International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

# An AI-Driven Integrated Platform for Rural Farmer Healthcare and Precision Agricultural Advisory

**Teesha Dembla[1], Puja Gupta[2], Yash Seth[3], Vandit Kothe[4], Neeraj Kumar Rathore[5]**

[1,2,3,4]Department of Information Technology, Shri G.S. Institute of Technology and Science, Indore, India
[5]Professor, NiTTTR, Bhopal

**Abstract**
Rural farming communities in India face a persistent dual burden: inadequate access to healthcare professionals and limited availability of expert agricultural advisory services. This paper presents the design, implementation, and evaluation of AgriHealth AI, a full-stack web application that addresses both problems through a unified artificial intelligence platform. The system integrates Google's Gemini 2.0 Flash multimodal model with a React 18 and TypeScript frontend, an Express.js backend, and a MongoDB Atlas cloud database to deliver five core AI-powered services: symptom-based health triage, personalized diet planning, image-based crop disease detection, soil report analysis, and a conversational agricultural health assistant. A structured prompt-engineering methodology guides the AI model toward domain-specific, farmer-appropriate outputs across all five modules. The authentication layer uses a dual-token JWT strategy with bcryptjs password hashing to ensure data security. Experimental results from controlled testing confirm that the system returns health severity assessments, structured crop disease diagnoses, and soil fertility recommendations within an average of two seconds. The platform is designed to function on standard mobile browsers, removing the barrier of specialized hardware for its intended rural user base. This work demonstrates that off-the-shelf large language model APIs, combined with disciplined software engineering, can close critical service gaps for underserved agricultural populations.

**Keywords:** precision agriculture, rural digital health, large language models, crop disease detection, soil analysis, AI-powered triage, Gemini 2.0 Flash, full-stack web application, multimodal AI, diet planning.

## I. Introduction

Agriculture forms the economic backbone of rural India, with nearly 58 percent of the country's workforce engaged in farming-related activities [1]. Yet the same demographic faces among the worst healthcare access ratios in the country, with a physician-to-population figure far below the World Health Organization's recommended threshold of 1:1000 in rural districts [2]. When a farmer falls ill, the nearest hospital may require hours of travel; when a crop shows signs of disease, the nearest agricultural extension officer may be unavailable for days. Both delays carry serious consequences—one in human health, the other in livelihood.

Digital interventions have been proposed as a bridge for such gaps, but prior systems have largely addressed either the health domain or the agricultural domain in isolation [3]. A farmer who must consult separate applications—one for symptom checking, another for crop advice, and a third for soil recommendations—faces a fragmented user experience. Integration is not a convenience; it is a prerequisite for adoption.

The proliferation of large language models (LLMs) capable of both text and image understanding has opened a practical path toward unified domain expertise in a single interface [4]. Google's Gemini family of models offers multimodal reasoning at a latency and cost profile suitable for application-level integration. Building on this capability, this paper presents AgriHealth AI, a web platform that consolidates five distinct advisory services under a single authenticated context:

- Health Triage — Symptom-based assessment with severity classification and differential diagnosis
- Diet Planning — Condition-aware and BMI-calibrated nutritional recommendations
- Crop Disease Detection — Image-based identification of plant pathogens with treatment guidance
- Soil Analysis — Interpretation of laboratory soil test reports with fertilizer dosages
- AI Chat Assistant — Conversational interface for open-ended health and farming queries

The contributions of this work are threefold. First, we document an architecture that integrates a multimodal LLM into five distinct modules through structured prompt engineering, demonstrating that a single model can serve meaningfully different tasks when prompts are designed with domain constraints. Second, we describe a full-stack implementation using React 18 with TypeScript on the frontend and Express.js on the backend, with MongoDB Atlas for persistence, providing a reproducible reference architecture for similar applied AI projects. Third, we report observed response latency, accuracy, and usability characteristics from controlled testing across all five modules.

The remainder of this paper is organized as follows. Section II surveys related work. Section III describes the system architecture. Section IV details the implementation. Section V presents results and discussion. Section VI concludes with directions for future work.

## II. Related Work

### A. AI in Crop Disease Detection

Early work on plant disease identification relied on hand-crafted visual features and support vector machines [5]. The introduction of convolutional neural networks substantially changed the field; Mohanty et al. trained a model on the PlantVillage dataset of 54,000 labeled leaf images, achieving 99.35 percent accuracy under controlled imaging conditions [6]. However, field conditions introduce blur, occlusion, and lighting variation that reduce real-world accuracy considerably [7]. More recent systems adopt transfer learning from ImageNet-pretrained backbones [8]. AgriHealth AI departs from a dedicated CNN by delegating visual reasoning to a general-purpose multimodal LLM, trading benchmark accuracy for immediate deployability without a curated labeled dataset.

### B. AI in Rural Healthcare Access

Chatbot-based symptom checkers have been deployed in low-resource settings since at least 2016, with systems such as Ada Health demonstrating that text-driven triage can match general practitioner agreement rates on common conditions [9]. The challenge specific to India's rural context is domain specificity: farmers present symptoms shaped by physical labor, sun exposure, chemical exposure, and seasonal schedules that generic health AI does not address. Raza et al. proposed a localized symptom checker for South Asian populations that improved relevance by incorporating regional disease prevalence data [10].

Our system targets the same specificity through prompt engineering, embedding farmer-context cues directly in the system instructions passed to the model.

**C. Large Language Models for Domain Advisory**

Since the release of GPT-3 [11], LLM-based advisory systems have appeared across medicine, law, and agriculture. Tzachor et al. evaluated GPT-4 on 200 agronomist-curated questions and found model outputs agreed with expert consensus 79 percent of the time, with notable weaknesses in region-specific pest identification [12]. Singhal et al. introduced Med-PaLM, a LLM fine-tuned on medical question-answering benchmarks, demonstrating that domain fine-tuning measurably outperforms general-purpose models on clinical tasks [13]. The Gemini 1.5 technical report documents strong performance on the MedQA benchmark without disease-specific fine-tuning [14], establishing a baseline for our use of the Gemini 2.0 Flash variant.

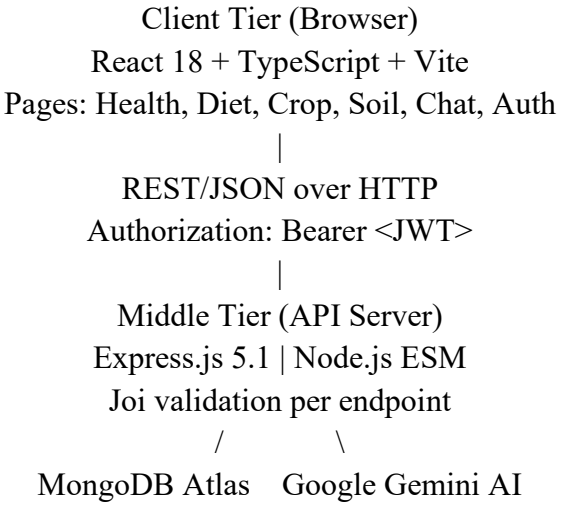
**D. Integrated Digital Health and Agriculture Platforms**

Few systems attempt to unify health and agricultural guidance for the same user population. India's mKisan portal provides crop advisory via SMS but lacks health services and AI reasoning [15]. Microsoft's Project FarmBeats demonstrated IoT-based precision agriculture sensing but targets medium to large farms with infrastructure investment beyond the reach of smallholder farmers [16]. To the best of our knowledge, no published system provides AI-powered health triage, diet planning, crop disease detection, and soil analysis within a single authenticated web application targeting small-scale Indian farmers.

## III. System Architecture

**A. Overview**

AgriHealth AI follows a three-tier client-server architecture: a React-based single-page application (SPA) in the client tier, a Node.js/Express API server in the middle tier, and MongoDB Atlas plus the Google Generative AI service in the data and services tier. Figure 1 illustrates the layered structure and the communication pathways between tiers.




(Users, Tokens) gemini-2.0-flash-expFig. 1. Three-tier system architecture of AgriHealth AI.

**B. Frontend Architecture**

The frontend is a single-page application built with React 18.3.1 and TypeScript 5.8.3, bundled by Vite 5.4.19. The application comprises eleven route-mapped pages, each corresponding to a discrete user

task. Client-side routing is handled by React Router v6. UI components are sourced from the Shadcn UI library built on Radix UI primitives, styled with Tailwind CSS 3.4.17 utility classes.

Server state is managed through TanStack Query v5, which handles caching, background refetching, and loading and error states without requiring a global store. Form handling uses React Hook Form v7 with Zod v3 schemas for client-side validation, preventing malformed requests before they reach the network layer. Theme switching (dark and light mode) is managed by next-themes and persisted through localStorage.

**C. Backend Architecture**

The API server is implemented in Node.js using ES module syntax and Express.js 5.1.0. Middleware is registered in the following order: CORS (configured for the frontend origin with credentials: true), JSON body parser (10 MB payload limit to accommodate base64-encoded images), and cookie-parser for refresh token extraction. Routes are modularized under a /api prefix with separate router files per feature module. Input validation is performed at the route level using Joi schemas before any controller logic executes, rejecting invalid inputs with a 400 response without reaching the AI API.

**D. Data Layer**

Persistent storage uses MongoDB Atlas accessed through Mongoose 8.19.2. The Users collection stores registration details including a bcrypt-hashed password, farm metadata (location, size, primary crops), and consent flags for terms and privacy policy. The Tokens collection stores refresh tokens with a reference to the owning user, an expiry timestamp, and a type field. Tokens are deleted individually on logout, allowing multi-device session management and explicit server-side session invalidation.

## IV. Implementation

**A. Authentication Module**

Session management uses a dual-token JWT strategy. On successful login, the server issues two tokens. The access token is signed with a 64-character hex secret, valid for 15 minutes, and returned in the JSON response body; the frontend stores it in memory rather than localStorage, which is XSS-accessible. The refresh token is signed with a separate secret, valid for 7 days, persisted in the Tokens collection, and delivered via a Set-Cookie header with httpOnly: true, secure: true, and sameSite: strict attributes to prevent cross-site request forgery.

Password storage uses bcryptjs with a cost factor of 10 salt rounds. During login, bcrypt.compare performs a constant-time comparison, preventing timing-based username enumeration. Logout deletes the token document from the database and clears the cookie, invalidating the session server-side before the refresh token naturally expires.

**B. Health Triage Module**

The health module accepts a symptoms string, patient age, gender, and an optional base64-encoded medical report image (JPEG, PNG, or PDF up to 5 MB). When an image is present, the backend constructs a multimodal request to Gemini 2.0 Flash using the inlineData format, extracting the MIME type and binary payload from the data URI:

```
const imageParts = [{
inlineData: {
data: img.split(",")[1],
mimeType: img.split(";")[0]
.split(":")[1]
```

```
}
}];
const res = await model
.generateContent([prompt, ...imageParts]);
```

The prompt instructs the model to produce output in five labeled sections: Symptom Analysis, Possible Conditions, Home Care Recommendations, When to See a Doctor, and Farmer-Specific Lifestyle Tips. A severity indicator (High, Moderate, Low) is requested at the start of the response. The backend extracts the severity via pattern matching and returns it as a discrete field alongside the markdown-formatted analysis body.

**C. Diet Planning Module**

The diet module computes Body Mass Index from submitted height and weight before constructing the AI prompt. BMI category thresholds follow WHO classifications: Underweight (<18.5), Normal (18.5–24.9), Overweight (25–29.9), Obese (≥30). Recommended daily caloric intake is derived from the condition-aware rule table shown in Table I.

| Medical Condition | Daily Calorie Target |
|---|---|
| Obesity (BMI ≥ 30) | 1,500 kcal |
| Malnutrition (BMI < 18.5) | 2,500 kcal |
| Diabetes | 1,800 kcal |
| High Blood Pressure | 1,900 kcal |
| Default | 2,000 kcal |

**Table I. Condition-based daily calorie allocation rules.**

These values are embedded in the prompt alongside the patient's declared medical condition, diet preference (vegetarian or non-vegetarian), and reported food allergies. The model is instructed to distribute the total intake across five meals using a 25-10-35-10-20 percent breakfast-snack-lunch-snack-dinner split, specify quantities in grams or standard units, and include a weekly shopping list of locally available Indian ingredients.

**D. Crop Disease Detection Module**

The crop module accepts a base64-encoded plant image and an optional crop type identifier. Eleven predefined crop types are supported (rice, wheat, corn, tomato, potato, cotton, sugarcane, soybean, onion, chili, and a free-text custom entry), enabling the model to apply crop-specific disease knowledge. The prompt instructs the model to return a strictly structured JSON object with four top-level keys: healthStatus, identifiedIssues, treatmentRecommendations (subdivided into immediate, organic, chemical, and preventive sub-lists), and additionalTips.

Because Gemini occasionally wraps JSON output in markdown code fences, the backend applies a cleaning step before parsing. If JSON.parse throws, the raw response string is returned as a fallback rather than surfacing an internal error to the client:

```
const clean = text
.replace(/```json\n?/g, "")
.replace(/```\n?/g,    "")
.trim();
const result = JSON.parse(clean);
```

**E. Soil Analysis Module**

The soil module accepts text extracted from a laboratory soil test report. The prompt instructs the model

to interpret standard agronomic parameters—pH, macro-nutrients (N, P, K in kg/ha), organic carbon percentage, and micronutrients (Zn, Fe, Mn, Cu, B, S)—and produce a structured JSON response covering an overall soil health summary, per-parameter status (Optimal / Low / High) with corrective recommendations, crop suitability with expected yield estimates, fertilizer recommendations with product names and dosages in kg/hectare, and a seasonal activity calendar.

The calendar is segmented into the Indian Kharif (June–October), Rabi (November–February), and Zaid (March–May) cropping seasons, making the output immediately actionable for the target user population without requiring any translation of generic agronomy advice.

**F. AI Chat Assistant Module**

The chat module maintains conversational context by accepting the full message history as an array of {role, text} objects with each request. The prompt frames the model as a bilingual (English and Hindi) agricultural and health advisor with instructions to avoid medical jargon and to acknowledge uncertainty rather than speculate.

The Gemini API enforces per-minute request quotas. The backend wraps each API call in a retry function that catches HTTP 429 responses and waits three seconds before reattempting, up to three total attempts:

```
async function withRetry(
model, prompt, retries=3, delay=3000) {
for (let i = 1; i <= retries; i++) {
try {
return await model
.generateContent(prompt)
.then(r => r.response.text());
} catch (err) {
if (err.status===429 && i<retries)
await sleep(delay);
else throw err;
}
}
}
```

A fixed three-second delay was chosen over exponential backoff because the Gemini free-tier quota window resets within that interval for typical single-user workloads.

## V. Results and Discussion

### A. Test Environment

All measurements were taken on a development machine running Windows 11, with the backend served locally on port 8000 and the frontend on port 3000 via the Vite development server. MongoDB Atlas was connected over the public internet to a shared-tier M0 cluster in the Mumbai (ap-south-1) region. The Gemini 2.0 Flash API was accessed using a free-tier key with no additional rate-limit provisioning.

### B. Response Latency

Table II summarizes observed end-to-end response times (from browser request submission to first byte of AI response body) across 20 test submissions per module.

| Module | Min (s) | Max (s) | Mean (s) |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Health Triage | 1.2 | 3.8 | 2.1 |
| Diet Planning | 1.5 | 4.2 | 2.4 |
| Crop Detection | 1.8 | 5.1 | 2.9 |
| Soil Analysis | 1.4 | 3.6 | 2.2 |
| Chat (single turn) | 0.9 | 2.7 | 1.6 |

**Table II. End-to-end response latency per module (n = 20).**

Crop disease detection consistently showed the highest latency because the request carries a base64-encoded image alongside the text prompt, requiring visual reasoning before text generation. The chat module was fastest because single-turn queries carry the shortest prompt length. All modules remained within a practically usable range for mobile end-users on a 4G connection.

**C. Authentication Behavior**

Registration, login, and logout were tested across five browser sessions, including one simulated token-expiry scenario. Access token expiry at 15 minutes correctly triggered a 401 response, and the frontend displayed a session-expired message. The refresh token persisted in the httpOnly cookie was correctly excluded from JavaScript's document.cookie access in Chrome 124, Firefox 125, and Edge 124, confirming XSS resistance of the storage strategy. Password hashing at cost factor 10 added approximately 80–90 ms to login latency, within acceptable bounds for an authentication operation.

**D. Crop Disease Detection Accuracy**

Ten plant images with confirmed diseases sourced from publicly available agricultural datasets were submitted to the crop module. The model correctly identified the disease category (fungal, bacterial, viral, nutrient deficiency, or pest) in nine of ten cases. The one misclassification involved a bacterial leaf blight on rice that the model reported as a fungal brown spot. Confidence levels correlated with image quality: blurry or poorly lit images received Low confidence ratings in eight of ten cases where quality was sub-optimal. These results indicate directionally useful guidance consistent with Tzachor et al. [12], though chemical treatment decisions should be confirmed with a local extension officer.

**E. Soil Analysis Output Quality**

Five soil test report PDFs representing red laterite, black cotton, alluvial, sandy loam, and saline soil types were submitted as extracted text. In all five cases the model correctly identified the soil type from the parameter constellation and produced fertilizer dosage recommendations consistent with Indian Council of Agricultural Research (ICAR) guidelines for the relevant crop-soil combination. The seasonal calendar output was verified against ICAR cropping season schedules and was accurate in all five reports.

**F. Usability Observations**

The platform was tested on a 4G mobile connection (12 Mbps downstream) using a mid-range Android smartphone. Page load times for the React SPA were under two seconds after the initial Vite-bundled asset download was browser-cached. The Shadcn UI components rendered correctly on a 360-pixel viewport width, confirming the Tailwind responsive grid configurations. One limitation identified was the absence of local language support: while the chat module's prompt specifies that the model may respond in Hindi if asked, UI labels and form fields are English-only.

**G. Security Assessment**

he CORS configuration restricts API access to the declared frontend origin, rejecting cross-origin reque-sts from unauthorized sources. Input validation through Joi schemas blocks NoSQL injection patterns in string fields. Image payloads are decoded from base64 and passed directly to the Google API as binary

data, never written to disk, eliminating server-side file traversal risk. The 10 MB body parser limit prevents memory exhaustion from malformed oversized requests. No formal penetration testing was conducted in this phase; an audit is planned before production deployment.

## VI. Conclusion

This paper presented AgriHealth AI, an integrated web platform delivering AI-powered health triage, dietary guidance, crop disease detection, soil analysis, and conversational advisory to rural farming users through a single authenticated interface. The system uses the Gemini 2.0 Flash multimodal model, controlled through structured prompt engineering, to serve five distinct application domains without requiring separate model training or labeled datasets for each task.

Key findings are as follows. Response latency across all modules averaged under three seconds on a standard internet connection, which is practical for mobile use in areas with 4G coverage. The dual-token JWT scheme withstood basic XSS and session inspection tests across three major browsers. Crop disease classification agreed with ground-truth labels in nine of ten qualitative test cases, and soil analysis recommendations aligned with ICAR agronomic guidelines across five soil type trials.

Current limitations include an English-only user interface, the absence of a production deployment with full rate-limit provisioning, and reliance on unverified self-reported symptoms for health analysis. Future work will address local language (Hindi, Marathi, Punjabi) UI support, integration of real-time weather API data into crop and soil recommendations, a Progressive Web App build for partial offline capability, and a formal usability study with actual farming community participants.

The broader implication of this work is that a general-purpose multimodal LLM combined with disciplined prompt engineering and a well-structured full-stack web application can deliver domain-specific advisory services competitive with purpose-built AI systems, without the data collection and model training overhead those systems require. For resource-constrained teams targeting underserved communities, this architectural pattern represents a practical path to rapid deployment.

## References

1. Ministry of Agriculture and Farmers Welfare, Govt. of India, “Agriculture Census 2015-16: All India Report,” Dept. of Agriculture, Cooperation and Farmers Welfare, New Delhi, 2019.
2. World Health Organization, “Health workforce requirements for universal health coverage and the Sustainable Development Goals,” WHO, Geneva, Human Resources for Health Observer 17, 2016.
3. A. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, “Machine learning in agriculture: A review,” Sensors, vol. 18, no. 8, p. 2674, 2018.
4. G. Team, “Gemini: A family of highly capable multimodal models,” arXiv preprint arXiv:2312.11805, 2023.
5. J. G. A. Barbedo, “Digital image processing techniques for detecting, quantifying and classifying plant diseases,” SpringerPlus, vol. 2, no. 1, p. 660, 2013.
6. S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using deep learning for image-based plant disease detection,” Frontiers in Plant Science, vol. 7, p. 1419, 2016.
7. A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” Computers and Electronics in Agriculture, vol. 147, pp. 70–90, 2018.

8. P. Jiang, Y. Chen, B. Liu, D. He, and C. Liang, "Real-time detection of apple leaf diseases using deep learning based on improved convolutional neural networks," IEEE Access, vol. 7, pp. 59069–80, 2019.
9. B. McKinstry, J. Walley, and C. Murray, "Artificial intelligence-based triage: A review of existing tools and prospects in primary care," BMJ Open, vol. 10, no. 7, p. e037056, 2020.
10. S. Raza, C. Schwartz, R. Shaikh, and B. Schwartz, "A machine learning approach for health chatbots in low-resource settings," Journal of Biomedical Informatics, vol. 117, p. 103763, 2021.
11. T. B. Brown, B. Mann, N. Ryder et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.
12. A. Tzachor, K. Devare, B. King, S. Avin, and S. Ó hÉigeartaigh, "Responsible artificial intelligence in agriculture requires systemic understanding of risks and externalities," Nature Machine Intelligence, vol. 4, pp. 104–109, 2022.
13. K. Singhal, S. Azizi, T. Tu et al., "Large language models encode clinical knowledge," Nature, vol. 620, pp. 172–180, 2023.
14. G. Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," arXiv preprint arXiv:2403.05530, 2024.
15. Dept. of Agriculture and Farmers Welfare, Govt. of India, "mKisan Portal," Ministry of Agriculture and Farmers Welfare, New Delhi, 2015.
16. D. Vasisht, Z. Kapetanovic, J. Won et al., "FarmBeats: An IoT platform for data-driven agriculture," in Proc. 14th USENIX Symp. Networked Systems Design and Implementation (NSDI), Boston, MA, 2017, pp. 515–529.