Krishi Samparkha: A Multilingual, Intelligent Agricultural Ecosystem & Direct Farmer-Market Platformfor Farmer Advisory Support

Divya C [1], Rajashekar K J[2], Rithu T K[3], Spoorthi B[4], T D Rakshitha[5], Varsha A L[6]
[1,2,3,4,5,6]*Department of Information Science and Engineering, Kalpataru Institute of Technology, Tiptur, Karnataka, India*

***Abstract*—Getting timely farming advice is still hard if you're a small or marginal farmer in rural India. Part of it is language, part of it is that a lot of farmers aren't comfortable navigating a complicated app, and part of it is that there simply aren't enough extension officers to go around. This paper describes KrishiSamparkha, a web-based assistant that tries to chip away at that problem by pairing a large language model with multilingual text, voice, and image input. It runs on a Flask backend with SQLite behind it. A farmer can register, chat in whichever Indian language they're most comfortable with, send a photo of a sick-looking crop for a first opinion on what might be wrong with it, and have the answer read out loud instead of having to read it. Chat sessions persist, so someone can come back a week later and still find the earlier advice. We walk through the architecture, the core algorithms, and the implementation of the system's four main modules authentication, conversational AI, image analysis, and voice output and report on unit, integration, system, and user-acceptance testing. Every functional test case we defined passed, and because the modules are fairly loosely coupled, a slow response from the AI service never took down anything else. The overall picture suggests that stitching conversational AI together with multilingual voice and image support is a workable, low-cost route toward reaching farmers that most existing digital tools don't quite serve.**

***Index Terms*—agricultural AI, chatbot, Flask, multilingual NLP, crop disease detection, text-to-speech, digital agriculture**

I. INTRODUCTION

Agriculture employs a huge share of India's workforce, and yet the sector keeps running into the same obstacles unpredictable rainfall, crop disease, pest outbreaks, and, underneath all of that, not nearly enough expert guidance reaching small and marginal farmers. Extension services technically exist, but coverage is patchy at best, and a farmer in a remote village often has no realistic way to get an agronomist's opinion the day a crop problem shows up. At the same time, smartphones and mobile data have spread across rural India far faster than most people expected even a decade ago, which is exactly what makes software-based advisory tools worth building now.

This is roughly where AI, and large language models in particular, can help. A conversational system that actually understands a farmer's question, replies in that farmer's own language, and combined with basic image recognition can offer a first read on a damaged leaf without anyone physically visiting the field, closes a real gap. Whether such a system is actually useful in practice comes down to a few unglamorous things: does it work in the local language, does it assume the user reads comfortably (it shouldn't), and is it cheap enough to actually deploy at scale.

KrishiSamparkha was built with those constraints in mind. It's a Flask-based web application where a farmer can register, talk to an AI assistant by text or voice, upload a crop photo for a preliminary diagnosis, and hear the reply read back. Every chat session is saved, so past advice doesn't just disappear. The rest of the paper is organised as follows: Section II looks at related work in agricultural chatbots, Section III lays out the system architecture and its four core modules, Section IV covers implementation and the underlying algorithms, Section V reports how testing went, and Section VI closes with conclusions and what we'd want to add next.

II. RELATED WORK

A few existing tools already try to bring agricultural expertise closer to farmers, though each one only really covers part of the problem. mKrishi, from TCS, connects farmers to expert advice but doesn't do much for language diversity or back-and-forth conversation. Kisan Suvidha handles weather and market-price updates reasonably well, but there's no AI layer nothing personalised about the guidance it gives. Plantix is fairly good at spotting disease from a leaf photo, though its regional-language support is thin and it has no general chatbot for the kind of open-ended question a farmer might actually want answered.

The research side tells a similar story. Pravinkrishnan et al. [1] surveyed chatbot architectures used in agriculture and found both rule-based and ML-based systems capable of supporting tasks like crop management and pest control, but they were candid about the open problems language barriers, unreliable connectivity, and the expectation of real-time answers. Vijayalakshmi and Meena [2] built an NLP-based chatbot that handled irrigation and fertiliser questions well, but it never touched multilingual support or images at all. Bang et al. [3] worked on chatbot memory to make long conversations feel more coherent over time, which is a genuinely useful idea for something like this that a farmer might keep coming back to, even though that work wasn't built with any particular domain in mind. Haller and Rebedea [4] built a dialogue-based educational chatbot that worked well for teaching history but has no obvious path into agriculture. Du Preez et al. [5], on the other hand, built a voice-enabled chatbot using keyword extraction and found that voice interfaces genuinely help users with limited literacy an observation that fed directly into how we designed the voice module here.

Table I. Comparative summary of related systems

Ref.	System	Strength	Limitation
[1]	Chatbot survey	Broad comparison	No deployable system
[2]	NLP chatbot	Accurate queries	No multilingual/image
[3]	Chatbot memory	Personalised dialogue	Not domain-specific
[4]	Edu. chatbot	Engaging learning	Not for agriculture
[5]	Voice chatbot	Low-literacy access	Weak context

Taken together, none of these systems really bring multilingual conversation, image-based diagnosis, and voice accessibility into one lightweight tool that's actually deployable at low cost. That's the gap KrishiSamparkha is aimed at.

III. PROPOSED SYSTEM

KrishiSamparkha follows a client–server split, mostly for the usual reasons it's easier to scale, secure, and maintain when the interface and the backend processing aren't tangled together. The client side stays deliberately thin, just HTML, CSS, and JavaScript for collecting input and showing responses back. All the actual work happens server-side, inside a Flask backend organised into five cooperating modules, shown in Fig. 1.

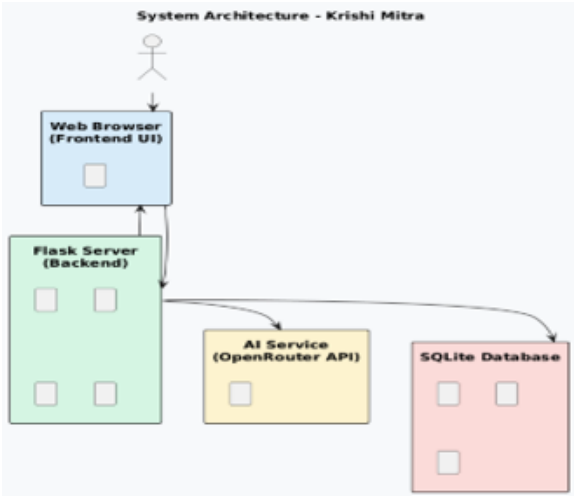

Fig. 1. System architecture of KrishiSamparkha.

A. Authentication Module

New users sign up with an email address, username, and password. A verification code goes out over SMTP, and login isn't allowed until that code gets confirmed. Sessions run through Flask's built-in session handling, so an expired or logged-out session simply can't reach anything it shouldn't.

B. Chat Management Module

Each user can keep several independent chat sessions going at once, each with its own name and timestamp. Sessions can be created, renamed, switched between, or deleted, and every exchange gets written to the database as it happens so a farmer's earlier questions, and whatever advice came with them, stay available whenever they want to look back.

C. AI Interaction and Image Analysis Module

Text queries, along with a system prompt tuned for agricultural context, go to a large language model through the OpenRouter API, which replies in whichever language the farmer used. When a farmer uploads a crop photo instead, the image is stored on the server and sent to the model with a prompt asking specifically for disease or pest identification plus a suggested remedy. That gives a farmer a reasonable first opinion on a damaged crop straight from a photo, without waiting on anyone to actually visit the field.

D. Voice Output Module

Every AI-generated response can be turned into speech through Google Text-to-Speech, matched to whichever language the farmer is using. This runs asynchronously, in a background thread, so it never holds up the text reply, and the finished audio file gets attached to its chat message once it's ready.

E. Database Module

SQLite handles user records, verification status, chat sessions, individual messages, and references to any images or audio involved. This was a deliberate choice over something heavier SQLite doesn't need its own server process, which keeps deployment simple and the footprint small enough to run on modest hardware, which matters for organisations working with tight IT budgets.

Table II lists the technology stack used across the application layers.

Layer	Technology
Frontend	HTML, CSS, JavaScript
Backend	Flask (Python 3)
Database	SQLite
AI engine	LLM via OpenRouter API
Voice service	Google TTS (gTTS)
Email service	SMTP (verification)

IV. IMPLEMENTATION

The workflow starts once a user registers and verifies their account logging in after that automatically spins up a default chat session. Every query, text or image, gets tagged with the user's chosen language and routed to the AI interaction module. Once a response comes back, it's queued for speech synthesis and written to the database at more or less the same time, and both the text and, a moment later, the audio gets sent to the client. Algorithm 1 walks through query handling; Algorithm 2 covers voice generation.

Algorithm 1: Chat Query Processing

1: Input: user message or uploaded image, language code
2: Determine input type (text or image)
3: Build an AI prompt using agricultural context
4: Send request to the language model API
5: if response received then
6: Queue response text for speech synthesis (async)
7: Persist message, response, language to database
8: else
9: Return a fallback error message to the user
10: end if
11: Return response (and audio ref.) to client

Algorithm 2: Voice Response Generation

1: Input: response text, language code
2: Map language code to a supported gTTS language
3: try
4: Synthesize speech and write to an audio file
5: catch synthesis error
6: Retry once with default synthesis settings
7: end try
8: Link the audio file path to the chat message

The point of splitting things this way is to keep the request–response loop short. The farmer gets a text answer right away, and the slower voice synthesis catches up a beat later in the background, without ever blocking the chat itself.

V. TESTING AND RESULTS

Testing happened at four levels: unit tests on individual functions (registration, verification, login, chat handling, image upload, voice generation), integration tests across the seams between Flask, SQLite, the AI API, and SMTP, system tests on full end-to-end workflows, and user-acceptance testing that looked at language clarity, response relevance, and how easy the system actually was for non-technical users to pick up. All of this ran under conditions close to a real deployment a Flask development server, an SQLite database, and a live internet connection for the API and email.

Table III lists the core functional test cases and how each one went. Every case passed. None of the error

conditions we tried bad credentials, a wrong verification code, a simulated API timeout produced an unhandled exception; each one surfaced a clear message instead of just failing silently.

Table III. Functional test cases

ID	Test Case	Expected	Status
TC01	User registration	Account created	Pass
TC02	Email verification	Account activated	Pass
TC03	Login auth.	Session established	Pass
TC04	Text chat	AI response returned	Pass
TC05	Image analysis	Diagnosis returned	Pass
TC06	Voice output	Audio linked	Pass

We didn't formally benchmark performance, but the qualitative picture was solid enough: AI responses came back at a pace that felt fine in actual use, voice synthesis running in the background never noticeably slowed the text reply, and database reads and writes stayed lag-free under the light, single-developer load we tested with. A tighter evaluation logging latency across many queries, checking diagnosis accuracy against a labelled crop-disease dataset, running an actual usability study with farmers in the field was outside the scope of this first deployment, and we flag it as the obvious next step in Section VI.

Informal feedback gathered during the acceptance phase suggested the multilingual responses came across as clear, and that voice playback felt like a genuine help rather than a gimmick, particularly for people who weren't comfortable typing much.

VI. CONCLUSION AND FUTURE WORK

KrishiSamparkha shows that even a fairly modest, Flask-based application can bring conversational AI, multilingual support, image-based crop analysis, and voice output together into something that offers real advisory value to farmers who currently have little practical access to expert guidance. The lightweight stack Flask and SQLite in particular keeps hardware requirements low enough for cheap systems, which matters for the rural service centres and community access points most likely to actually run this. Testing across all four levels showed the core functionality holding up reliably, and because the design is fairly modular, a failure in one part a slow AI response, say never dragged the rest of the system down with it.

A few things would make this stronger for real deployment. A formal accuracy check of the image-based disease detection against a labelled agricultural dataset would put that feature on much firmer ground. Pulling in live market prices, government scheme details, and localised fertiliser or irrigation advice would push the system well past just answering questions. A dedicated mobile app with offline caching would help it reach places where connectivity isn't reliable. And a larger field study across different regions and literacy levels would test these usability findings at a scale this first evaluation simply couldn't reach.

REFERENCES

[1]. Pravinkrishnan, P., et al., “A survey on chatbot systems for agriculture using machine learning,” International Journal of Advanced Research in Computer Science, 2022.

[2]. Vijayalakshmi, J., and K. P. Meena, “Agriculture TalkBot using AI,” International Journal of Recent Technology and Engineering, vol. 8, no. 2S5, pp. 186–190, Jul. 2023.

[3]. Bang, Y., et al., “Enhancing chatbot memory for personalized long-term interaction,” in Proc. Conference on Dialogue Systems, 2024.

[4]. Haller, E., and T. Rebedea, “Designing a chat-bot that simulates an historical figure,” in Proc. IEEE International Conference on Control Systems and Computer Science, 2021.

[5]. Du Preez, J., et al., “A voice-enabled, web-based chatbot supported by keyword extraction,” in Proc. Southern Africa Telecommunication Networks and Applications Conference, 2019.

[6]. Reddy, S., and T. Mohan, “Intent-based conversational agent using Rasa for agriculture,” in Proc. AI in Agriculture Conference, 2021.

[7]. Sharma, K., and P. R. Kumar, “Dynamic learning chatbot for precision farming,” Journal of Machine Learning Applications, vol. 14, no. 4, pp. 60–65, 2022.

[8]. Fatima, A., and N. Ahmed, “WhatsApp integrated chatbot for crop advisory services,” International Journal of Agricultural Informatics, vol. 12, no. 1, pp. 30–35, 2023.

[9]. Kuska, M. T., M. Wahabzada, and S. Paulus, "AI-chatbots for agriculture: Where can large language models provide substantial value?" SSRN, Jan. 2024.
[10]. Singh, N., et al., "Farmer.Chat: Scaling AI-powered agricultural services for smallholder farmers," arXiv preprint arXiv:2409.08916, Sep. 2024.
[11]. Darapaneni, N., S. Raj, R. V., V. Sivaraman, S. Mohan, and A. R. Paduri, "LSTM-RASA based agri farm assistant for farmers," arXiv preprint arXiv:2204.09717, Apr. 2022.