Grozo: An AI-Powered Online Grocery Delivery System Integrating Machine Learning, Dynamic Pricing, And Real-Time Logistics In A Full-Stack MERN Platform

V. Udhayakumar[1], M. Sadhish[2]
[1] Associate professor, Dept of Computer Applications
[2]Dept of Computer Applications
[1, 2] Sri Manakula Vinayagar Engineering College, Puducherry, India.

Abstract- *Widespread smartphone adoption and ubiquitous high-speed connectivity have collectively reshaped how consumers discover and purchase goods, driving sustained migration toward digital retail channels across all product categories. The grocery segment, long anchored to in-person shopping, is now experiencing accelerating structural change as customers increasingly prioritise doorstep delivery, broad product assortment, and faster fulfilment over the traditional in-store experience. This paper presents Grozo, a production-ready, AI-powered online grocery delivery platform built on the MERN (MongoDB, Express.js, React, Node.js) technology stack. Grozo integrates a suite of intelligent subsystems — including a multi-signal product recommendation engine, a real-time dynamic pricing model, an inventory forecasting module, a Haversine-based route optimisation algorithm, and a Gemini-powered conversational AI chatbot — to deliver a seamless and personalised shopping experience. The system supports over 8,000 product listings sourced from BigBasket, multi-modal payment processing via Razorpay, live order tracking through Leaflet.js/OpenStreetMap, a four-tier loyalty rewards programme, and a voice-command interface built on the Web Speech API. Experimental evaluation demonstrates that Grozo achieves sub-second API response times, a recommendation click-through rate improvement of 34% over static listings, and a dynamic pricing engine accuracy within 8% of market fluctuations. The architecture described herein offers a scalable blueprint for next-generation AI-driven grocery retail platforms.*

Keywords: E-commerce, Grocery Delivery, MERN Stack, Recommendation Engine, Dynamic Pricing, Chatbot, Route Optimisation, Real-Time Systems, Machine Learning, Web Speech API, Socket.io, Razorpay.

I. INTRODUCTION

The Indian online grocery market has expanded markedly over recent years, driven by a convergence of changed shopping habits following the COVID-19 pandemic, growing smartphone ownership across urban and semi-urban households, and the rise of sub-30-minute delivery services. Domestic market forecasts project this segment to reach USD 24 billion by 2027 (IBEF, 2024), with incumbents such as BigBasket, Blinkit, Zepto, and Swiggy Instamartintensifying competition through AI-driven personalisation and hyper-local logistics.

Despite sector growth, four recurring technical gaps are apparent in prior work: (i) price adjustment mechanisms that react to real-time demand signals and supply conditions rather than relying on static rules alone, (ii) hybrid recommendation approaches that merge collaborative filtering with content signals without incurring heavy computational overhead, (iii) unified platform designs that serve distinct roles — customer, administrator, and delivery partner — within a single cohesive codebase, and (iv) AI-driven chat interfaces that draw on live order and cart state rather than pre-scripted FAQ trees.

This paper addresses these gaps through the design, implementation, and evaluation of Grozo — a full-featured grocery delivery platform whose name connotes growth and grocery synergised. Grozo's key contributions are:

- A six-module AI algorithm engine spanning recommendations, smart search, cart abandonment detection, inventory forecasting, delivery route optimisation, and coupon targeting.
- A multi-factor dynamic pricing engine that adjusts prices every 30 minutes based on seasonal data, demand signals, stock levels, and time-of-day patterns.
- A real-time, role-based platform built with Socket.io supporting live order tracking, push notifications, and a delivery partner dashboard.

- A Gemini-powered conversational AI chatbot ("Genie") with contextual awareness of live cart, order history, and available coupon codes.
- A voice-command interface using the Web Speech API enabling hands-free product search, navigation, and cart management.

The remainder of this paper is structured as follows: Section 2 reviews related work; Section 3 describes system architecture; Section 4 details the AI and algorithmic modules; Section 5 covers the dynamic pricing engine; Section 6 presents implementation details; Section 7 reports experimental results; and Section 8 concludes with future directions.

II. RELATED WORK

Recommendation systems in e-commerce have historically leaned on collaborative filtering (CF), with Amazon's item-to-item CF approach (Linden et al., 2003) serving as a foundational benchmark. Later work brought matrix factorisation techniques (Koren et al., 2009) and neural methods such as Neural Collaborative Filtering (He et al., 2017) into the mainstream. Grocery recommendation carries sector-specific difficulties — notably high repurchase frequency, perishability constraints, and basket-level item correlations — that have been studied using the publicly released Instacart dataset and through RNN-based models for sequential basket prediction (Yu et al., 2016).

Dynamic pricing research has a rich history in yield-managed sectors such as aviation and hospitality (Talluri and Van Ryzin, 2004). Extending these principles to perishable groceries adds complexity: price adjustments must track live demand, accelerating stock depletion, and upstream commodity price movements. Rule-based engines dominate deployed retail systems for their predictability, while reinforcement learning formulations (Maestre et al., 2018) can achieve tighter accuracy at the cost of additional training infrastructure and latency.

MERN-stack e-commerce implementations appear primarily in instructional literature (Aggarwal, 2018); independently evaluated, production-grade grocery platforms combining MERN with integrated AI capabilities are notably absent from peer-reviewed venues. The use of large language models (LLMs) for e-commerce support channels has accelerated since GPT-3 demonstrated few-shot capabilities (Brown et al., 2020), and Google's Gemini API has lowered experimentation barriers through a free-tier offering suitable for academic prototyping.

Grozo differentiates itself from prior work by combining all of the above subsystems — recommendations, dynamic pricing, LLM chatbot, voice interface, real-time logistics, and loyalty mechanics — within a unified, open-source platform evaluated against real-world product data.

III. SYSTEM ARCHITECTURE

Grozo follows a three-tier client-server architecture with a React 18 single-page application (SPA) front-end, a Node.js/Express.js REST API back-end, and a MongoDB 7 document store. Real-time bidirectional communication is provided by Socket.io. Table I illustrates the high-level architecture layers.

TABLE I Grozo System Layer Architecture

Layer	Technology	Responsibility
Presentation	React 18, Vite, Tailwind CSS 3	Customer, Admin & Delivery UIs
API Gateway	Node.js 18+, Express.js	REST endpoints, JWT auth, WebSocket
Business Logic	Custom JS modules	Pricing engine, Algorithm engine
Data Access	Mongoose ODM	Schema validation, query optimisation
Persistence	MongoDB 7	Products, Orders, Users, Carts
External APIs	Gemini, Razorpay, Nominatim	AI, Payments, Geocoding
Real-time	Socket.io	Order status push, rider location

A. Front-End Architecture

The React front-end is organised into three role-specific portals — Customer, Admin, and Delivery Partner — each with its own navigation sidebar and protected route guards. React Router v6 enforces role-based access control (RBAC) on the client side, complemented by JWT-validated middleware on the server. State management is handled via React Context API and local component state, keeping the

bundle lightweight without Redux overhead. The design system employs a purple (#6E3DFF) primary palette with Inter as the system typeface.

B. Back-End Architecture

The Express.js server exposes RESTful endpoints grouped by domain: /api/auth, /api/products, /api/orders, /api/cart, /api/algo, /api/pricing, /api/chat, and /api/payment. Authentication uses stateless JWTs verified by a custom authMiddleware. The server integrates Socket.io on the same HTTP instance, emitting events for order status changes, rider location updates, and push notifications. An automated pricingScheduler runs the dynamic pricing engine every 30 minutes.

C. Data Models

Nine core Mongoose schemas underpin the system:

TABLE II Core Data Models

Model	Key Fields	Purpose
User	role, loyaltyPoints, tier, referralCode	Auth & loyalty
Product	price, originalPrice, dynamicPrice, demandScore	Catalogue & pricing
Order	items[], status, paymentStatus, razorpayOrderId	Order lifecycle
Cart	user, items[{product, quantity}]	Persistent basket
Flash Sale	product, discountPercent, startTime, endTime	Timed promotions
Coupon	code, type, value, minOrder, usedBy[]	Discount codes
Subscription	user, product, frequency, nextDelivery	Recurring orders
Notification	user, message, type, read	In-app alerts
Review	user, product, rating, comment	Product reviews

IV. AI AND ALGORITHM ENGINE

The Algorithm Engine (backend/utils/algorithms.js) encapsulates six intelligent modules exposed via /api/algo REST endpoints, each designed for low-latency real-time execution.

A. Product Recommendation Engine

The recommendation engine combines three signals to produce a ranked product list:

- **Co-purchase signal (weight = 3):** Queries paid orders containing the seed product (capped at 200 docs), aggregates co-occurring products, awards 3 per co-occurrence.
- **User purchase history (weight = 1 + 0.5 × rating):** Analyses last 10 paid orders to extract preferred categories; boosts products by average rating.
- **Trending signal (weight = log(count+1)):** Counts units sold in trailing 7 days; log-dampened to prevent popularity bias.

Final scores are sorted descending; top candidates are fetched from MongoDB, in-stock filtered, and returned. The hybrid scoring formula is:

$$\text{Score}(p) = 3 \times \text{Co}(p) + (1 + 0.5 \times r(p)) \times \text{Cat}(p) + \log(\text{Trend}(p)+1)$$

B. Smart Search Ranking

MongoDB text-index results are re-ranked using a composite score incorporating purchase frequency, average product rating, and a personalisation boost for historically preferred categories.

C. Cart Abandonment Predictor

Scans Cart documents not updated in 24 hours with at least one item, computes aggregate lost revenue per cart, and returns a ranked at-risk customer list to the admin dashboard on a pull basis.

D. Inventory Forecasting

Computes daily sales velocity from trailing 30-day order history. Days-until-stockout is estimated as:

$$\text{DaysToStockout} = \text{currentStock} / \text{avgDailyUnits}$$

Products with DaysToStockout< 7 are flagged high-risk; < 14 medium-risk, feeding the admin's Inventory Forecast dashboard.

E. Delivery Route Optimiser

Selects the nearest available delivery partner using the Haversine formula to compute great-circle distances between the partner's last known GPS coordinate and the customer's geocoded address. This O(n) heuristic runs synchronously within the assignment API call.

F. Smart Coupon Targeting

Segments users into three cohorts — churned (30+ days inactive), at-risk (14–30 days), and active — recommending personalised coupon codes. Admins can bulk-export targeted user lists for campaigns.

V. DYNAMIC PRICING ENGINE

Grozo's dynamic pricing engine (backend/utils/dynamicPricing.js) continuously adjusts product prices within configurable bounds. The final computed price is stored as dynamicPrice on each Product document with transparent original-price comparison labels.

TABLE III Dynamic Pricing Factors

Factor	Condition	Adj.
Seasonal (Veg)	Monsoon (Jun–Sep)	+15%
Seasonal (Veg)	Winter (Nov–Feb)	-10%
Seasonal (Fruit)	Mango season (Apr–Jun)	-15%
Seasonal (Dairy)	Summer (Mar–May)	+8%
Stock Scarcity	Stock ≤ 5 units	+18%
Stock Surplus	Stock ≥ 200 units	-7%
High Demand	demandScore ≥ 100	+20%
Low Demand	demandScore ≤ 10	-15%
Morning Fresh	6 AM – 9 AM	-3%
Evening Clear.	8 PM – 11 PM	-6%
Flash Sale	Active flash sale	Admin %

The engine fetches an inflation proxy from ExchangeRate-API and multiplies prices by an inflation factor when the rupee weakens. Prices are bounded: no product exceeds 130% of originalPrice or falls below 70%.

A. Flash Sales Integration

Flash sales override dynamic prices for specified windows. The FlashSale model stores product ID, discount percentage, start/end timestamps, and an isActive flag. A countdown timer on the customer home page broadcasts urgency via Socket.io.

VI. IMPLEMENTATION DETAILS

A. AI Chatbot — Genie

Genie is powered by Google Gemini API (gemini-pro, free tier). Each user message is enriched with live context: current cart contents, last three order statuses, and available coupon codes. Quick-action chips (Track Order, Show Cart, Available Coupons) reduce friction for first-time users.

B. Voice Assistant

Leverages the browser-native Web Speech API (SpeechRecognition and SpeechSynthesis), requiring no third-party API keys. Supported intents include product search, navigation, order tracking, and cart operations. A waveform animation provides visual feedback during listening.

C. Real-Time Order Tracking

Implemented via Leaflet.js with OpenStreetMap tiles. When a delivery partner marks an order Out for Delivery, their device emits GPS coordinates at 10-second intervals via Socket.io. Nominatim geocodes customer text addresses to latitude/longitude.

D. Loyalty Programme

Points are awarded at order completion: 1 point per ₹10 at Bronze tier, scaling to 3 points per ₹10 at Platinum (5,000+ pts). Silver (500+ pts) and above receive free delivery on qualifying orders; Gold (2,000+ pts) receives unconditional free delivery.

E. Payment Integration

Razorpay's Node.js SDK handles payment order creation and webhook verification. Supported methods include UPI (GPay, PhonePe, Paytm, BHIM), debit/credit cards (Visa,

Mastercard, RuPay, Amex), net banking (50+ banks), wallets, and Cash on Delivery.

VII. EXPERIMENTAL RESULTS AND EVALUATION

Grozo was evaluated against 8,247 product listings imported from BigBasket. Metrics were collected across three dimensions: system performance, recommendation quality, and pricing accuracy.

A. System Performance

TABLE IV API Performance (50 Concurrent Users)

Endpoint	Avg(ms)	P95(ms)	req/s
GET /api/products	42	87	210
GET /api/algo/reco	118	195	84
POST /api/cart	31	58	290
POST /api/orders	76	134	120
GET /api/loyalty	28	49	340
POST /api/pricing	1,240	1,890	0.8

All customer-facing endpoints responded under 120 ms at P95 under 50 concurrent users, within the 200 ms threshold for perceived instant response (Nielsen, 1993).

B. Recommendation Quality

The hybrid engine achieved Hit Rate@10 of 0.61 and NDCG@10 of 0.44, versus 0.47 and 0.31 for a popularity-only baseline. A/B testing over 30 days showed a 34% CTR increase on recommended products.

TABLE V Recommendation Evaluation (n=1,200)

Method	HR@10	NDCG@10	TR
Popularity Baseline	0.47	0.31	
Co-purchase CF	0.54	0.37	18%
Grozo Hybrid	0.61	0.44	34%

C. Dynamic Pricing Accuracy

Pricing accuracy was evaluated by comparing Grozo-generated prices for 50 vegetable and fruit SKUs against manually recorded market prices over 14 days. MAPE was 7.8%, acceptable for a rule-based retail engine.

D. User Experience Metrics

A usability study with 35 participants yielded a SUS score of 79.6 ("Good"). The voice assistant received the highest novelty rating (4.3/5), the loyalty dashboard the highest engagement (4.6/5), and chatbot task-completion rate was 88%.

VIII. DISCUSSION

Grozo demonstrates that a full-featured, AI-augmented grocery platform can be built cost-effectively on the MERN stack without proprietary ML infrastructure. The hybrid recommendation engine outperforms pure popularity and pure CF baselines through multiple lightweight signals.

The dynamic pricing engine's MAPE of 7.8% is competitive with published rule-based retail pricing results, though it falls short of RL-based approaches (~4–5% MAPE) at significantly lower computational cost.

The voice assistant's dependency on browser-native Web Speech API limits availability to Chrome and Edge. Future iterations will evaluate cloud-based ASR services (e.g., Whisper API) for broader compatibility.

Grozo's open-source codebase and modular architecture enable independent extensibility of each AI subsystem — a significant advantage over closed-source commercial platforms.

IX. CONCLUSION AND FUTURE WORK

This paper presented Grozo, a production-ready AI-powered online grocery delivery system on the MERN stack. Experimental evaluation demonstrated strong API performance, competitive recommendation quality (CTR +34%), acceptable pricing accuracy (MAPE 7.8%), and positive UX (SUS 79.6).

Planned future enhancements include:

- **Deep learning recommendations:** NCF or BERT4Rec model trained on full order history.
- **Reinforcement learning pricing:** Contextual bandit or Q-learning agent for adaptive price optimisation.

- **Multi-modal search:** Product search by photographing items using a vision model (e.g., Gemini Vision).
- **Federated delivery:** Multi-depot VRP formulation for warehouse-to-hub-to-customer logistics.
- **Cross-platform voice:** OpenAI Whisper or Google Cloud STT replacing browser-native Web Speech API.

X. ACKNOWLEDGMENT

The authors gratefully acknowledge the support of Sri Manakula Vinayagar Engineering College, Puducherry, and the open-source communities behind MongoDB, Express.js, React, Node.js, Leaflet.js, and the Google Gemini API.

REFERENCES

[1] Aggarwal, A. (2018). Web Development with MongoDB and Node.js. Packt Publishing.
[2] Brown, T., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners. NeurIPS 33, 1877–1901.
[3] He, X., Liao, L., Zhang, H., et al. (2017). Neural Collaborative Filtering. Proc. WWW 2017, 173–182.
[4] IBEF. (2024). Indian E-Commerce Industry Report. India Brand Equity Foundation. https://www.ibef.org
[5] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix Factorization for Recommender Systems. IEEE Computer, 42(8), 30–37.
[6] Linden, G., Smith, B., and York, J. (2003). Amazon.com Recommendations: Item-to-Item CF. IEEE Internet Computing, 7(1), 76–80.
[7] Maestre, R., Ramirez, J., Fernandez, F., et al. (2018). RL for Pricing Strategy Optimisation. Eng. Applications of AI, 76, 67–80.
[8] Nielsen, J. (1993). Usability Engineering. Academic Press.
[9] Talluri, K.T. and Van Ryzin, G.J. (2004). The Theory and Practice of Revenue Management. Springer.
[10] Yu, F., Liu, Q., Wu, S., et al. (2016). A Dynamic Recurrent Model for Next Basket Recommendation. SIGIR 2016, 729–732.
[11] Google. (2024). Gemini API Documentation. https://aistudio.google.com
[12] Razorpay. (2024). Payment Gateway Documentation. https://razorpay.com/docs
[13] [13] MongoDB. (2024). MongoDB 7.0 Documentation. https://www.mongodb.com/docs