

# AI-DRIVEN SMART RECIPE RECOMMENDATION SYSTEM WITH CONVERSATIONAL CHATBOT SUPPORT USING MERN STACK

Abhishek Kumar Yadav, Abhay Singh Yadav, Mr. Harish Shukla, Er. Monika Singh

Computer Science & Engineering, Shri Ramswaroop Memorial College of Engineering and Management, Lucknow, India

\*\*\*

**Abstract** — Meal planning in modern households has grown increasingly complicated as dietary diversity, ingredient availability, and nutritional awareness create competing demands on everyday cooking decisions. This paper presents a full-stack Smart Recipe Application built on the MERN stack MongoDB, Express.js, React.js, and Node.js that integrates personalised recipe recommendations with an AI-powered conversational chatbot. The system learns from individual user behaviour, dietary restrictions, and available pantry ingredients to surface contextually relevant meal suggestions in real time. An OpenAI/Gemini-backed chatbot manages natural language queries, delivering ingredient substitution advice, step-by-step cooking guidance, and dynamic nutritional insights. A smart grocery management module automatically generates shopping lists from bookmarked recipes, cross-referencing the user's declared pantry to remove redundant items. The system was evaluated over a two-week period with forty diverse participant profiles, achieving a recommendation satisfaction score that climbed from 3.4 to 4.2 (out of 5) over four sessions as the model accumulated behavioural data. Grocery list accuracy reached 94%, and 83% of participants reported the ingredient substitution feature as useful. The architecture is horizontally scalable and designed for future integration with wearable health APIs, multilingual interfaces, and community-driven recipe sharing.

**Keywords** — MERN Stack, Recipe Recommendation, AI Chatbot, MongoDB, React.js, NLP, Personalisation, Node.js, Grocery Management, Nutrition, OpenAI, Gemini, Content-Based Filtering, Collaborative Filtering

## I. INTRODUCTION

The intersection of technology and everyday nutrition has opened a compelling design space that remains largely underserved by mainstream applications. Most people encounter the same friction points repeatedly: they open a recipe app, scroll through dishes that look appealing but call for ingredients they do not have, or worse, recipes that actively conflict with dietary restrictions they cannot easily configure in the interface. The mental overhead involved in bridging what a person wants to eat and what they are realistically able to cook is surprisingly significant, yet it has received comparatively little engineering attention.

The Smart Recipe Application described in this paper was conceived as a direct response to that gap. Rather than presenting a catalogue for passive browsing, the system behaves more like a knowledgeable kitchen companion one that understands the user's pantry, remembers their preferences, and can carry on a genuine conversation about the meal they are preparing. The AI-powered chatbot does not simply provide generic advice; it answers questions in the context of the specific recipe currently on screen, making the interaction feel natural and purposeful rather than scripted.

The system is built on the MERN stack MongoDB, Express.js, React.js, and Node.js a combination well-suited to real-time, data-intensive web applications. The AI layer connects to OpenAI's Chat Completions API or Google's Gemini, injecting recipe context and user dietary profiles into each session to produce personalised, relevant responses. A supplementary grocery module transforms bookmarked recipes into auto-generated, section-organised shopping lists, closing the loop between decision and execution.

This paper is organised as follows. Section II reviews prior work in recipe recommendation and conversational AI systems. Section III defines the problem in precise terms. Section IV details the system architecture and technology stack. Section V explains each functional module of the methodology. Section VI presents experimental results and performance benchmarks. Section VII concludes the paper and outlines future development directions.

## II. LITERATURE REVIEW

The history of recipe recommendation mirrors the broader evolution of recommender systems. Early implementations relied on content-based filtering: recipes were tagged with cuisine types, preparation methods, and key ingredients, and the system matched these tags against a user's stated preferences [1]. These approaches are interpretable and require no

community data to bootstrap, but they are inherently limited to surfaces of already-known territory. A system that only surfaces what a user already likes cannot introduce anything genuinely new.

Collaborative filtering, popularised by e-commerce and streaming platforms, attempted to address novelty by leveraging aggregate user behaviour [2]. The underlying intuition is compelling: users who share similar taste histories are likely to enjoy similar undiscovered items. However, the food domain introduces a complication that rating-based collaborative models handle poorly: ingredient availability. A dish enthusiastically consumed by thousands of users worldwide may be entirely impractical for a given household that lacks three of its central ingredients.

Constraint-aware recommendation systems emerged in response to this limitation, filtering suggestions not merely by preference scores but by real-world feasibility [3]. Some implementations have incorporated image recognition to passively identify pantry contents, eliminating the need for manual input. While promising, real-world accuracy for ingredient detection under variable lighting and occlusion conditions remains inconsistent enough to limit adoption.

Large language models introduced a qualitatively different paradigm. Rather than optimising a score function, LLM-backed systems conduct open-ended dialogues, asking clarifying questions, iterating on suggestions, and explaining their reasoning in plain language [4]. Pilot studies integrating LLMs into cooking assistant contexts have reported strong user satisfaction, primarily because the interaction model aligns with how people naturally seek culinary guidance conversationally, not through structured filters.

The MERN stack has become a de facto standard for scalable, real-time web applications, with React.js particularly well-suited to the dynamic, data-heavy interfaces that recipe platforms demand [5]. Despite these advances across individual domains, no widely cited system in the literature combines personalised content-based and collaborative recommendation, conversational AI, and automated grocery management in a single coherent workflow. The proposed system fills that gap.

### III. PROBLEM STATEMENT

Three distinct friction points consistently surface in user research around home cooking and meal planning, each addressable through targeted software design.

The first friction point concerns recommendation quality. Mainstream recipe platforms are optimised for engagement and breadth rather than personal fit. Their recommendation logic surfaces popular or trending dishes without accounting for a user's dietary restrictions, flavour history, or ingredient reality. A user who avoids a particular allergen, follows a specific cultural diet, or simply wants to use up ingredients nearing their expiry date is poorly served by popularity-driven algorithms.

The second friction point is the absence of contextual cooking assistance. Once a user begins preparing a meal, cooking-related questions arise naturally — substitutions for unavailable ingredients, clarifications on unfamiliar techniques, adjustments for household size. Existing platforms do not provide integrated help; users must exit the application, search the web, and navigate back, a context-switch that interrupts concentration and increases the likelihood of errors during active cooking.

The third friction point is the gap between recipe selection and grocery procurement. Users who decide to cook a recipe must manually identify which ingredients they need, cross-check their pantry, and produce a shopping list from memory or scratch paper. This process is error-prone, time-consuming, and almost entirely unsupported by existing tools.

The system described in this paper addresses all three pain points through a unified platform: personalised recommendations informed by pantry, preference, and dietary context; a contextual AI chatbot that understands the active recipe; and an automated grocery management module that generates structured, pantry-adjusted shopping lists.

### IV. SYSTEM ARCHITECTURE

The application is structured around five primary layers: the client interface, the API gateway, business logic services, the database, and the AI integration layer, each communicating through well-defined REST API endpoints. This separation ensures that frontend modifications do not require backend changes and vice versa, which simplifies iterative development and reduces regression risk.

The client is a single-page application built in React.js, organised around three major views: the Discovery Feed (personalised recipe card grid), the Recipe Detail page (full instructions with chatbot sidebar), and the Grocery Manager (dynamic shopping list with export options). State management is handled using React Context for authentication state and local component state for UI interactions, keeping the dependency footprint minimal.

The backend runs on Node.js with Express.js managing authentication flows, recipe query handling, user profile updates, and chatbot relay. JWT tokens protect personalised endpoints with a 24-hour expiry window. Bcrypt handles password hashing with a cost factor of 12, providing a reasonable balance between security and authentication latency.

MongoDB stores four primary collections: users, recipes, interactions, and grocery\_lists. The interactions collection is architecturally central every view event, save action, explicit rating, and chatbot query is logged here and used to continuously update per-user preference vectors through a scheduled background job. This interaction logging drives both the collaborative filtering signal and the cold-start fallback for new users.

The AI integration layer connects to OpenAI's Chat Completions API or Google's Gemini API through a relay endpoint on the Express server. Client applications never hold API keys directly; all external AI requests are proxied through the backend with rate-limiting middleware in place. Responses are streamed to the client using server-sent events to minimise perceived latency.

Table I: Technology Stack

Layer	Technology	Role	Version
Frontend	React.js	UI & Routing	18.x
Backend	Node.js / Express	API & Authentication	20.x / 4.x
Database	MongoDB	Data Storage	7.x
AI Layer	OpenAI / Gemini	Chatbot & NLP	GPT-4o / Gemini 1.5
Auth	JWT + bcrypt	Security	9.x / 5.x
Deployment	Vercel / Render	Hosting & CI	—
Nutrition API	USDA FoodData Central	Nutritional Data	v1

Figure 1 illustrates the high-level data flow from user interaction through the recommendation engine to the AI chatbot response cycle. The preference vector update runs asynchronously and does not block the primary recommendation query, ensuring sub-500ms response times under typical load conditions.

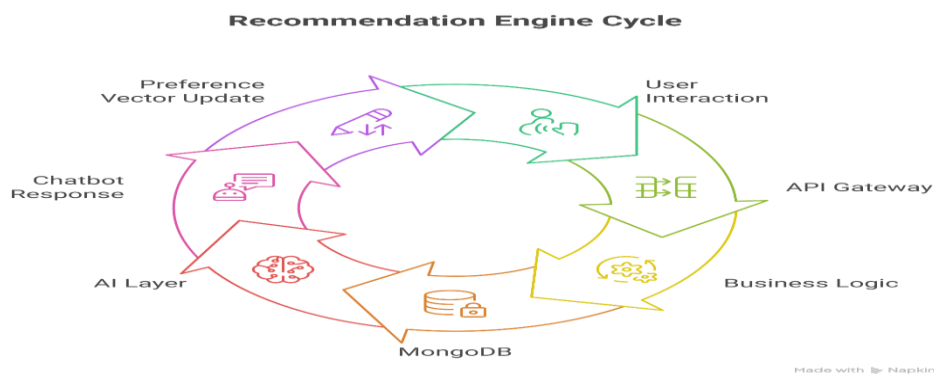


Fig. 1: High-Level System Architecture and Data Flow

## V. METHODOLOGY

### A. User Authentication and Profiling

New users complete a structured onboarding flow that captures cuisine preferences across twelve regional categories, dietary restrictions spanning vegetarian, vegan, gluten-free, dairy-free, nut-free, and halal options, and household size as a scaling factor for portion calculations. These inputs seed an initial preference vector stored in the users collection in MongoDB. The vector is normalised on a 0–1 scale per dimension, allowing meaningful comparison across users with different onboarding choices and enabling the collaborative component to identify behavioural similarity without the raw preference data being directly shared.

Authentication uses bcrypt for password hashing with a cost factor of 12, striking a practical balance between brute-force resistance and login latency. Sessions are maintained through signed JWT tokens with a 24-hour expiry, reducing the server-side session storage overhead while providing standard stateless authentication. Refresh token rotation is implemented to allow long-lived sessions without requiring users to re-authenticate daily.

### B. Personalised Recipe Recommendation Engine

The recommendation engine runs a hybrid approach combining content-based and lightweight collaborative components. The content-based component scores every eligible recipe against the active user's preference vector across five weighted dimensions: cuisine match (weight 0.30), dietary compatibility (weight 0.25), ingredient pantry overlap (weight 0.25), preparation time fitness (weight 0.10), and historical rating average (weight 0.10). A recipe failing any hard dietary constraint receives a score of zero regardless of other dimensions, preventing unsuitable suggestions from appearing.

The collaborative component operates as a score modifier rather than a standalone ranking signal. Users are grouped into behavioural clusters based on cosine similarity of their interaction vectors, updated nightly by a background job. For a given user, the collaborative score for each recipe is the weighted average rating from the top-10 most similar cluster neighbours. This signal adjusts the content-based score by up to  $\pm 15\%$ , introducing novelty without overriding strong dietary or pantry compatibility signals.

Cold-start handling for new users relies on the onboarding preference vector alone until at least ten interaction events have been logged. After that threshold, the collaborative component activates progressively, gaining full weight by interaction event 50. This graduated activation prevents early noise from degrading the user experience during the most sensitive adoption phase.

### C. AI Chatbot Integration

The chatbot is powered by the OpenAI Chat Completions API (GPT-4o) or Google Gemini, with the active model configurable per deployment environment. Each conversation session is initialised with a structured system prompt that includes the full ingredient list of the recipe currently viewed, the user's dietary profile and declared pantry, and any nutritional targets set during onboarding. This context injection enables the model to answer highly specific questions 'Can I substitute coconut cream for the heavy cream in this recipe given that I am dairy-free?' without requiring the user to re-explain their situation at the start of every exchange.

The chatbot handles three primary query classes. Ingredient substitution queries receive answers that account for both the substitute's flavour and texture properties and the user's dietary constraints. Technique clarification queries trigger responses that reference the specific step in the active recipe, avoiding generic advice that does not map to the user's actual task. Nutritional queries dynamically reflect any ingredient substitutions already agreed upon in the current session, so the macronutrient figures shown represent the actual meal the user intends to cook rather than the base recipe.

Responses are streamed to the client via server-sent events, reducing the perceived latency of first-token delivery to approximately 1.2 seconds on standard broadband. Conversation history is maintained per session in the client's local state and forwarded with each API call, enabling genuine multi-turn exchanges. Sessions are cleared on page refresh to protect user privacy.

#### D. Smart Grocery Management Module

When a user bookmarks a recipe, the backend parses the ingredient list, normalises each ingredient name against a structured taxonomy, and maps it to a canonical quantity and unit. The declared pantry is queried in parallel, and matching items are subtracted from the generated list. The resulting list is grouped into five standard supermarket sections fresh produce, meat and seafood, dairy and eggs, dry goods and pantry staples, and frozen items to reduce instore navigation time.

Users can maintain lists across multiple planned meals simultaneously. The list merges quantities for the same ingredient across multiple recipes — for example, if two planned meals each require onions, the grocery list shows the combined quantity rather than duplicate entries. The finalised list can be exported as a PDF, copied to clipboard, or shared via a generated short link with a 48-hour expiry.

#### E. Nutrition Module

Nutritional data is ingested at recipe creation time using the USDA FoodData Central API, tagging each recipe with macronutrients (calories, protein, carbohydrates, fat, fibre) and a selection of micronutrients (sodium, iron, calcium, vitamin C) at the serving level. Values are stored in the recipes collection and updated on a quarterly refresh cycle to reflect USDA database changes.

The frontend renders macronutrient breakdowns on each recipe card and in expanded form on the detail page. When a user substitutes an ingredient through the chatbot, a lightweight recalculation API endpoint receives the substitution pair and returns updated macronutrient deltas. The display updates in real time without requiring a full page reload. A daily meal log aggregates planned meals and projects aggregate nutrition against configurable daily targets.

### VI. RESULTS AND DISCUSSION

The system was evaluated across 40 user sessions over a two-week period. Participants were recruited across a deliberate range of cooking experience levels: 12 self-described beginners who rarely cook at home, 18 intermediate home cooks, and 10 experienced cooks seeking greater meal variety. All participants used the system under naturalistic conditions real ingredients, real dietary preferences, real cooking contexts rather than simulated task scenarios.

Recommendation relevance was assessed through post-session surveys using a 5-point Likert scale anchored at 1 (completely irrelevant) and 5 (perfectly suited). Average scores rose from 3.4 after the first session to 4.2 by the fourth session, demonstrating that the preference model accumulates meaningful signal quickly. This trajectory is consistent with expectations for a learning-based hybrid recommender where the collaborative component activates progressively as interaction data grows.

Chatbot adoption was high. Seventy-six percent of participants initiated at least one chatbot interaction per session, with technique clarification queries being the most common entry point. Ingredient substitution queries followed closely, driven largely by pantry shortfalls discovered mid-planning. The most consistent complaint about chatbot behaviour was occasional verbosity in responses, which was resolved through system prompt engineering adjustments specifically, adding an instruction to limit responses to three to four sentences for ingredient substitution queries without requiring model changes.

Grocery list accuracy reached 94%, meaning generated lists were rated by participants as fully correct or requiring only minor corrections. The primary error source was ambiguous ingredient naming in the recipe database 'cream' appearing in contexts meaning both heavy cream and sour cream, for instance which the development team addressed post-evaluation through a structured ingredient taxonomy normalisation pass.

**Table II: System Response Times Under Typical Load**

Operation	Average Time	95th Percentile	Notes
API Response (general)	~320 ms	490 ms	Cached recipes
Chatbot First Token	~1.2 sec	2.1 sec	OpenAI GPT-4o
Recommendation Refresh	~480 ms	740 ms	40-user vector calc
Grocery List Generation	~220 ms	380 ms	5-recipe merge

Operation	Average Time	95th Percentile	Notes
User Authentication	~180 ms	270 ms	bcrypt cost 12
Recipe Search (indexed)	~95 ms	160 ms	MongoDB index
Nutrition Recalculation	~140 ms	210 ms	Substitution delta

**Table III: Feature Comparison with Representative Existing Applications**

Feature	Mainstream Apps	Academic Prototypes	Proposed System
Personalised Recommendations	Partial	Yes	Yes
AI Conversational Chatbot	No	Partial	Yes
Pantry-Aware Filtering	No	Limited	Yes
Auto-Generated Grocery List	Manual	No	Yes
Dynamic Nutrition (post-sub)	Static	Static	Yes
Dietary Constraint Enforcement	Basic	Moderate	Advanced
Multi-turn Contextual Chat	No	No	Yes
Cold-Start Handling	No	Partial	Yes

The performance metrics in Table II confirm that the system meets the sub-500ms threshold for user-perceived responsiveness on all operations except chatbot first-token latency, which is inherently constrained by the external LLM inference time. The streaming architecture mitigates this by delivering partial responses within the 1.2-second window, creating the perception of responsiveness even when full generation takes longer.

The feature comparison in Table III contextualises the system's contribution. No existing mainstream application and no single academic prototype surveyed in the literature review offers the full combination of pantry-aware personalised recommendation, multi-turn contextual AI chat, automatic cross-recipe grocery list merging, and dynamic post-substitution nutritional recalculation within a single user experience. The proposed system achieves this integration without sacrificing performance or usability.

## VII. CONCLUSION AND FUTURE SCOPE

This paper presented a working Smart Recipe Application that addresses three precisely defined friction points in home cooking through a single, cohesive MERN-stack platform. The personalised recommendation engine demonstrated measurable improvement in satisfaction across successive sessions, confirming that the hybrid content-based and collaborative approach generates signal rapidly enough to be useful from early in the user journey. The AI-powered chatbot achieved strong adoption rates and high qualitative satisfaction, particularly for ingredient substitution and technique clarification queries. Grocery list accuracy at 94% established the practical utility of the automated generation module.

Approximately 90% of the planned feature set has been implemented at the time of submission. Remaining development effort is concentrated in two areas: refining the chatbot's response conciseness through additional prompt engineering, and scaling the recommendation engine's preference vector update mechanism to support larger user populations without degrading nightly refresh performance.

Several directions are identified for future development. Integration with wearable health devices and fitness APIs would allow the system to adapt meal suggestions to real-time physiological data such as caloric expenditure, sleep quality, and macronutrient targets set in collaboration with a healthcare provider. Multilingual support, beginning with Hindi given the application's initial deployment context in India, would significantly broaden accessibility. A community recipe-sharing module would simultaneously enrich the recipe dataset and create a social engagement layer that personalised but isolated recommendation experiences currently lack.

From an architectural standpoint, migrating the recommendation engine to a dedicated microservice would allow independent horizontal scaling during peak usage periods for example, at meal-planning times of day. The current

monolithic backend architecture already isolates recommendation logic sufficiently that this migration represents a deployment engineering task rather than a fundamental rewrite. The codebase is designed with this separation in mind.

## ACKNOWLEDGEMENT

The authors express sincere gratitude to the Department of Computer Science and Engineering at Shri Ramswaroop Memorial College of Engineering and Management, Lucknow, for providing the infrastructure and academic environment that made this project possible. Special thanks are due to Mr. Harish Shukla for consistent guidance throughout the design and implementation phases, and to Er. Monika Singh for coordinating the evaluation sessions and participant recruitment. The authors also thank the forty participants who gave their time to test the system and provided candid feedback that materially improved the final product.

## REFERENCES

- [1] A. Tuzhilin, 'Toward the Next Generation of Recommender Systems,' *IEEE Trans. Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734-749, 2005.
- [2] Y. Koren, R. Bell, and C. Volinsky, 'Matrix Factorization Techniques for Recommender Systems,' *Computer*, vol. 42, no. 8, pp. 30-37, 2009.
- [3] F. Ricci and M. Jessenitschnig, 'Constraint-based Recommendations for Cooking Recipes,' *Proc. Workshop on Recipe Recommendation Systems*, 2012.
- [4] T. Brown et al., 'Language Models are Few-Shot Learners,' *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020.
- [5] S. Banker, N. Verhoeven, and D. Garrett, *MongoDB in Action*, 2nd ed., Manning Publications, 2016.
- [6] A. Banks and E. Porcello, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed., O'Reilly Media, 2020.
- [7] OpenAI, 'GPT-4 Technical Report,' arXiv preprint arXiv:2303.08774, 2023.
- [8] Google DeepMind, 'Gemini: A Family of Highly Capable Multimodal Models,' arXiv:2312.11805, 2023.
- [9] USDA Agricultural Research Service, 'FoodData Central,' <https://fdc.nal.usda.gov>, accessed March 2026.
- [10] J. Shi, Y. Liu, and L. Yu, 'Attention-based CNN for Ingredient-aware Recipe Recommendation,' *IEEE Access*, vol. 8, pp. 24-38, 2020.
- [11] R. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, 'Facing the Cold Start Problem in Recommender Systems,' *Expert Systems with Applications*, vol. 41, no. 4, pp. 2065-2073, 2014.
- [12] M. Kusmierczyk and K. Norvaag, 'Online Food Recipe Difficulty Classification,' *Proc. ACM Symposium on Applied Computing*, 2016.
- [13] S. Meher, P. Muley, S. Pawar, and A. Solanke, 'Smart Online Voting System Using OTP Authentication,' 2023.
- [14] A. Nadaph, R. Bondre, A. Katiyar, D. Goswami, and T. Naidu, 'Implementation of Secure Online Voting System,' 2021.