



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14

Issue: X

Month of publication: October 2026

DOI:

www.ijraset.com

Call:  08813907089

E-mail ID: ijraset@gmail.com

ATLAS - Where Travel Meets AI: An AI-Powered Travel Planning and Decision-Support Platform with Live Data Grounding

Ashima Sahu¹, Pratik Rathore², Vaishnavi Kosarkar³, Aditya Doijad⁴, Rahul Kawariya⁵

¹⁻⁴Students, Department of B.Tech (Artificial Intelligence)

⁵Project Guide, Department of B.Tech (Artificial Intelligence)

G. H. Rasoni University, Amravati, Maharashtra, India

Abstract: *Travel planning is a fragmented, multi-domain decision process: a traveller must separately research destinations, transport, accommodation, food, activities and weather before assembling a feasible schedule. This paper presents ATLAS (“Where Travel Meets AI”), an implemented AI-powered travel planning and decision-support platform that pairs a large language model (LLM) planner with deterministic grounding against live geospatial and environmental data. Given a destination, an inclusive date range, a boarding location, a budget and traveller preferences, ATLAS generates a day-by-day itinerary with the open-weight gpt-oss-120b model served through Groq, while route, weather, food and accommodation agents run concurrently and record the provenance of each data source. Model output is validated rather than trusted: a schema enforces exactly one itinerary day per requested calendar date, generated place names are resolved through OpenStreetMap’s Nominatim geocoder, and road distances and travel times are computed with OSRM from the traveller’s real boarding location, with unknown legs reported as unavailable instead of zero. The system is implemented as a React and TypeScript single-page application backed by FastAPI and PostgreSQL, and includes a live weather outlook, real restaurant and attraction listings, a simulated booking flow that issues a PDF e-ticket by e-mail, and a community Lost & Found module. We describe the architecture and grounding method, report a verification suite of 477 automated tests and preliminary functional observations, and state the system’s current limitations explicitly.*

Keywords: *Multi-Agent Systems, Large Language Models, Travel Planning, Itinerary Generation, Geospatial Grounding, OpenStreetMap, Recommendation Systems, Decision Support.*

I. INTRODUCTION

Planning a trip conventionally requires a traveller to research destinations, compare transport options, evaluate hotels, identify restaurants and activities, check the weather, and then assemble all of it into a workable day-by-day schedule. Existing digital platforms usually specialise in one service — flight search, hotel booking or destination discovery — without an integrated reasoning layer that understands the traveller’s complete requirements and coordinates them into a single personalised plan.

ATLAS (“Where Travel Meets AI”) addresses this gap with an AI-powered travel planning and decision-support platform. A large language model drafts a constraint-aware itinerary from the traveller’s requirements, while specialised components handle routing, weather, food, accommodation and destination discovery. The platform is exposed through a structured trip planner, a conversational assistant, an itinerary and route viewer, a simulated booking flow and a personal dashboard.

LLMs write fluent plans, but fluency is not accuracy: a model can name restaurants that do not exist, compress a week into a few days, or describe travel times it never computed. During development, ATLAS exhibited each of these failures. Its central design principle is therefore that the model proposes and deterministic services verify — and that nothing estimated is ever presented as live. The contributions of this work are:

- 1) An implemented end-to-end travel planning platform (20 client routes, 36 REST endpoints) that integrates an LLM planner with seven external data services.
- 2) A grounding layer that enforces inclusive date coverage, resolves generated place names against OpenStreetMap, and computes road legs starting from the traveller’s actual boarding location.
- 3) A provenance model in which every agent and integration reports whether its output is live or estimated, carried through the API, the interface and the issued e-ticket.

4) An explicit account of what the system does and does not yet achieve, including measured failure modes.

Section II reviews related work. Section III states the problem. Section IV presents the system architecture and Section V the planning and grounding methodology. Section VI describes the implementation and Section VII reports evaluation results. Sections VIII and IX discuss limitations and future scope, and Section X concludes.

II. RELATED WORK

The reasoning capability underlying ATLAS builds on the transformer architecture [1], which underpins modern LLMs shown to generalise from few examples [2]. Chain-of-thought prompting [3] showed that eliciting intermediate reasoning improves multi-step problem solving, and the ReAct paradigm [4] interleaved reasoning with external actions so that a model can query tools and ground its output in retrieved information rather than relying on parametric knowledge alone.

Beyond single-model reasoning, recent frameworks coordinate multiple LLM-driven agents toward a shared goal. AutoGen [5] provides a conversation-based framework in which customisable agents exchange messages and invoke tools, while Generative Agents [6] demonstrated that LLM-backed agents equipped with memory, reflection and planning can produce believable coordinated behaviour. ATLAS draws on this line of work in decomposing travel planning into specialised components applied to a single domain. Retrieval-augmented generation (RAG) [7] grounds LLM output in external, up-to-date sources. ATLAS applies a related but simpler idea: rather than retrieving documents into the prompt, it validates the model's structured output against authoritative services after generation. The planner uses the open-weight gpt-oss-120b model [8], served through Groq's OpenAI-compatible inference API [17]. Geographic grounding relies on OpenStreetMap [12], a volunteered geographic information project, queried through the Nominatim geocoder [14], and on OSRM [13], which computes shortest road paths over OpenStreetMap data.

On the recommendation side, the Recommender Systems Handbook [9] provides the general foundation for preference-based ranking. Context-aware tourism recommenders have been surveyed extensively [10], and reviews of intelligent tourism recommender systems [11] identify the same fragmentation that motivates ATLAS: most deployed systems address a single sub-domain, such as accommodation or points of interest, rather than reasoning across the whole trip. ATLAS contributes not a new model or ranking algorithm, but the end-to-end integration of an LLM planner with deterministic grounding and explicit data provenance.

III. PROBLEM STATEMENT

A traveller planning a trip typically performs a sequence of independent tasks: choosing a destination, comparing transport, searching for accommodation within budget, finding restaurants that match dietary needs, discovering activities aligned with personal interests, checking the weather, and arranging the result into a time-feasible schedule. For example, a traveller departing from Nagpur Railway Station with a budget of ₹40,000, two travellers, a three-day window and an interest in culture and local food must currently satisfy each of these constraints manually across several platforms.

ATLAS frames travel planning as a constrained, multi-domain personalisation problem: given a structured description of the traveller's requirements, generate a day-by-day itinerary that respects the requested dates, origin, budget, traveller count and preferences while remaining internally consistent across transport, accommodation, food and activities.

A second, less visible problem arises once an LLM is used: fluent but unverifiable output. The problem therefore has two parts — (i) generating a coherent multi-domain plan under user constraints, and (ii) ensuring that every factual element shown to the traveller (dates, places, distances, weather) is either verified against an external source or explicitly labelled as an estimate.

IV. PROPOSED SYSTEM

A. System Overview

ATLAS follows the layered architecture shown in Fig. 1. The presentation layer is a React 18 and TypeScript single-page application. The application layer is a FastAPI service [16] exposing REST endpoints for authentication, trips, planning, routing, weather, places, bookings and Lost & Found. The planning core combines the LLM planner, a set of concurrent agents and a deterministic validation and grounding stage. Persistent state lives in PostgreSQL, and every third-party service is called server-side, so no external credential is ever sent to the browser.

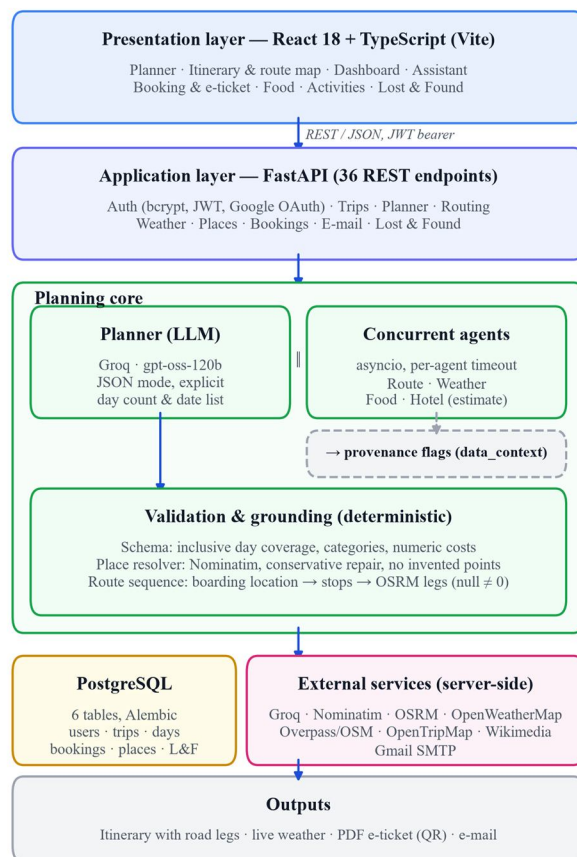


Figure 1: System architecture of ATLAS. Agents run concurrently with the planner and record data provenance; deterministic validation and grounding are applied to the planner’s output before it is stored.

B. Agent Layer

Four agents execute concurrently with the planner using Python’s *asyncio*. The Route agent geocodes the destination through Nominatim. The Weather agent retrieves a destination-specific forecast from OpenWeatherMap [15]. The Food agent queries the Overpass API for restaurants, cafés and similar amenities recorded in OpenStreetMap within 5 km of the destination. The Hotel agent produces a budget-proportional nightly accommodation estimate and labels it as an estimate, since no live inventory source is connected.

Each agent runs under its own timeout (2.5 s by default; 8 s for weather, which makes two sequential calls; 14 s for food, because Overpass is slower) and returns an *is_realtime_data* flag together with a *fallback_reason* when it degrades. These flags are stored with the trip as its provenance record and drive the interface’s “live” and “estimated” labels. In the present implementation the agents enrich and annotate the plan; their outputs are not yet fed back into the planner’s prompt (see Section VIII).

Destination discovery is handled by a separate service. Groq proposes candidate Indian destinations; each candidate is validated with Nominatim to confirm that it exists within India; and a photograph with author and licence attribution is retrieved from Wikimedia Commons. Results are cached for 6–10 hours. If the model is unavailable, a curated fallback set is returned and explicitly marked as such in the interface.

C. Planner

The planner prompt states the destination, the boarding location, the budget, the traveller count, the preferences and — explicitly — the number of itinerary days together with the full list of required dates. It instructs the model to name only real places that can be found on OpenStreetMap, to avoid vague qualifiers such as “near hotel”, and never to output coordinates, distances or travel times, which ATLAS derives itself. The model replies in JSON mode, and the reply is validated against a strict Pydantic schema. If validation fails, one retry is made that tells the model precisely what was wrong; if that also fails, a clearly labelled placeholder itinerary is stored with *is_realtime_data* = *false* rather than an incomplete one.

D. Conversational Assistant

The assistant accepts natural-language requests such as “plan a three-day trip from Nagpur for two people under ₹40,000, preferring culture and local food”. It is served by a single LLM call with a travel-planning system instruction. When a reply completes a plan, the model appends a structured trailer that the interface uses to offer a direct hand-off into the booking flow. The endpoint is rate-limited to protect the provider quota.

V. METHODOLOGY

A. Constraint Formulation

The complete generation and grounding pipeline is summarised in Fig. 2. A trip request comprises a destination, a start date d_s , an end date d_e , a boarding location b , a budget B , a traveller count T and a preference set P . Start and end dates are inclusive, so the required number of itinerary days is

$$N = (d_e - d_s) + 1 \quad (1)$$

A generated plan is accepted only if it contains exactly N days, numbered 1 to N in order, with day i dated $d_s + (i - 1)$. Every activity must carry one category from {accommodation, travel, food, activity} and a non-negative numeric cost. These constraints are enforced deterministically. Budget, interests, dietary requirements and accessibility are currently passed to the model as prompt-level constraints and are not independently verified.

B. Place Grounding

After validation, the distinct location names in the plan form a set L . Each name is qualified with the destination city — except a name that refers to the boarding location, which lies elsewhere — and resolved through Nominatim at no more than one request per second, in line with its usage policy, with results cached for 24 hours. A conservative repair step may only shorten a name (removing a parenthetical aside, a vague trailing qualifier, or a trailing descriptor such as “jetty”, which is then flagged as approximate); it never substitutes a different place or invents a coordinate. The resolution rate

$$r = |L_{resolved}| / |L| \quad (2)$$

is stored with the trip for diagnosis. Names that cannot be resolved are left exactly as written.

C. Route Computation

The canonical stop sequence is $S = [b, s_1, \dots, s_n]$, in which the boarding location is always the first point and consecutive repeats are collapsed. The boarding coordinates confirmed at planning time are reused, so a later change in the geocoder cannot move the trip’s origin. For each consecutive pair, OSRM’s route service returns a road distance and a driving duration:

$$\ell_i = (dist_i, dur_i) \text{ if } s_{i-1}, s_i \text{ are resolved; otherwise } \square \quad (3)$$

An unresolved endpoint or a routing failure yields an explicit null rather than zero, because zero is a genuine answer meaning two stops coincide. The itinerary timeline and the map both render from this single sequence, so the two can never disagree.

D. Weather

Weather is requested by coordinates obtained from Nominatim rather than by city name. The 3-hourly, 5-day forecast is grouped into days using the destination’s own UTC offset. When a trip falls beyond the forecast window, the interface shows the current outlook and states that a forecast for the travel dates will appear closer to departure; a forecast is never attached to dates the provider does not cover.

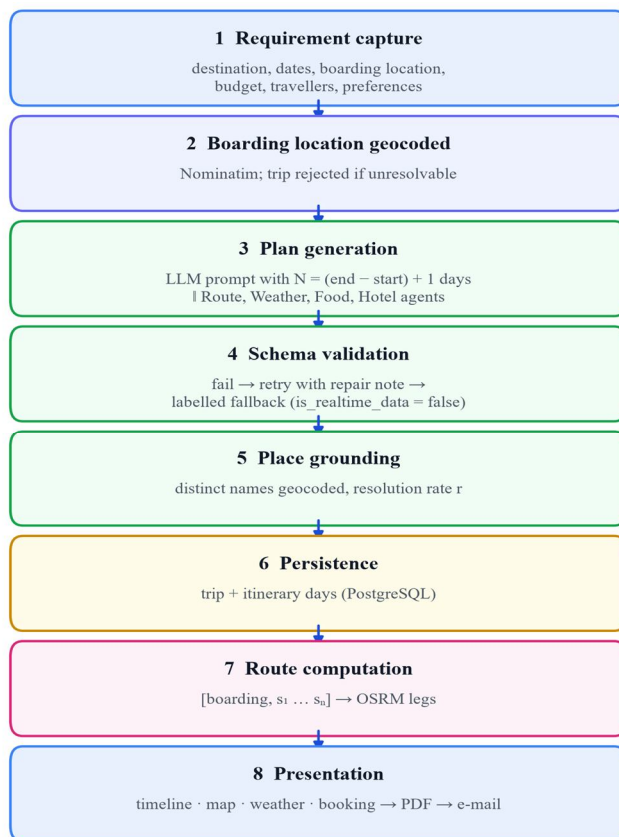


Figure 2: Itinerary generation and grounding pipeline.

VI. SYSTEM IMPLEMENTATION

A. Frontend

The client is a React 18 application written in strict TypeScript, built with Vite and styled with Tailwind CSS. It provides 20 routes across 18 pages, including the home page and destination discovery, a five-step trip planner, the itinerary viewer, the dashboard, food and activity pages, bookings, the assistant and Lost & Found. The interface is internationalised with i18next for nine languages — English, Hindi, Marathi, Bengali, Tamil, Telugu, Gujarati, Kannada and Malayalam — with Hindi the most complete translation. Fig. 3 shows AI-curated destinations with attributed photographs, and Fig. 4 the planner's boarding-location field with a geocoded suggestion.

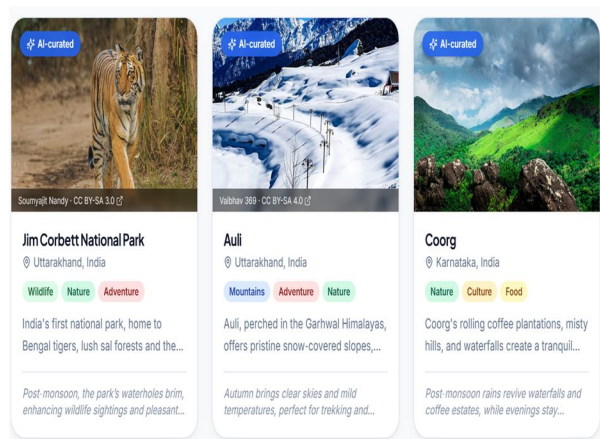


Figure 3: Discover India: AI-curated destinations validated with Nominatim and illustrated with Wikimedia Commons photographs, each credited with its author and licence.

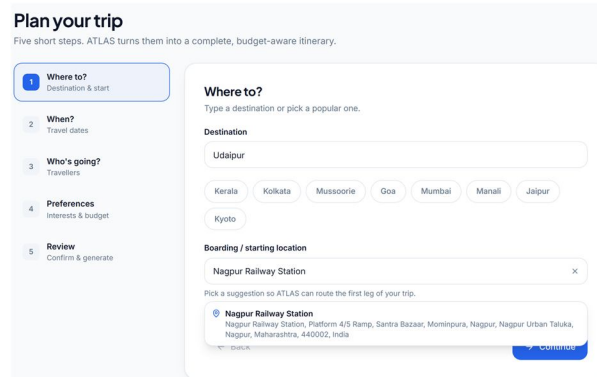


Figure 4: Trip planner, step 1. The boarding location must be a geocoded place chosen from suggestions; free text alone is not accepted.

B. Backend and Persistence

The backend is implemented in FastAPI with SQLAlchemy and Pydantic v2. PostgreSQL holds six tables (users, trips, itinerary days, saved places, bookings and Lost & Found reports) managed through six Alembic migrations. Authentication supports e-mail and password, with bcrypt hashing and HS256 JSON Web Tokens, and Google OAuth limited to the *openid*, *email* and *profile* scopes. Every trip, booking and report query is filtered by the authenticated owner, AI endpoints are rate-limited, and secrets are supplied only through environment variables. Table 1 lists the external services used.

Table 1: External services integrated in ATLAS

Function	Provider	Use in ATLAS	API key
LLM inference	Groq (gpt-oss-120b)	Itinerary, assistant, destination discovery	Yes
Geocoding	Nominatim (OSM)	Boarding location, place resolution, autocomplete	No
Routing	OSRM	Road distance and driving time	No
Weather	OpenWeather-Map	5-day destination forecast	Yes
Restaurants	Overpass API (OSM)	Food places near the destination	No
Attractions	OpenTripMap	Activities near the destination	Yes
Images	Wikimedia Commons	Attributed destination photographs	No
E-mail	Gmail SMTP	E-ticket delivery	App password

C. Itinerary and Route View

Fig. 5 shows a generated three-day itinerary for a trip from Nagpur to Udaipur. Each activity shows its time, its location and the road distance and driving time from the previous stop. The route card lists the canonical sequence beginning at the boarding location, marked START. Where a leg could not be grounded, the interface reads “Travel time unavailable” instead of showing an invented value. The first two legs in Fig. 5 show this: the model named the boarding station “Nagpur Junction”, an alias the sequence builder did not recognise as the boarding location, so those legs remained ungrounded.

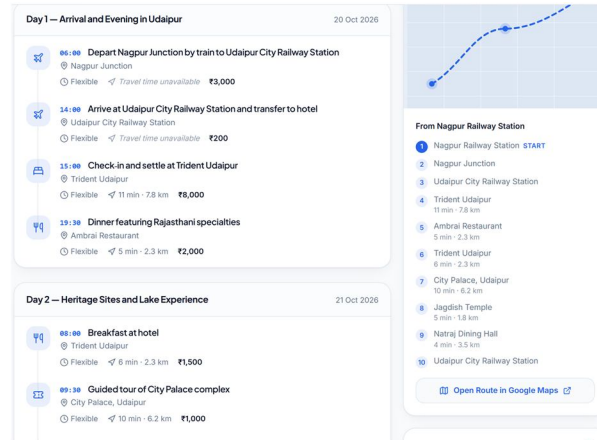


Figure 5: Generated itinerary (Nagpur → Udaipur, 20–22 October 2026). Per-stop road distances and driving times come from OSRM, and the route card begins at the boarding location.

D. Weather Outlook

Fig. 6 shows the weather card for the same trip. Because the trip begins beyond the provider's five-day window, the card presents the current outlook for Udaipur and says so explicitly, rather than presenting it as a forecast for the travel dates.

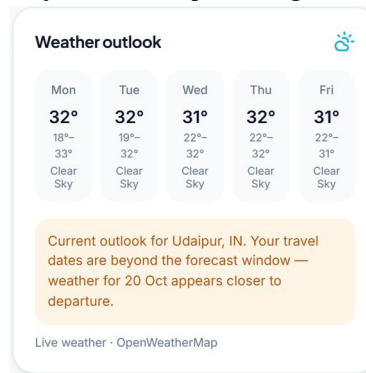


Figure 6: Live weather outlook from OpenWeatherMap, with an explicit notice that the travel dates lie beyond the forecast window.

E. Food and Activities

The food and activity pages list real places near the selected destination, obtained from OpenStreetMap through Overpass and from OpenTripMap respectively (Fig. 7). Cuisine filters are built from the returned data, and addresses, opening hours, phone numbers and websites are shown only where the source records them. No rating is displayed unless the provider publishes one; OpenStreetMap does not, and OpenTripMap's popularity tier is deliberately not presented as a star rating.

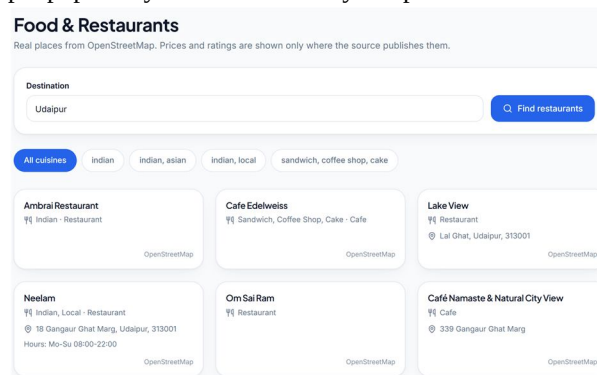


Figure 7: Food & Restaurants for Udaipur, listing OpenStreetMap places with only the attributes the source actually records.

F. Dashboard

The dashboard (Fig. 8) is derived entirely from persisted records: counts of upcoming and completed trips, bookings and saved places; the next trip, illustrated with a Wikimedia photograph of its destination; a budget overview; recent activity; and travel insights. Trip status is computed from trip dates when trips are read, so a trip whose end date has passed appears as completed.

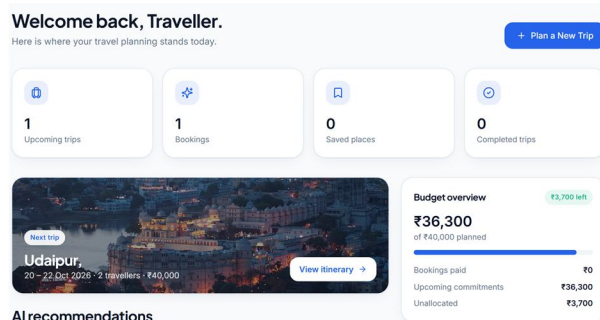


Figure 8: Dashboard summarising the traveller's persisted trips, bookings and budget.

G. Simulated Booking, E-Ticket and E-mail

A traveller can book a generated trip through a three-step simulated flow. The booking is stored with an *is_mock* flag, and an A4 e-ticket is generated server-side with ReportLab (Fig. 9). It contains the booking reference, a QR code, traveller details and a fare summary aggregated from the itinerary's categories, under a prominent "DEMO / MOCK BOOKING" notice. The same PDF bytes are offered as a download and can be e-mailed to the account holder through SMTP. E-mail delivery is independent of the booking, so a mail failure never invalidates a confirmed booking.

H. Lost & Found

A community Lost & Found board lets signed-in travellers report lost or found items with category, location, date, description and an optional photograph. Reports are stored in PostgreSQL. Photographs are validated by file signature rather than declared type, limited to 3 MB, saved under server-generated names, and deletable only by their owner.

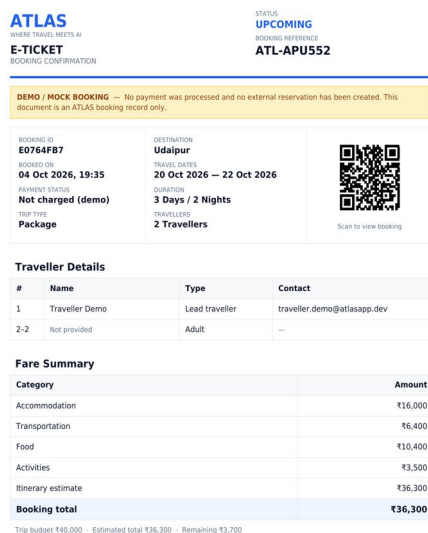


Figure 9: Generated e-ticket for the trip in Fig. 5, clearly marked as a demonstration booking.

VII. RESULTS AND EVALUATION

A. Verification Suite

Table 2 summarises the automated verification of the current build. The backend suite covers authentication and ownership isolation, schema validation, inclusive date coverage (single-day, cross-month and cross-year ranges), place resolution and repair, route-leg alignment and null handling, weather bucketing and failure modes, e-ticket generation and e-mail failure isolation. All external services are mocked in these tests, so the suite exercises ATLAS's own logic rather than provider availability.

Table 2: Automated verification results

Check	Tool	Result
Backend unit and integration tests	pytest	388 passed
Frontend unit tests	Vitest	89 passed
Static type checking	TypeScript (strict)	0 errors
Linting	ESLint	0 errors
Production build	Vite	Successful
Database migrations	Alembic	Linear chain, 6 revisions

B. Functional Observations

Table 3 records observations made while developing and testing the grounding layer. They come from a small number of generated trips and are illustrative, not a controlled benchmark.

Two results stand out. First, looking up the weather by city name let the provider silently choose a different “Manali” 2,000 km away, labelled with the expected name; pinning the request to geocoded coordinates corrected this. Second, instructing the model to use only real, findable places — combined with conservative name repair — raised the proportion of stops with genuine road distances markedly, while unresolvable names remained visibly unavailable rather than being filled in.

Table 3: Effect of the grounding layer (illustrative)

Observation	Before	After
Weather location for “Manali”	Chennai locality (13.17° N, 80.27° E), 35 °C	Manali, Himachal Pradesh (32.25° N, 77.19° E), 16.7 °C
Place resolution rate, Udaipur trips	67% (8 of 12)	75–100% across four trips
Route legs with road data	3 of 11	14 of 14 (best observed)
Days for a 1–8 October request	4, silently truncated	8, enforced by schema
First leg, Nagpur → Udaipur	not computed	863 km, 10 h 50 min (OSRM)

C. Itinerary Length

Three-day itineraries were generated successfully with real AI content. Two attempts to generate an eight-day itinerary (1–8 October) both failed validation: the model first returned only two days, and the repair retry was rejected by the provider. In both cases ATLAS stored an eight-day placeholder clearly marked as not AI-generated, rather than a truncated plan. The probable cause is the 4,000-token output limit combined with the reasoning model’s internal token use, but this has not yet been confirmed.

VIII. LIMITATIONS

- 1) The agents run concurrently with the planner, and their outputs annotate the plan rather than shaping it: weather, restaurant and route data are not yet supplied to the planner’s prompt.
- 2) Itineraries longer than about three days currently fail to generate within the model’s output budget and fall back to a labelled placeholder.
- 3) Budget, dietary, interest and accessibility constraints are enforced only at the prompt level; the planned total is not yet checked against the budget.

- 4) No review or rating data source is integrated, accommodation is an estimate, and live transport and hotel inventory are not connected.
- 5) Grounding depends on OpenStreetMap coverage. Genuine businesses absent from OSM, and aliases of the boarding point such as “Nagpur Junction”, remain unresolved.
- 6) The public OSRM demonstration server and the Nominatim usage policy are not intended for production traffic.
- 7) Interface translations are partial for most languages, AI responses are produced in English, and voice interaction is not implemented.
- 8) The booking module is a simulation and must not be represented as a payment or reservation system.
- 9) Evaluation is limited to automated tests and small-sample functional observations; no user study has yet been conducted.

IX. FUTURE SCOPE

The most direct improvement is to make the agents steer the planner: supplying the verified restaurant list, the weather outlook and geocoded landmarks to the prompt would let the model choose from real options and schedule outdoor activities around the weather. Longer trips can be supported by generating itineraries in daily chunks within the output budget. A deterministic budget checker, alias matching for the boarding location, review-data integration, persistent object storage for uploads and self-hosted routing and geocoding are planned engineering steps.

Beyond these, future directions include orchestration through a framework such as LangGraph with explicit inter-agent messages, dynamic replanning when conditions change (for example, substituting an indoor activity when rain is forecast), group travel planning, completion of the regional-language interface, voice interaction, and a structured user study comparing ATLAS with manual planning and with a single-prompt LLM baseline.

X. CONCLUSION

This paper presented ATLAS, an implemented AI-powered travel planning and decision-support platform that combines an LLM planner with deterministic grounding against live geographic and environmental data. Its core contribution is a practical discipline for using a generative model in a factual domain: the model drafts the plan, while schemas, geocoding and road routing verify dates, places and travel times, and every value carries a record of whether it is live or estimated. The evaluation shows that this discipline catches errors that would otherwise reach the traveller unnoticed, and the limitations section states plainly what remains, chief among them agent-to-planner coordination and generation of longer itineraries.

XI. ACKNOWLEDGMENT

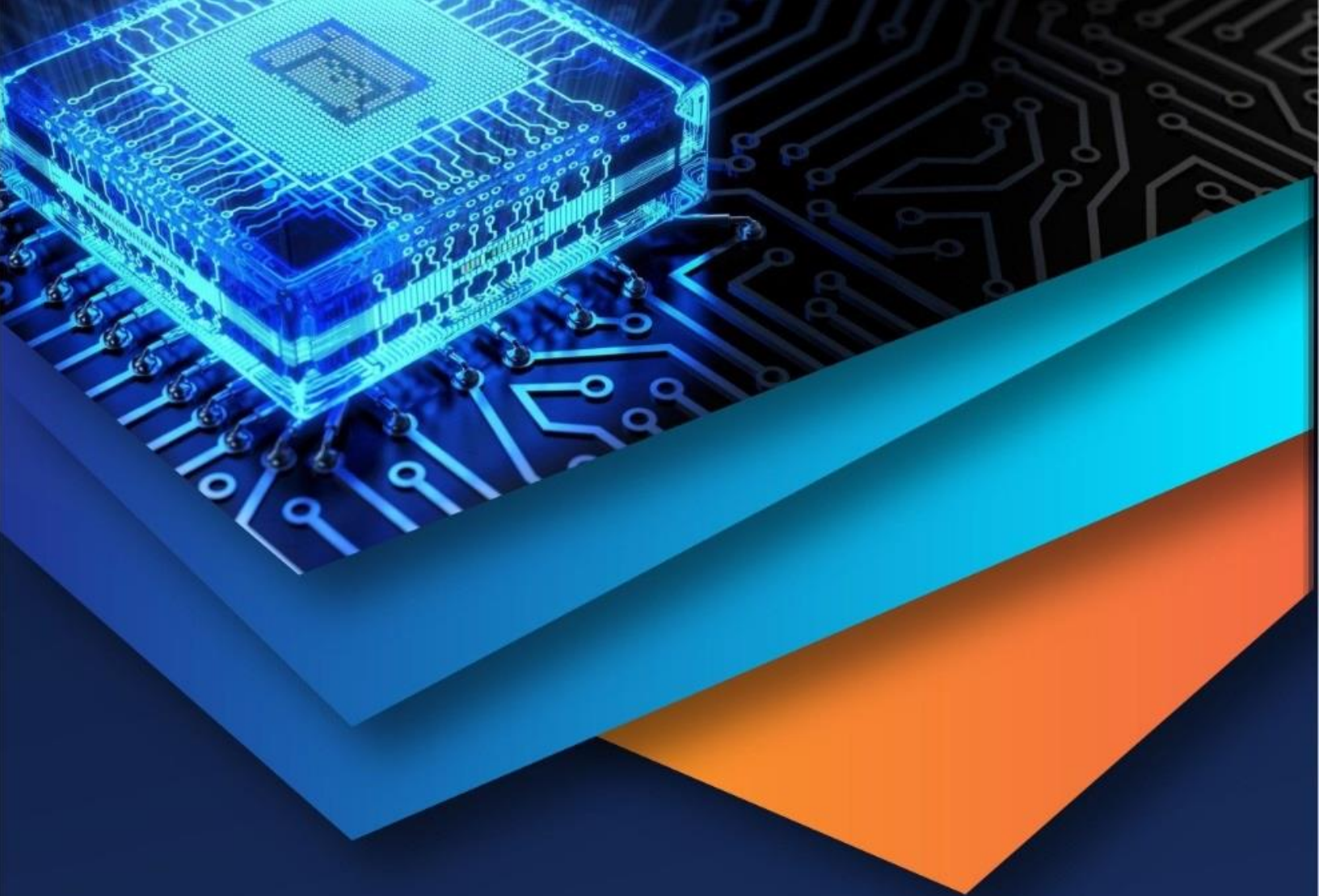
The authors thank their project guide, Mr. Rahul Kawariya, for his continued guidance and feedback throughout the design and development of ATLAS, and the Department of Artificial Intelligence, G. H. Rasoni University, Amravati, for the resources and support provided during this project.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2017.
- [2] T. Brown et al., “Language models are few-shot learners,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020.
- [3] J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022.
- [4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. R. Narasimhan, and Y. Cao, “ReAct: Synergizing reasoning and acting in language models,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.
- [5] Q. Wu et al., “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” Microsoft Research, Tech. Rep. MSR-TR-2023-33, 2023.
- [6] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proc. ACM Symp. User Interface Softw. Technol. (UIST)*, 2023.
- [7] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” *arXiv:2312.10997*, 2023.
- [8] OpenAI, “Introducing gpt-oss,” Aug. 2025. [Online]. Available: <https://openai.com/index/introducing-gpt-oss/>
- [9] F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*, 2nd ed. New York, NY, USA: Springer, 2015.
- [10] A. Bentalab, Y. El Bouzekri, A. Ait Lahcen, and M. Boulmal, “Context aware recommender systems for tourism: A concise review,” in *Proc. IEEE 5th Int. Congr. Inf. Sci. Technol. (CiST)*, 2018.
- [11] J. Borràs, A. Moreno, and A. Valls, “Intelligent tourism recommender systems: A survey,” *Expert Syst. Appl.*, vol. 41, no. 16, pp. 7370–7389, 2014.
- [12] M. Haklay and P. Weber, “OpenStreetMap: User-generated street maps,” *IEEE Pervasive Comput.*, vol. 7, no. 4, pp. 12–18, 2008.



- [13] D. Luxen and C. Vetter, "Real-time routing with OpenStreetMap data," in Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst., 2011, pp. 513–516.
- [14] OpenStreetMap Foundation, "Nominatim: Open-source geocoding with OpenStreetMap data." [Online]. Available: <https://nominatim.org/>
- [15] OpenWeather Ltd., "5 day / 3 hour forecast API." [Online]. Available: <https://openweathermap.org/forecast5>
- [16] S. Ramírez, "FastAPI." [Online]. Available: <https://fastapi.tiangolo.com/>
- [17] Groq, Inc., "GroqCloud documentation." [Online]. Available: <https://console.groq.com/docs>



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)