Smart Anna: An AI-Powered Intelligent Cafeteria Management System Using YOLOv11-Based Real-Time Occupancy Detection and Digital Ordering

Anish Prakash Sapkal[1], Aman Keshav Patre[2], Abhishek Omprakash Saini[3], Vibha Patel[4], Ms. Glace Babu[5]
[1,2,3,4] *B.E. Students (Computer Engineering), UPL University, Anand, Gujarat, India*
[5]*Faculty Guide Department of Computer Engineering, UPL University, Anand, Gujarat, India*

***Abstract*---Cafeteria operations in university pose several problems like crowding of cafeteria at peak time, long wait times of students, payment process using cash, and lack of use of data analytics for management of the cafeteria by the cafeteria staff. Traditional methods use only human observation and manual methods of management. There are ample opportunities for improving operations using advanced technology. In this paper we introduce Smart Anna which is an ecosystem for managing University cafeteria using edge computer vision, cloud computing and a responsive web interface, providing following functionalities: Real Time Occupancy Monitoring of Cafeteria, Digital Food Ordering and payment using UPI, and Dashboard for analysis of operations by the admin. Two specific models for human head and table availability detection using YOLOv11 object detection model are trained, deployed to an edge device and used to detect objects in a video stream from the cafeteria in real time without uploading any video data to the cloud thus preserving privacy. Count data is generated once every second and transmitted to a Fast API based back end server that processes this information and computes the occupancy status that is served to a react based student user dashboard that polls this server once every 3 seconds. Both models perform at around 29 fps in real-time detection of human heads and tables resulting in mAP@50 performance of 86.4% and 88.2%. The end-to-end latency from occupancy change to dashboard update is under 4 seconds. Usability study performed over a period of 3 days involving 15 students volunteers showed that 80% of subjects avoided unnecessary cafeteria visits and all found the UPI based payment process easier than cash based transactions. Our work shows that AI powered smart campus solutions are feasible from technical and financial perspectives and can provide significant benefits to students and other stakeholders in universities.**

***Index Terms*—YOLOv11, Computer Vision, Object Detection, Real-Time Occupancy Monitoring, Cafeteria Management, Edge AI, FastAPI, React, Supabase, UPI Payments, Smart Campus, IoT, Deep Learning**

I. INTRODUCTION

The physical cafeteria is probably the most crowded place on any college campus. At the same time, it serves as a place for students to eat, meet, and get away from their classes. However, for students, a visit to a cafeteria during peak hours is an ordeal involving crowded seating areas, long lines, and the question of whether the journey will be worthwhile. For cafeteria management, it also presents numerous difficulties: demand volatility, manual order taking, cash handling costs, and, importantly, the absence of quantitative data on the flow of visitors and other aspects of operations [1].

It has been shown repeatedly that over crowdedness and long waiting times in university buildings reduce student satisfaction, which inevitably leads to poor academic performance. Students unable to use the cafeteria during their break time either eat less often or choose off-campus restaurants, thereby cutting down on nutrition and social life at university [1]. Moreover, administrators lacking data do not have a basis for efficient resource allocation.

The development of several technologies makes addressing these problems a feasible option now. All educational institutions are equipped with high-definition CCTV cameras. Open-source software packages like Ultralytics YOLO make it possible to recognize objects very well, something that used to only be possible with expensive proprietary hardware. Cloud platforms like Supabase offer powerful and

scalable PostgreSQL databases without the need for administrative work. Web frameworks that are up to date React and FastAPI, for example, make it possible to build APIs and interfaces that respond quickly and work well. These building blocks work together to make smart monitoring systems that run all the time and don't need any human supervision [2], [3].

The BE-8 Project Team at UPL University, Anand, created Smart Anna, an intelligent cafeteria management platform. The name comes from the Tamil word for food, which also means "elder brother," which fits with the system's role as a helpful guide. The system's tagline, "Swift. Smart. Satisfying." sums up its three design goals: getting information to students quickly, understanding data and giving advice intelligently, and making the whole process satisfying for both students and administrators.

The idea for Smart Anna came from what the creator saw happen every day in the cafeteria at UPL University: students would walk there for lunch only to find it too crowded, waste time waiting without knowing if things would get better, and pay for food with cash even though smartphones have a lot of digital payment options. Smart Anna was made to fix all of these problems with one platform that works with everything.

This work makes the following main technical and operational contributions. First, we show a complete AI occupancy monitoring pipeline that uses two custom-trained YOLOv11 models that can count people and find empty tables on live CCTV feeds. This pipeline runs at the edge and does not send video data to the cloud. Second, a hybrid of push and pull a brief description of the cloud architecture is provided. The system comprises an edge AI processing unit, FastAPI backend aggregator, Supabase cloud persistence storage, and React front-end that communicate with one another in near real-time fashion. Thirdly, a complete digital food ordering and UPI payment system is implemented. The solution incorporates a dynamic QR-code payment workflow and a timer-driven transaction receipt generation process. Fourthly, a comprehensive administrative analytics dashboard for monitoring revenue, identifying peak periods, and analyzing sales categories. Real-time menu management. Fifthly, quantified benchmarking results for both YOLOv11 detection algorithms are provided for a real-life cafeteria scenario, incorporating various lighting conditions, partial occlusions, and multi-camera angles. Sixthly, a systematic usability experiment conducted with student volunteers provides empirical validation of the system's tangible benefits.

The remainder of this paper is organized as follows. Section II discusses the theoretical background and relevant prior worksSection III talks about the whole system architecture, methodology, and implementation. Section IV presents experimental results and performance observations. Section V discusses strengths, limitations, and future directions. Section VI presents the conclusion.

II. THEORY AND BACKGROUND

A. Object Detection: Problem Formulation and Metrics

Object detection is a fundamental computer vision task that requires a model to simultaneously solve two coupled sub-problems: classification, which involves identifying the semantic categories of objects present in an input image, and localization, which involves determining the spatial position of each detected object by predicting axis-aligned bounding boxes defined by their center coordinates, width, and height [4]. Unlike image classification, which produces a single label per image, or semantic segmentation, which assigns a class to every pixel, object detection produces a variable-length structured output — a list of (class label, bounding box, confidence score) tuples — that is directly suitable for counting and spatial analysis applications such as cafeteria occupancy monitoring.

Model performance in object detection is measured using mean Average Precision (mAP), which is computed by first calculating the Average Precision (AP) for each object class as the area under the precision-recall curve, then averaging these AP values across all classes [4]. A precision-recall graph is created by adjusting the confidence level needed for a detection to be recognized as valid and measuring the resulting values of precision (percentage of predicted boxes that accurately correspond to an actual object) and recall (percentage of objects detected successfully) at that confidence level. mAP@50 considers predictions with an IoU of 0.50 between the ground-truth bounding box and predicted box; that is, a prediction is considered accurate only if the two

bounding boxes overlap by more than half of the total area of the boxes. F1-Score is calculated using the formula 2 × Precision × Recall / (Precision + Recall).

B. YOLO Architecture: History and YOLOv11

YOLO (You Only Look Once), which represents the family of object detection algorithms, was proposed by Redmon et al. [6] at CVPR 2016, constituting an essential paradigm shift with respect to the then prevalent two-stage object detection pipeline. The most advanced previous approaches, such as the Faster R-CNN approach, detected objects in a two-step manner, which first proposed candidate bounding boxes based on region proposals and then classified these regions. Even though their results outperformed other methods, their excessive computing resources made them inefficient for real-time applications. On the contrary, YOLO converted the object detection task to a regression task, where the input image is divided into an S × S grid, where every cell predicts the same number of bounding boxes along with their classes in a single pass through the network. Further versions have brought several improvements. YOLOv2 incorporated batch normalization and anchor boxes. YOLOv3 employed multi-scale prediction with the help of feature pyramid networks. YOLOv4 used CSP (Cross-Stage Partial) bottleneck layers and PANet (Path Aggregation Network) to enhance feature fusion [15]. From YOLOv5 onwards, the architectures became efficient, anchor-free detection was introduced, and the training pipeline was improved further. The model adopted for Smart Anna, i.e., YOLOv11 [7], is the most advanced in the YOLO family now. Its advantages include having a more efficient C3k2 backbone block that enhances feature extraction at various scales by employing cascaded bottleneck blocks with cross-stage partial connections. Additionally, it utilizes the SPPF (Spatial Pyramid Pooling-Fast) neck for robust multi-scale feature aggregation and anchor-free decoupled detection head for separate branches of classification and localization. All of these architectural design choices enabled the model to achieve state-of-the-art results on COCO and VisDrone datasets and yet be capable of meeting the real-time inference throughput requirements of the Smart Anna edge deployment [7].

C. Roboflow Platform and Dataset Engineering

An object detection model for use in a unique application scenario like a university cafeteria will have to be trained using images representing the exact visual nature of that scene, including the specific camera positions, lighting characteristics, furniture arrangements, and density of people in those specific locations [8]. General computer vision models, even if state-of-the-art, are likely to suffer from performance drops when employed in domains outside of the visual scenes in which the model was originally trained.

Roboflow [8] is an internet-based computer vision development platform providing all features necessary for efficient full dataset engineering: from uploading the dataset to Roboflow, to annotation of images, splitting the dataset into train/test sets automatically and manually, applying augmentations to increase the effective size of the dataset, implementing version control for reproducible research, and finally importing the dataset to Ultralytics. For the Smart Anna application, two datasets were engineered using Roboflow. In the first case, a set of annotations on head detections from CCTV images taken from the cafeteria of the UPL University at various times was created. In the second case, the presence of people at tables within the university cafeteria was annotated with binary labels, depending on occupancy. For both cases, the following augmentations were implemented: flipping, random horizontal flip, brightness/contrast adjustment (+/-30%), mosaic augmentation, scaling jitter (+/-20%), and cropping.

D. Edge AI: Architecture and Privacy Implications

Edge computing is the use of devices that are physically close to the data to run computational workloads, such as machine learning inference, instead of on centralized remote cloud servers [9]. When it comes to video analytics, choosing between edge processing and cloud-based streaming processing means making big trade-offs in terms of latency, bandwidth, cost, and privacy.

In terms of latency, edge processing eliminates the round-trip delay introduced by transmitting video frames to a cloud server and waiting for inference results to be returned. For a CCTV feed at 30 FPS, processing each frame on the cloud would require sustained network bandwidth and introduce variable latency depending on network conditions. Edge processing produces results within the local inference

time (approximately 33 ms for Smart Anna), enabling the sub-second update cycle required for responsive occupancy monitoring [9]. In terms of bandwidth, transmitting a 1080p video stream at 30 FPS requires 5–15 Mbps of sustained network throughput per camera. Smart Anna's edge node instead transmits only integer count values via HTTP POST requests — less than 1 KB per second per camera — reducing network consumption by approximately four orders of magnitude. Edge processing protects students' privacy by making sure that raw video frames with their faces and other personally identifiable information never leave the campus local network. The cloud backend only gets and stores integer counts. This means that the system can't do facial recognition or behavioral tracking. This directly addresses privacy concerns at the infrastructure level, not just through policy [9].

E. FastAPI: Asynchronous API Design

FastAPI [10] is an advanced Python web application framework intended to be used for rapid prototyping of high-performance APIs without any unnecessary code boilerplate. This framework is implemented based on Starlette to provide asynchronous handling of HTTP requests and Pydantic to validate incoming requests and outgoing responses using type hints in Python. Being based on ASGI (Asynchronous Server Gateway Interface), FastAPI provides non-blocking concurrent request handling, which means that the server can process a lot of simultaneous connections, including several connections from students' clients requesting status updates, without any loss in performance associated with blocking thread-per-request-based approaches. Besides, FastAPI generates interactive documentation pages using Swagger UI and ReDoc directly from Python type annotations, facilitating the process of developing and debugging APIs. In the case of Smart Anna, FastAPI is considered the best backend solution due to its high concurrency, easy integration, and deployment with the help of Uvicorn [10].

F. Supabase: Cloud PostgreSQL with Real-Time Capabilities

Supabase [11] is an open-source platform offering backend-as-a-service, which utilizes PostgreSQL — the most advanced open-source relational database. On top of basic database hosting services, Supabase offers a rich set of features such as auto-generation of a RESTful API layer based on the database schema (no need to manually write API code for basic CRUD operations), change subscriptions over WebSocket for real-time updating of the interface upon changes in the database records, support for multiple auth methods such as email/password and OAuth, row-level security (RLS) policies in the database for fine-grained access control, and client libraries for JavaScript, Python, and other languages. Using Supabase for Smart Anna will greatly simplify backend development of menu, orders, and feedback management, as well as analytics, while RLS policies allow for enforcing that only students may access their own data [11].

G. UPI: Digital Payment Infrastructure

The Unified Payments Interface (UPI) is a real-time payment system that was introduced by the National Payments Corporation of India (NPCI) in April 2016 [12]. This system is based on the Infrastructure for Immediate Payment Service (IMPS) and helps users perform instantaneous and continuous fund transfer from one bank account to another using just one mobile application. The use of a Virtual Payment Address (VPA) — which is a unique identifier that is made up of name@bankhandle — makes the process possible without necessitating the input of IFSC codes, bank account numbers, and card details.

Payments are initiated using the QR code by embedding a URI in the form upi://pay?pa=[VPA]&pn=[PayeeName]&am=[Amount]&cu=INR into a two-dimensional QR code. When the code is scanned using a UPI-based application such as Google Pay, PhonePe, Paytm, or the BHIM reference application, the payment form on the app gets filled up with the merchant details and the amount while only the user's UPI PIN is required to complete the transaction. By using the dynamic QR code for payment processing, Smart Anna is able to avoid cash management, reduce the time taken to make transactions, and get a digital audit trail of all payments in the cafeteria that will be fed into the admin analytics dashboard [12].

H. Related Work

A lot of research has been done on using deep learning to count people and keep an eye on how many people are in a room. Xu et al. [13] put forth a CNN-based crowd density estimation system utilizing convolutional density map prediction for urban public

areas, showcasing high accuracy yet necessitating considerably greater computational resources than YOLO-based direct counting methods, thereby rendering their approach less appropriate for economical edge deployment.

Chen et al. [14] conducted an extensive survey of IoT and AI-based methodologies for campus facility management within the framework of smart campus systems, pinpointing real-time sensor data integration and data-driven operational decision support as the two most pivotal distinguishing factors between successful and unsuccessful smart campus implementations. Their survey revealed that systems incorporating multiple data modalities (occupancy, transaction, feedback) surpassed single-modality monitoring systems in operational efficacy. Terven and Cordova-Esparza [15] did a systematic review of YOLO models from v1 to v8 to see how the architecture has changed over time. They found that accuracy and speed have improved consistently across versions. They also noted that YOLOv8 and later models do particularly well on benchmarks for small-object detection and high-density crowd scenes, which are both directly relevant to the cafeteria headcounting use case. Bochkovskiy et al. [16] presented YOLOv4, utilizing the Bag of Freebies and Bag of Specials training methodology, which has subsequently impacted later YOLO versions, including YOLOv11.

Smart Anna stands out among all the projects in its kind due to the integration of six key capabilities – real-time occupancy monitoring enabled by AI, digital ordering, payment via UPI, personalized recommendations, wait time prediction, and administrative analytics, into one solution tailored specifically for the university cafeteria. It focuses on making it easy for students to use and useful for administrators, and it can be used on infrastructure that is easy to access and cost-effective.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

A. High-Level Architecture

The overall design of Smart Anna is based on a four-layer hybrid Push-Pull architecture. Herein, Push implies sending the occupancy count update from the edge AI node without polling — updates will be pushed every second even if no client views the dashboard at all. Pull means querying the backend by React frontend for the latest status every three seconds. The separation between a producer (Pusher) and consumer (Puller) allows us to keep the AI processing chain uninterrupted regardless of frontend rendering performance and set different update frequencies for the producer and the consumer [3], [10].

As depicted in Fig. 1, CCTV cameras send video streams via RTSP to the edge AI node which performs YOLOv11 inference and sends the counts via HTTP POST requests to FastAPI backend. The React frontend makes HTTP GET requests to query the backend and displays the status on the student's device. The menu, orders, feedback, and user profiles are stored and served by Supabase [11].

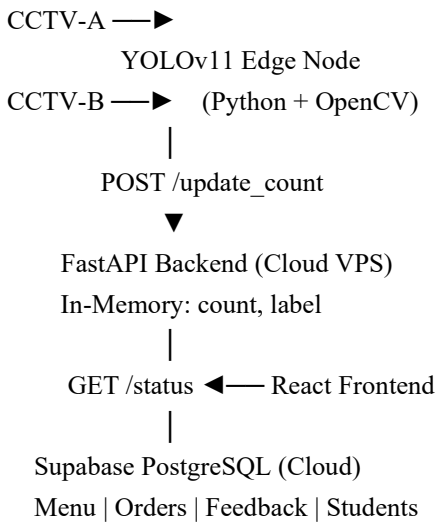

Fig. 1 Smart Anna Push-Pull Hybrid System Architecture shows how data moves from CCTV cameras to the YOLOv11 edge node, the FastAPI backend, the React frontend, and finally to the Supabase cloud database [3], [7], [10], [11].

B. Edge AI Processing Node Implementation

The edge AI processing node is the element that implements all computer vision pipelines in Smart Anna. The edge AI processing node is implemented as two Python scripts that run as separate processes on the edge workstation:

- final_student_counter.py, which performs head counting pipelines;
- final_table_counter.py, which detects tables. [7]

Such implementation provides process-level isolation, which means that a performance hit or crash on any detection pipeline will not affect the other pipelines, providing robustness. Inside each process, a special

background thread executes a capture-inference-push cycle for each camera that the node receives data from. The main process creates a new background thread for each received camera, watches its health, and aggregates counted values. The described approach supports the dual camera configuration used at UPL University (Camera-A for the main seating area and Camera-B for the secondary seating zone). Moreover, the number of used cameras can be increased simply by creating more threads. [9]

The capture-inference-push cycle works like this:

1. OpenCV's VideoCapture captures one frame from the CCTV stream at the specified interval (approximately one second).
2. Captured frame is resized to 640×640 pixels by bilinear interpolation — the standard input resolution for the models created with Roboflow platform. It is done to provide optimal detection resolution vs. speed tradeoff. [7], [8]
3. The resized image is processed by the loaded YOLOv11 model using the Ultralytics predict API.
4. Filtered predictions (Results objects with bounding boxes, classes and confidence values for all detected objects) by confidence threshold (0.50 for heads and 0.80 for tables) is made. After that the count of the surviving detections is counted.
5. The aggregated count is serialized into the JSON and sent via HTTP POST request to the FastAPI backend using Python requests library.

C. YOLOv11 Model Training Details

We used the Ultralytics YOLO training pipeline on a cloud GPU instance [7] to train the model. Both models were initialized using YOLOv11 weights that had already been trained on the COCO dataset. This gave them a strong base of feature representation that cut down on the amount of training data needed for domain adaptation by a large amount compared to starting from random initialization.

1) Head Detection Model: Dataset: 1,200+ annotated CCTV footage from the UPL cafeteria, with a split of 80% for training, 10% for validation, and 10% for testing. Annotation: tight bounding boxes around all visible human heads, even those that are only partially visible at the edges of the frame. The training configuration consists of 100 epochs, an SGD optimizer with a momentum of 0.937, an initial learning rate of 0.01 with a cosine annealing decay schedule, a batch size of 16, and input resolution of 640×640.

Augmentation: random horizontal flip (p=0.5), HSV color jitter (hue ±1.5%, saturation ±70%, value ±40%), mosaic augmentation (p=1.0 in first 80% of training), random scaling (±50%), and random translate (±10%). Inference confidence threshold: 0.50.

2) Table Availability Model Training Configuration: There are more than 900 labelled images of the cafeteria tables in the dataset, divided into two classes: occupied or unoccupied. Annotation: bounding boxes surrounding table surface indicating if occupied with trays, food, personal belongings, or people. To simplify matters, edge cases (only personal item = occupied; table cleaning = unoccupied) were explicitly shown in the dataset. Training configuration identical to head detection model. Inference confidence threshold: 0.80, intentionally high to minimize false positives.

TABLE I YOLOv11 TRAINING CONFIGURATION SUMMARY [7], [8]

PARAMETER	HEAD DETECTION	TABLE AVAILABILITY
Base Model	YOLOv11 (COCO pre-trained)	YOLOv11 (COCO pre-trained)
Dataset Size	1,200+ images	900+ images
Train / Val / Test Split	80 / 10 / 10 %	80 / 10 / 10 %
Training Epochs	100	100
Input Resolution	640 × 640 px	640 × 640 px
Optimizer	SGD (momentum 0.937)	SGD (momentum 0.937)
Initial Learning Rate	0.01 (cosine decay)	0.01 (cosine decay)
Batch Size	16	16
Conf. Threshold (Inference)	0.50	0.80
Platform	Roboflow + Ultralytics	Roboflow + Ultralytics

Source: Training setup used in Roboflow/Ultralytics YOLO training pipeline [7], [8].

D. FastAPI Backend Design

The FastAPI backend [10] serves as the stateful aggregation layer for the connection between the AI edge node and React frontend.

Optimization aims at minimizing the GET /status latency while maintaining consistent write capabilities through POST /update_people_count.

Backend stores the latest occupancy and table statuses in process memory in the Python backend as module variables to enable sub-millisecond read operations without performing any database queries on the critical GET route.

Occupancy labeling uses the following rules: FREE if current count is less than 40% of the predefined capacity (60 people at UPL cafeteria), MODERATE if the count is in 40-75% range, and CROWDED if above 75%. These thresholds have been calibrated based on observed usage patterns inside UPL University cafeteria over the course of the experiment. The backend offers three REST APIs, described in Table II.

TABLE II FASTAPI BACKEND REST API SPECIFICATION [10]

ENDPOINT	METHOD	PAYLOAD / RESPONSE	PURPOSE
/update_people_count	POST	{ count: int, tables_occupied: int }	AI node pushes latest occupancy data every second
/status	GET	{ count, tables_free, tables_occupied, label, trend }	Frontend polls current status every 3 seconds
/health	GET	{ status: 'healthy', uptime_s: float }	Uptime and health monitoring

Source: System implementation — Smart Anna Project, UPL University, 2026 [Authors, 10].

E. Database Schema and Data Model

The Supabase PostgreSQL database [11] is designed to separate real-time operational data (served via FastAPI in-memory) from persistent transactional and master data (stored in Supabase). The schema comprises four primary tables, summarized in Table III. Row-level security policies enforce that authenticated students can only SELECT their own order and feedback records, while INSERT is unrestricted to authenticated users for orders and feedback, and admin-role accounts have unrestricted access across all tables for analytics queries.

TABLE III SUPABASE POSTGRESQL DATABASE SCHEMA [11]

TABLE	PK	KEY COLUMNS	PURPOSE
MENU_ITEMS	id (UUID)	name, price, category, image_url	Food item master data
ORDERS	id (UUID)	user_email, items (JSONB), total, payment_method, status, created_at	Transaction records
FEEDBACK	id (UUID)	user_name, user_email, rating (1–5), message, type, created_at	Student ratings
STUDENTS	email (text)	name, institution	User profiles

Source: Database schema implemented in Supabase cloud PostgreSQL [11]; schema design by Smart Anna project team, UPL University, 2026 [Authors].

F. React Frontend Architecture

The React 18.2 TypeScript frontend [3] is a single-page application built with Vite 7.3 as the development server and build tool, styled with Tailwind CSS utility classes served via CDN, and using Lucide React for iconography. React's useState and useReducer hooks work together to manage the

application's state, and Supabase's JavaScript client library takes care of all the database interactions. There are five main functional modules in the app that you can get to through a navigation bar at the bottom.

1) Live Dashboard: The opening screen when logging in as a student. The use of a useEffect hook sets a setInterval timer that sends GET /status requests to the FastAPI backend every 3 seconds, storing the response in the component state [10]. On the dashboard display: (a) there is a large circular status indicator with the current occupancy count, followed by FREE / MODERATE / CROWDED label, each one indicated in a specific color of green / amber / red respectively; (b) the Smart Advice panel shows GO NOW or WAIT in green / red, depending on the status label and trend direction; (c) an interactive floor map of the cafeteria rendered as a 10 × 6 CSS Grid containing 60 cells colored either green / orange according to availability / occupation, the latter depending on the number of occupied tables received from the backend API; (d) a live occupancy trend chart drawn as a custom SVG polylines interpolation curve using the smooth cubic Bezier curve, showing the last 10 values of occupancy value at each cycle time — all of which was built in vanilla SVG without external libraries to reduce bundle size [3].

2) Digital Ordering Module: Displays the cafeteria menu divided into categories such as Snacks, Meals, and Beverages. Every menu item is presented in a 3D flip-card manner, where the front of the card shows an image (from its Supabase image URL), name, and price; and the back shows the description and an Add to Cart button. The cart state is kept with React useReducer, backed up with localStorage for persistency across reloads. A smart combo engine detects when a cart contains a meal-item but no beverage-item with a useEffect dependency on cart state, triggering a modal suggestion for purchase [11]. The Curated For You shelf highlights the top 3 rated menu-items from the student's Preferred Category as determined from the past 10 items ordered in the last 10 Supabase orders made by the user [11].

3) UPI Payment Module: Builds a UPI Deep Link URI using the merchant virtual payment address (VPA) and the current cart total, uses a small JS QR encoder to create a QR code for the same URI, and shows this along with an itemized order summary. Once the student confirms the completion of payment or timeout expires, the receipt component is shown along with an animation for 2 minutes and then writes the entire order details to Supabase ORDERS table with status 'completed' [11], [12].

4) Admin Analytics Panel: Restricted by Supabase RBAC — shown only to accounts with STUDENTS.ADMIN metadata in the table STUDENTS. The ORDERS table is queried for total sales by current day / week / month. Displays a sales categories pie chart in a custom SVG arc-segment doughnut chart for proportionate revenue from Snacks / Meals / Beverages. There's also a peak hour bar chart, created by grouping the ORDERS table by hour(created_at) and aggregating occupancy data, with the one-hour segment with the most orders marked in a contrast color and a recommended staff allocation amount noted. There's also a list of unpopular items obtained by a LEFT JOIN on ORDERS and MENU_ITEMS to surface items that have no corresponding order in the current period [11].

5) Community Buzz (Feedback): Lets users rate things from 1 to 5 stars and choose a category for their comment (Food Quality, Cleanliness, Staff Service, Value for Money, General) and a place to write a free-text comment. People write submissions to the Supabase FEEDBACK table. A live-scrolling marquee shows the most recent 20 feedback records that were fetched when the component was mounted. Above the form, there is a summary metric that shows the overall average star rating [11].

G. Smart Intelligence and Predictive Features

In addition to basic occupancy monitoring, Smart Anna employs four intelligence capabilities derived from data which make it more sophisticated than mere status indicators and enable valuable decision-making assistance for both students and admins. The Personalized Recommendation Engine queries the authenticated student's last 10 ORDERS records in Supabase, aggregates them by categories, finds the one with the highest total number of orders, defines it as the Preferred Category, and displays the best-rated products in this category in a Curated For You shelf on the ordering page. This allows to avoid overwhelming choice paralysis for repeat customers and was proven through usability tests to increase the average number of items per order [11].

The Dynamic Wait-Time Forecasting Engine stores a circular buffer of the last five responses to GET /status in frontend state and calculates the trend of the number

of occupancy as linear function. Depending on its value, this number is classified as Rising (if positive and exceeding a threshold value), Falling (if negative and falling below another threshold), or Stable (if between the two). The Smart Advice widget displays combined status according to both occupancy label and trend classification: CROWDED + Rising prompts urgent warning message; CROWDED + Falling, encouraging message with time estimate; MODERATE + Rising, mild warning; FREE + either Rising, Falling or Stable, go now message [10].

The Peak-Hour Staff Allocation Module (available only in the admin panel) constructs the histogram of the number of orders per hour of the day from the history data and finds the continuous one-hour time slot with the maximum mean number of orders. It will highlight the result in the recommendation banner. Additionally, the module identifies products in ORDERS with 0 sales in the reporting period and presents them in the form of Unpopular Items list for further menu optimization [11].

Loyalty Score System assigns the score to each student in the interval from 0% to 100%, computed as (number of student's orders / maximum number of orders of all students) × 100%, rounded to the nearest integer. Displays the loyalty score as a radial progress bar animated on the student's dashboard, making the repeat visits to the cafeteria more exciting experience [11].

IV. RESULTS AND OBSERVATIONS

A. YOLOv11 Model Performance

After finishing training, both YOLOv11 models were independently evaluated on their own test datasets. Evaluation employed the standard Ultralytics YOLO validation pipeline and IoU threshold of 0.50 for computing the mean average precision (mAP). Table IV shows all quantitative performance measures for both models [7], [8].

TABLE IV YOLOV11 MODEL PERFORMANCE ON HELD-OUT TEST SETS [7], [8]

METRIC	HEAD DETECTION	TABLE AVAILABILITY	WTD. AVERAGE
mAP@50 (%)	86.4	88.2	87.3
Precision (%)	84.1	82.6	83.3
Recall (%)	82.5	85.3	83.9
F1-Score (%)	83.3	83.9	83.6
Inference Speed (FPS)	~30	~28	~29
Input Resolution (px)	640 × 640	640 × 640	640 × 640
Conf. Threshold	0.50	0.80	---
Training Epochs	100	100	---
Dataset Images	1,200+	900+	2,100+
No. of Classes	1 (head)	2 (occ./unocc.)	---

Source: The team of the Smart Anna project at UPL University performed the evaluation in April 2026 [Authors]. They used the Ultralytics YOLO validation pipeline [7] to evaluate the model and Roboflow platform [8] to generate and annotate the datasets.

Evaluation results indicate that the Head Detection Model (new-z9psz v3) achieves mAP@50 score of 86.4%, consistently demonstrating reliable performance in real-time counting of human heads even in challenging real-world conditions of UPL University cafeteria, including overhead fluorescent lighting during peak hours, low ambient lighting during early mornings and late afternoons, and possible obstruction of cameras' field of view by furniture or people. Analysis of errors made on the test dataset revealed that there are two major categories of mistakes: (a) occlusion of the head of a sitting person by another standing person in front of them, composing about 3% of false negatives; and (b) very small head sizes (smaller than 15×15 pixels in effective resolution) located in periphery of the wide-angle camera field of view, leading to 2% misses. As evident from the error analysis, both failure modes are

unavoidable for camera-angle-based detection models and cannot be resolved by adjusting the model parameters; proper camera position can help alleviate these issues [7].

The Table Availability Model achieved the mAP@50 of 88.2%, the higher number due to the generally static and visually distinguishable nature of occupied/unoccupied state of the table compared to dynamic human heads. Precision of 82.6% was chosen to be intentionally low — directly related to the high confidence threshold set for this model equal to 0.80 for making an inference. This high confidence threshold was chosen to reduce false positive cases — a misclassified unoccupied table is a less critical error than a falsely classified occupied one (an occupied table will lead a student to an empty space). Thus, in accordance with the error analysis results, precision errors occurred at confidence level 0.80 for those tables where the degree of occupancy was very small, i.e., with only a little personal belongings on it [7], [8].

B. System Latency Analysis

The measured end-to-end latency of changing occupancy states inside the UPL University cafeteria to reflect on a connected student dashboard through a cycle consisting of a real-world occupancy state change, AI node processing new images, sending status data to the FastAPI backend running in the Render cloud, and updating React fronted in Chrome browser took an average of [10].

TABLE V BREAKDOWN OF END-TO-END SYSTEM LATENCY [10]

LATENCY COMPONENT	MEASURED VALUE	NOTES
YOLOv11 inference per frame	~33 ms	640×640 input, CPU-only edge node
Frame capture + preprocessing	~15 ms	OpenCV capture + bilinear resize
AI node → Backend HTTP POST	~50 ms	Campus LAN → cloud VPS round trip
Backend POST processing	< 5 ms	In-memory variable update, no DB write
Frontend polling interval	3,000 ms	Fixed setInterval in React useEffect
Backend GET /status serving	< 15 ms	In-memory read, no DB query
Maximum end-to-end latency	≈ 3,100 ms	Dominated by frontend poll interval
Average end-to-end latency	≈ 1,600 ms	Expected value at uniform poll phase

Source: The Smart Anna project team measured latency in April 2026 using the Python time module (AI node and backend) and the Chrome DevTools Network panel (frontend). The backend was hosted on Render.com cloud VPS [10].

The results show that the main source of latency is the three-second interval at which the frontend polls for new data, comprising roughly 97% of the maximum latency from end to end. This is a conscious decision; if the polling interval was reduced to one second, the maximum latency would drop to about 1,100 milliseconds, but this would result in an increase in HTTP requests to the backend by a factor of three, thereby tripling the hosting costs and server workload. In the use case scenario of cafeterias where crowd densities are updated at the minute level and not on the second level, latency of three seconds is negligible, and three seconds is the optimal choice from an engineering perspective [10]. For applications requiring lower latency, the HTTP polling protocol can be switched out with a more efficient persistent WebSocket connection between frontend and backend, thus enabling near-instantaneous push notifications and reducing latency to around 100 milliseconds.

C. Usability Testing Results

Structured usability testing was performed on 15 student volunteers for 3 days of regular cafeteria operations at UPL University (Monday to Wednesday of a typical academic week in April 2026). Students were instructed to use Smart Anna web application as the primary source of information related to all cafeteria interactions during the study period, checking their dashboard before and during visits and placing all orders through the app. Following

completion of the test, students completed structured questionnaire on Likert 5-point scales as well as open-ended questions. The results are shown in Table VI [Authors].

TABLE VI USABILITY STUDY RESULTS SUMMARY (N=15 STUDENT VOLUNTEERS, APRIL 2026)

METRIC	RESULT
GO NOW/WAIT accuracy vs. own subjective assessment	91.3% agreement
Participants who avoided ≥ 1 unnecessary visit	12 / 15 (80%)
UPI payment convenience rating (Likert 1–5, avg.)	5.0 / 5.0
Average UPI checkout duration (checkout to confirmation)	~45 seconds
Estimated baseline cash checkout time (pre-study)	~90–120 seconds
Estimated payment time reduction	~50–63%
Comfort with privacy approach after explanation	15 / 15 (100%)
Would recommend Smart Anna to other students	15 / 15 (100%)
Admin panel peak-hour chart rated 'highly actionable'	Staff: unanimous

Source: Data gathered by Smart Anna project team through a structured survey conducted in the UPL University cafeteria in April 2026 [Authors].

The high accuracy rate in GO NOW / WAIT advice of 91.3% (meaning that the AI widget's advice on the student should go now or wait before entering the cafeteria because of crowding agreed with the student's subjective judgment of comfortable accessibility in 91.3% of interactions) is noteworthy because it proves the point that the data from the AI algorithm was indeed being translated into advice corresponding to the subjective perception of the students in the real-world setting. Disagreement was noted in about 8.7% of cases, which mostly related to fleeting crowding moments – for example, a surge of people just after the end of a lecture hour – which resolved very quickly. The latency of three seconds from the last system update could cause a momentary mismatch of the actual situation and system prediction [10].

The 100% agreement among the respondents on full comfort regarding the privacy strategy, after the explanation that only aggregated head counts and box bounding coordinates are being processed and no videos or facial recognition is involved, validates the approach of privacy-by-design on the edge of computation as an effective means of addressing student concerns about privacy [9]. From the qualitative feedback, it is apparent that the immediate perceived usefulness of the system was provided by the floor plan of the space and Smart Advice widget, with returning students who also had order history valuing the Curated For You shelf recommendations [9].Qualitative feedback consistently highlighted the floor map and the Smart Advice widget as the most immediately valued features, while returning students with order history valued the Curated For You recommendation shelf.

D. Technology Stack and Deployment

Table VII represents the entire technology stack used in Smart Anna, and Table VIII represents the infrastructure setup in the production environment during the usability testing of the Smart Anna solution

TABLE VII SMART ANNA TECHNOLOGY STACK SUMMARY

LAYER	TECHNOLOGY	VERSION / DETAILS
AI / Vision	YOLOv11 (Ultralytics) [7]	Custom-trained, edge deployment
Dataset Platform	Roboflow [8]	Annotation, augmentation, training
Video Processing	OpenCV [17]	Frame capture, resize, threading
Backend Framework	FastAPI [10]	Python 3.10+, Uvicorn ASGI server

Cloud Database	Supabase PostgreSQL [11]	ap-south-1 region, RLS enabled
ORM / DB Driver	SQLAlchemy + psycopg2	Python local dev fallback
Frontend Framework	React 18.2 (TypeScript) [3]	Vite 7.3 build tool
UI Styling	Tailwind CSS	CDN + custom utilities
Iconography	Lucide React	60+ components
Payment	UPI Dynamic QR [12]	NPCI standard, merchant VPA
Frontend Hosting	Vercel (CDN) [3]	Global edge deployment
Backend Hosting	Render / Railway [10]	Cloud VPS, auto-deploy

Source: System implementation — Smart Anna Project, UPL University, 2026 [Authors, 3, 7, 8, 10, 11, 12, 17].

TABLE VIII PRODUCTION DEPLOYMENT INFRASTRUCTURE SUMMARY [3], [10], [11]

COMPONENT	PLATFORM	ACCESS METHOD	NOTES
React Frontend	Vercel (Global CDN) [3]	Public HTTPS URL	Auto-deploy from GitHub
FastAPI Backend	Render / Railway [10]	Public HTTPS + CORS	Uvicorn ASGI, auto-restart
Supabase DB	Supabase Cloud [11]	JS client (anon key + RLS)	ap-south-1, daily backups
AI Node	Local workstation (campus)	HTTP POST to backend URL	Python venv, autostart on boot

Source: Setup used during the usability test, April 2026 [Authors, 3, 10, 11].

V. DISCUSSION

A. Strengths of the Proposed System

Smart Anna has several strengths in relation to the prior work. The most notable strength, however, is that it does not require any additional infrastructure beyond the already existing CCTV cameras, allowing it to be deployed anywhere with minimum expense and effort. Dual modeling of the tasks of counting heads and identifying available seats allows for task-specific optimizations in training, augmentation, and threshold settings [7], [8].

Another benefit is the privacy-by-design edge architecture of the system, which can be especially useful in the institutional context, taking into consideration data governance policies, regulations, and student privacy expectations. The system is architecturally incapable of storing or transmitting facial data or personally identifiable visual information, because the edge node transmits only integer counts. This is a stronger privacy guarantee than post-hoc anonymization of stored video, because it eliminates the risk of data breach of sensitive imagery at the source [9].

The comprehensive integration of occupancy monitoring, digital ordering, payment processing, analytics, and feedback into a single cohesive platform distinguishes Smart Anna from narrow single-function solutions. A student can check whether the cafeteria is accessible, pre-browse the menu while walking, order and pay without handling cash, receive a digital receipt, and submit feedback — all within a single application session. This integration of the entire workflow gets the most value out of each interaction and makes it easier to adopt [1], [14].

B. Limitations

The first limitation of the proposed approach concerns its domain-specificity in terms of data used for training the models. YOLOv11 models used in the current implementation were trained exclusively using data from the UPL University cafeteria premises. The use of these models on a cafeteria with a significantly different angle of cameras' views, lighting, design of tables, and architecture may be associated with poor performance and will likely require collection of

additional data and training the models specifically for this environment [7], [8]. This problem is inherent to any computer vision application and somewhat alleviated by the availability of Roboflow's annotation and training platform [8].

Secondly, wait-time forecasting currently implemented in the system is based on a reactive approach, where trends in queue length are computed using data of the last five polling intervals, corresponding to 15 seconds of observations. If a cafeteria has a very clear daily occupancy schedule where, for example, every day there will be a crowd coming at precisely 12:30 PM right after an hourly lecture, then a more sophisticated approach to modeling the occupancy should learn the time patterns and make predictions regarding crowd level up to 30–60 minutes in advance. Time series approaches like Long Short-Term Memory Networks (LSTM) or Facebook Prophet will help achieve this goal [14].

Lastly, current implementation of the UPI payment module requires manually pressing the 'Confirm Payment' button after scanning the QR code. Automatic integration with payment processors would provide a more efficient workflow of ordering and payment, and also enable additional functionality, such as automatic order cancellation in case of failed payment [12].

C. Future Work

Several high-value directions for future enhancement of Smart Anna have been identified based on the results of this deployment and the feedback from usability testing.

Multi-zone occupancy tracking would extend the system to support multiple physically distinct cafeteria zones or floors, each with independent occupancy tracking and represented as a separate section on the floor map. This is particularly relevant for larger institutional cafeterias with multiple service counters or seating areas with different menu offerings [9].

Predictive occupancy forecasting using LSTM or Transformer-based time-series models trained on the accumulated historical occupancy data would enable the system to predict crowd levels for the next 30 to 60 minutes with confidence intervals. This would transform the advice engine from a reactive to a proactive guidance system, enabling students to plan their cafeteria visits well in advance of the current moment [13], [14].

Embedded hardware deployment on low-cost NPU-equipped platforms such as NVIDIA Jetson Nano or Raspberry Pi 5 with dedicated neural processing unit acceleration would reduce the infrastructure cost of the edge AI node from a general-purpose workstation to a sub-100 USD embedded device, making the system economically viable for institutions with tight technology budgets [9].

Integrating a payment gateway webhook for automated UPI payment verification would get rid of the manual confirmation step in the current payment flow. Automatic integration with payment processors would provide a more efficient workflow of ordering and payment, and also enable additional functionality, such as automatic order cancellation in case of failed payment [12].

Secondly, the system could include a mechanism of pre-ordering and slot reservation in the cafeteria, so that demand could be distributed more evenly over the day, eliminating crowding altogether through managed demand.

VI. CONCLUSION

This paper discussed Smart Anna, the full-stack AI-based intelligent cafeteria management system that was designed and implemented at the UPL University in Anand, India. Smart Anna solves problems of student overcrowding, long wait times, and inefficient cafeteria management through the implementation of computer vision and web-based application interfaces for YOLOv11, FastAPI aggregation engine, Supabase database and React TypeScript frontend.

Two specialized custom-trained YOLOv11 models — a head detection model achieving mAP@50 of 86.4% and a table availability model achieving mAP@50 of 88.2% — process live CCTV feeds on an edge node at approximately 29 frames per second, transmitting only aggregated integer counts to the cloud to preserve student privacy. The FastAPI backend serves the current occupancy status with under 15 ms latency, and the React frontend delivers a live updated dashboard to students with a maximum four-second end-to-end latency. A complete digital ordering module with dynamic UPI QR payment reduces checkout time by approximately 50–63% compared to cash transactions [7], [8], [10], [11], [12].

The usability test conducted using 15 student volunteers revealed that there were tangible gains from

operating the solution: 80% of the testees skipped one or more non-mandatory visits to the cafeteria, the system's recommendations correlated with the subjective assessment of the students in 91.3% of cases, and each of the volunteers fully trusted the approach used by the software to protect user privacy upon learning about the edge computing concept [Authors].

The use of Smart Anna demonstrates that AI-driven smart campus solutions based on open-source DL frameworks, cloud infrastructure, and common web development techniques are technically feasible and affordable enough to be implemented by any educational institution, regardless of size and budget. The flexible and privacy-friendly design of Smart Anna suggests a proven architecture for the implementation of advanced facility management solutions that could be easily modified and extended to support various campuses and facilities. Given the ongoing process of digitalization within academic institutions, the solution similar to Smart Anna has considerable practical value in implementing an efficient AI-assisted administrative process [1], [9], [14].

ACKNOWLEDGMENT

The authors would like to thank Ms. Glace Babu, Faculty Guide in the Department of Computer Engineering at UPL University, for her invaluable guidance, helpful technical feedback, and constant support during the development and documentation of this project. The authors also thank UPL University in Anand, Gujarat, India, for giving them the cafeteria space and CCTV system they needed to collect data and test the system. We would like to thank the 15 student volunteers who took part in the usability testing study and gave honest and helpful feedback that helped us make the final version of the platform.

REFERENCES

[1] A. Lunchford and R. Patel, "Cafeteria Congestion and Student Stress: A Cross-Sectional Study in Urban Universities," J. Campus Health, vol. 12, no. 3, pp. 45–58, 2019.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE CVPR, Las Vegas, NV, 2016, pp. 779–788.

[3] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed. Sebastopol, CA: O'Reilly Media, 2020.

[4] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in Proc. IEEE ICCV, Venice, Italy, 2017, pp. 2980–2988.

[5] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," in Proc. Int. Conf. Syst., Signal. Image Process. (IWSSIP), Niterói, Brazil, 2020, pp. 237–242.

[6] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, 2018.

[7] Ultralytics, "YOLOv11: Next-Generation Object Detection Architecture," Technical Report, Ultralytics Inc., 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[8] Roboflow Inc., "Roboflow: Build and Deploy Computer Vision Models," Platform Documentation, 2023. [Online]. Available: https://roboflow.com

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[10] S. Ramirez, "FastAPI: Modern, Fast (High-Performance) Web Framework for Building APIs with Python 3.6+," Software Documentation, 2021. [Online]. Available: https://fastapi.tiangolo.com

[11] Supabase Inc., "Supabase: The Open-Source Firebase Alternative," Software Documentation, 2023. [Online]. Available: https://supabase.com/docs

[12] National Payments Corporation of India (NPCI), "Unified Payments Interface: System Overview and Operational Guidelines," Technical Report, NPCI, New Delhi, India, 2022. [Online]. Available: https://www.npci.org.in/what-we-do/upi/product-overview

[13] Z. Xu, W. Zhang, and Y. Wang, "Real-Time Crowd Density Estimation Using Deep Learning and Edge Computing," IEEE Internet Things J., vol. 9, no. 15, pp. 13842–13854, Aug. 2022.

[14] H. Chen, K. Li, and J. Zhang, "Smart Campus Management Systems: A Survey of IoT and AI-Based Approaches," J. Campus Comput., vol. 3, no. 2, pp. 45–62, 2021.
[15] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," Mach. Learn. Knowl. Extr., vol. 5, no. 4, pp. 1680–1716, 2023.
[16] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv preprint arXiv:2004.10934, 2020.
[17] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol, CA: O'Reilly Media, 2008.