AURA: Software Architecture and System-One Decision Integration for an Edge Artificial Intelligence Departmental Assistant Robot

Siddhant Ramesh Dube[1] Abhishek Devanand Jadav[2] Rusha Gajanan Panzade[3] Aditya Mahadev Jaybhay[4] Prof. Dhiraj Tikar[5]
[1,2,3,4,5]Department of Electrical Engineering
[1,2,3,4,5]Trinity College of Engineering and Research, Pune, India

Abstract — University academic offices struggle with distributed, disconnected informational channels for scheduling, course curricula, instructor availability, circulars, and course attendance. Mobile robotics presents an integrated interactive medium, yet industrial mobile bases and anthropomorphic platforms remain excessively intricate and costly for localized departmental use. This paper conducts a software architectural review of AURA (Academic Utility Robotic Assistant), an edge-AI assistant configured for voice interaction and currently undergoing simulation-stage evaluation. The work reports no synthetic or unverified experimental metrics; instead, it outlines the system software infrastructure—incorporating ROS 2 Jazzy, Nav2, Gazebo Harmonic simulation, FastAPI service management, ChromaDB-indexed retrieval-augmented generation (RAG), dual-factor BLE/TOTP verification, and role-based security policies—alongside the introduction of TypeSafe's Jev System One decision model. Delegating input validation, task classification, and threshold-gated escalation away from generative large language models (LLMs) toward typed Choice, Score, and Noul questions reduces parsing errors, bounds query latency, and keeps every AI component outside the real-time ESP32 wheel control loop. A measurement plan and a set of unmeasured V1 targets are fixed in advance so that later prototype results can be judged against clear criteria.

Keywords: AURA, Edge-AI Robotics, ROS 2 Jazzy, Nav2, Retrieval-Augmented Generation, System One Decision Model, Jev by TypeSafe, Digital Twin

I. Introduction

Engineering departments handle constant informational traffic, ranging from syllabus inquiries and schedule verifications to classroom navigation and roster authentication. Conventional solutions scatter these activities across fragmented online portals and informal chat channels, imposing administrative friction on departmental staff and students alike. Dedicated service robots offer a practical physical alternative by providing localized voice interaction; nevertheless, unconstrained autonomous systems bring significant software overhead, factual hallucinations from conversational models, and compute demands that outstrip edge hardware capabilities.

AURA (Academic Utility Robotic Assistant) is designed as a focused, domain-bounded platform serving a single-floor Electrical Engineering department. Rather than chasing open-ended intelligence or large architectural scale, AURA's software framework couples localized edge services running on an ARM Single-Board Computer (SBC), the ROCK 5B+ 8GB, with hard real-time motor control executed on an ESP32. An Axon Vichaarak board is kept as an alternative host, but only if it clears the same compatibility checks. With prototype manufacturing and final deployment still pending, this paper provides a design-stage architectural review: we analyze the proposed software modules, lay out a digital-twin simulation method, and introduce TypeSafe's Jev System One model to make routing decisions deterministic, without presenting fabricated experimental figures or unverified performance claims.

The main contributions are: (1) a requirements-driven software architecture that keeps every AI component outside the motor-control loop; (2) a decision layer that places Jev in front of retrieval and generation, with explicit thresholds, a rule-based fallback and model-version pinning; (3) a dual-factor BLE/TOTP attendance flow whose codes are checked only on the server; and (4) a measurement plan and engineering targets fixed before any hardware result exists. Section II reviews related technologies and Section III states the design requirements. Sections IV to VI describe the architecture, the decision layer and the application pipelines, and Section VII covers evaluation and limitations.

II. Review of Related Technologies

A. ROS 2 and Nav2 for Indoor Service Robotics

The robotics community has moved broadly to Robot Operating System 2 (ROS 2), adopting Data Distribution Service (DDS) communication that replaces the old master-node design with peer-to-peer discovery, configurable quality-of-service profiles and lifecycle-managed nodes [1]. On top of ROS 2, the Nav2 stack uses behavior trees to coordinate global path planning, local obstacle avoidance, costmap integration and recovery routines in indoor spaces [2]. Map building for such a robot is commonly handled by SLAM Toolbox, a graph-based Simultaneous Localization and Mapping (SLAM) package maintained for ROS 2 [3], and Gazebo, which began as an open multi-robot simulator [4], lets the same launch files run on a virtual robot first. For a service robot moving at 0.15–0.25 m/s between pre-mapped departmental waypoints, ROS 2 Jazzy with Nav2 and Gazebo Harmonic gives a workable simulation framework before any hardware is assembled.

B. Domain-Constrained Retrieval-Augmented Generation

Standard parametric large language models (LLMs) frequently generate unsupported assertions [5] and cannot keep up with institutional information that changes often, such as room schedules or daily notices. Retrieval-Augmented Generation (RAG) addresses this by pairing an external dense vector store with the generator, so that answers stay tied to retrieved passages [6]. The dense search step normally relies on sentence-level embeddings of the Sentence-BERT family [7], which is also the approach AURA follows. In a department setting, combining vector search with metadata filters (unit, semester, course code) and strict domain filtering keeps the conversation from drifting into unrelated topics.

All rights reserved by www.ijsrd.com

C. System One Decision Models: Jev by TypeSafe

Conversational systems often push routine routing, filtering and classification through generative models, paying for it in latency, token cost and unstable JavaScript Object Notation (JSON) output. Jev, which TypeSafe AI describes as its first public System One model, takes a different route. The name borrows Daniel Kahneman's distinction between fast, intuitive System One judgments and slow, deliberate System Two reasoning [8]. Instead of generating tokens, Jev evaluates a set of typed questions against an application state and returns typed values with probability distributions; it does not write replies, produce code or explain its reasoning [9], [10]. Three question types matter for AURA: Choice selects one answer from a defined set, Score places an input on a defined scale, and Noul returns a yes/no-style probability [10]. A fourth type, Box, which returns an image region, arrived with version 1.14 [11]. All questions in one request are evaluated in parallel, so adding a question barely changes response time [9].

TypeSafe's own demonstration workflow reports a Jev run of 0.114 s against 8.566 s for a comparable LLM run [12]. These are vendor figures, so this paper treats them as a claim to be checked on AURA's own network, not as a result. TypeSafe also points out that calibration holds across groups of predictions and does not guarantee that any single answer is correct [10]; this is the reason AURA keeps a threshold and a fallback in front of every action that Jev triggers.

D. Comparison of Routing Approaches

Table I compares the three ways a departmental robot can decide what a request means. The comparison is qualitative on purpose; quantitative figures will come from the measurements planned in Section VII.

Approach	Output	Main weakness	Role in AURA
Keyword rules	Fixed match	Brittle to new phrasing	Offline fallback
Generative LLM	Free text or JSON	Slow on edge board; format drift	Phrasing answers only
System One (Jev)	Typed value + probability	Needs network; vendor-hosted	Primary router

Table I: Comparison Of Intent-Routing Approaches

III. Design Requirements

Before choosing software, the team fixed a short list of requirements taken from the project's V1 engineering overview. AURA serves one small, single-floor department and makes roughly 5–6 controlled trips a day, mostly for attendance, and its intelligence is limited on purpose to department matters. Three rules follow from this and shape every later section.

1) The database is the source of truth. The local language model phrases answers; it does not supply facts. Anything that can be answered from structured data, such as a timetable slot, a room or a faculty cabin, is answered from structured data.
2) No AI path reaches the motors. Cloud services and language models may suggest a destination, but motion always passes through ROS 2/Nav2, the /cmd_vel interface and the ESP32 safety controller. A dropped network link must never produce unsafe movement.
3) Cheap decisions stay cheap. Simple requests skip model inference altogether. On an 8 GB board this is the only realistic way to answer deterministic commands in under a second and keep general voice interaction near 5 s.

It is just as important to say what AURA V1 is not: it is not a household robot, a security robot, a multi-floor delivery platform or an unrestricted chatbot. Keeping that list written down has helped the team turn away features that would have pulled the software away from its actual job.

IV. Proposed AURA Software Architecture

A. Compute Responsibility and Safety Boundary

The architecture draws a hard line between cognitive tasks and low-level motor control. The Linux board runs the ROS 2 nodes, Nav2 planning, FastAPI endpoints, the relational database, local RAG retrieval, speech modules and OpenCV vision pipelines, and sends velocity commands on /cmd_vel over a USB serial link to the ESP32. The microcontroller independently runs the Proportional-Integral-Derivative (PID) loops, encoder servicing, ultrasonic monitoring, bumper sensing and physical emergency-stop (E-stop) checks. Cloud Application Programming Interfaces (APIs) and conversational models help with queries but have no access to the actuators, so a transmission delay or a stalled model cannot affect motor safety. In short, the host decides what should happen, and the ESP32 decides whether it can safely happen now. Fig. 1 shows this split together with the decision path above it.

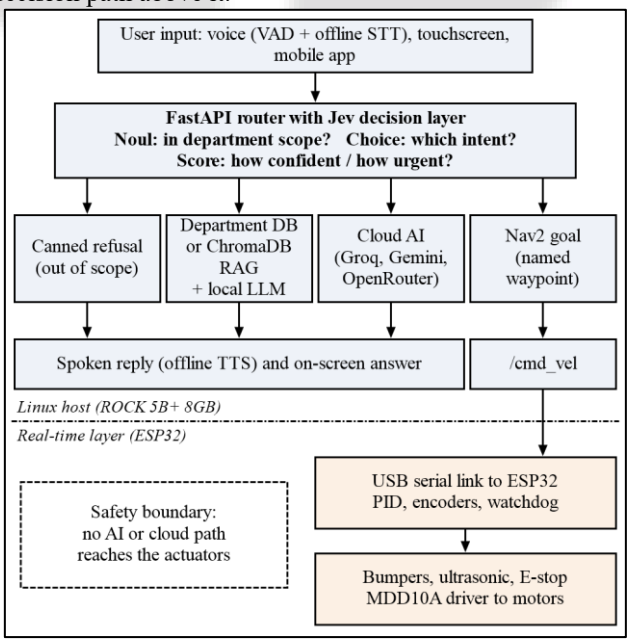


Fig. 1: Proposed AURA software decision flow and safety boundary.

B. Layered Software Stack

The software is divided into layers—Gazebo simulation, ROS navigation, FastAPI orchestration, local RAG services,

All rights reserved by www.ijsrd.com

speech interfaces and mobile endpoints—so that each component boundary can be verified before end-to-end integration. Table II lists the proposed V1 configuration.

Layer	Proposed Technologies
Robot OS	Linux (ARM) on ROCK 5B+ 8GB; Axon Vichaarak as validated alternative
Robotics & Sim	ROS 2 Jazzy, Nav2, SLAM, Gazebo Harmonic, ros_gz_bridge
Backend & DB	Python, FastAPI, SQLite (dev) / PostgreSQL (prod)
AI & RAG	sentence-transformers, ChromaDB, local small LLM, Jev router
Speech & CV	VAD, offline STT/TTS, OpenCV, MediaPipe face recognition
Client UI	7-inch HDMI kiosk UI; Flutter / React Native mobile app
Dev tooling	Ubuntu 24.04, WSL2, Docker, Git/GitHub

Table II: Proposed Aura V1 Software Stack

C. Backend Services and Data Model

FastAPI is the single-entry point for the touchscreen, the mobile app and the voice pipeline. It handles authentication, Role-Based Access Control (RBAC), attendance, QR generation, BLE state, robot status, navigation requests, AI routing, RAG access, notices, timetable and logging. Development starts on SQLite and moves to PostgreSQL for a shared deployment. The core tables are Students, Faculty, Subjects, Units, Rooms, Timetable, Notices, Attendance_Logs, Mentor_Issues, Robot_Status, Navigation_Points and AI_Logs.

Four roles are enforced. Free_Student and Paid_Student cover attendance, questions, notices and learning tools according to tier; Faculty adds attendance analytics, notice, syllabus and issue management and robot interaction; Admin manages users, the knowledge base, the robot, logs and security settings. The touchscreen on the robot never exposes administrative controls without an authenticated session.

V. Integrating JEV by TYPESAFE into AURA

A. Limitations of Pure Generative Routing

In the baseline voice design, incoming audio passes through Voice Activity Detection (VAD) and local Speech-to-Text (STT) before intent is resolved. Using a small on-device LLM for this first filtering and slot extraction creates clear bottlenecks on an 8 GB edge computer: multi-second generation for simple lookups, and occasional syntax errors when the model has to produce structured JSON routing parameters.

B. Proposed Jev System-One Decision Layer

To make routing deterministic and fast, we place Jev inside the FastAPI routing service as a first classification layer, ahead of any database lookup or LLM generation. Because Jev answers several typed questions over a single state payload in one request [9], the service can check the subject boundary, infer intent and flag escalation in one step. The gate itself is a plain threshold rule. For a request state s and a set of candidate actions a, the router picks

$$\hat{a} = \arg\max_a P(a \mid s), \quad (1)$$

and carries out the action only when $P(\hat{a} \mid s) \geq \tau$. Here $P(a \mid s)$ is the probability returned by Jev and τ is a cutoff chosen per decision point. This review does not fix values for τ; they will be set from logged simulation runs, with a stricter cutoff for motion goals than for informational answers. Table III maps each decision point to a question type, as follows:

1) Strict Domain Control (Noul):

Before any ChromaDB query or language model call, Jev decides whether the input concerns Electrical Engineering coursework or department logistics. Inputs below the Noul cutoff get an immediate canned refusal, so the robot does not drift into general chat. Table IV gives typical examples.

2) Deterministic Intent and Tier Routing (Choice):

Valid queries are mapped to one of a fixed set of paths: direct database lookups (schedules, faculty offices), navigation targets such as HOD_OFFICE (the Head of Department office) and ATTENDANCE_POINT, the local RAG pipeline, or cloud multimodal services (Groq → Gemini → OpenRouter). Database queries skip language generation entirely, which is what makes the sub-second target reachable.

3) Confidence-Gated Clarification and Escalation (Score / Noul):

When a voice input is ambiguous, the robot asks for a quick touch confirmation instead of executing an unverified motion goal. Student grievance tickets can also be ranked on an ordinal severity scale so that urgent ones reach faculty first.

Decision Point	Question Type	Software Action
Domain gating	Noul	Reject out-of-scope queries before any LLM call
Intent routing	Choice	Route to DB, Nav2, RAG or cloud AI
Waypoint match	Choice	Map speech to registered room IDs
Safety / clarify	Score	Ask the user if confidence is below the gate
Issue triage	Score	Rank student mentor issues for faculty

Table III: JEV Question-Type Mapping in the Aura Router

User Input	Expected Decision
"Explain how a transformer works."	Allowed (academic)
"What is today's timetable?"	Allowed (database)
"Where is the Electronics Lab?"	Allowed (navigation)
"Who won yesterday's cricket match?"	Refused (out of scope)

Table IV: Examples Of Domain Gating

C. Fallback, Versioning and Data Handling

Jev is reached through TypeSafe's HTTP API (POST /v1/systemone) [10], so it is an external dependency and the design treats it as one. If the call fails or times out, FastAPI falls back to hardcoded keyword rules that still cover navigation to registered waypoints and timetable lookups, and the robot stays useful during an internet outage. The router also pins a specific model version, for example jev-

All rights reserved by www.ijsrd.com

1.13.0, instead of the moving jev-latest alias [11]; otherwise a silent model update could shift the probabilities and quietly invalidate thresholds tuned in simulation. Only the transcribed text and a minimal state object are sent to the API. Audio, face data and student identifiers stay on the robot.

VI. Voice, RAG, Attendance, And Digital-Twin Pipelines

A. Voice Pipeline and Controlled Faculty Recognition

Speech is captured by a directional microphone mounted away from motor noise, silence is filtered by VAD, audio is transcribed locally, intent is resolved by Jev inside FastAPI, and the reply is spoken through offline Text-to-Speech (TTS) at a normalized volume. Face analysis with OpenCV and MediaPipe [13] runs asynchronously against a small registered set of 6–7 departmental faculty members, so it never holds up a spoken reply. Recognition is treated as a closed-set problem inside the department, not as general identification of the public.

B. Department Knowledge Base and RAG

The knowledge base covers 27 Electrical Engineering subjects across Semesters 3–8 (six units per subject), together with timetables, room directories, notices, FAQs and department rules. Source documents are cleaned, split into Markdown chunks, tagged with metadata, embedded with sentence-transformers [7] and stored in ChromaDB. A typical chunk carries its subject, semester, unit and source (for instance Digital Electronics, Semester 4, Unit 3, official syllabus), so the search space can be narrowed before any similarity scoring. At query time the chain is retrieval, a context check that discards weak matches, and only then generation. The local model acts as a phrasing layer over retrieved facts, which also makes each answer traceable to a department document.

C. Dual-Factor Attendance

Attendance combines device binding, BLE proximity and a dynamic code, as shown in Fig. 2. The ESP32 advertises a Bluetooth Low Energy (BLE) beacon, and the student app allows the QR step only when the phone is within about 3 m. This distance must be calibrated in the field, because received signal strength varies widely between handsets [14]. The robot's screen then shows a Time-Based One-Time Password (TOTP) Quick Response (QR) code [15] that changes every 10 s. Following RFC 6238 [15], which extends the HOTP algorithm of RFC 4226 [16], the code is built from a shared secret K and a time-step counter

$$T = \lfloor (t - T_0) / X \rfloor, \quad (2)$$

$$\text{code} = \text{Truncate}(\text{HMAC-SHA-1}(K, T)) \bmod 10^d, \quad (3)$$

where t is the current Unix time, T_0 the reference epoch, X the step length (10 s in AURA) and d the number of digits. The backend recomputes the code itself and never trusts a value reported by the phone, so a photo of an old QR soon becomes useless. Device binding ties each account to one phone to discourage proxy attendance; the exact identifier is still being chosen, because mobile platforms restrict access to hardware IDs for privacy reasons.

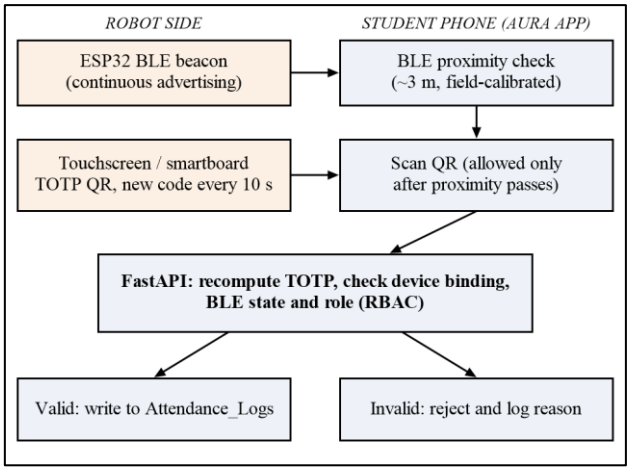


Fig. 2: Proposed dual-factor attendance flow with server-side validation.

In practice the robot does not need to enter classrooms. It drives to ATTENDANCE_POINT, stops, opens a session and shows the QR, and the department smartboards can mirror a larger copy of the code for the whole room.

D. Mobile Application and Service Tiers

The Flutter or React Native app is a separate subsystem that talks only to FastAPI. Free-tier students get attendance and attendance analytics, syllabus questions, notices, mentor issues and project archives. A paid tier adds priority routing and multimodal help through Gemini, such as PDF and handwritten-note analysis, circuit and schematic reading, and waveform interpretation from cathode-ray and digital storage oscilloscopes (CRO/DSO). Faculty use the app for lecture completion, syllabus coverage, attendance reports, notices and infrastructure tickets. Cloud requests follow a Groq → Gemini → OpenRouter fallback chain. This chain is purely an application-level choice: none of these providers sits on the motion path, and provider costs will be revalidated before any paid tier goes live.

E. Faculty Navigation Workflow

A typical faculty request shows how the pieces connect. A spoken command such as "AURA, take me to the HOD office" is transcribed, a Choice question maps it to the registered waypoint HOD_OFFICE, and if the probability clears the motion threshold the router sends a goal to Nav2. Nav2 plans the path, the ESP32 executes /cmd_vel under its own safety checks, and on arrival the robot announces, "We have reached the HOD office." If the match is uncertain, the robot asks for confirmation on the touchscreen before it moves.

F. Simulation-First Digital Twin

To check software workflows before physical assembly, the project uses a Unified Robot Description Format (URDF) model in Gazebo Harmonic, connected to ROS 2 Jazzy through ros_gz_bridge. The virtual setup exercises differential kinematics, simulated encoder odometry (/odom), synthetic 2D Light Detection and Ranging (LiDAR) scans (/scan), Nav2 route tracking and emergency stopping. The first-month twin specification records robot dimensions, wheel radius and separation, sensor positions, coordinate

All rights reserved by www.ijsrd.com

frames and ROS topics, and it already connects to FastAPI, the dashboard, a basic RAG service and the domain classifier. The twin is meant to stay in use as a regression bench, so any change to routing rules or Nav2 parameters can be replayed before it reaches the real robot.

VII. Evaluation Plan, Targets, and Limitations

A. Planned Measurements

No hardware results exist yet, so the evaluation plan is fixed now, before any result can influence it. Table V lists what will be measured for each subsystem, and Table VI lists the V1 targets. The router row is specific to this work: Jev will be compared against the rule-based fallback on the same logged requests, so that its value is shown by measurement rather than assumed.

Subsystem	Planned Measurements
Navigation	Position and stopping error, route completion, obstacle-avoidance success, trip time
Voice	Word error rate, command recognition rate, response latency
RAG	Retrieval accuracy, answer correctness, hallucination rate
Router (Jev)	Routing accuracy, clarification rate, latency vs. rule fallback
Attendance	False acceptance and rejection, QR validation time, BLE proximity reliability
System	CPU/RAM load, compute temperature, battery use, uptime

Table V: Planned Measurements Per Subsystem

B. Engineering Targets

Metric	Proposed Target
Route completion	≥ 95% in mapped single-floor runs
Safety stop	100% in controlled obstacle tests
Attendance validation	≥ 99% backend TOTP/BLE match
Dept. query correctness	≥ 95% grounded in DB/RAG
Voice command recognition	≥ 90% in normal dept. noise
Local command latency	< 1 s for deterministic routes
General voice interaction	≈ 5 s practical local V1 target

Table VI: Proposed Aura V1 Engineering Targets (Unmeasured)

C. Limitations and Security Roadmap

Several software limits are expected: precision loss in quantized local models, network delay for cloud multimodal work such as circuit and waveform analysis, BLE Received Signal Strength Indicator (RSSI) variation across student phones, and the effect of room lighting on face recognition. Jev's dependence on a hosted API is handled by the fallback described in Section V-C, but it remains a dependency. V1 security rests on authentication, RBAC, device binding, BLE and TOTP. Later versions are planned to add TLS on every link, certificate-based robot identity, signed firmware with over-the-air (OTA) updates and rollback, encrypted secrets, key rotation and audit logs.

VIII. Conclusion

This paper has set out the software architecture planned for AURA and shown how ROS 2/Nav2 navigation, FastAPI coordination, metadata-filtered RAG and dual-factor attendance can run together on one edge board. Placing TypeSafe's Jev System One model in the router separates fast, typed classification from language generation; this limits hallucination and keeps the real-time ESP32 firmware fully isolated from AI components. A pinned model version, explicit thresholds and a rule-based fallback keep the design honest about its dependence on a hosted model. The next step is to stress-test the full node network in the Gazebo Harmonic twin, collect the measurements in Table V, and only then move to physical chassis integration.

Acknowledgment

The authors thank Prof. Dhiraj Tikar, Department of Electrical Engineering, Trinity College of Engineering and Research, Pune, for his technical guidance and mentorship in shaping the AURA system architecture and this review.

References

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022.

[2] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Marathon 2: A navigation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2718–2725.

[3] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *J. Open-Source Softw.*, vol. 6, no. 61, p. 2783, 2021.

[4] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2004, pp. 2149–2154.

[5] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, et al., "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, art. 248, 2023.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[7] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.

[8] D. Kahneman, *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.

[9] TypeSafe AI, "Introduction," *TypeSafe Documentation*, 2026. [Online]. Available: https://docs.typesafe.ai/introduction (accessed Nov. 17, 2026).

[10] TypeSafe AI, "System One," *TypeSafe Documentation*, 2026. [Online]. Available: https://docs.typesafe.ai/concepts/system-one (accessed Nov. 17, 2026).

All rights reserved by www.ijsrd.com

[11] TypeSafe AI, "Models," *TypeSafe Documentation*, 2026. [Online]. Available: https://docs.typesafe.ai/models (accessed Nov. 17, 2026).
[12] TypeSafe AI, *TypeSafe AI* (product website, Jev workflow comparison), 2026. [Online]. Available: https://typesafe.ai/ (accessed Nov. 17, 2026).
[13] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, et al., "MediaPipe: A framework for building perception pipelines," arXiv:1906.08172, 2019.
[14] R. Faragher and R. Harle, "Location fingerprinting with Bluetooth Low Energy beacons," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 11, pp. 2418–2428, Nov. 2015.
[15] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-based one-time password algorithm," Internet Engineering Task Force (IETF), RFC 6238, May 2011.
[16] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "HOTP: An HMAC-based one-time password algorithm," Internet Engineering Task Force (IETF), RFC 4226, Dec. 2005.

All rights reserved by *www.ijsrd.com*