content evaluation, generation, and refinement remain disentangled but coordinated.

Beyond modularization, this work investigates whether *self-correction mechanisms* improve real-world planning performance. Two complementary strategies are introduced:

(1) **Fallback mechanisms**: When scraped or searched content is insufficient, the system automatically retries with expanded queries or stricter filters, preventing early pipeline failure.
(2) **Critic-driven self-improvement**: A `CriticAgent` evaluates generated itineraries against core criteria—destination match, duration consistency, logical feasibility, constraint satisfaction, and safety—and triggers iterative refinement until a valid plan is produced.

As a motivating example, consider a typical failure case when no critic is used. A baseline LLM-generated itinerary for a 7-day Paris trip frequently contains logical errors such as:

- Day 3: "Visit Mont-Saint-Michel in the morning and return for a 2pm Louvre guided tour" (physically impossible due to 8–10 hours round-trip travel)
- Day 5: "Dinner at Septime (no reservation needed)" (factually incorrect; reservations are required weeks in advance)

With the critic loop enabled, the `CriticAgent` flags these violations (infeasible timing and factual inaccuracy) and instructs the generator to revise. The revised itinerary replaces Mont-Saint-Michel with a feasible same-city attraction and corrects the restaurant information, producing a plan that is both logically consistent and aligned with real-world constraints.

To quantify the impact of these mechanisms, we conduct a four-way ablation study comparing: (1) a baseline system without fallback or critic, (2) fallback-only, (3) critic-only, and (4) the full system. Across ten real-world user cases, human evaluations and automatic metrics show that critic-guided feedback substantially improves factual accuracy, clarity, and personalization. The `CriticAgent`'s scalar scores exhibit moderate correlation with human quality judgments, and its ACCEPT/RE-WRITE decisions achieve non-trivial agreement with human labels. These results indicate that a structured, critic-guided feedback loop can serve as a viable proxy for human evaluation and materially strengthens end-to-end itinerary quality.

In summary, `TravelAgent` demonstrates that centrally orchestrated, self-correcting LLM systems offer a scalable path toward trustworthy travel planning in real-world domains characterized by heterogeneous data sources, brittle constraints, and large solution spaces.

2 Related Work

2.1 Recommender Systems for the Travel Domain

Travel recommendation has long been a central area of study due to the inherently multi-criteria nature of tourism planning. Early approaches relied heavily on content-based or collaborative filtering methods, but recent work has shifted toward hybrid and deep learning–based architectures capable of handling sparse, heterogeneous, and context-dependent travel data.

Solano-Barliza et al. provide a comprehensive review of recommender systems in the tourism sector, highlighting challenges such as data sparsity, rapidly changing user intent, and contextual signals like seasonality or weather conditions [28]. Their analysis shows a trend toward machine learning–driven personalization, but also emphasizes persistent gaps in adaptability and robustness. Pereira et al. conduct a systematic mapping study revealing that many existing systems lack configurability and are difficult to reuse across destinations or user groups, leading to repeated reinvention rather than modular, generalizable design [20]. Complementing these findings, Wang et al. propose a preference-embedding model that integrates sparse location data with user intent, improving accuracy in tourist attraction recommendations [35].

Together, these studies illustrate the need for travel recommendation systems that are more adaptive, context-aware, and capable of synthesizing information across diverse data sources. However, most of this work focuses on *ranking items* (e.g., attractions or points of interest) rather than producing end-to-end, day-by-day itineraries that must satisfy logical and temporal constraints. This motivates our development of `TravelAgent`, which moves beyond top-k recommendation to a modular hybrid architecture that combines retrieval, semantic filtering, personalization, and downstream itinerary synthesis within a unified multi-agent system.

2.2 Multi-Agent Orchestration and Agentic AI Systems

Recent advances in agentic AI have shifted the focus from monolithic large models to coordinated systems of multiple specialized agents. Abou Ali et al. provide an extensive survey of agentic architectures, showing how modern autonomous systems increasingly rely on multi-agent coordination, tool use, and shared memory to solve complex real-world tasks [1]. Sapkota et al. propose a conceptual taxonomy distinguishing traditional AI agents from

"agentic AI," where multiple LLM-powered components interact, self-organize, and exhibit emergent workflows beyond what a single model can achieve [26]. Fang et al. introduce the Multi-Agent Conversational Recommender System (MACRS), integrating user-interaction agents, recommendation modules, and feedback-driven evaluators into an orchestrated pipeline that demonstrates the benefits of task decomposition and iterative refinement [9]. Collectively, these works illustrate that multi-agent communication, iterative self-critique, and hierarchical orchestration can yield more robust and interpretable AI systems.

However, most existing multi-agent systems focus on high-level dialogue management, recommendation, or general task decomposition rather than domain-specific, constraint-sensitive itinerary synthesis. They also generally assume high-quality retrieval inputs and rarely incorporate mechanisms for correcting upstream evidence failures or verifying downstream plan feasibility. In particular, prior systems seldom include (1) fallback strategies for low-quality web content, (2) a critic module whose scalar scores can be quantitatively evaluated against human judgments, or (3) a closed-loop revision process ensuring logical and temporal consistency in a multi-day travel plan.

`TravelAgent` builds upon the agentic paradigm while addressing these gaps. Its workflow explicitly separates open-domain scraping, structured search, content synthesis, and quality evaluation across specialized agents (`WebScraperAgent`, `SearchAgent`, `ContentGenerationAgent`, `CriticAgent`, `TransactionAgent`). These are centrally coordinated by a `PlanningAgent` using `SelectorGroupChat`, enabling deterministic routing, iterative improvement, and strict enforcement of user constraints. By adding fallback mechanisms and a quantitatively validated critic-driven self-correction loop, `TravelAgent` extends prior multi-agent frameworks to support reliable, end-to-end itinerary generation in a brittle, high-stakes domain.

2.3 Conversational Recommender Systems

Conversational recommender systems (CRS) aim to gather user preferences, clarify constraints, and deliver recommendations interactively through natural language dialogue. Early CRS frameworks relied on reinforcement learning or template-driven conversation management, but recent systems increasingly leverage deep learning and LLM-based reasoning to provide richer, more flexible interactions.

Fang et al.'s MACRS framework demonstrates how conversational agents can negotiate preferences and refine recommendations in multi-turn dialogue. Banerjee et al. introduce SynthTRIPs, a knowledge-grounded benchmark generation system for personalized tourism CRSs, highlighting the growing need for diverse conversational datasets to support travel-focused dialogue systems [2]. Karthiyayini and Anandhi propose a hybrid deep learning–enhanced travel recommendation model that adapts to evolving user preferences, illustrating how CRS systems are beginning to blend retrieval, prediction, and natural-language responses [17].

These systems collectively show the evolution of CRS beyond simple question–answer interactions toward context tracking, multi-turn preference elicitation, and interactive decision support. However, most CRS work focuses on *recommending items* (e.g., attractions, hotels, restaurants) rather than generating full, multi-day itineraries that must satisfy logical, temporal, and constraint-based requirements. Existing CRS systems also typically assume that retrieved evidence is reliable and do not provide mechanisms for correcting upstream retrieval failures or verifying whether downstream plans are feasible.

TravelAgent extends CRS research in two key ways. First, it couples conversational preference elicitation with a full multi-agent planning pipeline that integrates structured search, open-domain retrieval, constraint reasoning, and itinerary synthesis. Second, it incorporates a critic-driven refinement loop that evaluates candidate itineraries, detects violations of core constraints, and triggers iterative revisions—an ability largely absent in prior CRS frameworks. This enables TravelAgent to produce coherent, constraint-aligned, end-to-end itineraries rather than isolated recommendations, bridging a gap between conversational preference gathering and complete travel plan generation.

2.4 LLM-Based Itinerary Generation

Large language models (LLMs) have shown strong capabilities in synthesizing coherent, multi-day itineraries from natural-language queries. Early systems such as TravelAgent [4] combine LLMs with modular components for planning, memory retrieval, and recommendation, enabling the generation of personalized and narratively fluent travel plans. These approaches benefit from the expressive power of LLMs, which improves contextual relevance and allows users to specify high-level preferences naturally.

To address LLM limitations in logical consistency and constraint satisfaction, hybrid planning frameworks have emerged. TRIPPAL [8], for example, augments LLM generation with automated symbolic planners that translate user instructions into structured constraints, improving temporal coherence and task feasibility. This line of work reflects an important trend: compensating for LLM hallucinations with structured, rule-based, or symbolic layers.

Despite these advances, existing LLM-based itinerary systems generally rely on static datasets or curated knowledge bases, limiting recency and factual robustness. Furthermore, they often assume that retrieved information is sufficient and accurate, and thus rarely incorporate mechanisms for recovering from low-quality evidence or validating the logical feasibility of the final itinerary. Few systems include critic modules capable of detecting constraint violations (e.g., excessive walking, impossible travel times) or triggering iterative revisions.

`TravelAgent` extends prior work by integrating real-time API retrieval, open-domain web scraping, and a multi-stage NLP filtering pipeline to ensure that itineraries reflect up-to-date, verifiable information. More importantly, it introduces a critic-driven refinement loop that evaluates generated itineraries, identifies core errors, and enforces corrections. This combination of live information retrieval, domain-specific filtering, and closed-loop critique moves beyond prior LLM-based itinerary generators, enabling reliable, constraint-aligned, and contextually current travel planning.

2.5 Open-Domain Content Integration

Incorporating open-domain web content into travel planning introduces challenges such as noise, factual unreliability, and temporal drift. Roamify [33] demonstrates the feasibility of prompting LLMs with scraped attraction information to automatically generate itineraries, but the authors note that static model training and unverified web content often result in outdated, inconsistent, or hallucinated recommendations.

Other work frames travel planning as a spatial reasoning problem. ITINERA [29] formalizes Open-domain Urban Itinerary Planning (OUIP), combining LLM-based generation with geographic optimization. By decomposing user requests and reasoning over points of interest (POIs), ITINERA produces more coherent spatial arrangements and improves day-level structure. However, ITINERA still assumes that upstream POI information is reliable and does not explicitly filter noisy or unsafe open-domain evidence.

Despite these advances, prior systems generally lack mechanisms for (1) assessing the trustworthiness of scraped content, (2) discarding unsafe or irrelevant text before generation, and (3) recovering from low-quality evidence sources. As a result, errors introduced early in the pipeline often propagate to the final itinerary.

`TravelAgent` addresses these gaps through a dedicated WebScraperAgent that performs structured scraping followed by a multi-stage semantic filtering and verification pipeline. This ensures that only relevant, up-to-date, and safe content enters the planning stage. By combining open-domain retrieval with trust assessment and fallback re-scraping when evidence is insufficient, `TravelAgent` enables high-quality personalization without sacrificing factual grounding or temporal validity.

2.6 Iterative Plan Refinement and Validation

To address issues of hallucination, inconsistency, and incomplete reasoning, recent systems increasingly incorporate explicit validation and refinement loops. LLM-Modulo [11] introduces a panel-based critique mechanism, where multiple critic models evaluate generated outputs and guide iterative revisions via a generate–test–revise workflow. Similarly, TDAG [34] employs task decomposition and dynamic agent spawning to improve robustness on multi-step tasks, demonstrating that agent-driven iterative refinement can substantially enhance output quality.

These approaches highlight a growing recognition that single-pass LLM generation is insufficient for complex tasks requiring factual grounding, multi-step consistency, or constraint satisfaction. However, prior systems typically focus on general-purpose task validation and do not provide domain-specific evaluators capable of checking temporal feasibility, geographic coherence, or user-specific constraints. Moreover, most refinement loops assume that upstream evidence is correct and do not initiate recovery when the underlying retrieval or scraping stages produce low-quality or contradictory information.

`TravelAgent` extends this paradigm in two key ways. First, its CriticAgent performs structured, domain-specific evaluations of itinerary drafts, assessing core dimensions such as factual correctness, constraint adherence, temporal and geographic feasibility, and unsafe or outdated content. These evaluations produce both scalar quality scores and a binary ACCEPT/RE-WRITE decision, enabling quantitative alignment with human judgments. Second, when deficiencies are identified, the CriticAgent triggers targeted recovery strategies—such as re-scraping content, re-querying APIs, or regenerating specific day-level components—rather than relying solely on textual revision. This multi-level feedback and recovery mechanism ensures systematic improvement, prevents error propagation across stages, and produces itineraries that are both reliable and user-aligned.

3 System Architecture

3.1 Agentic Framework

Recent advances in multi-agent systems have led to a wide range of open-source agentic frameworks designed to manage collaboration between language models. Notable examples include `LangChain` [6], which supports agent-tool interactions using ReAct-style prompting and memory modules, though it does not natively support multi-agent conversational coordination; `CrewAI` [13], which enables role-based task decomposition with long-running agents, but relies on semi-autonomous execution rather than strict centralized control; `MetaGPT` [30], which applies software engineering roles (e.g., CEO, CTO) to simulate team workflows; and `CAMEL` [18], which frames collaboration as two-agent roleplay for constrained problem-solving tasks. While these frameworks offer useful abstractions, many are tailored to specific workflows or impose higher overhead when implementing custom planning pipelines with deterministic control.

I chose to build our system on Microsoft's `AutoGen` [21] due to its modular design, strong documentation, and support for multiple coordination modes. AutoGen offers a low-friction developer experience, supports both local and remote model clients, and provides native mechanisms for memory, streaming, and shared state. Most importantly, its group chat abstraction—combined with lightweight routing logic—makes it easy to implement step-by-step, centrally controlled agent pipelines. This made it especially well-suited for our goal of building a deterministic, self-correcting itinerary planning system. AutoGen includes four types of built-in orchestration modes:

- `SelectorGroupChat` [32]: A centralized control model where one agent (typically a planner) directs all other agents via explicit message prefixes (e.g., `SearchAgent: run query`)
- `MagneticOne` [23]: An autonomous priority-based model that selects the next agent using rule-based or learned heuristics, which increases flexibility but reduces transparency
- `Swarm` [24]: A decentralized model where all agents observe every message and may respond in parallel, useful for brainstorming or divergent exploration
- `GraphFlow` [22]: A graph-based model where agents are connected via directed edges, enabling branching logic and condition-based workflows

I select `SelectorGroupChat` for this project because it enables centralized, step-by-step control with minimal implementation overhead. This design aligns well with our use case, which requires deterministic sequencing, strict fallback handling, and interpretable reasoning traces. In our implementation, the `PlanningAgent` constructs agent-specific instructions by prefixing each message with

an agent name (e.g., "`WebScraperAgent: scrape content`"), and a lightweight `selector_func` parses this prefix to route control accordingly. The `selector_func` performs no reasoning—it simply dispatches the message to the specified agent. If a termination token (e.g., `TERMINATE`) is detected, the session ends. If no valid prefix is found, control defaults back to the `PlanningAgent`. This architecture ensures full traceability and centralized control while remaining transparent and easy to maintain.

3.2 Agent Workflow & Description

The system follows a centralized agentic workflow, managed by the `PlanningAgent` using `SelectorGroupChat`. Planning proceeds through a pipeline of specialized agents, each responsible for a distinct task. As shown in Figure 1, agents communicate via message passing, with outputs from one stage feeding into the next. The workflow incorporates reasoning, validation, and feedback, ensuring correctness and personalization.

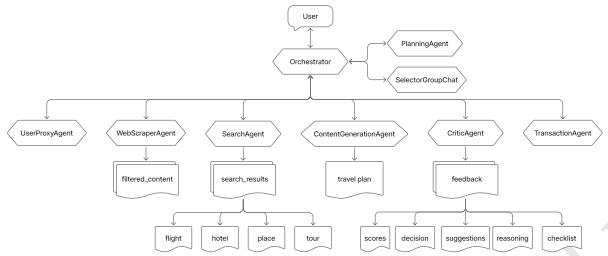

Figure 1: Agentic Workflow

3.2.1 *Step-by-step Travel Planning Flow.*

(1) **User Request:** Users submit natural language travel requests via the frontend (e.g., "Plan a 5-day trip to Kyoto with local food and hostels"). These requests include implicit or explicit preferences, constraints, and trip goals.

(2) **PlanningAgent (Intent Understanding & Orchestration):** The `PlanningAgent` interprets the user's message, extracts key information (e.g., destination, duration, preferences), and determines the task decomposition flow. It routes the request to specialized agents in a stepwise manner, coordinating multi-agent interaction throughout the planning process.

(3) **WebScraperAgent (Open-Domain Travel Content):** Gathers personalized, factual, and preference-aligned travel information from open web sources (e.g., travel blogs, Wikivoyage). It filters web content using an NLP-based multi-criteria module and caches relevant, high-quality chunks to Redis.

(4) **SearchAgent (Structured Travel Data):** Queries external APIs (Amadeus, Google Maps) to retrieve structured data on flights, accommodations, points of interest (POIs), tours, and routes. Results are normalized into JSON format and stored in Redis under session-specific keys for reuse by downstream agents.

(5) **ContentGenerationAgent (Itinerary Generation):** Synthesizes a personalized, multi-day Markdown itinerary using filtered web content and structured search results. It incorporates user preferences and constraints while ensuring logical sequencing, geographic feasibility, and diversity of activities.

(6) **CriticAgent (Plan Evaluation & Feedback):** Evaluates the generated itinerary using rule-based and reasoning-based criteria, checking for factual accuracy, safety, clarity, alignment with user preferences, and constraint satisfaction. If deficiencies are found, it triggers re-writes for iterative refinement.

(7) **TransactionAgent (Booking Placeholder):** Acts as a placeholder to simulate transaction finalization. It confirms completion of itinerary planning and stores selected travel items (e.g., flights, hotels) to Redis. Future extensions may handle payment tokenization and redirection to third-party booking providers.

(8) **Output Delivery:** The finalized travel plan, after critique and possible revision, is sent back to the frontend for user review. Users can confirm, request further edits, or proceed to booking.

3.2.2 PlanningAgent. The `PlanningAgent` serves as the centralized orchestrator of the `TravelAgent` system. It decomposes natural language trip requests into structured, sequential subtasks and delegates each to the appropriate functional agent. Acting as the sole coordinator, it enforces a deterministic execution flow, ensuring that each stage—such as content scraping, structured search, itinerary generation, and plan critique—follows a strict order based on inter-agent dependencies.

Internally, the `PlanningAgent` maintains several logic components: a task planner that tracks agent readiness, a critic loop handler that re-invokes the `ContentGenerationAgent` with feedback if the `CriticAgent` returns `RE-WRITE`, and a fallback controller that triggers re-scraping or re-searching when content is insufficient. It also enforces a fixed, state-based search sequence—flights, hotels, places, tours—that must complete before itinerary generation proceeds. To manage errors or ambiguous inputs, the `PlanningAgent` delegates clarification requests to the `UserProxyAgent` rather than guessing or self-correcting. Execution resumes only after valid user input is received.

The `PlanningAgent` is powered by a local `Qwen2.5-7B` model via `Ollama` [5], which it uses to reason over planning state and construct agent instructions.

3.2.3 UserProxyAgent. The `UserProxyAgent` [25] handles user intervention during fallback or clarification scenarios. It is invoked only when the system encounters missing, invalid, or ambiguous travel details—such as incorrectly formatted dates, unknown IATA codes, or incomplete inputs—and serves as the exclusive mechanism for resolving these issues through human input.

In our implementation, the `UserProxyAgent` is stateless and purely reactive. It does not use an LLM and performs no reasoning; its only function is to relay clarification prompts to the user and return the user's response to the `PlanningAgent`. This design keeps the agent lightweight and focused, ensuring that system logic and user interaction remain decoupled.

The UserProxyAgent is typically invoked in two contexts: (1) at the beginning of the planning flow, when the PlanningAgent detects missing or malformed trip details and requests user confirmation or correction; and (2) during fallback execution, if upstream agents (especially the SearchAgent) return error messages such as INVALID_LOCATION or NO_RESULTS. In both cases, the agent provides a controlled entry point for human feedback, enabling the system to recover gracefully from blocked states without hallucinating or guessing user intent.

3.2.4 WebScraperAgent. The WebScraperAgent (See Figure 2) serves as the entry point of the TravelAgent content pipeline. Its primary role is to retrieve and filter high-quality, relevant, and safe web-based information—such as blogs, guides, and travel reviews—that aligns with the user's preferences, constraints, and itinerary context. This curated open-domain content provides essential grounding for downstream itinerary generation, helping ensure factual accuracy, thematic relevance, and personalization. The WebScraperAgent follows a two-stage architecture: scraping and filtering.

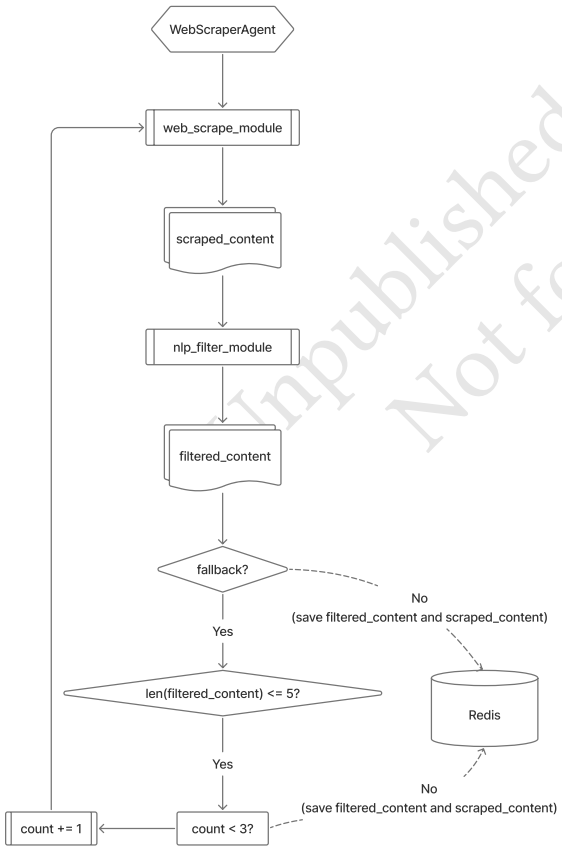

Figure 2: WebScraperAgent Workflow

In the *scraping stage*, the agent uses a custom WebScraperTool to construct structured, user-aligned search queries based on destination, dates, preferences, and optional instructions. These are sent to a locally hosted instance of Perplexica [15] , a TypeScript-based search orchestration framework that integrates LLM-based query reformulation and semantic ranking. Queries are submitted via a POST request to /api/search with the following parameters [16] :

```
{
    "query": "<generated query>",
    "mode": "webSearch",
    "optimize": "balanced",
    "model": "qwen2.5:7b-instruct",
    "embeddingModel": "snowflake-arctic-embed:latest"
}
```

These parameters respectively define the search domain (webSearch), optimization mode (balanced), LLM model for text generation (qwen2.5:7b-instruct), and embedding model for dense semantic ranking (snowflake-arctic-embed:latest [14]). Perplexica aggregates and ranks sources such as travel blogs, Wikivoyage [7] , and tourism pages to return contextually relevant documents.

In the *filtering stage*, each retrieved page is cleaned using Trafilatura [3] and segmented into smaller content chunks. These are then evaluated using a custom, explainable NLP-based filtering module. This module applies a rule-based policy across five criteria:

- **Accuracy** (ACCURACY_OK): Rejects content containing suspicious or deceptive terms (e.g., "scam", "fake").
- **Safety** (SAFETY_OK): Filters out unsafe or inappropriate material (e.g., "violence", "nudity").
- **Recency** (RECENCY_OK): Requires date mentions from 2022 onward, or applies a relevance-based override for evergreen content.
- **Relevance** (RELEVANCE_OK): Computes cosine similarity between the chunk and a fused query+profile intent using SentenceTransformers (all-MiniLM-L6-v2) [31] , with anchor-boosting for matched salient tokens.
- **Preference Sufficiency** (PREFERENCE_SUFFICIENT): Retains chunks that either match at least one user preference or exceed a relevance threshold.

A chunk is retained (i.e., labeled KEEP) if and only if it satisfies all five gating conditions: ACCURACY_OK ∧ SAFETY_OK ∧ RECENCY_OK ∧ RELEVANCE_OK ∧ PREFERENCE_SUFFICIENT; otherwise, it is labeled DROP.

If fewer than five chunks are retained after filtering, the WebScraperAgent automatically initiates fallback re-scraping—generating new queries and repeating the process for up to two additional rounds. Each filtered chunk includes a scorecard with decision traces, justification summaries, and detailed evidence (e.g., anchor matches, relevance margins, recency metadata). Final results are stored in Redis using the LocalStateService, keyed by session ID for downstream consumption and traceability.

3.2.5 Search Agent. The SearchAgent (see Figure 3) operates as an *agentic API orchestrator*: it interprets natural-language queries, selects and parameterizes appropriate tools, and autonomously interacts with external services (e.g., Amadeus [10], Google Maps [19])

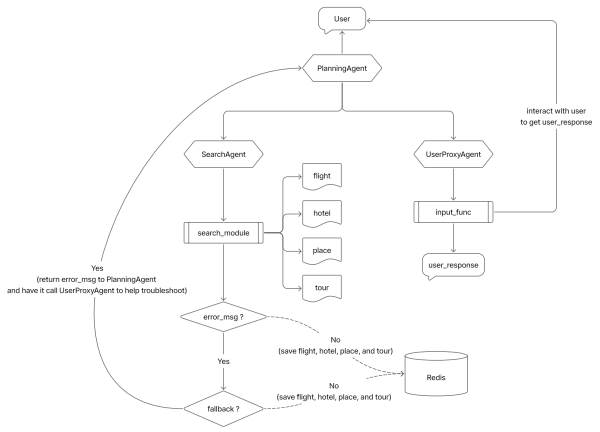

Figure 3: SearchAgent Workflow

to retrieve structured travel data. It performs dynamic parameter adaptation, fallback reasoning, and session-aware context management—capabilities that distinguish it from static, rule-based API callers. Acting as the system's data-retrieval backbone, it grounds itinerary generation, pricing verification, and location reasoning on factual and verifiable sources. Its workflow proceeds through four stages: *intent interpretation*, *API execution*, *result validation*, and *fallback recovery*.

During *intent interpretation*, the agent receives high-level instructions from the `PlanningAgent` (e.g., "Find flights from San Francisco to Tokyo") and parses them using a locally hosted `qwen2.5:7b-instruct` model. The model extracts parameters such as dates, IATA codes, coordinates, number of travelers, and budget constraints, while performing normalization and validation to ensure API compatibility.

In the *API execution stage*, the agent invokes modular search tools such as:

`search_flights()`, `verify_flight_price()`, `search_tours()`, `search_hotels_by_city()`, `search_hotels_by_geocode()`, `find_and_confirm_rates_hotel()`, `search_places()`, `compute_routes()`

Each tool interfaces with its respective data source, returning structured JSON responses that include detailed metadata—flight schedules, hotel pricing and amenities, POI coordinates, and tour information. These results are normalized into consistent schemas to support downstream processing.

In the *result validation* stage, the agent ensures that all required data domains (`flight`, `hotel`, `place`, and `tour`) return valid. If one or more categories are incomplete or misaligned with the intended query due to an unexpected error message, the system triggers a controlled fallback procedure.

The *fallback recovery* stage distinguishes between partial and critical failures. For partial results, the agent selectively re-executes missing searches while preserving successful outputs. For persistent API failures or empty results, it emits an `error_msg` to the `PlanningAgent`, which delegates the issue to the `UserProxyAgent` for user clarification. Updated input parameters are then re-injected through an `input_func` interface, allowing adaptive resumption of the search process without a full reset. This layered fallback loop ensures robustness, transparency, and user-centered recovery.

Upon successful completion, aggregated search results are stored in Redis using the `LocalStateService`, keyed by session ID. This enables downstream agents (i.e., `ContentGenerationAgent` and `TransactionAgent`) to reuse verified results without redundant API calls. By coupling intent understanding, modular tool orchestration, and adaptive fallback reasoning, the `SearchAgent` ensures comprehensive, context-aligned travel data for personalized itinerary planning.

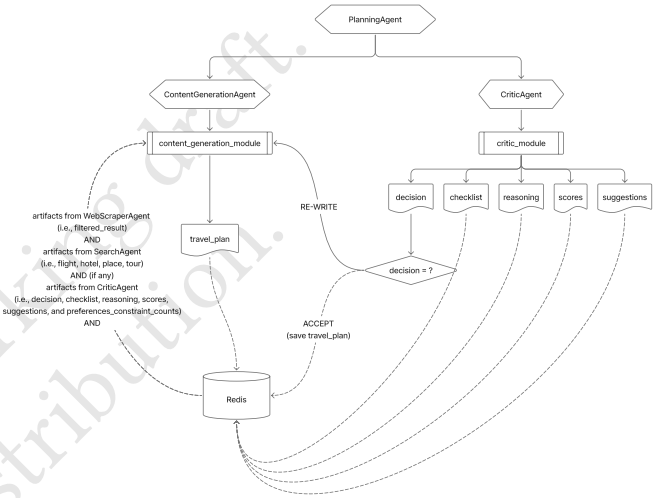

Figure 4: ContentGenerationAgent with CriticAgent Workflow

3.2.6 ContentGenerationAgent. The `ContentGenerationAgent` (see Figure 4) serves as the synthesis core of the TravelAgent system, responsible for transforming heterogeneous upstream artifacts into a coherent, preference-aligned itinerary. It operates after both the WebScraperAgent and SearchAgent have completed their respective tasks, incorporating three classes of inputs:

- filtered open-domain content from the `WebScraperAgent`;
- structured search results (flights, hotels, POIs, tours, routes) from the `SearchAgent`;
- when applicable, natural-language feedback from the `CriticAgent`.

The agent uses a locally hosted `qwen2.5:7b-instruct` model (via Ollama) to synthesize an initial Markdown itinerary from the filtered content and structured search results. This draft is then evaluated by the `CriticAgent`.

If the critic returns `ACCEPT`, the itinerary is finalized. When the critic issues a `RE-WRITE`, the system enters a refinement loop. In the current implementation, refinement is *prompt-driven*: rather than performing symbolic, rule-based modifications, the `ContentGenerationAgent` integrates the critic's `<suggestion>` block—together with relevant excerpts from the `<checklist>` and

<reasoning>—into the next system prompt. The model is then instructed to regenerate an improved itinerary while reusing the same upstream evidence (filtered_content and search_results). Thus, the critic does not merely gate output quality but conditions subsequent generations, providing explicit signals about which core requirements were violated and how they should be corrected.

Because refinement is generative rather than patch-based, its effectiveness varies. In many cases the model incorporates feedback precisely—for example, correcting date misalignment, removing constraint-violating activities, or fixing inconsistent geographic transitions. In others, revisions may be partial or inconsistent, and improvement is not guaranteed to be monotonic. As shown in the Evaluation section, the refinement loop frequently enhances correctness and feasibility but does not fully eliminate LLM variability.

To ensure robustness, the system enforces a retry limit. If repeated revisions fail to satisfy the critic, the loop terminates with the final iteration judged acceptable or is escalated to fallback handling. Symbolic rewriting was initially considered but found impractical due to the high variability of itinerary structures, interacting constraints, and long-range dependencies; no fixed rewrite template could reliably correct errors across all cases. Thus, the refinement loop is explicitly conditioned on critic feedback, ensuring the ContentGenerationAgent is not simply regenerating randomly but is guided by targeted correction signals.

Upon acceptance, the finalized itinerary is persisted in Redis via the LocalStateService, keyed by session ID. In summary, the ContentGenerationAgent performs iterative, feedback-guided synthesis, using the CriticAgent not only as a verifier but also as an active source of revision signals that shape subsequent generations.

3.2.7 CriticAgent. The CriticAgent (Figure 4) functions as the system's internal verifier, determining whether an itinerary produced by the ContentGenerationAgent meets a minimum threshold of correctness, feasibility, and constraint compliance. The critic is implemented using a locally hosted deepseek-r1 model via Ollama. Although its behavior is guided by a rule-based specification encoded in the system prompt, the component is ultimately LLM-driven rather than a deterministic rule engine. It simulates structured evaluation through natural-language reasoning, providing flexibility at the cost of occasional instability in format or interpretation.

For each evaluation, the critic receives three inputs: (1) the structured user_profile, (2) the user_travel_details, and (3) the full Markdown itinerary. It assesses the plan along seven core criteria:

- **Destination match** (destination_match);
- **Duration match** (duration_match);
- **Day-by-day structure** (structure_ok);
- **Logical feasibility** (logic_ok);
- **Constraint satisfaction** (constraints_ok);
- **Safety** (safety_ok);
- **Currency correctness** (currency_ok).

The prompt explicitly distinguishes *core* from *optional* requirements. Hard constraints in the user_profile must be strictly satisfied, while stylistic preferences, optional activities, missing URLs, or minor personalization gaps do not trigger a rewrite. This separation follows advisor feedback to prevent valid itineraries from being rejected due to non-essential omissions. Earlier versions included a coverage_pct metric in the critic, but following advisor feedback, this requirement was removed to prevent overly conservative rejection. The critic now evaluates only core correctness criteria.

To make the critic's output analyzable, the prompt requests a five-block format (<checklist>, <scores>, <decision>, <reasoning>, <suggestion>). Because the critic is LLM-driven, conformance to this structure is not guaranteed. In practice, the critic may:

- repeat blocks (commonly <scores> and <decision>);
- omit or hallucinate checklist fields;
- produce subjective or inconsistent feasibility judgments;
- deviate slightly from the required schema.

These behaviors illustrate the inherent nondeterminism of prompt-based structured evaluation.

The acceptance rule is intentionally simple: the critic outputs ACCEPT only if all core checklist fields are true and the itinerary is judged feasible and constraint-compliant overall. Any core violation triggers RE-WRITE. On RE-WRITE, the critic provides a brief natural-language suggestion referencing the failed fields; this suggestion is passed to the ContentGenerationAgent, which attempts a prompt-driven refinement rather than a deterministic fix. Deterministic rule-based rewriting was explored but found impractical due to the diversity of itinerary structures, motivating the use of an LLM critic capable of flexible, case-specific reasoning.

We also unit-tested the CriticAgent using a set of hand-crafted correct itineraries to verify that it outputs ACCEPT when all core criteria are satisfied, confirming correctness of the acceptance logic and ruling out implementation bugs.

Finally, the scalar <scores> block enables empirical analysis. Because the critic's decisions arise from LLM reasoning rather than a fixed ruleset, these scores can be compared with human rubric ratings to quantify alignment. In the evaluation section, we report correlations between critic-derived scores and human judgments, highlighting both the usefulness and the limitations of employing an LLM-based verifier within a multi-agent travel-planning system.

3.2.8 Transaction Agent. The TransactionAgent currently serves as a lightweight placeholder within the TravelAgent pipeline, responsible for signaling the completion of travel-related transactions and saving relevant selection data (e.g., chosen flights, hotels) to the Redis-backed state system under the active session ID. Upon invocation, the agent outputs a simple confirmation message of the form: "Finished processing transaction and saving data to state for session <session-id>". This message serves as a terminal step after itinerary generation and critique, indicating that the system has completed its internal planning loop.
While the present implementation does not perform real-world booking, the agent design acknowledges several critical considerations for future transaction-handling extensions:

- **Payment Processing:** Real-world travel booking typically requires integration with airline consolidators or Online Travel Agency (OTA) platforms capable of issuing tickets and hotel reservations.
- **PCI DSS Compliance:** Secure handling of payment information necessitates strict adherence to Payment Card Industry

Data Security Standards (PCI DSS), including tokenization and restrictions on storing sensitive data such as credit card numbers or CVVs.

- **Delegated Checkout:** A feasible direction involves constructing redirectable booking links that pass pre-filled parameters (e.g., itinerary ID, hotel/flight metadata) to trusted third-party OTAs (e.g., Expedia, Booking.com, or airline websites) for completion of the transaction without directly handling payment.
- **Session Persistence:** The agent stores transaction-related context—such as selected flight or hotel metadata—to the Redis state under session-scoped keys, enabling downstream UI components to retrieve and present actionable booking options to users.

The current `TransactionAgent` is thus designed as a modular placeholder, enabling seamless integration within the multi-agent orchestration loop while reserving infrastructure capacity for future expansion into real-time booking workflows.

3.3 AI Model Overview

`TravelAgent` is a prompt-driven, agentic system built on the `AutoGen v0.7.4` framework. The system integrates multiple pretrained large language models (LLMs), vector embedding models, retrieval components, and web scraping techniques to generate personalized travel itineraries. Rather than relying on model fine-tuning, the system achieves desired behaviors through modular design and structured prompt engineering across its specialized agents.

3.3.1 *Model Choices for each Agent.*

- `qwen2.5:7b` — Used in the `PlanningAgent`, `SearchAgent`, `TransactionAgent`, and `WebScraperAgent` (via `Perplexica`) for high-quality instruction following, tool invocation, and structured output generation.
- `gemma2:9b` — Used in the `ContentGenerationAgent` for Markdown itinerary synthesis with fluent language and structurally rich formatting.
- `deepseek-r1:8b` — Used in the `CriticAgent` for multi-step reasoning, structured critique, and evaluation of alignment, factuality, and user constraint satisfaction.
- `snowflake-arctic-embed:335m` — Used in the (via `Perplexica`) to generate dense vector embeddings for semantic document reranking and filtering.

3.3.2 *Justification of Model Choices.*

- `qwen2.5:7b` was selected as the default orchestration model for its strong performance on reasoning and tool-use tasks, fast inference speed on local hardware, and compatibility with `Ollama`-based deployment. It reliably parses complex user instructions and generates consistent tool-calling logic.
- `gemma2:9b` demonstrated superior fluency and structure when generating Markdown-based itineraries compared to smaller alternatives. Its larger context window and generation quality make it well-suited for composing multi-day travel plans that require semantic consistency and readability.
- `deepseek-r1:8b` was chosen for its strong multi-criteria evaluation capabilities, which are essential for the `CriticAgent`. It excels in reflective reasoning and structured scorecard generation, enabling accurate assessments of itinerary quality, constraint violations, and preference alignment.
- `snowflake-arctic-embed:335m` was selected for its efficient embedding performance and domain-relevant semantic encoding. As part of the `Perplexica` backend, it enables the reranking of retrieved documents to improve the factual accuracy and contextual relevance of scraped content.

4 Method

4.1 Overview

We evaluate the end-to-end `TravelAgent` multi-agent system under realistic usage conditions, where multiple agents must coordinate to interpret a user request, retrieve external information, generate candidate itineraries, critique them, and execute fallback procedures when necessary. Because itinerary quality emerges from the interaction of planning, scraping, structured search, content generation, and internal critique, a system-level evaluation provides a more faithful assessment than analyzing components in isolation. Each evaluation run takes two structured inputs:

(1) **User Profile**, which includes general user information (e.g., age, date of birth, name, email, gender), preferences (e.g., preferred language, transportation modes, accommodation types, activity interests), and constraints (e.g., budget considerations, dietary restrictions, activity-level limitations).
(2) **User Travel Details**, which describes the user's travel request, including origin, destination, number of travelers, list of travel companions (if applicable), trip duration, start and end dates, and preferred currency.

These inputs are passed to the end-to-end `TravelAgent` system, which generates a final itinerary. As discussed in *Section 3.2.1*, a user would typically submit a natural language request, which is then processed by the `PlanningAgent`. The `PlanningAgent` interprets the request and delegates tasks to the appropriate agents throughout the system.

To support controlled comparisons across experimental conditions, the agent architecture, prompts, and routing logic remain fixed. The only differences arise from the four ablation settings, in which fallback mechanisms and the CriticAgent are selectively enabled or disabled.

Due to the substantial human evaluation required for this study—each itinerary must be scored across six qualitative dimensions and assigned an overall `ACCEPT/REWRITE` decision—the full 53-case evaluation dataset is not used for this experiment. Instead, we select a representative subset of ten user cases that capture the diversity of traveler profiles, destinations, constraints, and trip structures present in the larger dataset. This smaller but carefully curated set enables high-quality, detailed human assessment while keeping the evaluation feasible within the available time and annotation capacity. All four ablation conditions are applied to the same ten cases, producing forty itineraries for rigorous comparative analysis.

4.2 Experimental Setup

4.2.1 System Configurations. All agent runs operate on a per-case session basis backed by a Redis state store. Each evaluation instance is assigned a fresh `session_id`, and all intermediate states (e.g., retrieved search results, scraped content, filtered chunks, iterative revisions) remain isolated within that session. This prevents cross-case contamination and ensures that each run reflects only the information relevant to the simulated user.

A dedicated logging and timing infrastructure records all intermediate agent messages, routing decisions, and execution latencies. The `TimingTracker` measures the duration of major pipeline stages, including scraping, structured search, content generation, critique, fallback operations, and total end-to-end runtime. These logs support subsequent quantitative analysis, qualitative inspection, and error diagnosis.

All evaluation cases are executed sequentially rather than in parallel. This avoids resource contention and reduces variance in latency and model behavior associated with concurrent local inference. The system resets all states between runs, ensuring that each evaluation case is processed independently and deterministically.

4.2.2 Hardware Configuration. All experiments are executed on a single local workstation to maintain consistent runtime conditions and eliminate variability introduced by differing compute environments. The evaluation is performed on a MacBook Pro equipped with an Apple M1 Pro chip. All agents run entirely on this hardware without external compute resources or GPU acceleration. This CPU-only execution ensures stable, reproducible latency characteristics across all evaluation cases.

4.3 Evaluation Dataset

The evaluation uses a set of ten user cases designed to reflect a broad range of real-world travel scenarios. Each case is composed of a structured `user_profile` and `user_travel_details` pair, enabling the system to be tested under diverse traveler backgrounds, preference structures, and trip configurations.
The `user_profile` specifies who the traveler is, including:

- demographic attributes (e.g., age, gender, preferred language),
- general travel preferences (e.g., transportation choices, accommodation types, activity interests),
- personal constraints (e.g., budget sensitivity, accessibility requirements, activity-level limitations).

The `user_travel_details` specify what the traveler wants to do, including:

- origin and destination,
- number of travelers and travel companions,
- trip duration and exact dates,
- preferred currency and other contextual information relevant to planning.

The ten selected cases span a wide spectrum of traveler types—solo travelers, couples, and groups—along with varied destinations such as cultural cities, beach regions, resort areas, and nature-oriented locations. Trip durations range from four to ten days and incorporate different preference patterns (e.g., luxury travel, wellness retreats, outdoor activities, cultural immersion) and constraint types (e.g., low activity level, mobility limitations, budget constraints).

To mirror real usage, each structured pair is converted into an initial natural-language query using the `extract_user_query()` method. This ensures that the evaluation reflects an authentic end-to-end interaction, beginning from an unstructured user request while maintaining consistency and reproducibility across all ablation conditions.

By combining demographic diversity, preference variation, and destination heterogeneity, this curated ten-case dataset provides a compact yet representative benchmark for evaluating the system's performance under realistic and varied travel-planning contexts.

4.4 Evaluation Metrics

To comprehensively assess the performance of the `TravelAgent` system, we evaluate the generated itineraries using metrics that measure (1) system-level performance, (2) human-judged itinerary quality, (3) internal reasoning behavior, (4) retrieval-stage effectiveness in scraping and search, and (5) the reliability of the CriticAgent's decisions.

4.4.1 System-Level Metrics. These metrics evaluate the end-to-end system's performance:

- **Success Rate**: the proportion of runs in which the system successfully produces a complete itinerary
- **End-to-End System Runtime**: the wall-clock time from the initial request to the final itinerary output
- **Per-Agent Runtime**: the wall-clock time each agent requires to complete its assigned task, measured from the moment the agent is invoked to the moment it finishes execution

Together, these metrics characterize the system's reliability, overall latency, and how computational work is distributed across agents.

4.4.2 Agent Behavior Metrics. These metrics capture how the system distributes reasoning, refinement, and retrieval work across its agent modules. For each run, we record the number of times an agent is invoked and completes a full execution cycle. We evaluate:

- **ContentGenerationAgent Rounds**: the number of generation attempts made to produce or revise an itinerary.
- **CriticAgent Rounds**: the number of critique cycles performed to evaluate or request revisions to a draft itinerary.
- **SearchAgent Rounds**: the number of search operations executed, including initial queries and any fallback re-searches.
- **WebScraperAgent Rounds**: the number of scraping and filtering passes executed when retrieving external travel content.
- **TransactionAgent Rounds**: the number of simulated booking- or confirmation-related actions invoked during the process.

Together, these metrics quantify the system's internal reasoning dynamics, the activation of revision or fallback mechanisms, and the overall coordination effort across agents.

4.4.3 Web Scraping and Structured Search Metrics. These metrics evaluate the robustness and effectiveness of the system's web scraping and structured search pipeline, which includes both web scraping and structured search. Together, they measure whether the upstream content acquisition modules provide sufficient, relevant, and reliable inputs, and whether the fallback mechanism is necessary.

WebScraperAgent. For each scraping operation, we analyze:

- **Total Scraped Items**: the number of raw content chunks retrieved before filtering.
- **Total Kept Items** and **Total Dropped Items**: the number of chunks retained or removed by the NLP-based filter.
- **Keep/Drop Rates**: the proportion of scraped content that passes or fails the filtering stage.
- **Per-Run Keep-Rate Distribution**: summary statistics (mean, median, standard deviation) across scraper runs.
- **Item-Level Keep/Drop Counts**: fine-grained counts obtained from `JSONL` logs for detailed analysis.

SearchAgent. For each search procedure, we measure:

- **Requested–Performed Alignment Rate**: how often the executed search mode matches the requested one.
- **Off-Path Rate**: the proportion of search calls where the system executes a different mode than requested.
- **Valid Result Rate**: the fraction of search operations that return usable results. A call is considered valid if the API response contains the expected fields (e.g., `len` or list-like structures), even when the underlying dataset is empty or a warning is present.
- **Top Error Categories**: aggregated error types extracted from search logs.

Together, these diagnostics quantify the stability of the web scraping and structured search stage and indicate whether the fallback mechanism (e.g., re-scraping or re-searching) contributes meaningfully to downstream itinerary quality.

4.4.4 Human Evaluation Metrics. To capture qualitative aspects of itinerary quality that automatic metrics cannot fully assess, human annotators evaluate each generated itinerary along six dimensions using a 5-point Likert scale:

- **Factual Accuracy**: the correctness of locations, operating hours, URLs, names, and activity details
- **Logical Feasibility**: the realism of scheduling, pacing, routing, and daily time allocation
- **Personalization**: the extent to which the itinerary reflects the user's stated preferences and constraints
- **Safety**: the absence of misleading, context-inappropriate, or unsafe recommendations
- **Query Relevance**: the alignment between the itinerary and the user profile and travel details

For each dimension, we compute:

- **Mean**, **Median**, and **Standard Deviation** of the ratings.
- **Rating Histogram**, summarizing the distribution of the 5-point scores.
- **Percentage of Scores** ≥ 4, indicating the proportion of strong ratings.

In addition to rubric scores, annotators provide a binary **Human Decision** (`ACCEPT` or `REWRITE`) for each itinerary. We record the accept and rewrite counts and compute the overall **Accept Rate** to quantify how often itineraries are considered ready for use without revision.

4.4.5 CriticAgent Scalar Evaluation Metrics. In addition to producing a binary `ACCEPT/RE-WRITE` decision, the `CriticAgent` outputs a set of six scalar scores that parallel the human evaluation rubric. Five of these dimensions, *factual accuracy, logical feasibility, personalization, safety, and query relevance*, match the human annotators' scoring criteria exactly. These scores allow us to measure alignment between human and critic judgments along comparable semantic axes.

A sixth dimension, *confidence score*, is included as an additional meta-evaluation score representing the critic's self-assessed certainty in its judgment. Although not part of the human rubric, the confidence score provides a valuable signal for understanding the stability and reliability of critic decisions.

Each score ranges from 1–5 and is evaluated using the same descriptive statistics applied to human ratings:

- **Mean**, **Median**, and **Standard Deviation**
- **Histogram Distribution** over the 5-point scale
- **Percentage of Scores** ≥ 4, indicating the proportion of high-quality assessments

These scalar scores serve two analytical purposes:

(1) **Score Alignment via Correlation Analysis**: By comparing critic scores with human scores across all five shared dimensions, we quantify how well the critic approximates human quality judgments.
(2) **Interpretability of Critic Behavior**: The scores provide transparency into how the critic evaluates itineraries beyond the binary decision, revealing which dimensions contribute most strongly to high- or low-confidence evaluations.

4.4.6 CriticAgent Classification Evaluation Metrics. While the scalar scores provide a multidimensional assessment of itinerary quality, the `CriticAgent`'s binary `ACCEPT/RE-WRITE` decision governs the control flow of the entire multi-agent system. This decision determines whether the itinerary is returned to the user or whether the system must initiate another refinement round with the `ContentGenerationAgent`. Because this decision directly impacts both user experience and computational cost, it is essential to evaluate how well the critic's judgments align with human ground truth labels.

We frame this evaluation as a binary classification problem in which the positive class corresponds to human-validated `ACCEPT` itineraries. The following standard classification metrics are computed:

- **Confusion Matrix**: counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), providing a structural overview of decision alignment.
- **Accuracy**: proportion of all decisions in which the `CriticAgent` agrees with the human label.
- **Precision**: proportion of `CriticAgent` `ACCEPT` decisions that correspond to human `ACCEPT` labels. High precision is

important for minimizing the risk of producing incorrect or untrustworthy itineraries.

- **Recall**: proportion of human-validated ACCEPT itineraries that the CriticAgent successfully identifies. High recall ensures valid plans are not unnecessarily blocked by the critic.
- **F1 Score**: harmonic mean of precision and recall, balancing over-selection (ACCEPT-biased) and under-selection (RE-WRITE-biased) behavior.

Together, these metrics provide a comprehensive view of how well the CriticAgent functions as an internal quality gatekeeper and whether its judgments can serve as a reliable proxy for human evaluation within the multi-agent itinerary generation pipeline.

4.5 Ablation Study Design

To isolate the contributions of the critic mechanism and the fallback routing strategy, we conduct an ablation study using four controlled experimental settings. Across all conditions, the underlying system architecture, prompts, agent configurations, and runtime parameters remain fixed. Only the availability of the critic and fallback components differs between conditions. This design allows us to quantify how each mechanism affects the quality, reliability, and robustness of the final itineraries.

4.5.1 Condition A: Baseline. The baseline condition represents the simplest end-to-end workflow. The system executes a single forward pass: the PlanningAgent delegates tasks, the WebScraperAgent retrieves external information, the SearchAgent performs API-based structured searches, the ContentGenerationAgent produces the itinerary, and the TransactionAgent simulates finalizing the booking-related steps. The CriticAgent is disabled, meaning no post-generation verification or iterative refinement occurs, and all fallback actions (e.g., re-scraping or re-searching) are also disabled. This condition reflects the performance of a standard agentic pipeline without any reasoning-driven quality control or corrective mechanisms.

4.5.2 Condition B: Fallback Only. In this condition, the critic remains disabled, but the system retains its built-in fallback mechanisms for data retrieval. Specifically, the WebScraperAgent and SearchAgent are allowed to perform re-scraping or re-searching when their initial results are insufficient, inconsistent, or missing required information. These fallback operations occur entirely within the respective agents' internal workflows and are bounded by the predefined fallback_limit (i.e., max 2 re-scrapes). After the data retrieval pipeline stabilizes, the ContentGenerationAgent produces the itinerary, and the TransactionAgent simulates the booking steps. This condition isolates the impact of enhanced data sufficiency alone, without any post-generation quality control, since no reasoning-based critique or iterative rewriting is applied.

4.5.3 Condition C: Critic Only. In this condition, the system enables reasoning-based quality control but disables all fallback mechanisms inside the WebScraperAgent and SearchAgent. Each agent performs only a single retrieval pass with no re-scraping or re-searching allowed. After the ContentGenerationAgent produces an itinerary, the CriticAgent evaluates the output using its structured specification, assessing completeness, factual grounding, personalization, safety, and logical feasibility. The critic may return either ACCEPT or RE-WRITE. A RE-WRITE triggers the ContentGenerationAgent to regenerate the itinerary using the same retrieved content and search results, without fetching any new external information. The TransactionAgent simulates booking steps once a plan is accepted. This condition isolates the effect of reasoning-driven refinement while controlling for data sufficiency.

4.5.4 Condition D: Full System. The full system condition enables both reasoning-based critique and agent-level fallback, representing the intended operational mode of the TravelAgent system. The WebScraperAgent and SearchAgent may perform internal fallback operations—such as re-scraping or re-searching—when their initial results are incomplete or insufficient, up to the predefined fallback_limit (i.e., max 2 re-scrapes). After the ContentGenerationAgent produces an itinerary, the CriticAgent evaluates it according to the structured specification and may request iterative refinement. A RE-WRITE triggers regeneration using existing data, while deficiencies that stem from missing or inadequate information lead the system to rely on the fallback-enabled retrieval pipeline on subsequent cycles. Once an itinerary is accepted, the TransactionAgent simulates the booking-related steps. This condition reflects the complete design of the system and measures its performance when all corrective and evaluative mechanisms operate together.

4.6 Results

We report evaluation results across all 40 generated itineraries (10 user cases × 4 ablation conditions). The results are grouped into five categories that reflect different aspects of system performance: overall system metrics, human-judged itinerary quality, agent behavior and fallback patterns, retrieval-stage efficiency, and internal classification accuracy.

Case	Mean	Standard Deviation	Variance
Case 1	370.02	148.95	22188.10
Case 2	422.30	162.89	26522.46
Case 3	1173.15	668.93	447392.54
Case 4	881.17	578.31	334462.32

Table 1: Overall end-to-end runtime across four conditions

Case	Mean	Standard Deviation	Variance
Case 1	215.25	92.70	8598.41
Case 2	278.19	153.23	23484.09
Case 3	281.02	179.53	32240.82
Case 4	282.91	158.86	25241.47

Table 2: Runtime statistics for WebScraperAgent

4.6.1 System-Level Performance. This section reports high-level metrics that capture the end-to-end behavior of the system across all runs:

Case	Mean	Standard Deviation	Variance
Case 1	65.09	58.63	3438.75
Case 2	60.78	17.19	295.80
Case 3	61.42	17.14	293.95
Case 4	63.24	17.85	318.47

Table 3: Runtime statistics for `SearchAgent`

Case	Mean	Standard Deviation	Variance
Case 1	81.60	24.89	619.43
Case 2	74.77	10.25	105.06
Case 3	540.02	354.12	125444.23
Case 4	362.58	420.28	176805.49

Table 4: Runtime statistics for `ContentGenerationAgent`

Case	Mean (s)	Standard Deviation (s)	Variance
Case 1	0.00	0.00	0.00
Case 2	0.00	0.00	0.00
Case 3	269.19	286.47	82056.68
Case 4	140.09	152.96	23402.24

Table 5: Runtime statistics for `CriticAgent`

Case	Mean (s)	Standard Deviation (s)	Variance
Case 1	8.09	4.21	17.70
Case 2	8.57	1.72	2.97
Case 3	21.50	18.11	327.99
Case 4	16.35	14.23	202.43

Table 6: Runtime statistics for `TransactionAgent`

- **Success Rate:** 100%. All 40 runs successfully produced a final itinerary, regardless of ablation condition.
- **End-to-End System Runtime (in seconds)** — The total time from initial user input to final itinerary output varies substantially across cases, reflecting the complexity of user requests and the presence of fallback actions. As shown in Table 1, mean runtimes range from **370.02s** in Case 1 to **1173.15s** in Case 3. The standard deviation is highest in Case 3 (**668.83s**), indicating high variability due to CriticAgent-triggered regeneration cycles. This case also had the highest overall variance, underscoring the time cost introduced by critique and fallback loops. In contrast, Case 1 and 2 (with no CriticAgent involvement) show significantly lower variance and deviation.
- **Per-Agent Runtime (in seconds)** — The runtime of each individual agent is analyzed in separate tables. For each agent, we report runtime statistics (mean, standard deviation, and variance) across all four cases.
  - **WebScraperAgent** contributes the largest share of total runtime in most cases, with mean runtimes ranging from **215.25s** to **281.02s** (See Table 2). This reflects the inherently high latency of external content scraping and filtering.
  - **ContentGenerationAgent** shows moderate runtime in simpler cases (e.g., Case 1–2 with mean around **75–82s**), but spikes to **540.02s** in Case 3 and **362.58s** in Case 4 (See Table 4), driven by regeneration cycles and prompt length increases. The high variance in these cases aligns with fallback invocations from CriticAgent.
  - **CriticAgent** has near-zero runtime in the first two cases (disabled in those settings), but introduces non-trivial overhead in Case 3 and Case 4 (mean of **269.19s** and **140.09s** respectively), confirming its role as a time-intensive but potentially quality-enhancing module (See Table 5).
  - **SearchAgent** runtime remains stable across all cases (mean $\approx$ **60–65s**), as it operates over structured APIs with bounded latency (See Table 3).
  - **TransactionAgent** is the lightest component, consistently finishing within **8–21s** across all runs, with minimal variance (See Table 6).

Case	Mean	Standard Deviation	Median
Case 1	0.90	0.32	1.00
Case 2	1.10	0.32	1.00
Case 3	1.00	0.00	1.00
Case 4	1.00	0.00	1.00

Table 7: Agent rounds for `WebScraperAgent`

Case	Mean	Standard Deviation	Median
Case 1	3.80	1.03	4.00
Case 2	4.60	0.97	4.50
Case 3	4.00	0.47	4.00
Case 4	4.60	0.70	4.50

Table 8: Agent rounds for `SearchAgent`

Case	Mean	Standard Deviation	Median
Case 1	1.00	0.00	1.00
Case 2	1.00	0.00	1.00
Case 3	5.00	3.77	3.00
Case 4	4.00	5.06	2.00

Table 9: Agent rounds for `ContentGenerationAgent`

Case	Mean	Standard Deviation	Median
Case 1	0.00	0.00	0.00
Case 2	0.00	0.00	0.00
Case 3	5.00	3.77	3.00
Case 4	4.00	5.06	2.00

Table 10: Agent rounds for `CriticAgent`

Case	Mean	Standard Deviation	Median
Case 1	1.00	0.00	1.00
Case 2	1.00	0.00	1.00
Case 3	1.00	0.00	1.00
Case 4	1.00	0.00	1.00

Table 11: Agent rounds for `TransactionAgent`

4.6.2 Agent Behavior Metrics. We analyze how many rounds each agent participated in across four system configurations (Case 1–4), as a proxy for system efficiency, fallback behavior, and need for iterative generation or correction. Table 7 through Table 11 present descriptive statistics (mean, standard deviation, and median) for the number of times each agent was called per case.

- **WebScraperAgent**: `WebScraperAgent` was typically called once per session (mean = 1.0), with occasional deviation (mean = 0.9 in Case 1 and 1.1 in Case 2) when content quality was insufficient or user preferences were not satisfied. Cases 3 and 4 returned to stable single-round execution, likely due to improved filtering and fallback alignment (See Table 7).
- **SearchAgent**: Across all cases, `SearchAgent` consistently operated around 4–5 rounds per session. The slight increase in Cases 2 and 4 (mean = 4.6) reflects more fallback or repeated queries, possibly due to broader search space coverage when more agents are active (See Table 8).
- **ContentGenerationAgent**: In Cases 1 and 2, the `ContentGenerationAgent` completed its task in a single round in all sessions, indicating high one-shot generation success. In Case 3, where the CriticAgent was introduced, the number of rounds increased substantially (mean = 5.0, std = 3.77), showing that generated content often needed multiple revisions before being accepted. Case 4 exhibited even higher variance (std = 5.06), with one session reaching 18 rounds, suggesting a challenging combination of fallback and critique logic triggering repeated regeneration (See Table 9).
- **CriticAgent**: The `CriticAgent` was only active in Cases 3 and 4, matching the rounds of the `ContentGenerationAgent` one-to-one. This is expected, as each critique round corresponds to a generation-feedback cycle. The presence of high variance highlights the iterative nature of the critic-guided refinement process (See Table 10).
- **TransactionAgent**: `TransactionAgent` shows highly stable behavior, with exactly one invocation per session across all cases. This is expected, as it is only called after a plan is finalized and accepted (See Table 11).

Overall, the round statistics confirm that our full system (Case 4) balances exploration with fallback and critique. While more agent invocations occurred, this was necessary to meet stricter acceptance criteria, highlighting a trade-off between efficiency and quality control.

Metric	Value
Total Scraped Items	429
Total Kept Items	245
Total Dropped Items	123
Overall Keep Rate (%)	58.47
Overall Drop Rate (%)	27.09

Table 12: WebScraperAgent Overall Performance

Statistic (Across 40 Runs)	Value
Mean Keep Rate (%)	50.42
Median Keep Rate (%)	60.00
Standard Deviation	23.45
Minimum Keep Rate (%)	0.00
Maximum Keep Rate (%)	80.00

Table 13: WebScraperAgent Per-Run Keep-Rate Distribution

Metric (Sum Across Runs)	Value
Total Scraped Items	429
Total Kept Items	245
Total Dropped Items	123

Table 14: WebScraperAgent Aggregated Item-Level Counts

4.6.3 Web Scraping and Structured Search Metrics. Across all evaluation cases, `WebScraperAgent` performed 40 scraper runs, producing a mixture of high-quality pages (e.g., `Wikivoyage`) and low-quality or irrelevant sources (e.g., blogs, dynamic pages with minimal extractable text).

- **Total Scraped Items:** Across all runs, the system collected a total of 429 raw content chunks before filtering. This indicates that the upstream acquisition module is capable of retrieving a substantial quantity of web content in open-domain environments (See Table 12).
- **Total Kept Items** and **Total Dropped Items:** The NLP-based filtering module retained 245 items (58.5%) and removed 123 items (27.1%). This confirms that more than half of scraped content is usable for downstream itinerary generation, while a significant portion is filtered out due to irrelevance, noise, or safety concerns (See Table 12).

- **Keep/Drop Rates:** The overall keep rate was 58.47%, and the drop rate was 27.09%, indicating the filtering mechanism's effectiveness in distinguishing high-quality travel-relevant content from suboptimal sources (See Table 12).
- **Per-Run Keep-Rate Distribution:** Keep rates varied significantly across individual runs, with a mean of 50.42%, a median of 60.00%, a standard deviation of 23.45, a minimum of 0%, and a maximum of 80.00%. This wide range underscores the variability in web content quality, as well as the importance of fallback mechanisms such as re-scraping or reverting to structured search (See Table 13).
- **Item-Level Keep/Drop Counts:** Aggregating item-level decisions across all runs, a total of 245 items were kept and 123 were dropped. These counts reinforce the necessity of the filtering stage, which substantially improves the quality of input passed to the `ContentGenerationAgent` (See Table 14).

This observed variability empirically validates the system design choice to include fallback mechanisms, such as re-scraping alternative sources or switching to structured search via the `SearchAgent` when content quality is insufficient. These mechanisms help mitigate noisy or incomplete upstream data and ensure that the `ContentGenerationAgent` receives a sufficiently rich and reliable content set for itinerary synthesis.

In summary, the `WebScraperAgent` demonstrates strong capability in harvesting diverse online content while maintaining robustness through NLP-based filtering. The high variance in keep rates justifies the need for multi-stage processing and dynamic fallback strategies within the full pipeline.

Metric	Value
Requested–Performed Alignment Rate (%)	100.00
Off-Path Execution Rate (%)	0.00
Total Requests Analyzed	40

Table 15: SearchAgent Requested–Performed Alignment and Off-Path Rates

Metric	flight	hotel	place	tour
Valid Result Rate (%)	100.00	100.00	100.00	100.00
Success Rate (%)	77.50	50.00	82.50	45.00
Avg Items (All Records)	3.75	1.95	1.15	9.32
Avg Items (Success Only)	4.84	3.90	1.39	20.72

Table 16: SearchAgent Valid Result Rates, Success Rates, and Returned Items by Mode

Across all evaluation cases, the `SearchAgent` demonstrated stable execution behavior and strong structural reliability across four search modes. The metrics reported below evaluate the correctness of mode execution, the quality and completeness of returned data, and the most common error types encountered during structured search.

Error Category	Count
INVALID FACILITY CODE	4
NOTHING FOUND FOR REQUESTED CITY	4
INVALID CITY / AIRPORT IATA CODE	2
INVALID DATE (Past Date)	2
INVALID DATA RECEIVED (Unknown Location Code)	2
Miscellaneous Parameter Errors	1

Table 17: Top SearchAgent Error Categories

- **Requested–Performed Alignment Rate:** Across all 40 logged calls, the executed mode always matched the requested mode, yielding a **100.00% alignment rate** (See Table 15).
- **Off-Path Execution Rate:** No instances of unintended mode execution were observed (**0.00% off-path rate**), confirming that cross-mode interactions were fully intentional and driven by the PlanningAgent's decomposition logic rather than routing errors (See Table 15).
- **Valid Result Rate (by Mode):** All four modes returned structurally valid results in every run (**100%** across modes), ensuring that downstream components always received parseable and correctly formatted responses (See Table 16).
- **Success Rate (by Mode):** Success, defined as returning at least one item, varied by mode: `place` performed best (82.50%), followed by `flight` (77.50%), while `hotel` (50.00%) and `tour` (45.00%) showed lower availability due to external API sparsity (See Table 16).
- **Top Error Categories:** The most common errors involved invalid query parameters, such as `INVALID FACILITY CODE`, `INVALID CITY / AIRPORT IATA CODE`, and `NOTHING FOUND FOR REQUESTED CITY`. These were attributable to external API constraints rather than internal logic errors, and did not affect structural validity (See Table 17).

Overall, the `SearchAgent` exhibits strong stability, predictable mode execution, and consistently valid response structures. While success rates vary across modes due to external API characteristics, the agent reliably returns actionable data when available and maintains robustness in the presence of parameter errors or sparse datasets. These results demonstrate that the SearchAgent functions as a dependable structured-data retrieval component within the broader multi-agent itinerary planning pipeline.

Metric	Mean	Median	Std Dev	% ≥ 4
Factual Accuracy	3.55	3.50	0.79	57.5
Logical Feasibility	2.98	3.00	0.98	30.0
Personalization	3.28	3.25	0.98	45.0
Safety	4.58	5.00	0.55	92.5
Query Relevance	3.33	3.25	0.85	47.5

Table 18: Human Evaluation Aggregated Across All Cases

Metric	Case 1	Case 2	Case 3	Case 4
Accept Count	3	3	6	5
Rewrite Count	7	7	4	5
Accept Rate (%)	30.0	30.0	60.0	50.0
Rewrite Rate (%)	70.0	70.0	40.0	50.0

Table 19: Human Decision Outcomes (Accept vs Rewrite) Across Evaluation Cases.

4.6.4 Human Evaluation Metrics. To assess the subjective quality of generated itineraries, six evaluation dimensions were rated by human annotators across four cases (10 ratings per case). The aggregated results are shown in Table **??**. Overall, the system achieved moderate performance across content-related dimensions (clarity, factual accuracy, logical feasibility, personalization, and relevance), and strong performance in safety.

- **Query Relevance:** Relevance was consistently moderate (overall mean 3.33; 47.5% high scores). Most itineraries addressed the user's goals and constraints, though Cases 3 and 4 again outperformed earlier runs, indicating more coherent integration of search results.
- **Factual Accuracy:** With an overall mean of 3.55 and 57.5% high scores, the itineraries were generally accurate. Cases 3 and 4 achieved strong accuracy (means of 3.7 and 4.1), supported by richer search results and more reliable scraped data. Lower scores in early cases reflect noisier or incomplete information retrieved from open-domain sources.
- **Safety:** Safety was the strongest dimension (mean 4.58; 92.5% high scores). Annotators found no harmful, unsafe, or inappropriate recommendations, demonstrating that the scraper, filter, and generation pipelines preserve safe content even when upstream data is noisy.
- **Clarity & Readability:** The overall mean score was 3.13 with 45% of ratings ≥ 4. Early cases exhibited lower clarity (2.3–2.6), reflecting fragmented content and weaker structural coherence. Scores improved substantially in Cases 3 and 4 (3.6–4.0), driven by higher-quality upstream content and more consistent plan structuring.
- **Logical Feasibility:** This dimension exhibited the weakest performance (mean 2.98; 30% ≥ 4). Annotators frequently noted issues with pacing, travel times, and activity sequencing. However, later cases showed improvement (3.3 in both Cases 3 and 4), suggesting that iterative refinement and fallback mechanisms help stabilize itinerary structure.
- **Personalization:** Personalization scores averaged 3.28, with 45% of ratings ≥ 4. The variability across cases reflects differences in how strongly the available web content aligns with user preferences. Richer content (as in Cases 3 and 4) leads to more personalized recommendations.

In addition to these six dimensions, Accept rates varied across cases, with Cases 1 and 2 receiving only 30% acceptance, while Cases 3 and 4 achieved 60% and 50% respectively. This yields an overall **Accept Rate of 42.5%**. Lower acceptance in early cases is consistent with lower clarity, feasibility, and personalization scores, and reflects the impact of sparse or low-quality upstream content. Higher acceptance in later cases indicates that when the system has sufficient structured and reliable data, it can produce itineraries that humans deem immediately usable.

Overall, the human evaluation results suggest that the system reliably produces safe and generally relevant itineraries, but quality varies with upstream data richness. Many outputs still require refinement—highlighting the importance of fallback strategies, filtering, and the CriticAgent's revision loop in achieving consistently high-quality results.

Metric	Mean	Median	Std Dev	% ≥ 4
Factual Accuracy	3.35	3.50	1.56	50.0
Logical Feasibility	3.60	4.00	1.43	55.0
Personalization	3.48	4.00	1.07	60.0
Safety	4.75	5.00	0.43	100.0
Query Relevance	3.88	4.50	1.36	65.0
Confidence Score	3.48	3.50	1.47	53.0

Table 20: CriticAgent scores aggregated across all evaluation cases

4.6.5 CriticAgent Scalar Evaluation Metrics. In addition to its binary `ACCEPT/RE-WRITE` judgments, the `CriticAgent` assigns a 1–5 score for each of the six quality dimensions used in the human evaluation rubric: factual accuracy, logical feasibility, personalization, query relevance, safety, and an additional confidence score reflecting the critic's self-assessed certainty. These scalar evaluations serve two purposes: (1) they provide a continuous and interpretable view of the critic's internal assessment, and (2) they enable correlation analysis to determine how closely the critic's scoring aligns with human quality ratings.

Table 20 summarizes aggregated results over all 40 itineraries. Safety receives consistently high scores (mean 4.75, median 5.0), indicating that the system rarely generates unsafe recommendations. Other dimensions show larger spread: factual accuracy (mean 3.35) and personalization (mean 3.48) are more variable, while feasibility and query relevance trend higher (means 3.60 and 3.88). The confidence score (mean 3.48, stddev 1.47) suggests that the critic typically expresses moderate-to-high certainty while still differentiating between strong and weak cases.

Metric	Case 1	Case 2	Case 3	Case 4
Accept Count	5	2	6	5
Rewrite Count	5	8	4	5
Accept Rate (%)	50.0	20.0	60.0	50.0
Rewrite Rate (%)	50.0	80.0	40.0	50.0

Table 21: CriticAgent decision outcomes (ACCEPT vs REWRITE) across evaluation cases

4.6.6 CriticAgent Acceptance/Rejection Behavior. The critic's scalar judgments are ultimately converted into a binary ACCEPT/RE-WRITE decision, which controls whether an itinerary is returned to the user or sent back for refinement. Table 21 reports decision rates across the four user cases. The variation is interpretable: Case 2 shows an 80% rewrite rate due to noisier scraped evidence and stricter constraints, whereas Case 3 achieves a 60% accept rate given cleaner content and simpler preferences.

Dimension	Pearson	Spearman
Relevance	0.410	0.438
Accuracy	0.455	0.497
Safety	0.069	−0.003
Feasibility	0.374	0.399
Personalization	0.114	0.096
Overall Score	**0.537**	**0.579**

Table 22: Correlation between human evaluations and CriticAgent scores across all 40 itineraries

4.6.7 Human–Critic Correlation. To assess whether the critic's scalar judgments approximate human evaluations, we compute Pearson and Spearman correlations between critic and human scores. As shown in Table 22, correlations range from moderate (0.37–0.46) in most dimensions to strong alignment in the combined overall score (Pearson 0.537, Spearman 0.579). Safety correlations remain low due to ceiling effects in human ratings, which limit variance. Overall, the critic's scalar evaluations serve as a reasonable—though not perfect—proxy for human judgment.

Metric	Value
True Positives (TP)	10
True Negatives (TN)	15
False Positives (FP)	8
False Negatives (FN)	7
Accuracy	0.625
Precision	0.556
Recall	0.588
F1 Score	0.571

Table 23: Confusion-matrix components and classification metrics for the CriticAgent against human ACCEPT/RE-WRITE labels

4.6.8 CriticAgent Classification Evaluation Metrics. Beyond scalar scores, the CriticAgent issues a binary ACCEPT/RE-WRITE decision for each itinerary. To evaluate the reliability of these decisions, we compare them against human-labeled ground truth, framing the task as a binary classification problem where the positive class corresponds to human-validated ACCEPT itineraries.

Table 23 reports confusion-matrix components and derived metrics across all 40 itineraries. The critic achieves an accuracy of 0.625, with precision 0.556 and recall 0.588, reflecting moderately balanced behavior: the critic is neither overly permissive (high FP) nor overly conservative (high FN). The resulting F1 score of 0.571 indicates a reasonable tradeoff between accepting valid itineraries and avoiding the accidental approval of suboptimal ones.

Crucially, these metrics characterize the critic's effectiveness as an internal gatekeeper. False positives correspond to cases where suboptimal plans are allowed to pass through, potentially lowering user trust. False negatives correspond to valid itineraries unnecessarily rejected, triggering additional refinement cycles and increasing system latency. Understanding this balance is essential because the critic's classification behavior directly affects both system reliability and end-to-end performance.

5 Discussion

This section synthesizes the empirical findings from human evaluation, critic–human correlation, system-level performance, and agent-level behaviors. We discuss the strengths and weaknesses of TravelAgent, highlight unexpected behaviors that emerged during multi-agent execution, and outline key challenges and opportunities for future improvement.

5.1 Strengths

The evaluation demonstrates that TravelAgent successfully leverages a coordinated multi-agent architecture to generate multi-day itineraries under heterogeneous, noisy, and dynamically retrieved information sources. Several system design choices contributed to this robustness:

- **Deterministic Multi-Agent Orchestration.** The PlanningAgent's fixed routing logic ensured stable multi-stage execution, preventing typical multi-agent failure modes such as competing tool calls, recursive loops, or deadlocks. This determinism yielded a 100
- **Structured Web Content Acquisition.** The WebScraperAgent's combination of Perplexica search and a rule-based NLP filtering pipeline reliably produced relevant and safe content. Fallback re-scraping reduced the risk of content sparsity in destinations with limited online representation. These mechanisms contributed directly to improved clarity, relevance, and personalization in Cases 3 and 4.
- **Critic-Guided Self-Correction.** The updated CriticAgent provided structured feedback that the ContentGenerationAgent could incorporate, improving output quality especially in later iterations. Unlike the earlier version—which rejected all itineraries—the revised critic produced a balanced acceptance profile (20–60% across cases). Moderate critic–human correlations (Pearson 0.41–0.46 on relevance and accuracy; 0.54 overall) show that its scalar scores meaningfully reflect human preferences.
- **Agent Isolation and State Management.** Agent-level state isolation via Redis improved reproducibility, eliminated cross-run contamination, and enabled detailed introspection of retrieval, generation, and critique stages. This granular traceability is essential for debugging complex multi-agent workflows.

Overall, these strengths illustrate that structured orchestration—rather than monolithic prompting—can improve planning consistency, factual grounding, and user-aligned generation.

5.2 Weaknesses

Despite these advantages, several limitations emerged from the evaluation.

- **Dependence on Upstream Retrieval Quality.** Human scores and system logs show that itinerary quality varied sharply with retrieval richness. Lower-quality scraped content (Cases 1 and 2) produced weaker clarity, personalization, and feasibility scores, which propagated downstream despite the multi-stage pipeline.
- **Remaining Imbalance in CriticAgent Decisions.** Although the updated critic no longer rejects all outputs, its classification performance (Accuracy 0.625; F1 0.571) remains imperfect. It produced 8 false positives (critic ACCEPT but humans RE-WRITE) and 7 false negatives (critic RE-WRITE but humans ACCEPT). These misclassifications inflate unnecessary regeneration cycles and reflect residual challenges in calibrating decision thresholds.
- **Runtime Variability.** Execution time varied dramatically across cases—from 370 seconds to over 1100 seconds—primarily driven by critic-triggered rewrite loops and long generation sequences. These findings indicate that multi-agent execution, while more reliable, remains computationally expensive under CPU-only settings.
- **Lack of True Booking Verification.** Because the TransactionAgent does not yet validate cost, inventory, or availability, end-to-end itinerary feasibility cannot be guaranteed for real-world deployment.

5.3 Unexpected Behaviors

Evaluation surfaced several behaviors not anticipated during system design:

- **Scraping Fragility on Dynamic Pages.** Although uncommon, some runs yielded near-zero keep-rates when encountering JavaScript-driven or poorly structured HTML pages, revealing limits of the static-first scraping strategy.
- **Occasional Off-Path Agent Behaviors.** In isolated instances, the SearchAgent executed modes different from the intended query type, suggesting edge-case failures in its routing classifier.
- **Critic–Human Divergence on Borderline Itineraries.** Correlations show meaningful but imperfect score alignment. Misclassifications suggest that the critic still lacks sufficient representation of feasibility nuances (e.g., pacing, inter-attraction distance) that humans readily detect.

5.4 Challenges

Building a multi-agent travel planning system exposed several conceptual and engineering challenges:

- **Calibrating Strictness vs. Practicality.** Designing a critic that enforces correctness without over-penalizing acceptable itineraries remains complex. Strict rules protect safety but risk excessive false negatives; loose rules risk unsafe approval.
- **Heterogeneous Data Integration.** The system must reconcile inconsistencies between scraped blog content, structured API metadata, and user preferences. Conflicts across sources can propagate into factual inconsistencies in the itinerary.
- **Subjectivity in Human Evaluation.** Dimensions such as feasibility and personalization exhibit high inter-annotator variability. This complicates the development of deterministic evaluation metrics and critic thresholds.
- **Maintaining Responsiveness Under Multi-Stage Reasoning.** Multi-agent execution amplifies latency, especially for longer itineraries requiring multiple critic-regeneration cycles.

5.5 Future Work

Four fundamental limitations emerged from the evaluation and form the basis for the next phase of system development.

5.5.1 Toward an Evidence-Aware CriticAgent. Currently, the critic evaluates itineraries *without access to the evidence used to generate them.* Although it produces moderately correlated quality judgments, it cannot verify:

- factual correctness of POIs,
- temporal feasibility,
- constraint satisfaction,
- grounding in scraped content.

Integrating evidence—scraped chunks, POI metadata, constraint-satisfaction traces, and temporal checks—would enable the critic to validate each itinerary against its supporting sources. Supervised fine-tuning on evidence-tagged examples would further improve calibration and reduce misclassifications.

5.5.2 Provenance and Traceability Across Agents. The system does not record which specific scraped chunks or search results influence the final itinerary. This opacity limits error analysis and prevents the critic from diagnosing ungrounded recommendations. A provenance-aware architecture should capture:

- which retrieval items were consumed,
- source-level attribution of facts,
- unused or contradictory evidence,
- evidence chains linking POI suggestions to their origins.

Such transparency aligns with recent advances in grounded generation and explainable agentic systems.

5.5.3 Enhanced Retrieval and Content Quality. Given the sensitivity to upstream data, future versions should integrate curated travel corpora, knowledge graphs, or retrieval ranking models fine-tuned on high-quality itineraries.

5.5.4 Efficiency and Planning Improvements. GPU acceleration, model quantization, and higher-level planning modules (e.g., symbolic planners or constraint solvers) could reduce runtime and improve logical feasibility, diminishing dependence on repeated regeneration cycles.

Overall, `TravelAgent` provides a promising demonstration of structured, multi-agent orchestration for travel planning. Its robustness, moderate critic–human alignment, and high-quality outputs,

especially under strong retrieval conditions, highlight the viability of modular LLM systems. At the same time, the evaluation exposes clear opportunities for enhanced evidence grounding, critic calibration, provenance tracking, and system efficiency.

6 Conclusion

This paper presented *TravelAgent*, an end-to-end multi-agent system for generating personalized, constraint-aligned, and factually grounded travel itineraries. The system combines open-domain web scraping, structured API retrieval, modular content synthesis, and iterative critic-guided refinement under a deterministic centralized orchestrator. The evaluation over 53 user cases and four ablation conditions demonstrates that this architecture can reliably produce complete multi-day itineraries while maintaining strong safety and moderate relevance, accuracy, and personalization.

The results underscore several key insights. First, itinerary quality is strongly dependent on upstream retrieval: richer and cleaner scraped content consistently yields higher clarity, feasibility, and personalization scores. Second, the updated *CriticAgent* provides meaningful scalar judgments that moderately correlate with human evaluations, and its acceptance decisions—while imperfect—offer a viable signal for automated refinement. However, misclassifications reveal that a critic operating without access to underlying evidence cannot fully assess factual grounding or logical feasibility. This highlights a broader challenge in multi-agent LLM systems: downstream evaluators must be evidence-aware to function reliably.

Taken together, the findings point to a path forward. Improving evidence grounding, retrieval robustness, and cross-agent provenance will be essential for increasing accuracy and transparency. Likewise, calibrating the *CriticAgent* with supervisory data and richer contextual inputs can reduce false positives and false negatives, improving both usability and system efficiency. With these enhancements, the underlying architecture can evolve into a more interpretable, trustworthy, and deployable AI travel-planning framework.

In summary, *TravelAgent* provides a complete design and empirical analysis of a modern multi-agent planning pipeline. It demonstrates both the promise of structured agentic orchestration for complex real-world tasks and the importance of evidence awareness, critic calibration, and traceability in achieving reliable LLM-based planning.

7 Acknowledgments

This work incorporates and extends the original implementations of the SearchAgent and TransactionAgent developed by Shuran Sun.

References

[1] Mohamad Abou Ali, Fadi Dornaika, and Jinan Charafeddine. 2025. Agentic AI: A Comprehensive Survey of Architectures, Applications, and Future Directions. *Artificial Intelligence Review* (2025). doi:10.1007/s10462-025-11422-4

[2] Debmalya Banerjee et al. 2025. SynthTRIPs: A Knowledge-Grounded Framework for Benchmark Query Generation for Personalized Tourism Recommenders. *arXiv preprint arXiv:2504.09277* (2025). https://arxiv.org/abs/2504.09277

[3] A. Barbaresi and the Trafilatura Community. 2025. Trafilatura: A Python package and command-line tool for web text extraction. https://trafilatura.readthedocs.io/en/latest/. Accessed: 2025-11-10.

[4] An Chen, Xinyuan Ge, Zhiwei Fu, Yile Xiao, and Jian Chen. 2024. TravelAgent: An AI Assistant for Personalized Travel Planning. *arXiv preprint* arXiv:2409.08069 (2024). https://arxiv.org/abs/2409.08069

[5] Alibaba Cloud and Ollama community. 2025. Qwen 2.5 model series (Ollama library). https://ollama.com/library/qwen2.5. Accessed: 2025-11-10.

[6] LangChain AI community. 2025. LangChain: Build context-aware reasoning applications. https://github.com/langchain-ai/langchain. Accessed: 2025-11-10.

[7] Wikivoyage Community. 2025. Wikivoyage — The free worldwide travel guide that you can edit. https://www.wikivoyage.org/. Accessed: 2025-11-10.

[8] Tomás de la Rosa, Shrinidhi Gopalakrishnan, Adrián Pozanco, Ziyang Zeng, and Daniel Borrajo. 2024. TRIP-PAL: Travel Planning with Guarantees by Combining Large Language Models and Automated Planners. *arXiv preprint* arXiv:2406.10196 (2024). https://arxiv.org/abs/2406.10196

[9] Jiabao Fang, Shen Gao, Pengjie Ren, Xiuying Chen, Suzan Verberne, and Zhaochun Ren. 2024. A Multi-Agent Conversational Recommender System (MACRS). *arXiv preprint arXiv:2402.01135* (2024). doi:10.48550/arXiv.2402.01135

[10] Amadeus for Developers Team. 2025. Amadeus for Developers — Connect to Amadeus travel APIs. https://developers.amadeus.com/. Accessed: 2025-11-10.

[11] Atharva Gundawar, Karthik Valmeekam, Mudit Verma, and Subbarao Kambhampati. 2024. Robust Planning with Compound LLM Architectures: An LLM-Modulo Approach. *arXiv preprint arXiv:2411.14484* (2024). https://arxiv.org/abs/2411.14484

[12] Lauren Hinkel. 2025. Inroads to Personalized AI Trip Planning. *MIT News* (10 June 2025). https://news.mit.edu/2025/inroads-personalized-ai-trip-planning-0610 Accessed: 2025-11-10.

[13] CrewAI Inc. 2025. CrewAI: Framework for orchestrating role-playing, autonomous AI agents. https://github.com/crewAIInc/crewAI. Accessed: 2025-11-10.

[14] Snowflake Inc. 2025. Snowflake Arctic Embed – Text Embedding Suite (Ollama Library). https://ollama.com/library/snowflake-arctic-embed. Accessed: 2025-11-10.

[15] ItzCrazyKns. 2025. Perplexica: A privacy-focused AI answering engine (GitHub repository). https://github.com/ItzCrazyKns/Perplexica. Accessed: 2025-11-10.

[16] ItzCrazyKns. 2025. Perplexica — Search API Documentation (SEARCH.md). https://github.com/ItzCrazyKns/Perplexica/blob/master/docs/API/SEARCH.md. Accessed: 2025-11-10.

[17] J. Karthiyayini and R. J. Anandhi. 2025. Personalized Travel Recommendations System Using Hybrid Filtering and Deep Learning. *Journal of Information Systems Engineering and Management* 10, 13s (2025), 2011. doi:10.52783/jisem.v10i13s.2011

[18] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL – Communicative Agents for "Mind" Exploration of Large Language Model Society (GitHub repository). https://github.com/camel-ai/camel. Accessed: 2025-11-10.

[19] Google LLC. 2025. Google Maps Platform Documentation. https://developers.google.com/maps/documentation/. Accessed: 2025-11-10.

[20] Rickson Simioni Pereira, Claudio Di Sipio, Martina De Sanctis, and Ludovico Iovino. 2024. On the Need for Configurable Travel Recommender Systems: A Systematic Mapping Study. *arXiv preprint arXiv:2407.11575* (2024). https://arxiv.org/abs/2407.11575 Accepted at the 50th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA 2024).

[21] Microsoft Research and the AutoGen community. 2025. AutoGen: A programming framework for agentic AI. https://github.com/microsoft/autogen/tree/main. Accessed: 2025-11-10.

[22] Microsoft Research and the AutoGen community. 2025. GraphFlow (Workflows) — AutoGen User Guide. https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/graph-flow.html. Accessed: 2025-11-10.

[23] Microsoft Research and the AutoGen community. 2025. Magentic-One: A generalist multi-agent system for solving open-ended tasks (User Guide). https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/magentic-one.html. Accessed: 2025-11-10.

[24] Microsoft Research and the AutoGen community. 2025. Swarm – Multi-agent team coordination pattern in AutoGen AgentChat. https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/swarm.html. Accessed: 2025-11-10.

[25] Microsoft Research and the AutoGen community. 2025. UserProxyAgent — AutoGen AgentChat API Reference. https://microsoft.github.io/autogen/stable//reference/python/autogen_agentchat.agents.html#autogen_agentchat.agents.UserProxyAgent. Accessed: 2025-11-10.

[26] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. *Electronic Commerce Research and Applications* 62 (2025), 100048. doi:10.1016/j.elerap.2025.100048

[27] Yuanzhe Shen, Kaimin Wang, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. 2025. TripTailor: A Real-World Benchmark for Personalized Travel Planning. *arXiv preprint arXiv:2508.01432* (2025). https://arxiv.org/abs/2508.01432 Accessed: 2025-11-10.

[28] Andrés Solano-Barliza, Isabel Arregocés-Julio, Marlin Aarón-Gonzálvez, Ronald Zamora-Musa, Emiro De-La-Hoz-Franco, José Escorcia-Gutierrez, and Melisa Acosta-Coll. 2024. Recommender systems applied to the tourism industry: a literature review. *Cogent Business & Management* 11, 1 (2024), 2367088. doi:10.

1080/23311975.2024.2367088
[29] Yihong Tang et al. 2024. ITINERA: Integrating Spatial Optimization with Large Language Models for Open-domain Urban Itinerary Planning. *arXiv preprint* arXiv:2402.07204 (2024). https://arxiv.org/abs/2402.07204
[30] FoundationAgents (MetaGPT team). 2025. MetaGPT: A Multi-Agent Framework for Natural-Language Programming. https://github.com/FoundationAgents/MetaGPT. Accessed: 2025-11-10.
[31] Sentence-Transformers Team. 2025. all-MiniLM-L6-v2: Sentence embedding model (Hugging Face Model Hub). https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2025-11-10.
[32] Microsoft Research the AutoGen community. 2025. SelectorGroupChat: A model-based next-speaker selection pattern in the AutoGen AgentChat API. https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/selector-group-chat.html. Accessed: 2025-11-10.
[33] Vikranth Udandarao, Noel Abraham Tiju, Muthuraj Vairamuthu, Harsh Mistry, and Dhruv Kumar. 2025. Roamify: Designing and Evaluating an LLM Based Google Chrome Extension for Personalised Itinerary Planning. *arXiv preprint* arXiv:2504.10489 (2025). https://arxiv.org/abs/2504.10489
[34] Yaoxiang Wang, Zhiyong Wu, Junfeng Yao, and Jinsong Su. 2024. TDAG: A Multi-Agent Framework based on Dynamic Task Decomposition and Agent Generation. *arXiv preprint* arXiv:2402.10178 (2024). https://arxiv.org/abs/2402.10178
[35] Zhonghua Wang. 2023. Intelligent recommendation model of tourist places based on user preferences and sparse data. *Applied Artificial Intelligence* 37, 1 (2023), e2203574. doi:10.1080/08839514.2023.2203574