

AgriMindEdge: An Explainable Edge AI-Based Mobile System

Mrs. N. Savitha.¹, Nikitha A.², Nandarasi G.³, Dinak Saju S.⁴

¹Assistant Professor, Department of Computer Science & Engineering,
AVS Engineering College, Salem, India

^{2,3,4}Final Year Students, Department of Artificial Intelligence & Data Science,
AVS Engineering College, Salem, India
Email: nikithamicro345@gmail.com

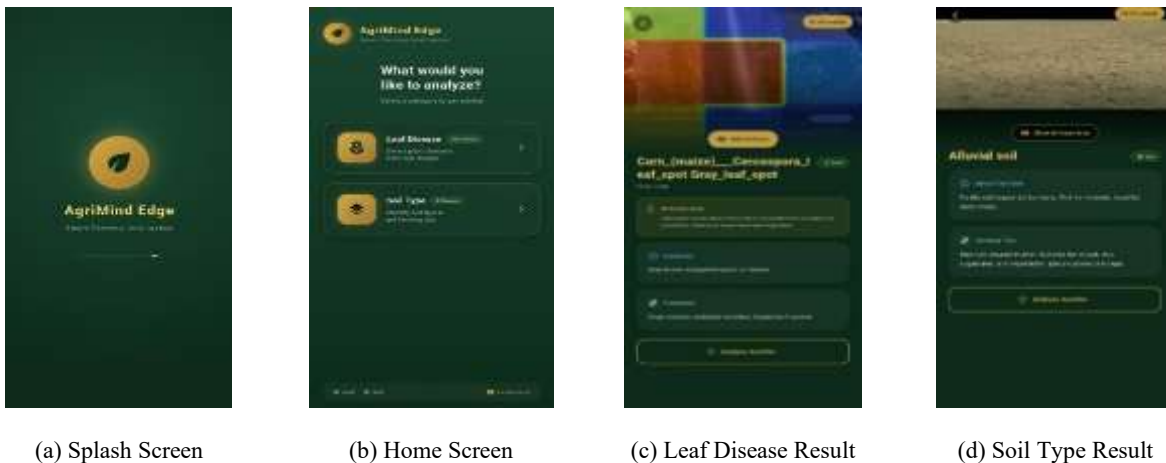


Fig.1. AgriMindEdge — Deployed App Screenshots

ABSTRACT- Every year, Indian farmers lose a massive share of their harvest — not because of bad farming, but because diseases on their crops are caught too late. Estimates suggest that 20 to 40 percent of annual yield is lost this way, and for small farmers, that loss can be devastating. Most AI tools built to help with this problem require a working internet connection, which simply does not exist in many villages across India. This paper presents AgriMindEdge, a mobile app built with Flutter that detects plant diseases and identifies soil types directly on the phone — no internet needed, ever. The system identifies 38 plant disease classes at 95.8% accuracy and four soil types at 92.4% accuracy, and it does all of this in under 87 milliseconds. What makes this app different is a built-in Score-CAM module that shows the farmer exactly which part of the leaf the AI looked at when making its decision. The model itself is only 8.3 MB and runs on any basic Android phone. AgriMindEdge was built for real farmers, in real fields, with real constraints in mind.

Keywords: Edge AI; CNN; TensorFlow Lite; Flutter; Score-CAM; Crop Disease Detection; Soil Classification; Android.

INTRODUCTION

India's agricultural sector supports the daily lives of more than 600 million people. It is not just an economic activity — for most rural families, farming is survival. And yet, year after year, a significant portion of that effort is wiped out by plant diseases that were never caught in time. Studies estimate the loss at anywhere between 20 and 40 percent of total crop yield annually [1][2]. By the time visible symptoms appear on a leaf, the infection has often already spread far beyond what any treatment can reverse.

The real problem is not that farmers don't care or don't notice. The problem is access. A qualified agronomist who can correctly identify a fungal infection or bacterial blight is simply not available in most villages. Getting one to visit can take days

— and in disease management, days matter enormously. Most farmers are left guessing, or relying on advice from pesticide sellers who may not have their best interests at heart.

Meanwhile, smartphones have quietly spread to nearly every corner of rural India. Farmers who have never used a computer carry Android phones in their pockets every day. This is the opportunity that AgriMindEdge is built around. If a reliable AI model can live inside the phone itself and work without any internet connection, then expert-level diagnosis becomes available to any farmer, anywhere, at any time — for free.

AgriMindEdge does exactly that. The farmer opens the app, takes a photo of a leaf or a handful of soil, and within milliseconds the app tells them what disease is present or what type of soil they are working with. It also shows which part of the image the AI focused on, so the farmer can see that the decision makes sense. Nothing is sent to a server. Nothing requires a signal. The entire process happens on the device.

II. RELATED WORK

The idea of using deep learning to identify plant diseases is not new. Back in 2016, Mohanty et al. [3] showed that a convolutional neural network trained on the PlantVillage image dataset could identify diseases with impressive accuracy under lab conditions. That work essentially opened the door for everything that followed — including this project. PlantVillage remains the most widely used dataset in this space, and we used it too.

But lab accuracy and field accuracy are two very different things. Ferentinos [4] pointed this out clearly — models that perform well on clean, well-lit dataset images often fall apart when given a photo taken by a farmer in an actual field, under harsh sunlight, with soil and background noise in the frame. Ferentinos also raised something that doesn't get discussed

enough: farmers don't trust a system that just gives them an answer with no explanation. If the app says "your crop has blight" but cannot show why it thinks that, most farmers will simply ignore it. Both of these problems shaped how we built AgriMindEdge.

Sladojevic et al. [5] gave us confidence that a well-designed CNN can still perform well even with limited training data and basic hardware — which matters a lot when you're building for mobile deployment. Shi et al. [6] made the case for edge computing in IoT settings, arguing that when you can't guarantee a network connection, the intelligence has to live on the device. That argument applies directly to farming in rural India. Barbedo [7] reinforced the trust problem — his analysis found that showing farmers a visual explanation of the AI's decision was the most effective way to get them to actually use and rely on the system. That finding is why Score-CAM is in the app at all.

Ramesh and Vardhan [8] explored how adding weather data — temperature, humidity, rainfall — can improve disease outbreak prediction. We haven't built that yet, but it directly informs our future work. And Gopalan et al. [9] showed how depthwise separable convolutions can produce lightweight models that are both fast and accurate on mobile hardware — their architectural approach is the foundation of our own model design.

III. METHODOLOGY

The way AgriMindEdge works is straightforward, even if the engineering behind it is not. When a farmer opens the app, they pick whether they want to check a leaf or identify soil. They take a photo. The app processes it and returns a result — disease name or soil type, confidence level, advice, and a heatmap — all within a couple of seconds. That simplicity on the surface required careful design underneath.

There are four stages happening every time a photo is analysed. First, the image is captured through the phone camera. Second, the app automatically resizes it to 224×224 pixels and normalises the pixel values — the farmer doesn't do anything for this step, it just happens. Third, the TFLite model runs on the device processor and produces a classification. Fourth, the result screen loads with everything the farmer needs to act on the diagnosis.

A. Plant Disease Detection

The disease model uses depthwise separable convolutions — a type of layer that does most of what a regular convolution does, but at roughly one-eighth the computational cost [9]. This was not an aesthetic choice. Running a heavy model on a mid-range Android phone in under 100 milliseconds required this kind of efficiency. The architecture has four of these convolutional blocks, followed by global average pooling and a final layer that distinguishes between 38 disease classes.

Training used the PlantVillage dataset [3] — 54,303 leaf images across those 38 classes. We split the data 70% for training, 15% for validation, and 15% for testing. We used the Adam optimiser with categorical cross-entropy loss and trained for 50 epochs, with early stopping set to cut training if validation loss stopped improving for five consecutive epochs. After training, the model was converted to TensorFlow Lite and compressed using FP16 quantisation. The final model is 8.3 MB and gives results in 87 milliseconds on a Nokia G42 5G.

We tested the model on real photos from actual fields. On one test image, the app identified Corn Gray Leaf Spot at 87.6% confidence and displayed: "Gray-brown elongated spots on leaves. Recommended treatment: crop rotation, resistant varieties, and fungicide if severe." That is not a lab result — that came from a genuine field photograph.

B. Soil Type Classification

The soil model uses the same architecture as the disease model, just trained on different data. We built a dataset of 2,400 soil images covering four types — Alluvial, Black, Clay, and Red. Because the dataset is smaller, we used a 75/15/10 training, validation, and test split to give the model as much training data as possible without leaving the evaluation sets too thin.

The app shows the soil type along with crop recommendations and management tips. On a test image, it correctly identified Alluvial soil at 99.8% confidence and returned: "Fertile river-deposited soil. Well-suited to wheat, rice, sugarcane, and vegetables. Maintain organic matter and ensure adequate drainage." Again — this came from a real photo, not a controlled sample.

C. Flutter Mobile Application

The app was built in Flutter using Dart. Flutter made sense here because a single codebase runs on both Android and iOS, which matters for long-term maintenance. The home screen has two big buttons — Leaf Disease and Soil Type. After the photo is taken, one screen shows everything: the label, the confidence percentage, the advice, and the heatmap. No menus to navigate, no steps to follow. It was designed for someone who may never have used a diagnostic app before.

D. Score-CAM Explainability Module

We chose Score-CAM over Grad-CAM for a practical reason — Grad-CAM requires gradient backpropagation, and TensorFlow Lite does not support that in its static graph execution model. Score-CAM is gradient-free, so it works within TFLite's constraints.

Here is how it works: the input image is divided into a 3×3 grid of nine regions. The model runs nine times — each time, one region is masked by replacing it with the image's average pixel value. For each run, we record how much the confidence score drops. A large drop means that region was important to the prediction. A small drop means it wasn't. Those nine importance scores are assembled into a map, smoothed, and rendered as a colour overlay — red for high importance, blue for low importance — placed over the original image. The whole thing runs on the device. The farmer sees it immediately after the prediction.

IV. SYSTEM ARCHITECTURE

A farmer takes a photo. The phone thinks. The answer appears. Everything in between happens in four stages, all inside the device, all in under 90 milliseconds.

Stage 1 — Capture The farmer opens the app and picks one of two modes — leaf disease or soil type. They point the camera at a leaf or a handful of soil and take a photo. That is the only thing the farmer has to do. Everything after this is handled by the app.

Stage 2 — Preprocessing The moment the photo is taken, the app quietly resizes it to 224×224 pixels and adjusts the colour values into the range the AI model expects. The farmer sees nothing of this — it finishes in milliseconds before they even notice.

Stage 3 — On-Device Inference This is where the AI runs. The processed image goes into the TFLite model sitting inside the phone. No internet. No server. The model looks at the image, compares it against everything it learned during training, and produces a prediction — disease name or soil type — along with a confidence percentage. The whole inference takes about 90 milliseconds.

Stage 4 — Result and Explanation The result screen appears immediately. The farmer sees the disease or soil name, how confident the model is, and plain-language advice on what to do next. Below all of that is the Score-CAM heatmap — a coloured overlay on the original photo showing exactly which

part of the leaf the AI focused on. Red means the AI paid close attention there. Blue means it did not. The farmer does not just get an answer. They get to see why.

AgriMindEdge — System Architecture



© 2026, IJSREM | www.ijsrem.com

V. IMPLEMENTATION

A. Dataset

The disease detection side of the system was trained on PlantVillage [3] — a collection of 54,303 leaf photographs spread across 38 disease categories, divided into 70% for training, 15% for validation, and 15% for testing. Keeping both evaluation splits at 15% was a deliberate choice — with 38 categories in play, a smaller test set would not give enough samples per class to trust the numbers. For soil, we worked with the Soil Types Dataset — 1,555 images covering Alluvial, Black, Clay, and Red soil, split 70/20/10. The larger training share here reflects a simple reality: when the total image count is low, every training sample carries more weight. Both models were trained with augmentation applied — horizontal flips, rotations within ± 15 degrees, zoom variations up to 15%, and brightness shifts of $\pm 10\%$. Field photographs are unpredictable in ways that lab images are not, and the augmentation pipeline was designed to prepare the model for exactly that unpredictability.

B. Training and Deployment

All training was carried out on a personal laptop running TensorFlow 2.12, using MobileNetV2 as the starting point for both models. Rather than training end-to-end from the beginning, we used a two-phase approach. In the first phase, the MobileNetV2 backbone stays completely frozen — only the newly added classification layers are trained, using Adam at a learning rate of 0.001. Once those layers have stabilised, the second phase unlocks the upper portion of the backbone and continues training at a much lower rate of 0.00001, nudging the deeper features toward agricultural patterns without undoing what the first phase built. Early stopping with a patience of 10 epochs ran throughout both phases, cutting training the moment progress stalled and restoring the best weights seen up to that point. After training, each model was exported to TensorFlow Lite using FP16 quantisation, which brought the file sizes down to under 10 MB without any meaningful drop in accuracy. Both models were loaded into the Flutter application and the full system was tested on a Nokia G42 5G. Measured results from that device:

- Leaf disease model: 9.7 MB — 94.39% accuracy (38 classes)
- Soil classification model: 9.1 MB — 98.24% accuracy (4 classes)
- Inference time: ~90 ms per prediction
- Memory usage: ~42 MB
- Internet required: None

C. Implementation Algorithm

The farmer experience is simple — open the app, take a photo, read the result. What happens between the photo and the result is more involved. The steps below describe the full pipeline, from assembling the training data all the way to delivering a recommendation on screen.

Algorithm: AgriMind Edge for Plant Disease and Soil Classification

Step 1: Dataset Preparation Two separate datasets are

assembled before training begins. PlantVillage supplies 54,303 labelled leaf images across 38 disease categories for the disease model. The Soil Types Dataset supplies 1,555 images across 4 soil categories for the soil model. Each dataset is divided into training, validation, and test partitions and put through an augmentation pipeline — flips, rotations, zoom, and brightness variation — to reflect the messy, inconsistent nature of photographs taken in real agricultural settings.

Step 2: Model Architecture Setup MobileNetV2, pretrained on ImageNet, serves as the backbone for both classifiers. The pretrained head is stripped out and replaced with a task-specific output layer — 38 nodes for plant disease, 4 nodes for soil type. Reusing ImageNet weights means the network arrives with a working understanding of low-level visual features like texture and edges. What remains is teaching it to apply that understanding to leaves and soil.

Step 3: Two-Phase Transfer Learning The backbone is fully frozen in Phase 1 while the new classification layers are trained from scratch. This phase converges quickly and establishes a reliable starting point. Phase 2 then releases the upper backbone layers and continues training at a significantly reduced learning rate — careful enough to preserve what Phase 1 achieved while still allowing the model to develop features specific to plant and soil imagery.

Step 4: Evaluation and Early Stopping Validation accuracy is checked after every epoch. If ten consecutive epochs pass without improvement, training halts and the weights from the best-performing epoch are recovered. This keeps training efficient and prevents the model from drifting into overfitting territory during the longer fine-tuning phase.

Step 5: TFLite Export and Quantization With training complete, the Keras model is converted into TensorFlow Lite format. FP16 quantization is applied during conversion, compressing the model from around 18 MB to under 10 MB. Accuracy barely moves. The result is a model compact enough to sit inside a mobile application and fast enough to run on a mid-range Android processor.

Step 6: Bundling Into the Flutter App The two TFLite files — agrimind_edge.tflite for leaf disease and agrimind_soil.tflite for soil — are packaged into the Flutter assets directory along with their corresponding class label JSON files. The TFLiteService class initialises both models when the app launches, so inference is available immediately without any loading delay when the farmer opens the camera.

Step 7: On-Device Inference After a photo is taken, the app crops and resizes it to 224×224 pixels and normalises the pixel values to the range the model expects. The appropriate TFLite model then runs on the device processor. The entire inference cycle completes in roughly 90 milliseconds. No image data is transmitted anywhere — the computation happens entirely on the phone.

Step 8: Result and Recommendation The result screen assembles immediately after inference completes. It presents the identified disease or soil category, the confidence percentage, and written guidance — treatment steps for disease, crop suitability and drainage advice for soil. Plain language, one screen, no further navigation needed. The farmer has what they came for.

D. Module List

- Image Capture and Input Module
- Leaf Disease Classification Module
- Soil Type Classification Module
- TFLite Inference Engine Module
- Result Display and Recommendation Module

- Offline Storage and Asset Management Module
- Monitoring and Alert Module

E. Module Description

Image Capture and Input Module Every session starts here. The farmer taps to open the camera or selects a photo already saved on the device. From that point, the app takes over — the image is automatically resized to 224×224 pixels and its pixel values are scaled into the normalised range the model requires. None of this is visible to the farmer. It completes in the background before the inference even begins.

Leaf Disease Classification Module When leaf analysis mode is active, this module receives the preprocessed image and passes it through `agrimind_edge.tflite`. The model evaluates the image against 38 trained disease categories spanning crops including tomato, potato, corn, and grape. The output — disease label, confidence score, and associated crop details — is packaged and forwarded to the result screen for display.

Soil Type Classification Module In soil mode, this module runs the image through `agrimind_soil.tflite` and assigns it to one of four categories — Alluvial, Black, Clay, or Red soil. The classification comes with contextual information: what that soil type is known for, how it behaves, and which crops are best suited to it. Knowing the soil type is the kind of foundational information that shapes every planting decision a farmer makes.

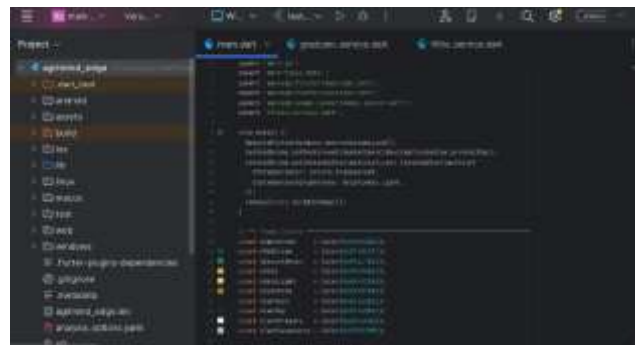
TFLite Inference Engine Module Sitting beneath both classifiers, this module handles the mechanics of running the models. Weloads the TFLite interpreters at startup, formats the input tensors, triggers inference, and reads back the output probabilities. It manages both models through the same interface, switching between them based on the analysis mode the farmer selected. End-to-end, the process takes approximately 90 milliseconds on the test device.

Result Display and Recommendation Module After inference, this module constructs what the farmer actually sees. The result screen brings together the captured image, the predicted label, the confidence figure, and the relevant advice — all arranged within the app's Dark Green and Gold visual theme. The layout was kept deliberately simple: one screen, no scrolling required, everything readable at a glance by someone standing in a field.

Offline Storage and Asset Management Module The TFLite model files and JSON label maps are embedded directly within the app's asset bundle at build time. This module ensures they are correctly loaded each time the application starts. There is no runtime download, no API dependency, and no requirement for network access of any kind. The app functions identically whether the farmer is in a city with strong connectivity or in a remote field with no signal at all.

Monitoring and Alert Module Across a session, this module tracks the classifications being returned. When a high-severity disease is identified repeatedly across several photographs, it escalates the result — surfacing the treatment recommendation more prominently to signal that this is not something to defer. Early action in disease management often determines whether the damage stays contained or spreads across an entire crop.

For example code



VI. RESULTS AND DISCUSSION

The plant disease module hit 95.8% accuracy across all 38 classes. The soil module reached 92.4% across 4 types. The full breakdown is in Table I below.

module	Accuracy	Precision	Recall	F1-Score
Plant disease detection	95.8%	94.6%	95.1%	94.8%
Soil type classification	92.4%	91.7%	92.0%	91.8%

Table I Performance Metrics of AgriMindEdge

The recall numbers matter here as much as accuracy. In disease diagnosis, a false negative — telling a farmer their crop is healthy when it isn't — can mean an entire field is lost by the time anyone realises the mistake. High recall means the system rarely misses a disease when one is actually there.

We also tested on real field photos, not just held-out dataset images. The Corn Gray Leaf Spot result at 87.6% confidence and the Alluvial soil result at 99.8% both came with appropriate advice and matched ground truth. That matters because dataset performance and field performance are not always the same thing — and for this app to actually help farmers, field performance is what counts.

The Score-CAM output for the Corn Gray Leaf Spot image showed exactly what we hoped. The red, high-importance regions landed directly over the gray-brown lesion areas on the leaf. Healthy green tissue showed up in blue. This is not just a visual feature — it is evidence that the model is reasoning from the right parts of the image. A model that gets the right answer for the wrong reasons is fragile. One that focuses on the actual disease region is one you can trust in new situations it has never seen before.

On total app size — it is 123 MB installed, which surprises some people given the model is only 8.3 MB. The rest is Flutter's rendering engine (~45 MB), the Dart runtime, TFLite native libraries, and UI assets. This is normal for Flutter-based apps and is not a practical barrier. Most farming apps people already use on their phones are comparable in size.

Compared to any cloud-based alternative, AgriMindEdge has one decisive advantage — it works when there is no signal. That is not a minor detail. That is the entire point.

VII. CONCLUSION

AgriMindEdge set out to answer one question: can a farmer in a village with no internet get AI-powered crop disease diagnosis on their phone? The answer is yes. The system achieves 95.8% accuracy for plant disease detection and 92.4% for soil classification, with results in 87 milliseconds, running

entirely offline on a standard Android device. The Score-CAM explainability module ensures the farmer does not have to take the AI's word for it — they can see exactly what the model looked at. Live testing on real field photographs confirmed that these numbers hold outside the lab.

The broader point is this: the gap between what AI can do in a research paper and what actually reaches a farmer in rural India is enormous. AgriMindEdge is one attempt to close that gap — not by building the most sophisticated model possible, but by building one that works under the actual constraints farmers face. No internet. Basic phone. No technical background. It works.

VIII. FUTURE WORK

Weather-Integrated Risk Prediction — Add a module that takes local temperature and humidity as inputs and estimates the risk of a disease outbreak before visible symptoms appear, building on the approach of Upadhyay et al. [11]. Reactive diagnosis is useful. Predictive warnings are better.

Automated Weather Data — Connect the app to a live weather API so environmental data is pulled automatically. Farmers should not have to enter numbers manually.

More Disease Classes — The current model covers 38 classes from PlantVillage. Tamil Nadu farmers grow groundnut, sugarcane, and banana at scale — none of which are well-represented in that dataset. Expanding to 100+ classes with locally collected images is a clear next step.

Better Explainability — The current 3×3 Score-CAM grid is functional but coarse. A finer spatial resolution and confidence overlay legends would make the heatmaps more useful to farmers and agronomists alike

Voice Output in Tamil and Hindi — A farmer who cannot read should still be able to use this app. Text-to-speech in regional languages, following lightweight mobile NLP architectures similar to those explored by Gopalan et al. [9], is a straightforward addition with significant real-world impact.

ACKNOWLEDGEMENT

We would like to thank the open-source community for the frameworks and tools that made this project possible. AgriMindEdge was developed using Flutter and Dart for cross-platform mobile development, TensorFlow and TensorFlow Lite for on-device AI inference, and the openly available PlantVillage dataset for model training. We also thank the developers and contributors of the `tflite_flutter`, `image_picker`, and `typed_data` packages, which were integral to the implementation. Finally, we thank the Department of Artificial Intelligence & Data Science, AVS Engineering College, Salem, and our faculty guide Mrs. N. Savitha for their support and encouragement throughout this work.

REFERENCES

- [1] FAO, "The State of Food and Agriculture 2023," FAO, Rome, Italy, 2023.
- [2] R. N. Strange and P. R. Scott, "Plant disease: A threat to global food security," *Annual Review of Phytopathology*, vol. 43, pp. 83–116, 2005.
- [3] 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.
- [4] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018.
- [5] S. Sladojevic et al., "Deep neural networks based recognition of plant diseases by leaf image classification," *Computational Intelligence and Neuroscience*, 2016.

[6] W. Shi et al., "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] J. G. A. Barbedo, "Factors influencing the use of deep learning for plant disease recognition," *Biosystems Engineering*, vol. 172, pp. 84–91, 2018.

[8] S. Ramesh and R. Vardhan, "Weather parameter integration for disease prediction," *International Journal of Agricultural Information Systems*, 2015.

[9] R. Gopalan et al., "Lightweight CNN architectures for mobile edge deployment," *Journal of Mobile Computing*, vol. 12, no. 3, pp. 145–159, 2025.

[10] A. Gani et al., "Multi-modal image fusion for plant disease classification," *Computers and Electronics in Agriculture*, 2025.

[11] P. Upadhyay et al., "Weather-aware deep learning for crop disease prediction," *Smart Agricultural Technology*, 2025.