IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 8, August 2026

DOI: 10.17148/IJARCCE.2026.15844

ARYA: An AI-Powered Campus Assistant Robot Integrating Real-Time Computer Vision, Conversational AI, and 3D DigitalTwin Simulation

Shankarprasad KS[1], Vignesh[2], Rakshitha KN[3], Sowjanya[4], Dr. Jeevitha B K[5]

Student, Department of Computer Science and Engineering, Vivekananda College of Engineering and Technology[1,2,3,4]

Associate Professor, Department of Computer Science and Engineering,

Vivekananda College of Engineering and Technology[5]

Abstract: Campus environments increasingly require interactive systems that can perceive surroundings, recognize individuals, converse naturally, and communicate robotic state. This paper presents ARYA, an AI-powered campus assistant robot platform integrating real-time computer vision, face recognition, conversational AI, and 3D digital-twin simulation. A Python/FastAPI backend runs YOLOv8-based object and person detection with an InsightFace recognition pipeline, while a React, TypeScript, and Three.js frontend provides a real-time robot avatar and browsable floor-plan digital twin. The conversational subsystem combines rule-based campus-information handling with a Gemini-based generative-AI fallback. Voice and typed commands control the simulated avatar and, where available, GPIO-based hardware. The paper describes the system architecture, perception and greeting pipelines, REST, WebSocket and MJPEG communication, and simulation subsystem. The resulting modular platform enables human-facing robot interactions to be developed and evaluated in simulation before physical deployment.

Index Terms: service robot, human–robot interaction, computer vision, YOLOv8, face recognition, InsightFace, conversational AI, digital twin, FastAPI, React Three Fiber.

I. INTRODUCTION

Educational campuses are large, socially dense environments where visitors, students, and staff routinely need directions, information, and a point of first contact. Traditional signage and static kiosks are non-interactive, while general-purpose voice assistants lack any awareness of physical space or the people in front of them. ARYA (AI-powered Robotic Assistant) was built to close this gap: a single software platform that lets a robot see who and what is around it, hold a natural conversation, and move — either physically or as a live 3D avatar — through a modeled version of its environment.

Prior to this work, the project existed as two disconnected halves: a desktop Python application ("AURA") capable of vision and speech processing but with no networkaccessible interface, and a React/Three.js frontend that was populated entirely with mock data. Neither could demonstrate an end-to-end interaction. The contribution of this paper is the design and implementation of the integration layer — a FastAPI backend exposing REST, WebSocket, and MJPEG interfaces, a headless (windowless) vision pipeline suitable for continuous background operation, and a rebuilt 3D simulation that mirrors a real building layout — that turns these two halves into one operable robot control center.

The remainder of this paper is organized as follows. Section II reviews related work in campus and service robotics. Section III describes the overall system architecture. Section IV details the computer-vision and face-recognition pipeline. Section V covers the conversational-AI subsystem. Section VI describes the 3D digital-twin simulation and movement control. Section VII discusses implementation and deployment considerations. Section VIII evaluates the system qualitatively and discusses limitations, and Section IX concludes with directions for future work.

II. RELATED WORK

Service and receptionist robots have been an active research area for over two decades, with early systems focused primarily on navigation and basic speech interfaces. More recent systems combine deep-learningbased perception with cloud or on-device conversational models, reflecting the broader shift from rule-based dialogue to large language model (LLM)-backed assistants. Real-time object detectors from the YOLO (You Only Look Once) family are widely used in

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 8, August 2026

DOI: 10.17148/IJARCCE.2026.15844

robotics because they offer a favorable accuracy-to-latency tradeoff for single-camera, single-pass detection, which is important when the same camera feed must simultaneously drive person detection, tracking, and a live video stream to a browser.

Face recognition in service-robot contexts typically relies on deep embedding models trained with margin-based losses to separate identities in embedding space; ARYA uses one such open-source pipeline (InsightFace) rather than training a bespoke recognizer, which is a common and pragmatic choice for small-gallery, campus-scale deployments where only a limited set of known individuals needs to be recognized. On the interaction side, digital-twin and simulation-first approaches — testing robot behavior in a rendered 3D model of the deployment site before committing to physical hardware — have become increasingly common in both industrial and social robotics, since they decouple software validation from hardware availability and reduce the risk of testing directly on people. ARYA follows this simulation-first philosophy: the 3D environment in the frontend is modeled on the actual building the robot will operate in, so navigation and interaction logic can be validated before any GPIO wiring is enabled.

III. SYSTEM ARCHITECTURE

ARYA is deliberately split into two independent processes — a Python backend and a TypeScript frontend — that communicate exclusively over HTTP-based protocols on localhost or a local network, with no shared build step. This separation allows the perception and hardware-facing code to run on the robot's own compute (or a Windows machine driving external hardware) while the interface can be served to any browser, including phones on the same network.

The frontend is a React 18 and TypeScript single-page application built with Vite, and exposes four principal views: a Home dashboard, a live-camera/detection view (labelled “VCET Live Demo” for the campus deployment), a 3D Simulation view built with Three.js and React Three Fiber, and a Talk with ARYA conversational panel. The backend is a Python FastAPI application (served with Uvicorn) organized into a Vision Service, a Chat Service, a robot command endpoint, and a greeting-management module, sitting on top of the preexisting desktop-application logic for movement and speech that the project reuses rather than re-implements.

Fig. 1. High-level ARYA system architecture: frontend, transport layer, and backend services. (Frontend — React + TypeScript + Vite + Three.js: Home | Live Camera | 3D Simulation | Talk with ARYA → REST / WebSocket / MJPEG stream → FastAPI Backend — Vision Service | Chat Service |

Robot Command API | Greeting Manager → YOLOv8 detection → InsightFace recognition → Gemini chat fallback

→ Movement / GPIO controller.)

IV. COMPUTER VISION AND FACE RECOGNITION PIPELINE

The vision subsystem runs as a headless background loop — it never opens a native OpenCV display window — so that its only observable output is the annotated MJPEG stream it serves to the browser at GET /api/vision/mjpeg. This design choice was necessary because the robot's camera must run continuously from the moment the backend starts, independent of whether any client is currently viewing the stream, and a blocking display window would be incompatible with that requirement.

Each captured frame is passed once through a bundled YOLOv8 nano model (yolov8n.pt) with no class filter, so a single forward pass simultaneously yields general object detections and person detections (COCO class 0); this avoids running two separate models per frame and keeps per-frame latency low enough for near-real-time streaming. Every detected person bounding box is then cropped and passed to an InsightFace recognition pipeline, which compares the extracted face embedding against a gallery of known individuals stored in a known_faces/ directory (one sub-folder of reference images per person).

A. Presence and Greeting State Machine

Because the same person remains in frame across many consecutive video frames, naively acting on every detection would cause ARYA to greet the same individual repeatedly. To prevent this, each tracked identity is advanced through an explicit state machine: DETECTED
→ RECOGNIZED → GREETED → PRESENT →
LEFT. A person who has already been greeted is not regreeted while PRESENT; if a second, unrecognized or newly detected person appears while a greeting is already in progress, they are placed in a greeting queue and processed once the current greeting completes. A person is transitioned to LEFT, and their state cleared, once they are no longer detected in the frame for a sufficient number of consecutive cycles.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 8, August 2026

DOI: 10.17148/IJARCCE.2026.15844

B. Real-Time Delivery

Detection results (object list, recognized-person list, and a rolling event log) are pushed to connected clients over a WebSocket endpoint (WS /ws/vision), while the annotated video itself is delivered separately as an MJPEG multipart stream. Splitting structured data from pixel data in this way lets the frontend update text-based UI elements (detected-object counts, recognized-person names) independently of, and at a different rate from, the video render loop.

V. CONVERSATIONAL AI SUBSYSTEM

The Chat Service reuses the existing wake-word, introduction, college-information, and campus-locationmatching logic from the original desktop application's configuration object, so that deterministic, previously validated responses (e.g., “Where is the library?”) are not re-implemented or duplicated by hand — they are read directly from the same configuration the desktop app used. A bare wake word (“Arya”) is handled entirely by this rule-based layer and receives a fixed acknowledgement (“Yes, how can I help you today?”).

For open-ended questions that fall outside this rule-based coverage, the Chat Service falls back to a generative-AI model (Google Gemini) via a text completion call, configured through a GEMINI_API_KEY environment variable that is read only on the backend and never exposed to the frontend or committed to version control. This hybrid design — deterministic handling for known campus-specific intents, generative fallback for everything else — keeps latency and cost low for the common case while still allowing ARYA to answer arbitrary questions.

Movement instructions can also be issued conversationally (e.g., typing or saying “Arya, move front” inside the chat panel), which the Chat Service recognizes as a command rather than a question and forwards to the same movement-handling code path used by the dedicated command interface, ensuring a single consistent implementation of “what a movement instruction means” regardless of which UI surface it arrived from.

VI. 3D DIGITAL-TWIN SIMULATION AND MOVEMENT CONTROL

The Simulation view renders an interactive 3D reconstruction of the robot's actual deployment building — a reception lobby opening into a corridor with doors, signage, lockers, and a window — built with Three.js and React Three Fiber, rather than an abstract grid. ARYA's avatar spawns at the door and can be driven through the corridor in three modes:

- Controls — a press-and-hold on-screen directional pad or WASD/arrow-key input, with continuous acceleration and deceleration and real collision detection against the modeled walls;
- Voice — spoken commands (“Arya, move forward”, “turn left”, “stop”) recognized in-browser and mapped to the same continuous-drive logic as the on-screen pad;
- Automatic — either a one-click Corridor Patrol, in which ARYA autonomously drives door → corridor end → door with no user input, or a named-destination auto-navigate mode.

A live floor-plan mini-map component renders ARYA's current position, heading, and a trail of recently travelled positions in every drive mode, giving an at-a-glance topdown view alongside the first-person 3D render. Earlier iterations of the movement code applied a fixed, discrete nudge (approximately 45° of rotation or 0.6 m of translation) per command and then stopped, which made spoken or typed commands feel unresponsive; this was replaced with continuous, physically smoothed motion that persists until an explicit stop or an overriding command is received.

The 3D avatar's position, heading, battery level, and travel trail are maintained entirely client-side as a software simulation, since the project does not assume a physical chassis is always present. Movement commands issued through any interface (on-screen pad, voice, or chat) are nonetheless also forwarded to the backend's MovementController and GestureController, which are reused unmodified from the original desktop application and will drive real GPIO-connected motors and servos when the backend is run on the robot's own hardware with GPIO output explicitly enabled; on any other machine, the same code path is a no-op, allowing the entire system to be developed, demonstrated, and evaluated without physical hardware.

VII. IMPLEMENTATION AND DEPLOYMENT

The backend and frontend are started as two independent processes and communicate over a configurable base URL (VITE_API_BASE_URL), defaulting to localhost:8000; running the frontend with its network-exposed dev flag additionally allows the interface to be opened from a phone on the same campus network, which was used to validate the mobile layout described below.

Two deployment concerns specific to a campus setting influenced implementation choices. First, station and location

DOI: 10.17148/IJARCCE.2026.15844

labels in the 3D scene were originally rendered using a text component that fetched a web font at runtime; because campus networks are frequently locked down or the robot may be demonstrated offline, this was replaced with a self-contained, canvas-drawn label that requires no network access to render. Second, the frontend's WebGL canvas could permanently lose its rendering context under memory pressure — a failure mode that disproportionately affects phones and low-end integrated GPUs rather than desktop demonstration machines — and the simulation view was made to detect and automatically recover from this condition rather than requiring a manual page reload.

For narrow viewports (below approximately 900 px), the sidebar navigation used on desktop is hidden and replaced with a bottom tab bar, and the Simulation view's drivemode tabs and mini-map are laid out to fit a single phone screen, so the same build serves both a desktop demonstration and a hand-held walkthrough without a separate mobile build.

VIII. DISCUSSION AND LIMITATIONS

The system successfully demonstrates end-to-end integration across four previously siloed capabilities, and the simulation-first approach allows the interaction logic — greeting management, conversational flow, and navigation — to be fully exercised without physical hardware, which is valuable in an academic setting where robot chassis availability is often limited or shared across teams. The reuse of validated desktop-application logic for movement, gesture, and campus-information handling, rather than reimplementing it, reduced the risk of introducing new bugs into already-working behavior while the integration layer was built.

The current implementation has known limitations. Face recognition accuracy depends directly on the size and image quality of each individual's reference gallery in known_faces/, and the system currently performs no explicit liveness or anti-spoofing check, which would be necessary before any access-control use case beyond greeting. The single YOLOv8-nano pass optimizes for latency over accuracy relative to larger YOLO variants, which is an appropriate trade-off for a live-streamed greeting robot but would need revisiting for tasks requiring finer-grained detection. Physical odometry is not yet implemented — the 3D avatar's telemetry is simulated even when GPIO output is enabled — so on real hardware there is currently no closed-loop feedback confirming that a commanded movement was physically executed as intended.

IX. CONCLUSION AND FUTURE WORK

This paper presented ARYA, an integrated software platform for a campus assistant robot that combines a headless YOLOv8/InsightFace vision pipeline, a hybrid rule-based and generative-AI conversational engine, and a Three.js-based 3D digital twin, connected through a FastAPI backend over REST, WebSocket, and MJPEG interfaces. The architecture treats physical hardware as optional rather than assumed, allowing the full interaction experience to be developed, demonstrated, and evaluated in simulation before any deployment to a physical chassis.

Future work includes closing the odometry loop with real sensor feedback when physical hardware is available, adding liveness detection to the face-recognition pipeline, extending the digital twin to additional floors and buildings, and evaluating recognition accuracy and conversational response quality quantitatively against a labelled campus dataset rather than the qualitative validation performed in this iteration.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788.

[2] G. Jocher, A. Chaurasia, and J. Qiu, Ultralytics YOLOv8, Ultralytics, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[3] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “ArcFace: Additive Angular Margin Loss for Deep Face Recognition,” in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4690–4699.

[4] J. Guo, J. Deng, A. Lattas, and S. Zafeiriou, InsightFace: 2D and 3D Face Analysis Project, 2021. [Online]. Available:https://github.com/deepinsight/insightface

[5] S. Ramírez, FastAPI: Modern, Fast (HighPerformance) Web Framework for Building APIs with Python, 2023. [Online]. Available: https://fastapi.tiangolo.com

[6] Google DeepMind, “Gemini: A Family of Highly Capable Multimodal Models,” Technical Report, Google, 2023.

[7] Poly Haven / Three.js contributors, Three.js: JavaScript 3D Library, and React Three Fiber, a React renderer for Three.js. [Online]. Available: https://threejs.org, https://docs.pmnd.rs/react-threefiber

[8] T. Kanda, T. Hirano, D. Eaton, and H. Ishiguro, “Interactive Robots as Social Partners and Peer Tutors for Children: A Field Trial,” Human–Computer Interaction, vol. 19, no. 1–2, pp. 61–84, 2004.

[9] R. Gockley et al., “Designing Robots for Long-Term Social Interaction,” in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), 2005, pp. 1338–1343.

