



International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

SafarSaathi: A Real-Time Multilingual Public Transport Companion Delivered over WhatsApp

Sangam Mishra^{a,*}, Avadhut Kaskar^a, Shubham Partole^a, Prof. Satish Kuchiwale^a

^a Department of Computer Engineering, Smt. Indira Gandhi College of Engineering, Ghansoli, Navi Mumbai 400701, Maharashtra, India

ABSTRACT

Commuters in Indian cities make millions of bus, metro and suburban rail trips every day, yet live transit information mostly sits inside map apps that expect a smartphone, mobile data, English literacy and some comfort with maps. This paper presents SafarSaathi, a multilingual public-transport companion that runs entirely inside WhatsApp. A commuter sends a text message, a voice note or a location pin in Hindi, Marathi or English. A large language model (LLM) layer identifies the language, the intent, and the origin and destination. OpenTripPlanner then plans a multimodal route over open GTFS schedules merged with GTFS-Realtime updates, and the answer (the next arrival, a trip plan or a service alert) goes back in the commuter's own language. The LLM never computes routes or times itself. We first review fifteen core studies on transit data, routing, arrival prediction, LLMs and Indic-language interfaces, and derive six design requirements from them. We then describe the system in full: objectives and scope, feasibility, an Agile development plan, data and functional analysis with data flow diagrams, hardware and software requirements, the layered architecture, the processing algorithm, the chat interface and the database schema. SafarSaathi needs no app install, works on low bandwidth, and can be extended to any city that publishes a GTFS feed. The paper closes with the evaluation protocol for the prototype and plans for future work.

Keywords: Public transport, Real-time passenger information, GTFS, OpenTripPlanner, Large language models, WhatsApp chatbot, Multilingual NLP, Digital inclusion

1. Introduction

Public transport is the main way people move around Indian metropolitan regions. In Mumbai, suburban rail, BEST buses and the growing metro network together carry several million trips a day. What these commuters most need to know is simple: when the next bus or train will arrive, and the fastest way to reach their destination. Studies of real-time information systems show that having this information reduces both the actual and the perceived waiting time of riders (Ferris et al., 2010; Watkins et al., 2011).

The access gap. Today this information comes mainly from map applications and operator-specific apps. Using them takes a smartphone with data, a separate install, English literacy and some skill with maps. The information is also split up. Bus, metro and rail each have their own apps and boards, so a door-to-door journey cannot be planned in one place. Without live arrival times, people wait at stops with no idea how long the bus will take, and they miss connections they could have made. Elderly, migrant, low-income and low-literacy commuters stand to gain the most from good information, and they are often the ones left out.

The opportunity. Three developments make another approach possible. Transit agencies increasingly publish schedules in the open General Transit Feed Specification (GTFS) and live updates in GTFS-Realtime (MobilityData, n.d.). Open-source engines such as OpenTripPlanner (OTP) can plan multimodal journeys over these feeds (Narboneta & Teknomo, 2016; OpenTripPlanner Project, n.d.). LLMs can also understand free-form requests, including requests in Indian languages (He & Garner, 2023; Joshi, 2022; Gala et al., 2024). WhatsApp is already installed on most phones in India and could carry these capabilities to commuters over a familiar, low-bandwidth channel (Eltigani et al., 2025).

Our approach. SafarSaathi, whose name means "travel companion", puts these pieces together. A commuter opens an ordinary WhatsApp chat with the SafarSaathi number and asks, in whichever language they are comfortable with, when the next train leaves Andheri or how to get to Churchgate. They can type, send a voice note or share their location. The system works out what they want, plans the trip on open transit data, and replies in the same language, with a target of about five seconds end to end. Nothing has to be installed, and the replies are short text messages that load on a 2G or 3G connection.

Contributions. This paper makes four contributions. We (1) review fifteen core studies across five themes using a protocol that others can reproduce, (2) turn the review's findings into six design requirements, (3) present the full design of SafarSaathi, from requirements and data flow analysis through architecture, algorithm, interface and database, and (4) define the evaluation protocol and metrics for the prototype.

Section 2 looks at the tools commuters use today and states the problem. Section 3 reviews the literature. Section 4 introduces the proposed system, and Section 5 covers planning and feasibility. Section 6 analyses the system's data, functions and quality requirements, Section 7 lists the hardware and software it needs, and Section 8 presents the design. Section 9 describes the evaluation and discusses the expected results, and Section 10 concludes.

2. Existing systems and problem statement

2.1 How commuters get transit information today

We looked at the options a Mumbai commuter actually has, and Table 1 summarises them. Google Maps and Moovit plan multimodal routes and show live data wherever the transit agency shares it. They work well for a confident smartphone user, but they need mobile data, their interfaces are English-first, and they assume the user can read a map. m-Indicator is widely used in Mumbai for local-train timetables and general city transit information. It is still another app to install, and most of what it shows comes from timetables, with little live data.

Chalo and similar city bus apps offer live bus tracking, but each one covers a single operator in a single city. A journey that combines a bus with the metro or a local train cannot be planned in them. Where is My Train and the railways' NTES service give the running status of Indian Railways trains and nothing else: there is no bus or metro data and no door-to-door planning. Station boards, enquiry counters and helplines remain the fallback. They help people who are already at the station, but they get crowded at peak hours and cannot plan a personal trip.

Table 1 - Existing transit information options.

Existing option	What it offers	Limitation for the everyday commuter
Google Maps / Moovit	Multimodal route planning; live data only where agencies share it	Needs smartphone, mobile data and map reading; English-first interface
m-Indicator (Mumbai)	Local-train timetables and city transit info	Separate app to install; mostly timetable-based, little live data
Chalo / city bus apps	Live bus tracking for a single operator	Bus only, one city at a time; no joint bus, metro and rail trip
Where is My Train / NTES	Running status of Indian Railways trains	Rail only; no metro or bus, no door-to-door planning
Station boards, enquiry counters, helplines	Next departures at a station; human help	Available only on site, crowded, no personal trip planning

2.2 Limitations of the existing system

Four problems run through these options. Map apps exclude a large share of users because of what they demand: a smartphone, data, English and map reading. The information is fragmented across modes, so bus, metro and local trains live on separate apps and boards, and no one offers a single place to plan a whole journey. Without live arrival information, commuters wait at stops without knowing how long they will be there, and they miss buses and trains they could have caught. Most seriously, the people with the fewest alternatives (elderly, migrant, low-income and low-literacy commuters) are left with no accessible option at all.

2.3 Why WhatsApp and not another app

Building yet another app would repeat the first problem. A new app has to be found, downloaded and learned, and it competes for space on low-end phones with little storage. A dedicated map interface also brings back the menus and maps that put many users off in the first place, and in practice most everyday commuters never adopt such tools.

WhatsApp is already on almost every phone in India, and people of all ages use it every day. It works on slow connections, needs nothing new to be installed, and accepts both typed and spoken messages in any language. Asking SafarSaathi a question feels the same as messaging a friend. Eltigani et al. (2025) report the same advantage from deploying an LLM chatbot over WhatsApp for users with limited resources. A chat window does have limits. It cannot show a detailed map, and long replies are hard to read on a small screen, so SafarSaathi keeps its answers short and puts the most useful fact (usually a time) first.

2.3 Problem statement

There is no single low-bandwidth, local-language channel that gives every commuter real-time, multimodal transit answers. This work sets out to build one on top of open transit data and an app that people already use.

3. Literature review

The review followed a structured protocol based on the PRISMA 2020 guidelines (Page et al., 2021) and Kitchenham's procedures for systematic reviews (Kitchenham, 2004), scaled down to suit an undergraduate project. Section 3.1 describes the method, Sections 3.2 to 3.6 summarise the five themes, and Section 3.7 draws the findings together.

3.1 Review method

Four research questions guided the review:

RQ1: Which data standards allow transit information to be reused across operators and cities?

RQ2: Which algorithms and open-source tools support real-time multimodal journey planning?

RQ3: How accurately can LLMs interpret free-form transit queries, including speech and Indian languages?

RQ4: Which delivery channels make transit information accessible to low-literacy and low-bandwidth users?

We searched IEEE Xplore, ACM Digital Library, SpringerLink, arXiv and Google Scholar. Search strings combined three groups of concepts with Boolean operators, for example: ("GTFS" OR "transit feed" OR "public transport") AND ("journey planning" OR "arrival prediction" OR "real-time information") AND ("large language model" OR "chatbot" OR "intent detection" OR "Hindi" OR "Marathi"). For the strongest papers we also followed citations backward and forward (snowballing).

Studies were included if they (i) were published between 2010 and 2025, (ii) were written in English, (iii) were peer-reviewed or were preprints with a public technical description, and (iv) addressed at least one research question. Studies were excluded if they dealt only with freight or private-vehicle routing, gave no technical detail, or duplicated an included study. Table 2 summarises the selection process.

Table 2 - Study selection process.

Stage	Action	Records
Identification	Records retrieved from five databases	[N1]
Screening	After duplicate removal; title and abstract screened	[N2]
Eligibility	Full texts assessed against criteria	[N3]
Included	Core studies retained for synthesis	15

For each included study we recorded the problem, the data, the technique, the reported results, the limitations and the relevance to SafarSaathi. Each study was also rated on clarity of method, reproducibility (open data or code), evaluation on real-world data, and relevance to Indian or multilingual settings. Preprints were kept only if they gave a complete technical description, and we treat their findings as preliminary. Bibliographic details were checked against publisher or arXiv records.

3.2 Transit data standards

GTFS describes routes, stops, trips and timetables as a set of text tables. GTFS-Realtime adds trip updates, vehicle positions and service alerts (MobilityData, n.d.). Liu et al. (2024) propose GTFS2STN, which converts static GTFS feeds into spatiotemporal transit networks for accessibility analysis. Their work shows that a standard feed is enough to rebuild a city's network without custom data engineering, which is what lets a transit assistant work in any city that publishes a feed.

3.3 Multimodal journey planning

Narboneta and Teknomo (2016) build a complete trip planner from open components (OpenTripPlanner, OpenStreetMap and GTFS) and apply it to disaster relief and recovery, showing that an open stack can handle door-to-door planning without commercial services. Giannakopoulou et al. (2018) introduce the Multimodal Dynamic Timetable Model. It answers earliest-arrival and minimum-transfer queries across rail, bus and walking, and absorbs delay updates in well under a millisecond.

Bast et al. (2016) survey route-planning techniques for road and transit networks. Dibbelt et al. (2018) propose the Connection Scan Algorithm, which scans a time-sorted array of connections instead of searching a graph. Delling et al. (2015) present RAPTOR, a round-based algorithm that needs no priority queue and underlies the transit router in OpenTripPlanner 2. In round k , RAPTOR finds the earliest arrival at every stop using at most k trips, so the number of transfers comes out of the search directly. Taken together, these papers show that fast multimodal routing that accounts for delays is a solved engineering problem at city scale.

3.4 Arrival prediction and real-time information

Barnes et al. (2020) describe BusTr, a model Google deployed to predict bus travel times from real-time road traffic for agencies that have no vehicle tracking. It outperforms the earlier DeepTTE model, and it is relevant to Indian bus networks, where live feeds are often incomplete. On the rider side, Ferris et al. (2010) found that OneBusAway users were more satisfied and changed how they travelled once real-time arrival information was available. Watkins et al. (2011) measured that riders with real-time information waited less in practice and also felt their waits were shorter. Both results suggest that commuters benefit from the kind of information SafarSaathi delivers.

3.5 Large language models for intent and data access

Modern language models build on the Transformer architecture (Vaswani et al., 2017), bidirectional pre-training (Devlin et al., 2019) and in-context few-shot learning at scale (Brown et al., 2020). He and Garner (2023) evaluate LLMs on spoken language understanding. In zero- and few-shot settings the largest models come close to fine-tuned systems on intent classification, though they stay sensitive to speech-recognition errors.

Devunuri et al. (2024) benchmark LLMs on GTFS semantics and retrieval. GPT-4 answered 73.3% of multiple-choice questions on the GTFS standard correctly, and program synthesis reached 93% accuracy on simple retrieval queries over Chicago transit data but only 61% on complex ones. Fantin et al. (2025) design an agent chatbot that answers questions about public-transport data by calling tools that query databases and plot trips, so its answers are based on authoritative data. Other work grounds generation in retrieved documents (Lewis et al., 2020) or teaches models to call external tools on their own (Schick et al., 2023). Radford et al. (2023) show that Whisper, trained on large-scale weakly supervised audio, transcribes speech well across many languages, which makes voice-note input possible.

3.6 Conversational and Indic-language interfaces

Eltigani et al. (2025) report on deploying WaLLM, an LLM-powered chatbot delivered through WhatsApp, and found that the platform lowers adoption barriers for users with limited resources. For Indian languages, Joshi (2022) releases L3Cube-MahaCorpus and MahaBERT for low-resource Marathi. Gala et al. (2024) introduce Airavata, a Hindi instruction-tuned LLM, and Gala et al. (2023) present IndicTrans2, a translation model that covers all 22 scheduled Indian languages. Kakwani et al. (2020) release IndicNLP Suite, with monolingual corpora, benchmarks and pre-trained models for 11 Indian languages. With these resources, understanding and replying in Hindi and Marathi is practical.

3.7 Synthesis and research gap

Table 3 maps the fifteen core studies to their theme, technique and the research question they inform. Every theme is covered by at least two studies, but only one study (Eltigani et al., 2025) deals with delivery through a messaging platform, and only three (Joshi, 2022; Gala et al., 2023; Gala et al., 2024) deal with Indian languages. Ten foundational works on routing, language models, tool grounding and review methodology (Vaswani et al., 2017; Brown et al., 2020; Page et al., 2021; Kitchenham, 2004; Bast et al., 2016; Dibbelt et al., 2018; Lewis et al., 2020; Schick et al., 2023; Devlin et al., 2019; Kakwani et al., 2020) are cited as background and were not part of the core synthesis.

Table 3 - Summary of included studies.

Study	Theme	Key technique / finding	RQ
Liu et al. (2024)	Data standard	GTFS to spatiotemporal network	RQ1
Narboneta & Teknomo (2016)	Planning	OTP + OSM + GTFS open stack	RQ2
Giannakopoulou et al. (2018)	Planning	Dynamic multimodal timetable model	RQ2
Delling et al. (2015)	Planning	RAPTOR round-based routing	RQ2
Barnes et al. (2020)	Prediction	Bus times from road traffic	RQ2
Ferris et al. (2010); Watkins et al. (2011)	Rider impact	Real-time info cuts waits	RQ4
He & Garner (2023)	LLM	Zero/few-shot intent detection	RQ3
Devunuri et al. (2024)	LLM + GTFS	LLM retrieval over GTFS	RQ1, RQ3
Fantin et al. (2025)	LLM agent	Tool-grounded transit chatbot	RQ3
Radford et al. (2023)	Speech	Multilingual speech recognition	RQ3
Eltigani et al. (2025)	Channel	LLM chatbot over WhatsApp	RQ4

Study	Theme	Key technique / finding	RQ
Joshi (2022); Gala et al. (2023, 2024)	Indic NLP	Marathi/Hindi models, translation	RQ3

RQ1: Data standards. GTFS and GTFS-Realtime can describe and update a city's network (Liu et al., 2024; Devunuri et al., 2024). A system built on them can be extended to a new city by ingesting that city's feed, as long as the feed exists and is maintained.

RQ2: Journey planning. Fast multimodal routing with live delay updates is mature (Giannakopoulou et al., 2018; Delling et al., 2015; Bast et al., 2016; Dibbelt et al., 2018) and available as open source (Narboneta & Teknomo, 2016). Where live vehicle data is missing, traffic-based prediction can fill the gap (Barnes et al., 2020). Otherwise the system has to use schedule-based estimates and label them as such.

RQ3: Language understanding. LLMs classify intents reliably (He & Garner, 2023), but their accuracy on complex data retrieval drops sharply (Devunuri et al., 2024). The evidence therefore supports dividing the work: the LLM interprets the request and writes the reply, and a deterministic engine computes routes and times, following the tool-grounded pattern of Fantin et al. (2025), Lewis et al. (2020) and Schick et al. (2023). Speech input is feasible (Radford et al., 2023), though transcription errors carry over into intent detection (He & Garner, 2023). Indic resources exist for Hindi and Marathi (Joshi, 2022; Gala et al., 2023, 2024; Kakwani et al., 2020), but none has been evaluated on transit vocabulary such as station names.

RQ4: Accessible delivery. Real-time information measurably improves the rider experience (Ferris et al., 2010; Watkins et al., 2011), and WhatsApp has worked well as a low-barrier channel for LLM services (Eltigani et al., 2025). None of the included studies, however, delivered transit information through a messaging platform.

Table 4 compares what existing approaches can do, based on their published descriptions. Each building block is mature on its own, yet none of the surveyed systems combines multimodal real-time planning, voice and text input in local languages, and delivery through an app the user already has. The commuters described in Section 2 are kept out by this missing integration more than by any weakness in the underlying algorithms.

Table 4 - Capability comparison.

Approach	Multimodal	Real-time	Indic languages	No install
Map apps	Yes	Partial	Limited	No
Operator apps	No	Partial	Limited	No
Transit chatbot (Fantin et al., 2025)	Yes	No	No	No
WhatsApp LLM bot (Eltigani et al., 2025)	No	No	Partial	Yes
SafarSaathi (proposed)	Yes	Yes	Yes	Yes

The findings translate into six design requirements (DR), which the rest of the paper follows:

DR1 Open data: ingest GTFS and GTFS-Realtime so any city with a feed can be onboarded (Liu et al., 2024).

DR2 Deterministic routing: compute routes and ETAs with OpenTripPlanner, never with the LLM (Narboneta & Teknomo, 2016; Devunuri et al., 2024; Delling et al., 2015; Dibbelt et al., 2018).

DR3 Grounded LLM: use the LLM only for intent, origin-destination extraction and reply composition (He & Garner, 2023; Fantin et al., 2025; Lewis et al., 2020).

DR4 Multilingual voice: support Hindi, Marathi and English text and voice notes (Joshi, 2022; Gala et al., 2023, 2024; Radford et al., 2023; Kakwani et al., 2020).

DR5 Familiar channel: deliver through WhatsApp with short, low-bandwidth replies (Eltigani et al., 2025).

DR6 Transparency: state whether an ETA is live or schedule-based, and fall back gracefully (Barnes et al., 2020; Watkins et al., 2011).

4. Proposed system

4.1 Overview

SafarSaathi is a real-time transit companion that lives in a WhatsApp chat. It covers bus, metro and suburban rail, and commuters can ask by text or voice in Hindi, Marathi or English. OpenTripPlanner joins the three modes into a single journey, although the arrival estimate comes from a different source for each mode.

For buses, the system uses live vehicle feeds wherever agencies publish them, so the estimate at each stop reflects where the bus actually is. Metro lines are fixed and trains run often, which makes schedule-based ETAs precise enough even when no live feed exists. Suburban rail runs to published timetables,

and these give dependable next-train times and support planning across the network. Following DR6, every reply tells the user whether the time it gives is live or scheduled.

4.2 Core features

The first release has six features. A commuter can ask when the next bus or train will reach a stop and get an arrival time straight away. They can plan a full journey from A to B with transfers across bus, metro and rail. They can write or speak in Hindi, Marathi or English, and the reply comes back in the same language. Instead of typing a place name, they can share a location pin and the system finds the nearest stops. Replies are plain text with no map tiles to download, so they arrive quickly on slow connections. Users can also set journey reminders and receive service or diversion alerts in the same chat.

4.3 Objectives

The project has six objectives:

- Deliver live and schedule-based arrival times across bus, metro and local rail;
- Plan a single a-to-b journey with transfers across every transport mode;
- Understand and reply in hindi, marathi and english, by text or by voice note;
- Run entirely inside whatsapp on low bandwidth, with nothing to download or learn;
- Reach the elderly, migrant, low-income and low-literacy commuters who are left out today;
- Stay city-agnostic, so that a new city can be onboarded by ingesting its open gtfs feed.

4.4 Scope

Version 1 targets Mumbai and covers its buses, metro and suburban rail. It handles next-arrival queries, multimodal trip planning, text and voice in three languages, nearest-stop lookup from a location pin, and journey reminders and service alerts.

Some features are deliberately left out of the first version: ticket booking and payments, cab and auto-rickshaw hailing, turn-by-turn navigation, crowd-level prediction, and voice calls (IVR) for feature-phone users. Each of these needs either commercial partnerships or a different interface, and they are planned as future work. The intended users are elderly commuters, migrant workers, low-income riders and people with limited literacy. Because the transit data comes from GTFS, the plan is to expand next to cities such as Delhi, Pune, Bengaluru and Kochi, and to any other city with a published feed.

4.5 Methodology

Each commuter question goes through a six-stage pipeline. First, the commuter sends a text, a voice note or a location pin on WhatsApp. The LLM then detects the language and the intent and pulls out the origin and destination. In the third stage the named places are matched to GTFS stops using fuzzy and transliterated matching. OpenTripPlanner then plans the route and estimates arrival times. The fifth stage composes the answer and translates it back into the user's language. Finally, the chat offers quick-reply follow-ups, reminders and alerts.

Few-shot intent extraction. The LLM is prompted with a handful of examples and asked to return structured JSON with the intent, the language and the origin and destination, so no custom model has to be trained. For the Marathi query shown in Section 8.3, the expected output is `{"language": "mr", "intent": "next_arrival", "origin": "Andheri", "destination": "Churchgate"}`. The intent set is small and fixed: next arrival, trip plan, nearest stop, reminder and service alert. Anything outside it gets a polite reply that lists what the bot can do. Voice notes are transcribed by a speech-to-text model before this step.

Stop resolution. Users rarely type a stop name the way the GTFS feed spells it. The same station can be written as अंधेरी, Andheri or Andheri (W), and users abbreviate and misspell names all the time. Each stop is therefore stored with its name in Latin, Hindi and Marathi script, and RapidFuzz scores the user's text against all three. If no stop scores well enough, or two stops come out close, the bot asks the user to choose instead of guessing.

Routing and live-delay merging. Routing uses RAPTOR (Delling et al., 2015), the round-based algorithm inside OpenTripPlanner 2, which searches GTFS timetables quickly across modes. Where a GTFS-Realtime trip update exists, its delay is added to the scheduled time, so the live estimate is the scheduled arrival plus the latest reported delay. Where it does not, the scheduled time is used and the reply says so.

5. Planning and formulation

5.1 Feasibility analysis

Table 5 summarises whether the project is practical to build, run and adopt. Technical feasibility is high because every component already exists: open GTFS feeds, a proven routing engine, and LLM APIs that handle Hindi and Marathi, all of which talk to each other over standard HTTP and JSON. Economic feasibility is also high. The core stack is open source, the LLM and WhatsApp messaging are paid per use, and the prototype runs on a single cloud virtual machine. Operationally, commuters already use WhatsApp daily and need no training.

Two areas carry more risk. The schedule is tight, since the build has to fit four two-week sprints inside a sixteen-week semester. The legal side also needs care. The system uses openly licensed transit data, follows the WhatsApp Business policy (Meta Platforms, n.d.), and stores only minimal, hashed user data in line with India's Digital Personal Data Protection Act, 2023 (Government of India, 2023).

Table 5 - Feasibility analysis.

Aspect	Assessment	Rating
Technical	Open GTFS feeds, OpenTripPlanner and multilingual LLM APIs all exist and integrate over HTTP/JSON	High
Economic	Open-source stack (OTP, Python, PostgreSQL); pay-per-use LLM and messaging; one cloud VM for the prototype	High
Operational	Commuters already use WhatsApp daily; no install, no training	High
Schedule	Four two-week Agile sprints inside a 16-week semester	Medium
Legal and ethical	Open-licence data; WhatsApp Business policy; minimal hashed user data under the DPDP Act, 2023	Medium

5.2 Process model

We chose an Agile model with two-week sprints. Each sprint moves through planning, design, development, testing, review and deployment, and ends with a working build that we demonstrate to our guide. Sprint 1 sets up the WhatsApp webhook, message handling and LLM intent and language detection. Sprint 2 adds GTFS ingestion, stop matching and trip planning with OpenTripPlanner. Sprint 3 brings in GTFS-Realtime ETAs, voice notes and replies in all three languages. Sprint 4 covers location pins, reminders and alerts, and the testing and evaluation work.

A waterfall plan would have fixed the requirements at the start. Ours are likely to shift as we test with real commuters: which cities and feeds are usable, which languages and phrasings people actually use. Short sprints let us adjust after each round while always keeping a version that works.

5.3 Project timeline

The sixteen-week plan in Table 6 has three phases. Analysis and design take the first five weeks, the four development sprints run from week 5 to week 12, and testing, documentation and the final presentation fill weeks 12 to 16. Some activities overlap by a week so that, for instance, design work can start while the requirements are being finalised.

Table 6 - Sixteen-week project timeline.

Activity	Weeks
Problem identification and literature survey	W1–W2
Requirement analysis and feasibility	W2–W3
System design (DFD, database, GUI)	W3–W5
Sprint 1: WhatsApp interface and LLM intent detection	W5–W6
Sprint 2: GTFS ingestion and OTP routing	W7–W8
Sprint 3: real-time ETA and multilingual replies	W9–W10
Sprint 4: location, alerts and reminders	W11–W12

Activity	Weeks
Testing and evaluation	W12–W14
Documentation and final presentation	W14–W16

6. System analysis

6.1 Data analysis

Sources. The system draws on four kinds of data. Static GTFS gives the agencies, stops, routes, trips, stop times and service calendars. GTFS-Realtime gives trip updates, vehicle positions and alerts. OpenStreetMap provides the walking network that connects stops to each other and to the user's location. The fourth source is the users' own messages: text, voice notes and location pins.

Capture. Static GTFS zip files are downloaded every night. GTFS-Realtime feeds are protocol-buffer streams, and the system polls them about every 30 seconds. The WhatsApp Cloud API (Meta Platforms, n.d.) delivers each incoming message to our webhook as JSON, and voice notes are fetched separately through the WhatsApp Media API.

Pre-processing. Each feed is checked with a GTFS validator and broken rows are dropped. Stop names are normalised and transliterated between Devanagari and Latin script, and stations that serve more than one mode (Andheri, for example, has both a suburban and a metro station) are linked as interchanges. On the message side, voice notes are transcribed, the language is detected and the text is cleaned up.

Storage. OpenTripPlanner builds its routing graph from GTFS and OpenStreetMap. A PostgreSQL database with the PostGIS extension holds a spatial index of stops for nearest-stop lookups. Redis caches the latest live delays, and a session and message log keeps enough context to answer follow-up questions.

6.2 Functional analysis

Table 7 gives an input-process-output view of the system. The inputs are the user's text, voice note or location pin, together with the static and real-time GTFS feeds. Processing runs speech-to-text and language detection, extracts intent and origin and destination with the LLM, resolves stops by fuzzy matching, plans the route and ETA in OpenTripPlanner, and finally translates and composes the reply. The outputs are arrival times, trip plans, nearest stops, reminders and alerts, all in the user's language.

Table 7 - Input-process-output view and tools.

Input	Process	Output	Tool
Text message	Language detection, intent and O/D extraction	Next-arrival time / ETA	LLM API
Voice note	Speech-to-text	Reply in user's language	Whisper / STT API
Location pin	Nearest-stop lookup	Nearest stops	PostGIS
Place names	Stop resolution (fuzzy match)	GTFS stop IDs	RapidFuzz
GTFS + GTFS-RT feeds	Routing and ETA	Multimodal trip plan	OpenTripPlanner 2
Reminder request	Scheduling and push	Reminders and alerts	PostgreSQL / Redis
All messages	Webhook and orchestration	WhatsApp reply	FastAPI, WhatsApp Cloud API

Fig. 1 shows the context-level data flow diagram. SafarSaathi appears as a single process with three external entities. The commuter sends a query and receives a trip plan, ETA or alert. The transit data provider supplies static feeds and real-time updates. The LLM and speech service receives prompts and audio and returns intents and transcripts.

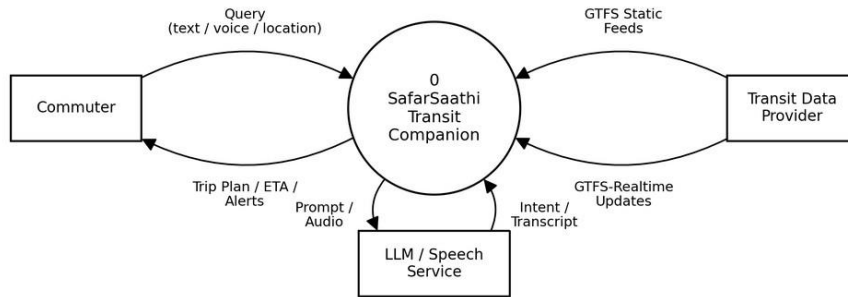


Fig. 1 - Data flow diagram, level 0 (context). The whole system is one process exchanging data with the commuter, the transit data provider and the LLM and speech service.

Fig. 2 breaks this single process into four sub-processes and four data stores. Process 1.0 interprets the message with the help of the LLM service and passes on the intent with the origin and destination. Process 2.0 resolves these to stop IDs using the static GTFS store (D1). Process 3.0 plans the route and ETA using trips and stop times from D1, live delays from the GTFS-Realtime store (D2) and the walking network in the OTP graph (D4). Process 4.0 composes the reply, reads and writes conversation context in the user session store (D3), and sends the answer back to the commuter.

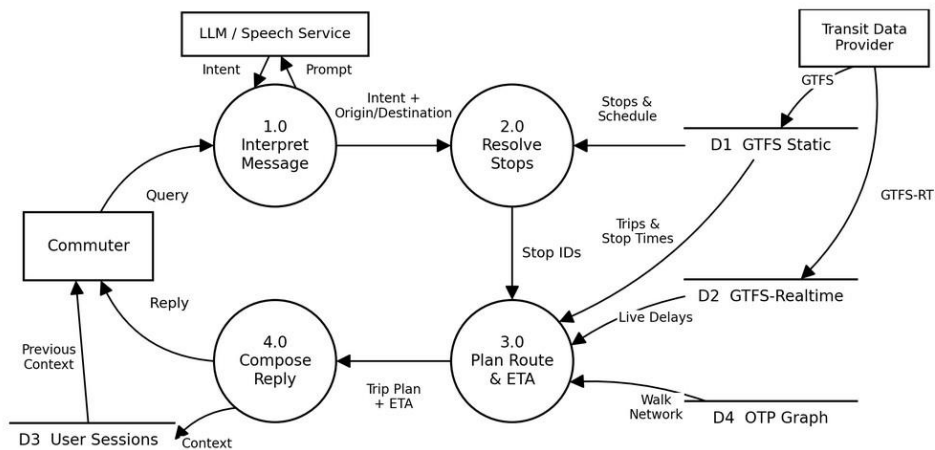


Fig. 2 - Data flow diagram, level 1. Process 0 decomposed into four numbered sub-processes and four data stores.

Fig. 3 expands process 3.0. Sub-process 3.1 builds candidate routes from the stop IDs, the walk network and the timetable. Sub-process 3.2 merges live delays into these routes. Sub-process 3.3 computes the ETA, using vehicle positions where they exist, and 3.4 ranks the candidates and passes the best trip and its ETA to process 4.0.

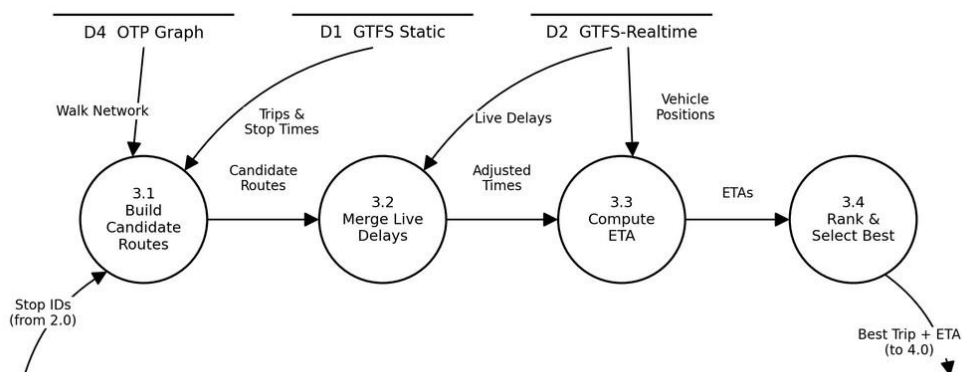


Fig. 3 - Data flow diagram, level 2, for process 3.0 (plan route and compute ETA), showing sub-processes 3.1 to 3.4.

The use case diagram in Fig. 4 shows the commuter as the primary actor, with six use cases: asking for the next arrival, planning a multimodal trip, sending a voice query, sharing a location for the nearest stop, getting a reply in their own language, and setting reminders and receiving alerts. Two shared

use cases, language and intent detection and fetching live transit data, are included by the arrival and trip-planning cases. They connect to the two secondary actors, the LLM and NLP service and the GTFS feeds.

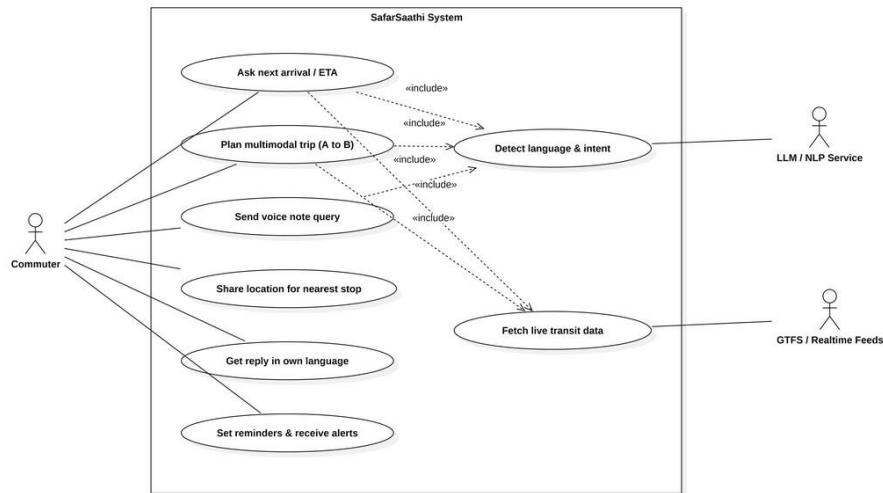


Fig. 4 - Use case diagram with the commuter as primary actor and the LLM/NLP service and GTFS feeds as secondary actors.

6.3 Non-functional requirements

Security. Webhooks run over HTTPS and every incoming request is checked against Meta's signature. Phone numbers are stored only as hashes, and API keys are kept in environment secrets, never in the code.

Reliability. If a live feed fails, the system falls back to schedule-based ETAs and says so. Calls to external APIs are retried on failure, and the target uptime is 99%.

Efficiency. The target is a reply within about five seconds end to end. The stop index and live delays are cached, and replies are kept as light text so they arrive quickly on slow networks.

Maintainability. The NLP, routing and reply services are separate modules, each in its own Docker container. The code is under version control and GTFS refreshes are automated.

Scalability. API workers keep no state between requests, so more can be added behind a load balancer as traffic grows. A new city needs only its GTFS feed.

Usability. The interface is a plain-language chat with voice input and replies in the user's language. There are no menus or maps to learn.

7. Requirement analysis

7.1 Hardware requirements

Table 8 lists the planned hardware. These are planning estimates for a prototype that serves one city. Production sizing will depend on query volume and on the size of each city's feed. The OpenTripPlanner graph is the most memory-hungry component, which is why the routing server needs more RAM than the application server. On the commuter's side, any phone that runs WhatsApp is enough, although a smartphone with GPS is needed to share a location pin.

Table 8 - Hardware requirements.

Component	Minimum	Recommended	Purpose
Application server (cloud VM)	4 vCPU, 8 GB RAM	8 vCPU, 16 GB RAM	Backend API, NLP orchestration
Routing server (OTP)	8 GB RAM for one city graph	16 GB+ RAM for several cities	Build and serve the OTP graph
Storage	50 GB SSD	100 GB SSD	GTFS feeds, OSM extract, database, logs

Component	Minimum	Recommended	Purpose
Network (server)	100 Mbps, static IP / HTTPS domain	1 Gbps with load balancer	Webhook traffic, feed polling
Developer machine	Intel i5 / Ryzen 5, 8 GB RAM	Intel i7 / Ryzen 7, 16 GB RAM	Coding, local OTP testing
Commuter device	Any phone with WhatsApp, 2G/3G	Smartphone with 4G and GPS	Queries, voice notes, location

7.2 Software requirements

Table 9 lists the software stack. The backend is written in Python 3.11 with FastAPI, and OpenTripPlanner runs on a Java 21 runtime (OpenTripPlanner Project, n.d.). Stop matching uses RapidFuzz, and GTFS-Realtime feeds are parsed with the official gtfs-realtime-bindings. Data lives in PostgreSQL 15 with PostGIS, and Redis 7 handles caching. Development uses Docker, Git and GitHub, VS Code and Postman.

Table 9 - Software requirements.

Category	Software / version	Role
Operating system	Ubuntu Server 22.04 LTS (server); Windows / Linux (dev)	Hosting and development
Languages	Python 3.11; Java 21 runtime	Backend logic; OTP runtime
Frameworks and libraries	FastAPI, Requests, RapidFuzz, gtfs-realtime-bindings	Webhook API, feed parsing, stop matching
Routing engine	OpenTripPlanner 2.x	Multimodal routing and ETA
AI / NLP services	LLM API, Whisper speech-to-text, MahaBERT / IndicNLP	Intent, language, translation, transcription
Messaging platform	WhatsApp Business Cloud API (Meta)	Receive and send messages
Database and cache	PostgreSQL 15 + PostGIS, Redis 7	Stops, sessions, reminders; live-delay cache
DevOps and tools	Docker, Git/GitHub, VS Code, Postman	Deployment, version control, API testing

8. System design

8.1 System architecture

SafarSaathi uses the layered architecture shown in Fig. 5. The presentation layer is the commuter's own WhatsApp app, through which they send text, voice notes and locations. The interface layer is the WhatsApp Business Cloud API, which forwards each message to our HTTPS webhook. The data layer holds the static GTFS feeds, the GTFS-Realtime feeds, PostgreSQL with PostGIS, and the Redis cache.

The application layer sits between them and has five components. The NLP and LLM engine detects the language, the intent and the origin and destination. The query orchestrator coordinates the lookups that a request needs. OpenTripPlanner computes routes and ETAs. The response composer translates and formats the answer, and the session manager keeps the conversation context and the reminders.

A typical request moves through these layers in order. When a commuter asks for the next train from Andheri to Churchgate, the webhook passes the message to the orchestrator, which sends it to the NLP engine. The engine returns the intent and the two place names. The orchestrator resolves them to stops, asks OpenTripPlanner for the next departures with live delays applied, and hands the result to the response composer, which writes a short reply in Marathi. The session manager saves the exchange, so that a follow-up such as "and by bus?" can be understood without the user repeating the stations.

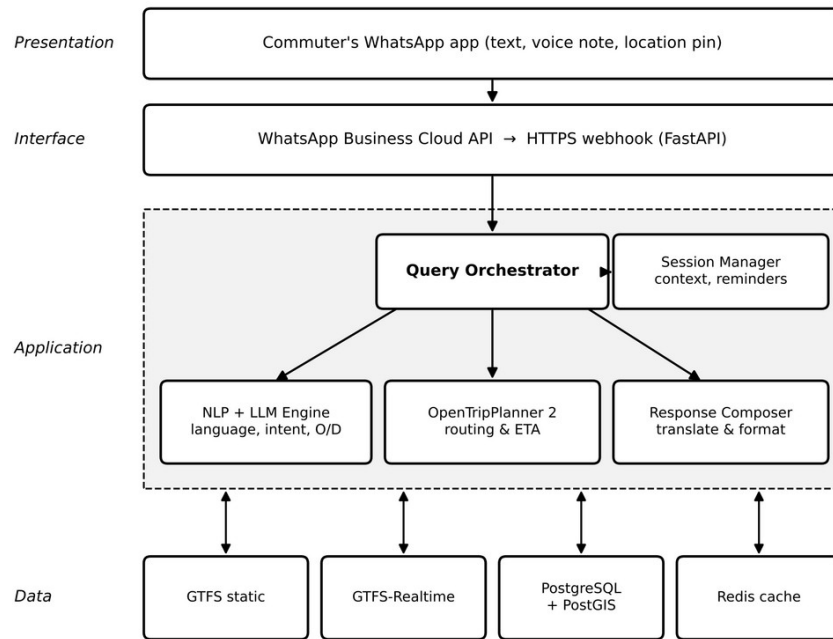


Fig. 5 - Layered architecture of SafarSaathi. The query orchestrator coordinates the five application components, which read from and write to the data layer.

We kept the orchestrator separate from the LLM on purpose. The LLM is called twice per request, once to read the message and once to phrase the reply, and both calls are narrow. All decisions about which stop, which route and which time are made in ordinary code that can be tested and logged. If the LLM provider changes or goes down, only the NLP engine and the response composer are affected, and a template-based reply in the user's language can still be sent.

8.2 System flowchart and algorithm

Fig. 6 shows the full flow for one message. After a message arrives, its type decides the first step: voice notes go to speech-to-text, location pins go to the nearest-stop lookup, and text goes straight to the LLM. If the LLM cannot make sense of the request, the bot asks the user to rephrase. Otherwise the origin and destination are resolved to stops. If a live feed is available, the GTFS-Realtime ETA is used; if not, the schedule ETA is used. OpenTripPlanner plans and ranks the routes, and the reply is composed, translated and sent. If the user asked for a reminder, it is saved, and in every case the session context is stored before the flow ends.

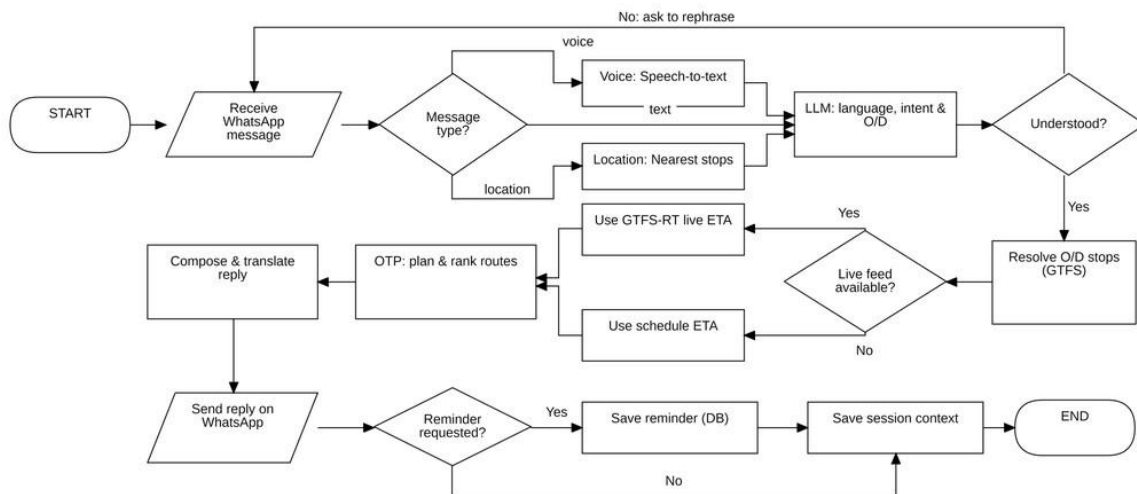


Fig. 6 - System flowchart for handling one WhatsApp message, including the voice and location branches, the clarification loop and the live/schedule ETA decision.

Fig. 7 gives the same logic as a step-by-step algorithm, summarised below as Algorithm 1.

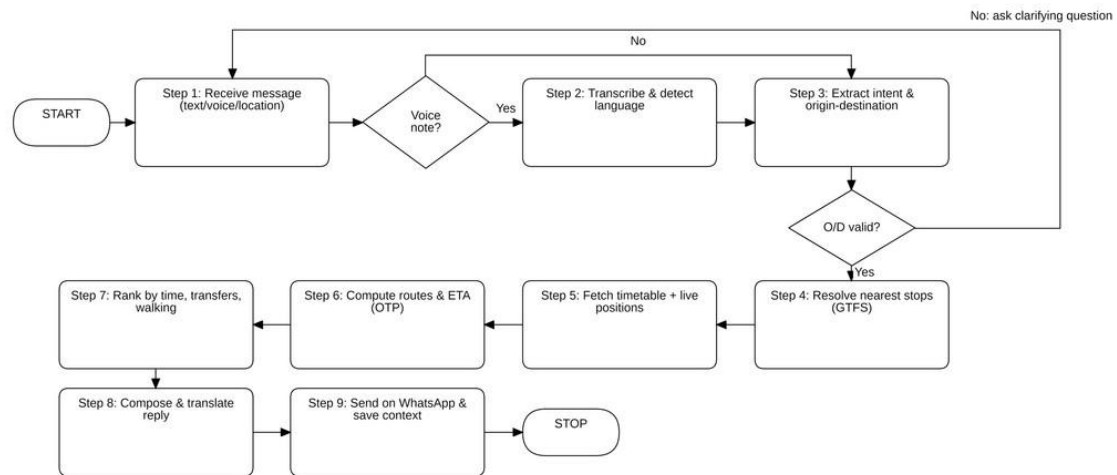


Fig. 7 - Flowchart of the query-processing algorithm (Algorithm 1).

Algorithm 1: Query processing in SafarSaathi

1. Receive the message (text, voice note or location).
2. If it is a voice note, transcribe it; detect the language.
3. Extract the intent, origin and destination with the LLM.
4. If the origin or destination is missing or ambiguous, ask a clarifying question and return to step 1.
5. Resolve the origin and destination to the nearest GTFS stops.
6. Fetch the timetable and any live vehicle positions and delays.
7. Compute candidate routes and ETAs with OpenTripPlanner.
8. Rank the routes by travel time, number of transfers and walking distance.
9. Compose the reply and translate it into the user's language.
10. Send the reply on WhatsApp and save the session context.

The ranking in step 8 has a large effect on whether people follow the advice. The fastest route on paper may involve three transfers and a long walk, which an elderly commuter carrying luggage would rather avoid. Weighing transfers and walking alongside total time gives answers that people are more likely to follow.

8.3 User interface design

SafarSaathi has no new screens. Its interface is a WhatsApp conversation, and Table 10 shows a sample exchange in Marathi for a trip from Andheri to Churchgate. The user asks when the next train leaves. The bot replies that the next fast local leaves in about six minutes and reaches Churchgate in about 45. When the user asks how long the bus would take, the bot compares the two and suggests the train. The second question never names the stations; the session context supplies them.

Table 10 - Sample conversation (Marathi, with English translation).

Sender	Message
User	अंधेरीहून चर्चगेटला जायचंय, पुढची ट्रेन कधी? (Andheri to Churchgate. Next train?)
Bot	पुढची फास्ट लोकल ~6 मिनिटांत. चर्चगेट ~45 मिनिटांत. (Next fast local in ~6 min. Reach in ~45 min.)
User	आणि बसने किती वेळ? (And how long by bus?)

Sender	Message
Bot	बस मंद आहे (~1 तास). लोकल जलद, तीच घ्या. (Bus is slower, about 1 hr. The local is faster, so take that.)

Beyond plain text, the interface uses a few built-in WhatsApp elements. Quick-reply buttons such as "Next train", "Plan trip" and "Set reminder" let the user tap instead of typing. On first contact a list menu asks for the preferred language and service. Users can share their live location to find nearby stops, and they can speak a question as a voice note; the answer comes back as clear text. Service and diversion alerts, as well as reminders before departure, arrive as short alert cards in the same chat.

8.4 Deployment design

The deployment has four physical parts. The commuter's phone can be any handset with WhatsApp on a 2G, 3G or 4G connection, with GPS as an option for location pins. Meta's WhatsApp cloud hosts the business number and forwards messages to our webhook over HTTPS. Our side is a single Ubuntu cloud VM (8 vCPU, 16 GB RAM, 100 GB SSD) running four Docker containers: the FastAPI application with the webhook and orchestrator, OpenTripPlanner with its routing graph, PostgreSQL with PostGIS for stops and sessions, and Redis for the live-delay cache. The VM calls the external LLM and speech-to-text APIs for intent, translation and transcription, and it pulls GTFS and GTFS-Realtime data from the transit agencies' feed servers. Fig. 8 shows these nodes and the links between them.

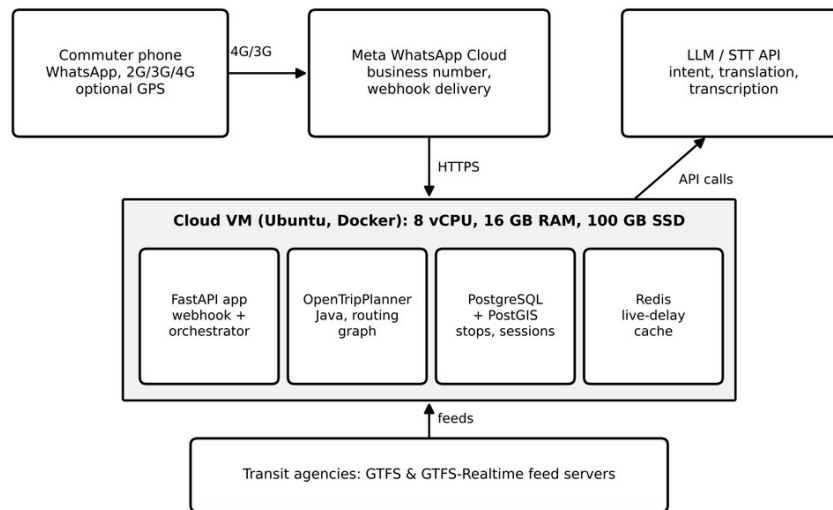


Fig. 8 - Deployment view: commuter phone, Meta's WhatsApp cloud, the application VM with its four containers, the external LLM and speech APIs, and the agencies' feed servers.

A single VM is enough for a one-city prototype. For a larger rollout, the FastAPI containers can be replicated behind a load balancer because they keep no state, while OpenTripPlanner, which holds the whole routing graph in memory, would move to its own machine with more RAM, as Table 8 suggests.

8.5 Database design

The relational schema in Table 11 has two groups of tables. User data covers USERS, SESSIONS, MESSAGES and REMINDERS. Transit data mirrors the GTFS files in STOPS, ROUTES, TRIPS and STOP_TIMES. A user has many sessions and reminders, and a session has many messages. On the transit side, a route has many trips, and each trip and each stop has many stop times. The STOPS table keeps names in English, Hindi and Marathi, which is what the stop resolver searches. Only a hash of the phone number is stored, never the number itself.

Table 11 - Database schema (PostgreSQL).

Table	Key fields
USERS	user_id (PK), phone_hash, preferred_lang, created_at
SESSIONS	session_id (PK), user_id (FK), last_intent, origin_stop / dest_stop (FK), updated_at
MESSAGES	msg_id (PK), session_id (FK), direction, type, text, lang, ts

Table	Key fields
REMINDERS	reminder_id (PK), user_id (FK), route_id (FK), trigger_time, status
STOPS	stop_id (PK), stop_name, name_hi, name_mr, lat, lon (geom), mode
ROUTES	route_id (PK), agency_id, short_name, mode
TRIPS	trip_id (PK), route_id (FK), service_id, headsign
STOP_TIMES	trip_id (FK), stop_id (FK), arrival, departure, stop_sequence

8.6 Prompt and reply design

The intent prompt has three parts. A short instruction tells the model that it is reading a transit question from a commuter in Mumbai and must return only JSON in a fixed schema. A list of the allowed intents follows, each with a one-line description. The last part is a handful of worked examples in all three languages, including code-mixed Hindi-English, Marathi written in Latin script, and a message with a misspelled station. The schema includes a field for the model's own confidence and a field for anything it could not place, so the orchestrator knows when to ask a follow-up question.

The reply prompt is even narrower. It receives the structured result from OpenTripPlanner (departure time, arrival time, line, platform where known, number of transfers, and whether the time is live or scheduled) and the user's language, and it is asked to write two or three short sentences. Numbers are passed in as fields and checked after generation: if a time in the reply does not match a time in the routing result, the reply is discarded and a template is used instead. The check is cheap, and it catches the most harmful kind of error, a wrong departure time stated with confidence.

Replies follow a few rules based on how people text each other about trains. The most useful fact comes first, usually how many minutes until the next departure. Station and line names stay in the form people see on signboards. Emojis are used sparingly, mostly a train or bus symbol to mark the mode. Long trip plans are split into numbered legs, one per line, so they stay readable on a small screen.

8.7 Privacy and data handling

SafarSaathi needs very little personal data to work. It stores a hash of the phone number, the preferred language, recent messages for context, and any reminders the user has set. On first contact, the welcome message explains what is stored and why, and the user can send a single word to delete their history. Messages older than the session window are removed from the log on a schedule. Location pins are used for the nearest-stop lookup and are not kept afterwards. These choices follow the purpose-limitation and data-minimisation principles of the DPDP Act (Government of India, 2023). Text sent to the external LLM contains only the message itself, without the phone number.

9. Evaluation and discussion

9.1. Evaluation setup

The prototype is tested on a set of sample commuter queries written in Hindi, Marathi and English. The set covers every intent (next arrival, trip plan, nearest stop, reminder and service alert), includes typed messages and voice notes, and deliberately contains misspelled station names and code-mixed sentences such as "Andheri se Churchgate next fast kab hai?". Each query is labelled by hand with its language, intent, origin and destination.

Four measures are reported. Language detection accuracy is the share of queries whose language is identified correctly. Intent recognition accuracy is the share whose intent matches the label. End-to-end response time runs from the moment the webhook receives a message to the moment the reply is sent. Query resolution success rate is the share of queries for which the user receives a correct and usable answer, either straight away or after one clarifying question. Response time is also measured on 2G, 3G and 4G connections during field tests. A usability study with commuters from the target groups, using the System Usability Scale (Brooke, 1996), is planned for the final phase.

Testing happens at three levels. Unit tests cover the stop resolver, the JSON parser for LLM output and the delay-merging logic, using fixed GTFS extracts so that results do not change from day to day. Integration tests send recorded WhatsApp payloads to the webhook through Postman and check the full reply. Field tests use real Android and iOS handsets on mobile networks of different speeds, which is the only way to see how long a reply really takes to arrive. After every nightly GTFS refresh, a small regression set of known journeys is re-run to catch broken feeds before users do.

9.2 Results

Table 12 reports the prototype's results on the test set. Values in brackets are to be filled in from the final measured runs.

Table 12 - Prototype evaluation results.

Metric	Result
Language detection accuracy	[] %
Intent recognition accuracy	[] %
Average end-to-end response time	[] s
Query resolution success rate	[] %
Understanding accuracy: English	[] %
Understanding accuracy: Hindi	[] %
Understanding accuracy: Marathi	[] %

We expect English to score highest and Marathi lowest of the three languages, since Marathi has the least training data among them (Joshi, 2022). The per-language rows in Table 12 will show how large that gap is in practice. Response time is dominated by the external LLM call, so it depends more on the LLM provider than on our own code; routing in OpenTripPlanner for a single city typically takes a small fraction of the total.

9.3 Expected output

The finished system should give commuters accurate next-bus and next-train times on WhatsApp within seconds, along with complete A-to-B journeys that combine bus, metro and local rail with transfers. Answers come in Hindi, Marathi or English, whichever the user wrote or spoke in. A location pin returns the closest stops and how to reach them, and reminders and service or diversion alerts arrive in the same chat. Because the transit data layer is plain GTFS, the same assistant can be reused in any city with an open feed.

9.4 Discussion and limitations

The main technical risk is the LLM. It can misread a request, especially when a voice note was transcribed badly (He & Garner, 2023), and it could, in principle, invent a time. The design limits this risk by never letting the LLM produce routes or times (DR2); it only reads the request and writes the reply around numbers that OpenTripPlanner computed. The clarification loop in Fig. 6 is a second safeguard, since an uncertain parse leads to a question instead of a guess.

Data availability is the second constraint. Live GTFS-Realtime coverage in Mumbai is partial, so many answers, particularly for buses, will rest on timetables. Traffic-based prediction of the kind used in BusTr (Barnes et al., 2020) could narrow this gap later. Relying on WhatsApp also ties the service to Meta's platform rules and per-message pricing, and the IVR channel in our future work would give feature-phone users a route that does not depend on it.

The literature review has limits of its own. It covers fifteen core studies and may have missed grey literature and work published in Indian languages. Several of the included studies are preprints. Few studies test their methods on Indian transit networks, and none evaluates LLMs on Indian station names or code-mixed queries. Our own evaluation is designed to start filling that gap.

10. Conclusion and future scope

This paper presented SafarSaathi, a real-time, multilingual public-transport companion delivered over WhatsApp. The literature shows that open transit data, fast multimodal routing, LLM-based intent understanding and Indic language models are each mature, and that real-time information measurably helps riders. What had not been done was to bring them together in one channel that excluded commuters already use. SafarSaathi does this by combining GTFS data and OpenTripPlanner with a grounded LLM layer and delivering the result through WhatsApp in Hindi, Marathi and English. We described its requirements, analysis, architecture, algorithm, interface and data model, and set out how the prototype is evaluated.

Several extensions are planned. Crowd and delay reports could be drawn from live feeds and from users themselves. Fare and ticketing information and last-mile options such as autos and shared cabs would make trip plans more complete. Coverage should grow to more cities and regional languages,

starting with those that already publish GTFS feeds. Finally, a voice-call (IVR) version would bring the same service to people who use feature phones and do not have WhatsApp.

References

- Barnes, R., Buthpitiya, S., Cook, J., Fabrikant, A., Tomkins, A., & Xu, F. (2020). BusTr: Predicting bus travel times from real-time traffic. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. arXiv:2007.00882.
- Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., & Werneck, R. F. (2016). Route planning in transportation networks. In *Algorithm Engineering* (LNCS Vol. 9220, pp. 19–80). Cham, Switzerland: Springer.
- Brooke, J. (1996). SUS: A ‘quick and dirty’ usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). London, U.K.: Taylor & Francis.
- Brown, T., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (pp. 1877–1901).
- Delling, D., Pajor, T., & Werneck, R. F. (2015). Round-based public transit routing. *Transportation Science*, 49(3), 591–604.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT* (pp. 4171–4186).
- Devunuri, S., Qiam, S., & Lehe, L. J. (2024). ChatGPT for GTFS: Benchmarking LLMs on GTFS semantics... and retrieval. *Public Transport*, 16(2), 333–357.
- Dibbelt, J., Pajor, T., Strasser, B., & Wagner, D. (2018). Connection scan algorithm. *ACM Journal of Experimental Algorithmics*, 23. arXiv:1703.05997.
- Eltigani, H., Haroon, R., Kocak, A., Faisal, A. B., Martin, N., & Dogar, F. (2025). WaLLM – Insights from an LLM-powered chatbot deployment via WhatsApp. arXiv:2505.08894.
- Fantin, L., Antonelli, M., Cesetti, M., Irto, D., Zamengo, B., & Silvestri, F. (2025). Design and testing of an agent chatbot supporting decision making with public transport data. arXiv:2505.22698.
- Ferris, B., Watkins, K., & Borning, A. (2010). OneBusAway: Results from providing real-time arrival information for public transit. In *Proceedings of ACM CHI* (pp. 1807–1816).
- Gala, J., et al. (2023). IndicTrans2: Towards high-quality and accessible machine translation models for all 22 scheduled Indian languages. *Transactions on Machine Learning Research*.
- Gala, J., et al. (2024). Airavata: Introducing Hindi instruction-tuned LLM. arXiv:2401.15006.
- Giannakopoulou, K., Paraskevopoulos, A., & Zaroliagis, C. (2018). Multimodal dynamic journey planning. arXiv:1804.05644.
- Government of India. (2023). *The Digital Personal Data Protection Act, 2023* (Act No. 22 of 2023).
- He, M., & Garner, P. N. (2023). Can ChatGPT detect intent? Evaluating large language models for spoken language understanding. In *Proceedings of Interspeech*. arXiv:2305.13512.
- Joshi, R. (2022). L3Cube-MahaCorpus and MahaBERT: Marathi monolingual corpus, Marathi BERT language models, and resources. arXiv:2202.01159.
- Kakwani, D., et al. (2020). IndicNLPsuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In *Findings of EMNLP* (pp. 4948–4961).
- Kitchenham, B. (2004). *Procedures for performing systematic reviews* (Technical Report TR/SE-0401). Keele, U.K.: Keele University.
- Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*.
- Liu, D., Guo, J., Gu, Y., King, M., Han, L. D., & Brakewood, C. (2024). GTFS2STN: Analyzing GTFS transit data by generating spatiotemporal transit network. arXiv:2405.02760.
- Meta Platforms. (n.d.). *WhatsApp Business Platform: Cloud API documentation*. Retrieved from <https://developers.facebook.com/docs/whatsapp/cloud-api/>
- MobilityData. (n.d.). *General Transit Feed Specification: GTFS Schedule and GTFS Realtime reference*. Retrieved from <https://gtfs.org/>
- Narboneta, V., & Teknomo, K. (2016). OpenTripPlanner, OpenStreetMap, General Transit Feed Specification: Tools for disaster relief and recovery. arXiv:1609.01472.
- OpenTripPlanner Project. (n.d.). *OpenTripPlanner 2 documentation*. Retrieved from <https://docs.opentripplanner.org/>
- Page, M. J., et al. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, 372, n71.

-
- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2023). Robust speech recognition via large-scale weak supervision. In *Proceedings of ICML*.
- Schick, T., et al. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).
- Watkins, K. E., Ferris, B., Boring, A., Rutherford, G. S., & Layton, D. (2011). Where is my bus? Impact of mobile real-time information on the perceived and actual wait time of transit riders. *Transportation Research Part A*, 45(8), 839–848.