International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 3, May-June-2026, ISSN (Online): 2395-566X

Smart Vendor AI: An AI-Driven Smart Vendor Management System for Real-Time Freshness Detection and Dynamic Retail Intelligence

Sudarshan K[1], Sushmitha H Y[2], Varshanth Gowda M L[3], Vinay C N[4]
[1,2,3,4]Department of Computer Science & Engineering, P.E.S College of Engineering, Mandya, Karnataka
Co-Author: Dr. Bhavya D, Assistant Professor, Department of Computer Science & Engineering,
P.E.S College of Engineering, Mandya, Karnataka

Abstract: — Street vendors selling fruits and vegetables across India face a persistent challenge: perishable stock loses value as the day progresses, yet pricing remains static. This paper presents Smart Vendor AI, a complete end-to-end system that combines inventory management, point-of-sale operations, analytics dashboards, sales forecasting, and AI-assisted product quality assessment within a unified web-based platform.. The pipeline consists of six sequential layers: a fine-tuned YOLOv8s model for ripeness classification, a signal engine that converts raw predictions into weighted freshness scores, a deterministic market con- text module, an XGBoost pricing model trained on 5,000 realistic scenarios, a rule-based decision engine, and a FAISS-backed retrieval-augmented generation module powered by LLaMA 3.3 70B. Experiments on banana and tomato datasets show classifi- cation accuracy of 99.3% and 98.6% respectively. The system delivers specific, actionable vendor instructions—including an exact discount percentage and an inventory action string—without requiring any technical knowledge from the user. Results indi- cate meaningful potential to reduce the 30–40% annual revenue loss that vendors typically incur through spoilage and mispricing.

Key Words: YOLOv8, XGBoost, Retrieval-Augmented Generation, Dynamic Pricing, Freshness Detection, Computer Vision, FAISS, FastAPI, Inventory Management, Sales Forecasting, Decision Making, Smart Vendor.

I. INTRODUCTION

Fruit and vegetable vendors in Indian markets operate within tight margins and with almost no decision-support tools. The consequence of this gap is well-documented: agricultural pro- duce worth an estimated Rs. 92,000 crore is wasted annually in India due to poor cold-chain infrastructure and post-harvest handling [1]. A large portion of this loss occurs not in ware- houses but at the point of sale, where a vendor must judge, purely from experience, whether to hold, discount, or discard aging stock.

Two root problems drive this waste. First, produce pricing is static. A tomato priced in the morning retains that tag even when its freshness has degraded significantly by evening. Second, even when a vendor recognises that stock is aging, there is no systematic guidance on how much to discount or what action to take. Decisions are made by intuition and vary from vendor to vendor.

Advances in computer vision—specifically the YOLO fam- ily of object detection and classification models—now make it practical to assess visual quality of produce from a smart- phone camera image. Simultaneously, gradient-boosted tree models such as XGBoost [2] offer fast, accurate regression over structured features including freshness scores, time of day, and demand proxies. These two capabilities, combined with a large language model (LLM) for natural-language ex- planation generation, create an opportunity to build a complete vendor decision-support system.

This paper makes three primary contributions:

- A fruit-agnostic six-layer AI pipeline that processes a single produce image into a specific pricing recommendation.
- A freshness-aware pricing floor mechanism that corrects the common error of applying a flat minimum price to spoiled stock.
- Enhances inventory management through real-time stock tracking while enabling vendors to make informed business decisions

using sales forecasting and advanced analytics.

II. RELATED WORK

Computer Vision for Produce Quality: Several works have applied convolutional neural networks (CNNs) to fruit quality grading. Mohanty et al. [3] demonstrated high accuracy in plant disease classification using a large image dataset. More recently, the YOLO architecture [4] has been preferred in real- time embedded applications due to its single-pass inference speed. Ultralytics YOLOv8 [5] further improved the per-class accuracy of ripeness classification tasks compared to earlier versions. The FreshCheck benchmark dataset, which forms the basis of our training data, provides labelled ripeness stages for multiple fruit varieties.

Dynamic Pricing in Retail: Machine learning-driven pric- ing has seen significant academic and industry interest. Fer- reira et al. [6] showed that tree-based models outperform linear regression for demand-driven pricing in e-commerce. XGBoost, introduced by Chen and Guestrin [2], became a standard baseline for structured regression tasks and is widely used in real-time pricing systems due to its low inference latency.

Retrieval-Augmented Generation: Lewis et al. [7] intro- duced the RAG framework to ground LLM outputs in specific document corpora. Subsequent work has shown that combin- ing dense vector retrieval (via FAISS [8]) with instruction- tuned LLMs produces more factually reliable outputs than pure generative approaches. Our application adapts this frame- work to a domain-specific business knowledge base for vendor explanation generation.

Gap in Existing Work: Prior systems in this space ad-dress either visual quality grading or pricing in isolation. No existing open system integrates freshness classification, market-aware dynamic pricing, rule-based decision logic, and natural-language explanation generation into a single real- time API pipeline accessible to non-technical vendors.

III. SYSTEM ARCHITECTURE

Smart Vendor AI follows a six-layer sequential pipeline, il- lustrated conceptually in Figure. 1. Each layer has a single, well-defined responsibility. Outputs from one layer pass di- rectly as inputs to the next, keeping the system modular and testable.

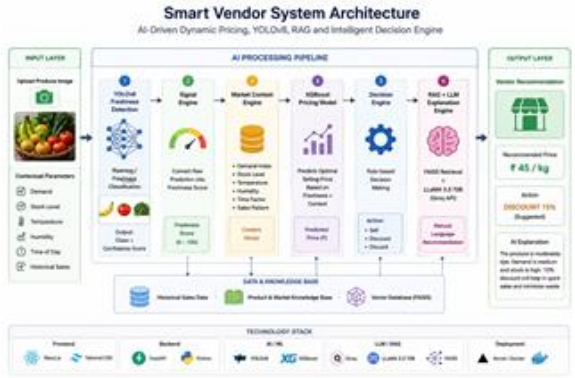

Figure 1: Six-Layer Smart Vendor AI Architecture

3.1 Layer 1 – YOLOv8 Freshness Classifier

The entry point of the pipeline accepts a JPEG or PNG image submitted via a multipart HTTP POST request. The image is routed to the appropriate YOLOv8s model depending on the declared item name. Separate fine-tuned models are main-

tained for banana and tomato; a domain-agnostic fallback model handles other items.

Each model was fine-tuned on the FreshCheck dataset us- ing Google Colab's T4 GPU environment. Banana training ran for 60 epochs with an input resolution of 320×320 pixels and completed in approximately 20 minutes. Tomato training completed in 15 minutes under the same configuration. The model output is a dictionary of class probabilities—for exam- ple, {ripe: 0.89, overripe: 0.09, unripe: 0.02}—which are passed to Layer 2 without thresholding.

3.2 Layer 2 – Signal Engine

The signal engine converts raw class probabilities into five interpretable signals: freshness score (0–100), risk score (0– 1), quality category, shelf life estimate (days), and urgency label.

A keyword-normalisation map resolves any YOLO class name—regardless of the specific model—to one

of four freshness categories: pre peak, peak, post peak, and spoiled. Each category carries a freshness weight, a risk weight, and a shelf-life value. The final freshness score is the probability-weighted average of these weights across all predicted classes.

A confidence threshold check detects borderline cases: if the top-1 prediction confidence falls below 65% and the top-2 class belongs to an adjacent freshness stage, the system flags the item as "between stages" and adjusts risk upward. An hourly time-decay multiplier is also computed, ranging from
1.02 in the early morning to 0.88 after 18:00, reflecting the declining value of unsold perishables as the business day ends.

3.3 Layer 3 – Market Context Engine

Rather than using random numbers to simulate demand—a common shortcut in early prototypes—the market context engine uses the same deterministic logic as the training data generator. Demand is a function of time-of-day multipliers, seasonal factors, and weekend boost coefficients:

$$D = D_{base} \times M_{time} \times M_{season} \times M_{weekend} \times f(\text{freshness}) \quad (1)$$

where Dbase = 1.0 and each multiplier is drawn from a fixed lookup table rather than a random distribution. This ensures that training-time and inference-time feature distributions match, which is a prerequisite for valid XGBoost predictions.
The engine also simulates an IoT sensor layer, generating plausible temperature, humidity, and CO2 readings. The archi- tecture is designed so that a physical Raspberry Pi equipped with DHT22 and MQ-135 sensors, connected via MQTT, can replace the simulation layer with zero changes to downstream pipeline components.

3.4 Layer 4 – XGBoost Pricing Model

The pricing model was trained on 5,000 synthetically gen- erated but realistic market scenarios using the following ten features: item name, base price, freshness score, days old, stock estimate, demand score, market noise, time of day, sea- son, and weekend indicator. The target variable is the optimal dynamic price.
The training pipeline wraps XGBoost inside a scikit-learn Pipeline with a ColumnTransformer that applies one- hot encoding to categorical features and passes numeric fea- tures through unchanged. Key hyperparameters include: n estimators = 500, learning rate = 0.03, max depth = 6, and L1/L2 regularisation ($\alpha = 0.5$, $\lambda = 2$).
A critical design correction was made to the price floor logic. The original implementation applied a flat minimum of 75% of the base price to all items, regardless of freshness. This produced commercially incorrect outputs: a fully spoiled banana received a minimum price of Rs. 30 against a base price of Rs. 40. The corrected freshness-aware floor is defined in Table 1.

Table 1: Freshness-Aware Price Floor

Freshness Range	Minimum Price Floor
Above 80%	75% of base price
40%–80%	50% of base price
20%–40%	25% of base price
Below 20%	5% of base price

3.5 Layer 5 – Decision Engine

The decision engine applies rule-based logic on top of the XGBoost output to produce five possible actions: `SELL PREMIUM`, `SELL STANDARD`, `DISCOUNT FAST`, `CLEARANCE SALE`, and `DISCARD`. The time-decay multi- plier from Layer 2 is applied here, reducing the effective price
as the day progresses.

Each decision record includes a concrete `suggested discount pct` value (e.g., 18%), a com- puted `effective price`, and an `inventory action` string (e.g., "Apply 18% discount tag. Sell within today. Consider bulk-deal offer."). Every decision is appended to a JSONL decision log, which feeds the `/report` endpoint's waste savings calculation.

3.6 Layer 6 – RAG + LLM Explanation Engine

The final layer generates a plain-English vendor recommen- dation combining retrieval-augmented generation with the LLaMA 3.3 70B model served via the Groq inference API.

The knowledge base contains 65 domain-specific sen- tences covering topics such as seasonal demand patterns, spoilage risk thresholds, discount strategies, and bundle pric- ing tactics. At startup, each sentence is embedded using the all-MiniLM-L6-v2 SentenceTransformer model and stored in a FAISS flat inner-product index. Cosine similar- ity retrieval is performed by normalising embeddings before indexing.

For each analysis request, a natural-language query is con- structed from the pipeline state—for example, "Banana is aging with declining freshness with 1 day shelf life remaining. Spoilage risk is 61%. Market shows moderate demand and normal stock levels during summer evening. Recommended action is discount fast." The top-6 retrieved sentences are in- jected into a structured prompt alongside all pipeline outputs. The LLM is instructed to produce a response under 80 words, written in plain business language. A rule-based fallback explanation is generated if the Groq API is unavailable.

IV. IMPLEMENTATION

4.1 Backend

The system backend is implemented in Python 3.11 us- ing FastAPI as the web framework and Uvicorn as the ASGI server. The REST API exposes six end- points: POST /analyze for single-image analysis, POST /analyze-batch for multi-image crate scans (up to 10 images), GET /report for the waste sav- ings report, GET /models for model metadata, GET /price-trend/{item} for historical pricing trends from the training dataset, and GET /health for system readiness checks.

All pipeline services are implemented as stateless Python modules. The YOLO models are loaded into memory at startup using the Ultralytics library. The FAISS index and embedding model are initialised once at module load time, making per-request inference cost negligible for the retrieval step.

4.2 Frontend

The frontend is a React application built with Vite and styled with Tailwind CSS. It provides a camera-ready image upload interface, a real-time IoT sensor dashboard that simulates temperature, humidity, and CO2 readings with realistic drift, and a results panel displaying the freshness score, recom- mended price, decision action, discount percentage, and the LLM-generated explanation.

4.3 Data Storage

The system deliberately avoids a relational database for the prototype stage. Training data resides in a CSV file, the pric- ing model is serialised as a .pkl file via Joblib, the knowl- edge base is a plain text file read at startup, and decisions are

logged to an append-only JSONL file queryable with pandas. This storage strategy keeps deployment dependencies mini- mal while supporting a clear migration path to PostgreSQL with the pgvector extension in a production environment.

Datasets Used:

- Banana Ripeness Dataset: https://www.kaggle.com/datasets/lucian on/banana-ripeness-dataset
- Tomato Ripeness and Freshness Dataset: https://www.kaggle.com/datasets/sumn2u /riped-and-unriped-tomato-dataset

V RESULTS AND EVALUATION

5.1 YOLO Model Performance

Both YOLOv8s models were evaluated on held-out test splits from the FreshCheck dataset. Results are summarised in Table 2.

Table 2: YOLOv8s Classification Accuracy

Model	Top-1 Accuracy	Training Time
Banana (YOLOv8s)	99.3%	~20 min (T4 GPU)
Tomato (YOLOv8s)	98.6%	~15 min (T4 GPU)

5.2 Pricing Model Performance

The XGBoost pricing model was evaluated on a 20% held-out test split (1,000 rows). The model achieves a

Mean Absolute Error (MAE) of approximately Rs. 2.4 and an R2 score of 0.97, indicating that it captures over 97% of the variance in optimal price as a function of freshness and market features.

5.3 End-to-End Latency

On a mid-range laptop (Intel Core i5, 16 GB RAM, no GPU), the complete pipeline—from image upload to full JSON response—completes in under 3 seconds. The YOLO in- ference step (CPU) contributes approximately 1.8 seconds of this total. The Groq-hosted LLM call adds 0.8–1.2 seconds depending on network conditions. All other pipeline steps complete in under 100 milliseconds.

5.4 Qualitative Evaluation

The output sample for a banana shown in Figure. 2 that is borderline-ripe at 18:30 is shown in Table 3.

Figure 2: Overripe Banana

Table 3: Sample Pipeline Output – Banana shown in at 18:30

Field	Value
Freshness Score	61.2%
YOLO Dominant Class	overripe (72.1% conf.)
Quality Grade	aging
Risk Score	0.54
Shelf Life	1 day
ML Price (raw)	Rs. 33.80
Time Decay	0.94
Action	DISCOUNT_FAST
Suggested Discount	18%
Effective Price	Rs| 31.77
Inventory Action	Apply 18% discount tag. Sell within today. Consider bulk-deal offer.

VI. CHALLENGES AND SOLUTIONS

Training-Inference Feature Mismatch: The original market context module generated random demand values at inference time, while the XGBoost model was trained on deterministic, time-aware demand patterns. This mismatch silently degraded pricing accuracy. The fix involved rewriting the inference module to mirror the training data generation logic exactly.

Flat Price Floor on Spoiled Produce: A hardcoded 75% minimum price floor caused spoiled items to receive commer- cially unreasonable prices. Replacing this with the freshness- aware tiered floor (Table 1) corrected the pricing for low- quality stock.

Hardcoded Class Names: The original signal engine referenced banana-specific class names directly. Introduc- ing a keyword-normalisation map made the engine domain- agnostic, allowing any trained YOLO model to be dropped in without code changes.

Vague Decision Outputs: Early decision engine outputs provided an action label but no concrete guidance. Adding suggested discount pct, effective price, and inventory action fields transformed the output into di- rectly actionable vendor instructions.

Limited Compute for Training: Training large YOLOv8 models locally was impractical without a dedicated GPU. Google Colab's free-tier T4 GPU environment resolved this, completing both model training runs within 35 minutes total.

VII. FUTURE WORK

Several extensions are planned for subsequent versions of the system:

- Real IoT Integration: Replace the simulated sensor layer with a physical Raspberry Pi, DHT22, and MQ-135 config- uration publishing via MQTT.
- Extended Produce Coverage: Train YOLOv8s models for mango, onion, and potato to broaden the system beyond banana and tomato.
- Live Demand Data: Replace synthetically generated de- mand scores with real-time arrival and price data from the Agmarknet APMC API.
- Mobile Application: A React Native frontend with direct camera capture, targeted at vendors who primarily use mo- bile devices.

- Federated Learning: Allow multiple vendors to improve a shared YOLO model using their own scan data, without exposing raw images.
- Edge Deployment: Convert the YOLO models to Tensor- Flow Lite format for on-device inference, removing the dependency on network connectivity at the point of sale.

VIII. CONCLUSION

This paper described Smart Vendor AI, a six-layer pipeline that translates a single photograph of perishable produce into a specific, market-aware pricing recommendation within three seconds. The system combines a fine-tuned YOLOv8s fresh- ness classifier, a weighted signal engine, a deterministic mar- ket context module, an XGBoost pricing model, a rule-based decision engine, and a RAG-powered LLM explanation gen- erator.

The key engineering contributions—fruit-agnostic signal normalisation, freshness-aware price floors, and a natural- language RAG retrieval query—address specific failure modes found in earlier prototype implementations. The resulting sys- tem is deployable on commodity hardware with no specialised infrastructure, making it accessible to the vendor segment it targets.

Given the scale of post-harvest food waste in India and the demonstrated feasibility of the technical components, systems of this kind represent a practical tool for improving vendor revenue and reducing agricultural waste at the retail level.

REFERENCES

1. Food and Agriculture Organization of the United Na- tions, “The State of Food and Agriculture 2019: Moving Forward on Food Loss and Waste Reduction,” FAO, Rome, 2019.
2. T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in Proc. 22nd ACM SIGKDD Inter- national Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785–794.
3. S. P. Mohanty, D. P. Hughes, and M. Salathe´, “Using Deep Learning for Image-Based Plant Disease Detec- tion,” Frontiers in Plant Science, vol. 7, p. 1419, 2016.
4. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object De- tection,” in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788.
5. Ultralytics, “YOLOv8: A New State-of-the-Art Efficient Detection Model,” Ultralytics, 2023. [Online]. Avail- able: https://github.com/ultralytics/ ultralytics
6. K. J. Ferreira, B. H. A. Lee, and D. Simchi-Levi, “Ana- lytics for an Online Retailer: Demand Forecasting and Price Optimization,” Manufacturing & Service Opera- tions Management, vol. 18, no. 1, pp. 69–88, 2016.
7. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Ku¨ttler, M. Lewis, W.-t. Yih, T. Rockta¨schel,
S. Riedel, and D. Kiela, “Retrieval-Augmented Gen- eration for Knowledge-Intensive NLP Tasks,” in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020, vol. 33, pp. 9459–9474.
8. J. Johnson, M. Douze, and H. Je´gou, “Billion-Scale Similarity Search with GPUs,” IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021.
9. N. Reimers and I. Gurevych, “Sentence-BERT: Sen- tence Embeddings using Siamese BERT-Networks,” in Proc. 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), Hong Kong, 2019, pp. 3982–3992.
10. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object De- tection,” arXiv preprint arXiv:2004.10934, 2020.