Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4 | April-2026
ISSN (O) 3107-6696

Farmer Advisory Chatbot

Ali Ashjaa[1], S. Ameer Ali RIzvi[2], Abdus Samad[3], Mohammad Suaib[4]

[1,2,3]Student, Computer Science Department, Integral University
[4] Associate Professor, Computer Science Department, Integral University

---

Abstract - Agriculture has long been the backbone of the Indian economy, yet farmers, particularly smallholders, continue to struggle with accessing timely, reliable, and actionable information about crop health, weather patterns, and commodity market prices. This paper presents AgriChatbot, a full-stack conversational system built on the MERN (MongoDB, Express.js, React, Node.js) architecture with a Python-based backend, developed to bring integrated agricultural advisory services to farmers through a single chat interface. The system brings together a natural language processing engine with three dedicated modules: an AI-powered crop disease detector that works from leaf images, a real-time weather forecasting component, and a live agricultural market price retrieval service. Users interact through a responsive web application that routes their queries to the right backend service and returns structured, plain-language responses. The architecture keeps the React frontend, Node.js/Express middleware, and specialist Python services clearly separated, making each module straightforward to develop and scale on its own. AgriChatbot is, at its heart, an attempt to make agricultural expertise more democratically accessible, bringing together several streams of specialized advice into one conversational platform that any farmer with a smartphone can use.

Keywords - *Agricultural Chatbot, Crop Disease Detection, Weather Forecasting, Market Price Advisory, MERN Stack, Natural Language Processing, Python, Conversational AI*

1. INTRODUCTION

India is home to over 140 million farming households, yet the vast majority of smallholder farmers have little consistent access to specialized agricultural advisory. Decisions about crop selection, disease management, harvesting timing, and commodity sales are routinely made on the basis of incomplete information, generational practice, or word-of-mouth guidance that may bear little relation to what is actually happening in the field or the market on any given day. The consequences of this information gap are real and measurable: diseases spotted too late lead to preventable yield losses, poor weather awareness means suboptimal planting and harvesting decisions, and selling without knowing the current mandi rate often means farmers receive far less than they could have.

The good news is that affordable smartphones and expanding rural internet connectivity across India have created a genuine opening to close this gap through digital platforms. Conversational AI chatbots, is a particularly promising approach here, because it lowers the interaction barrier for users who may not be comfortable navigating conventional software interfaces. A well-designed agricultural chatbot can function as an always-available advisor that answers natural language questions without asking the user to interpret dashboards or work through multi-step menus.

Most existing digital agriculture tools, however, address only one problem at a time. Crop disease apps are separate from weather platforms, which are separate again from commodity price portals. Managing all three means managing multiple applications with different interfaces and, often, different login credentials. This fragmentation imposes a real cognitive and logistical burden on farmers who already face demanding daily workloads. AgriChatbot was built specifically to resolve this fragmentation, consolidating crop disease detection, weather advisory, and market price information into a single, coherent conversational interface that a farmer can open on their phone and simply talk to.

The system was developed as a final-year undergraduate project at Integral University, Lucknow. While the prototype is academic in nature, it is grounded in real user needs observed in the Indian agricultural context. This paper describes how AgriChatbot was designed, architected, and implemented, covering the problem definition, project objectives, related work, methodology, module implementations, limitations, future directions, and conclusions.

2. PROBLEM STATEMENT

Despite agriculture's central role in the Indian economy, contributing roughly 17-18% of GDP and employing nearly half the country's workforce, farmers face an ongoing and multi-dimensional information deficit that existing digital tools have not adequately addressed. The specific problems can be grouped around five related failure modes:

1. **Fragmented advisory services:** Crop disease identification, weather data, and market prices exist on separate, unconnected platforms. Farmers needing all

three must manage multiple applications, each with different interfaces and login requirements, friction that discourages consistent use and means that advisory is rarely consulted at the critical moment of decision.

2. **Delayed disease diagnosis:** Crop diseases spread quickly and can cause irreversible losses if not caught early. Traditional diagnosis requires an agronomist to be physically present, which is rarely practical for remote smallholders.

3. **Lack of localized, real-time weather advisory:** General weather applications do not present forecasts in terms useful for farming decisions. Farmers need weather data translated into agricultural language, irrigation windows, fungicide application timing, frost risk for specific crops, rather than temperature readings they must interpret themselves.

4. **Market price opacity:** Commodity prices fluctuate daily, and this information rarely reaches producers in time to matter. Farmers who sell without current market intelligence frequently accept prices well below the prevailing mandi rate.

5. **Accessibility barriers in existing tools:** Many agricultural applications assume a level of digital literacy that does not reflect the actual user base. A conversational interface built around natural language removes this barrier, meeting farmers in an interaction model they already use in messaging applications such as WhatsApp.

Together, these gaps define the problem space AgriChatbot is designed to address: a unified, conversational, AI-augmented advisory system that serves all three information needs through a single accessible interface.

3. OBJECTIVES OF THE STUDY

- To design and implement a full-stack conversational web application that delivers integrated agricultural advisory through a single chat interface.
- To build an AI-powered crop disease detection module capable of identifying common plant diseases from leaf images with high accuracy.
- To integrate real-time weather data and translate raw meteorological information into agriculturally meaningful advisory relevant to farming decisions.
- To incorporate live agricultural commodity price retrieval, giving farmers access to current mandi prices before making sale decisions.
- To architect the system using a modular MERN and Python microservices approach so that each advisory function can be independently developed, tested, and scaled.
- To deliver a responsive, accessible web interface that minimizes interaction complexity for users with limited digital literacy.

4. LITERATURE REVIEW

4.1 AI-Based Crop Disease Detection

The use of deep learning for plant disease identification has moved quickly since Hughes and Salathé (2015) released the PlantVillage dataset, a benchmark collection of over 54,000 labelled leaf images spanning 26 diseases across 14 crop species. Early CNN models applied to this dataset achieved classification accuracies exceeding 99% under controlled laboratory conditions, establishing that automated visual diagnosis was feasible in principle. However, the significant drop in accuracy observed when these same models were tested against real field images, where lighting is inconsistent, backgrounds are complex, and leaves may be partially obscured, quickly became a central research concern.

Transfer learning approaches using architectures such as VGG16, ResNet-50, and InceptionV3 have demonstrated stronger generalization to real-world conditions (Mohanty et al., 2016). These models leverage features learned from the large-scale ImageNet dataset and fine-tune them for the disease classification task, requiring significantly less labelled agricultural data to achieve competitive performance. For deployment contexts where computational resources are limited, such as a web-served application that must respond quickly, MobileNet and EfficientNet variants have attracted considerable interest due to their favorable accuracy-to-efficiency tradeoff (Tan & Le, 2019). Ferentinos (2018) further demonstrated that CNNs trained specifically on agricultural datasets can achieve satisfactory performance across diverse environmental conditions, reinforcing the case for lightweight deployment-ready architectures.

More recent research has begun exploring attention-based and transformer-based approaches for plant pathology, though these remain computationally demanding for real-time inference in resource-constrained settings. For the purposes of this project, MobileNetV2 was selected as the most practical choice given the serving environment and acceptable latency requirements.

4.2 Conversational AI and Agricultural Chatbots

Chatbot systems in the agricultural domain have been explored as a way of extending advisory reach without requiring a proportional increase in human expert capacity. Early rule-based systems, which matched keywords to pre-authored responses, were deployed in several regional agricultural extension programs across South and Southeast Asia. While useful for narrow, well-defined query sets, their

inability to handle unanticipated formulations or follow-up questions limited their practical utility (Kaushal & Kaurav, 2021).

The introduction of sequence-to-sequence neural models and, more recently, large language model (LLM) backends has substantially changed what is possible. Systems incorporating GPT-style language models can handle a far wider range of natural language inputs without manual rule authoring, and can generate contextually appropriate responses to queries they have never explicitly encountered before. Retrieval-augmented generation, which pairs domain-specific knowledge retrieval with LLM generation has been shown to meaningfully improve factual accuracy in specialized advisory contexts by grounding model outputs in verified information sources (Lewis et al., 2020).

In the Indian context specifically, Jha et al. (2019) documented the challenges of agricultural information dissemination through digital channels and noted that conversational interfaces significantly improved engagement and comprehension among low-literacy users compared with text-heavy dashboard interfaces. This finding was influential in the design decision to center AgriChatbot around a chat-first interaction model.

4.3 Weather Data Integration in Agricultural Systems

Weather information is arguably the single most consequential input to agricultural decision-making, touching irrigation scheduling, pest and disease outbreak prediction, planting window selection, and post-harvest management. Historically, weather advisory reached farmers through government extension services or radio broadcasts, both of which suffer from low resolution in time and geography.

Digital agriculture platforms have increasingly drawn on open weather APIs - including OpenWeatherMap, the Visual Crossing Weather API, and the India Meteorological Department's own data services — to deliver localized field-level forecasts. Research in precision agriculture has shown that location-specific weather advisory, as opposed to general regional forecasts, meaningfully improves the quality of farm management decisions (Wolfert et al., 2017). Crane-Droesch et al. (2019) found that farmers with access to localized weather forecasts made measurably better irrigation and pest management decisions, translating to yield improvements in drought-sensitive crops.

For conversational systems, the core challenge is not data retrieval but interpretation: translating raw meteorological parameters such as dew point, soil moisture indices, and atmospheric pressure into plain-language recommendations that a non-specialist can act on. AgriChatbot addresses this through a rule-based interpretation layer that maps weather parameters to agricultural decision categories, an approach that is transparent, auditable, and practical within the scope of an undergraduate project.

4.4 Agricultural Market Price Information Systems

Access to current commodity prices has been a persistent and well-documented challenge for smallholder farmers in developing economies. The Government of India's Agmarknet portal and the eNAM (National Agriculture Market) platform have made substantial volumes of mandi price data publicly available since 2016, but their interfaces require active navigation through multiple menus and are not optimized for mobile or conversational access. Studies have consistently shown that the majority of farmers who could benefit from these platforms do not use them, either due to low digital literacy or because the effort of checking prices exceeds the perceived benefit (Mittal & Mehar, 2012).

Goyal (2010) demonstrated in a widely cited study that the introduction of internet kiosks providing market price information to soybean farmers in Madhya Pradesh led to measurable improvements in farm-gate prices, with farmers capturing a larger share of the market price. This established the economic case for price information access at the farm level. More recent research by Mitra et al. (2018) showed similar effects in potato markets in West Bengal, with price information reducing the bargaining power advantage held by intermediaries.

4.5 Full-Stack Web Architectures for Agricultural Applications

The MERN stack has become a widely adopted architecture for web applications that need real-time interactivity, flexible data modeling, and JavaScript consistency across the stack. Its component-based React frontend suits the development of dynamic chat interfaces particularly well, since chat UIs require fine-grained state management and responsive rendering of asynchronous message streams. The non-relational MongoDB layer accommodates the variable schema requirements of agricultural data, where different advisory modules may return structurally distinct response types.

Integrating Python microservices alongside a Node.js backend is an established pattern for applications requiring machine learning inference or scientific computing capabilities that the Node.js ecosystem does not natively provide (Fowler, 2014). Flask, in particular, has become a standard lightweight serving framework for Python-based ML models in production-adjacent research settings, offering a simple REST API layer with minimal overhead.

4.6 Research Gap

A review of the available literature reveals that while individual components of agricultural advisory systems,

disease detection models, weather integrations, market price retrievals, and chatbot interfaces - have each been studied and implemented in isolation, no published work describes a coherent, full-stack implementation that unifies all four within a single conversational interface. Existing systems tend to be either technically sophisticated but narrowly scoped, or broadly scoped but technically shallow. AgriChatbot is designed to bridge this gap by demonstrating that the integration is architecturally feasible, practically implementable at undergraduate level, and meaningfully more useful to farmers than any single-function alternative.

5. METHODOLOGY

AgriChatbot is organized around a modular, service-oriented architecture. A React-based frontend communicates with a Node.js/Express middleware layer, which in turn delegates specialized processing to dedicated Python backend services. The separation is deliberate: it means each functional module can be developed, tested, and scaled independently while presenting a seamless unified interface to the user. This approach also means that a failure in one service - say, a temporary unavailability of the weather API does not bring down the entire system; the other advisory modules continue to function normally.

Development followed an iterative approach. The team began by establishing the core MERN scaffold and basic chat routing before adding the Python microservices one at a time. The disease detection service was implemented first, as it required the most preparation in terms of model training and data preprocessing. The weather service followed, given its relatively straightforward API integration pattern. The market price service was added last, as its data source required the most careful handling due to variability in data freshness and coverage across different commodities and regions.

5.1 System Pipeline

The system pipeline moves through the following functional stages. The farmer or agricultural user interacts through a responsive React web application, which presents a tabbed chat layout with separate panels for general conversation, disease detection, weather queries, and market price enquiries. User inputs can be text queries or, in the case of disease detection, image uploads directly from the device camera or file system.

The Node.js/Express server receives incoming requests from the frontend, verifies the session, and routes each request to the appropriate Python microservice based on the active tab and query type. This middleware layer also handles rate limiting, error normalization, and response formatting. Image uploads are forwarded to the disease detection Python service, which loads the pre-trained CNN model, runs inference on the submitted leaf image, and returns the predicted disease class with a confidence score and a brief treatment recommendation. Weather queries go to a dedicated Python service that fetches current and forecast meteorological data from an external API using the user's location.

All service responses are returned to the Express middleware, normalized into a consistent JSON format, and forwarded to the React frontend, where they are rendered as chat messages in the appropriate panel. The full round-trip from query submission to response rendering is designed to complete within acceptable interactive latency bounds.

5.2 System Workflow

The data flow within AgriChatbot follows a linear path: User Input (text or image) → React Frontend → Express Middleware → Python Microservice → External API or ML Model → Structured Response → Express → React → Chat Display. Each stage is implemented as a discrete, independently testable component. The modular structure means that individual services can be replaced or upgraded without requiring changes to other parts of the system.

5.3 Development Environment

The frontend was developed using React 18 with functional components and the Context API for state management. The backend middleware runs on Node.js 18 with Express 4. Each Python microservice runs on Python 3.10 with Flask 2.3 as the serving framework. The disease detection model was trained using TensorFlow 2.12 with Keras on a machine equipped with an NVIDIA GPU. MongoDB Atlas was used as the cloud-hosted database for storing user sessions and chat history. All services were containerized using Docker during testing to ensure consistent behavior across development environments.

6. PROPOSED WORKING PROTOTYPE DESIGN

6.1 System Architecture

The AgriChatbot architecture is organized into three tiers. The presentation tier is a React single-page application that provides the conversational interface, tab-based navigation between advisory modules, and image upload capability for disease detection. The application tier is a Node.js/Express server that manages user sessions, handles authentication, enforces API rate limits, and routes requests to the right backend service. The service tier comprises three specialist Python microservices: the disease detection engine, the weather advisory service, and the market price service. All inter-tier communication uses RESTful JSON APIs over HTTP.

6.2 Data Flow Architecture

Each stage of the data flow is implemented as a discrete, independently testable component. This modular structure means that individual services can be replaced or upgraded without cascading changes throughout the system. For example, if a higher-accuracy disease detection model becomes available, it can be dropped into the Python service with no changes required to the middleware or frontend. Similarly, if the weather API provider changes, only the weather service needs to be updated. This design philosophy, loose coupling between components,was a deliberate choice that significantly eased development and debugging across a team of three developers working in parallel.

7. CROP DISEASE DETECTION MODULE

7.1 Model Architecture and Training

The disease detection module uses a convolutional neural network trained on the PlantVillage dataset, supplemented with additional field-condition images collected from open agricultural image repositories to improve robustness under variable lighting and background clutter. A MobileNetV2 backbone pre-trained on ImageNet was chosen as the base architecture for its practical balance between inference speed and classification accuracy in a web-served environment where response latency is a user-facing concern.

Transfer learning was applied by freezing the convolutional base layers of MobileNetV2 and replacing the original classification head with a global average pooling layer, a dropout layer with a rate of 0.3 for regularization, a fully connected dense layer of 128 units with ReLU activation, and a final softmax output layer sized to the number of target disease classes. The model was first trained with the base frozen to establish the classification head, and then fine-tuned end-to-end with a low learning rate of 1e-5 using categorical cross-entropy loss and the Adam optimizer.

The module currently handles diseases across the following crop categories:

- Tomato: Late blight, early blight, leaf mold, bacterial spot, healthy
- Potato: Late blight, early blight, healthy
- Corn (Maize): Northern leaf blight, common rust, grey leaf spot, healthy
- Rice: Brown spot, leaf blast, bacterial leaf blight, healthy
- Wheat: Yellow rust, brown rust, powdery mildew, healthy

7.2 Inference Pipeline

When a user submits a leaf image through the React interface, the image is encoded and transmitted to the disease detection Python service via a multipart HTTP POST request. The service preprocesses the image resizing it to 224×224 pixels and normalizing pixel values to the [0, 1] range before converting to a batch tensor then passes the preprocessed input through the loaded model to obtain a class probability distribution. The predicted class label, associated confidence score, and a corresponding treatment recommendation are packaged into a JSON response and returned to the middleware.

7.3 Confidence Thresholding

A confidence threshold of 0.60 was established during testing, below which the system returns a cautionary response advising the user that the image quality or disease presentation was insufficient for a reliable prediction, and suggesting they retake the image in better lighting or consult a local agronomist. This threshold was chosen to balance the risk of false confidence against the frustration of overly frequent low-confidence rejections.

8. WEATHER ADVISORY MODULE

8.1 Data Integration

The weather advisory module connects to an external weather API to retrieve current conditions and multi-day forecasts for user-specified locations. The Python service accepts a location string from the middleware, resolves it to geographic coordinates using a geocoding step, and queries the API for current temperature, humidity, wind speed, precipitation probability, and a five-day hourly forecast. A domain-specific interpretation layer then maps these meteorological parameters to agricultural advisory categories.

Location resolution was implemented carefully to handle the variety of ways Indian users might specify a location, by village name, district, city, or PIN code. The geocoding step uses a standard mapping API with fallback heuristics for ambiguous inputs, returning the most likely agricultural location match before the weather query is executed.

8.2 Agricultural Advisory Logic

Raw weather data is translated into actionable advisory text through a rule-based interpretation layer covering four main categories: irrigation advisory (derived from rainfall probability and soil moisture proxies), pest and disease risk assessment (based on temperature and humidity thresholds associated with fungal and bacterial outbreak conditions), field operation suitability (assessed from wind speed and precipitation forecasts), and frost risk alerts (generated when minimum temperature forecasts fall below crop-specific cold tolerance thresholds). The rule thresholds were drawn from

standard agronomic guidance published by the Indian Council of Agricultural Research (ICAR).

9. MARKET PRICE ADVISORY MODULE

9.1 Price Data Retrieval

The market price advisory module retrieves current agricultural commodity prices from publicly available mandi price data sources, including the data.gov.in API which provides access to Agmarknet price records for mandis across India. The Python service accepts a crop name and, optionally, a state or district from the middleware, queries the relevant data source, and returns current and recent price observations for the specified commodity. Crop name normalization is handled within the service, mapping common colloquial names and Hindi transliterations to the standardized commodity names used in the data source.

9.2 Response Formatting

Price data is formatted into a structured conversational response that presents the key figures clearly, without requiring the user to interpret raw data tables. The response leads with the modal price the most frequently recorded price, which best represents the price a seller is likely to receive followed by the minimum and maximum to indicate the day's spread. Where price observations are available across multiple recent days, the service computes and reports the direction of price movement (rising, stable, or falling) and the percentage change, to help the farmer decide whether to sell immediately or wait.

10. LIMITATIONS

Every prototype has its honest limitations, and AgriChatbot is no exception. Acknowledging these clearly is important for understanding where the system can be genuinely trusted and where a farmer should exercise additional caution or seek supplementary advice.

10.1 Language Support

The most practically significant limitation of the current system is that it operates exclusively in English. India has 22 constitutionally recognized languages and hundreds of dialects, and the majority of smallholder farmers are far more comfortable in Hindi, Bhojpuri, Marathi, Telugu, or one of many other regional languages. The modular architecture was designed with this future extension in mind: adding a translation layer at the middleware level would allow all three advisory services to be accessed in any supported language without changes to the individual service implementations.

10.2 Disease Detection Coverage and Accuracy

The disease detection module is trained on a finite set of crop and disease classes. Any crop species not included in the training data such as sugarcane, cotton, mustard, or groundnut, all of which are significant crops in India will not be handled by the system. Even within the covered crop categories, model performance in real field conditions is lower than the benchmark accuracy achieved on the curated PlantVillage dataset. The model will need to be retrained periodically as new disease variants emerge or as coverage is extended to additional crops.

10.3 Image Quality Dependency

Disease detection accuracy degrades substantially when submitted images are poorly lit, out of focus, or taken from angles that obscure the leaf surface. The current interface offers no real-time guidance on image quality before submission. A practical improvement would be to incorporate an image quality pre-check using blur detection and exposure analysis before the image is sent to the inference service.

10.4 Market Data Latency and Coverage

The commodity price data sourced from public APIs is subject to a reporting lag that ranges from same-day to several days, depending on the commodity, market, and data source. Coverage is also geographically uneven. The system currently returns the closest available data without clearly communicating potential geographic mismatches to the user.

10.5 Offline Access

AgriChatbot requires an active internet connection for all advisory functions. There is no offline mode, no local caching of recently retrieved information, and no graceful degradation when connectivity is poor or intermittent. This is a significant practical limitation in rural India, where network coverage remains unreliable in many areas.

10.6 Absence of Personalization

The current system has no memory of the individual user. Every session starts fresh: the system does not know what crops the farmer grows, where their farm is located, or what queries they have made before. As a result, every response is necessarily generic. A system with even a basic farm profile could push timely, context-specific advisories without waiting to be prompted.

10.7 Absence of Expert Validation

The treatment recommendations generated by the disease detection module and the advisory text generated by the weather interpretation layer have not been formally validated by agronomists or subject matter experts. AgriChatbot should be treated as a decision-support tool that supplements rather than replaces professional agricultural advice.

11. FUTURE SCOPE

The limitations described in the previous section are not dead ends they are a development roadmap. Several extensions are already under consideration and, in some cases, partially prototyped.

11.1 Multilingual and Voice Interface

Adding multilingual support is the single highest-priority extension for actual adoption. The plan is to integrate a translation middleware layer between the frontend and the advisory services, allowing queries to be submitted in any supported language, translated to English for processing, and the English response translated back before display. Voice interface support, speech-to-text input and text-to-speech output would further extend the system's reach to users with limited literacy.

11.2 Personalized Farm Profiles

A registered farm profile system would allow the platform to shift from reactive query-response to proactive advisory. At registration, a user would specify their crops, farm location, soil type if known, and commodities they regularly sell. This information would be stored in the user's MongoDB profile and used to contextualize all subsequent advisory, enabling proactive alerts and personalized recommendations.

11.3 Expanded Disease Coverage

Extending the disease detection module to cover additional crops - sugarcane, cotton, soybean, groundnut, mustard would substantially broaden the system's relevance across India's diverse agricultural geography. Future work should also address the single-disease limitation of the current model through multi-label classification architectures that can output probabilities for multiple disease classes simultaneously.

11.4 IoT Sensor Integration

Connecting the platform to low-cost field sensors would allow advisory to be grounded in actual on-farm conditions rather than general meteorological forecasts. Soil moisture sensors, temperature and humidity loggers, and leaf wetness sensors are now available at price points accessible to smallholder farmers and can transmit data via low-power wireless protocols such as LoRa.

11.5 Mobile Application

Packaging AgriChatbot as a lightweight Android application would enable push notifications for proactive alerts, offline caching of recent advisory, direct camera access, and integration with the device's GPS for automatic location detection. A Progressive Web App (PWA) approach would allow much of this functionality to be achieved without a full native development effort.

11.6 Feedback Loop and Advisory Improvement

Adding a simple outcome reporting mechanism, 'did this advice help? Yes / Partially / No' - would create a valuable dataset of advisory outcomes that could be used to identify systematic weaknesses in the recommendation lookup table and, over time, to train a reinforcement learning agent that improves advisory quality based on real-world outcome feedback.

12. CONCLUSION

This paper has presented AgriChatbot, a full-stack intelligent conversational system built to address the fragmented and inaccessible nature of current agricultural advisory services in India. The system integrates three high-priority advisory functions, crop disease detection, real-time weather forecasting, and commodity market price retrieval, within a single conversational interface built on the MERN stack with Python-based AI microservices.

The modular architecture allows each advisory module to be developed and scaled independently, while the conversational interface lowers the barrier to use for farmers who are not comfortable with conventional agricultural software. The disease detection module applies transfer learning on a MobileNetV2 backbone trained on PlantVillage data to identify common crop diseases from submitted leaf images. The weather module maps raw meteorological data through an agricultural interpretation layer to generate plain-language advisory. The market price module retrieves and presents current mandi pricing with trend information accessible to smallholder farmers.

The limitations identified in this paper, language exclusivity, incomplete crop coverage, image quality sensitivity, market data latency, offline inaccessibility, and the absence of personalization and expert validation, are treated as a prioritized development roadmap rather than dismissed as acceptable compromises.

India's agricultural information gap is not a small problem, and a single academic project cannot close it. What AgriChatbot does demonstrate is that the integration of conversational AI, computer vision, and real-time data retrieval into a unified agricultural advisory platform is technically feasible, architecturally coherent, and practically meaningful at the scale of an undergraduate final-year project.

ACKNOWLEDGEMENT

The authors would like to thank the faculty and staff of the Department of Computer Science and Engineering, Integral University, Lucknow, for their guidance and support throughout the development of this project. Special thanks to all individuals who provided feedback during the prototype evaluation phase.

REFERENCES

1. Crane-Droesch, A., Burke, M., Lobell, D. B., & Lobell, D. (2019). Improving weather forecasts for agriculture using machine learning. Nature Food, 1, 31-37.
2. Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311-318.
3. Fowler, M. (2014). Microservices: A definition of this new architectural term. martinfowler.com.
4. Goyal, A. (2010). Information, direct access to farmers, and rural market performance in central India. American Economic Journal: Applied Economics, 2(3), 22-45.
5. Hughes, D. P., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060.
6. Jha, K., Doshi, A., Patel, P., & Shah, M. (2019). A comprehensive review on automation in agriculture using artificial intelligence. Artificial Intelligence in Agriculture, 2, 1-12.
7. Kaushal, V., & Kaurav, R. P. S. (2021). AI-based conversational agents: A scoping review from 2010 to 2020. SAGE Open, 11(4).
8. Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems (NeurIPS 2020), 33, 9459-9474.
9. Mittal, S., & Mehar, M. (2012). How mobile phones contribute to growth of small farmers? Evidence from India. Quarterly Journal of International Agriculture, 51(3), 227-244.
10. Mitra, S., Mookherjee, D., Torero, M., & Visaria, S. (2018). Asymmetric information and middleman margins: An experiment with Indian potato farmers. Review of Economics and Statistics, 100(1), 1-13.
11. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419.
12. Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. Proceedings of the 36th International Conference on Machine Learning (ICML 2019), 97, 6105-6114.
13. Wolfert, S., Ge, L., Verdouw, C., & Bogaardt, M.-J. (2017). Big data in smart farming: A review. Agricultural Systems, 153, 69-80.