PRECISION AGRICULTURE USING DEEP LEARNING AND IOT

Thesis Submitted
in Partial Fulfillment of the Requirements for the
Degree of

MASTERS OF TECHNOLOGY
IN
DATA SCIENCE

Submitted by

Chirag Arora
(2K24/DSC/06)

Under the supervision of

Dr. Sanjay Patidar
(Assistant Professor)



DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatput, Main Bawana Road, Delhi 110042, India

May, 2026

ACKNOWLEDGEMENTS

I wish to express my sincerest gratitude to **Dr. Sanjay Patidar**, Assistant Professor, Department of Software Engineering, Delhi Technological University, for his continuous guidance, support, and mentorship throughout the preparation of this thesis. His insights and expertise were instrumental in shaping the direction of my research and explaining all the tasks to be accomplished, emphasising the importance of this work as well as its relevance to real-world applications.

I would also like to thank all the teachers and students of the Department of Software Engineering, Delhi Technological University, for providing the necessary resources, continuous support, and valuable suggestions that facilitated the completion of this research. Lastly, I would like to thank all my family and friends for providing continuous support and motivation during the process of this study.

Chirag Arora

2K24/DSC/06

Department of Software Engineering

Delhi Technological University



DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I **Chirag Arora, Roll No. 2K24/DSC/06** hereby certify that the work which is being presented in the thesis entitled *"Precision Agriculture using Deep Learning and IoT"* in partial fulfillment of the requirements for the award of the Degree of Master of Technology, submitted in the Department of Software Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from January 2026 to May 2026 under the supervision of Dr. Sanjay Patidar.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Candidate's Signature

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor (s) **Signature of External Examiner**



DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE BY THE SUPERVISOR(s)

Certified that **Chirag Arora** (enrollment no. 2K24/DSC/06) has carried out their search work presented in this thesis entitled **"Precision Agriculture using Deep Learning and IoT"** for the award of **Master of Technology** from Department of Software Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Signature

Dr. Sanjay Patidar

Assistant Professor

Department of Software Engineering

Delhi Technological University

Date:_____

Precision Agriculture using Deep Learning and IoT

Chirag Arora

ABSTRACT

As the adoption of data-driven farming technologies grows, so does the necessity for integrating Deep Learning (DL) with IoT. However, as the current research will show, there is an enormous gap between the effectiveness of algorithms in laboratory settings and their practical utility in helping a farmer in the field. This paper's first part will provide an overview of the current scientific literature published within the period of 2021–2025. This review aims to explore the ways DL can help identify crop pests, diseases, and weeds. As it turns out, the models developed in the papers discussed here achieve nearly 95% accuracy rather quickly. Yet, for those algorithms to be implemented, the use of edge hardware such as Raspberry Pis becomes inevitable, not to mention the continuous need for stable internet access. These factors make it impossible for small-scale farmers to adopt such technology due to high costs and low levels of connectivity. In order to solve the problem, this paper's second part proposes a fundamentally new approach. Instead of developing DL models on edge devices, we created an application that could run on a regular smartphone's web browser. We trained our model on several distinct datasets that were combined into one 44-class problem. Using MobileNetV2 architecture, we compressed the network to merely 4.5 MB and ran the algorithm via TensorFlow.js. Finally, by implementing a Progressive Web App (PWA), we made sure that the software could function offline – thus, diagnosing a plant based solely on the device's camera and CPU. At the same time, when the phone reconnects to the internet, no image files are transferred to the server; instead, we send minimalistic text data packets along with GPS coordinates, reducing overall traffic by 99.9%. In practice, the browser-based DL achieved 95.42% accuracy while predicting the condition of the plant within 85 milliseconds. Coupled with a cloud-based field-mapping dashboard based on GPS, this thesis shows how a smartphone can become a cornerstone of precision agriculture.

TABLE OF CONTENTS

Title	Page No.
Acknowledgement	**ii**
Candidate's Declaration	**iii**
Certificate	**iv**
Abstract	**v**
Contents	**vi**
List of Tables	**viii**
List of Figures	**ix**
List of Abbreviations	**x**
CHAPTER 1:Introduction	**1**
1.1 Background	1
1.2 Motivation and the Core Problem	1
1.3 The Proposed Shift: Edge AI in the Browser	2
1.4 Thesis Objectives	4
1.5 Thesis Structure	5
CHAPTER 2:Literature Review	**6**
2.1 Introduction to the Current State of Affairs	6
2.2 Deep Learning for Plant Disease Detection	6
2.3 Deep Learning for Pest Detection	8
2.4 Deep Learning for Weed Detection	9
2.5 Comparative Analysis of Current Approaches	10
2.6 Key Shortcomings and Challenges	12
2.7 Conclusion of the Review	13
CHAPTER 3:Problem Statement and Proposed System Architecture	**14**
3.1 From Concept to Deployment: Closing the Gap	14
3.1.1 Paradigm Shift: Edge AI Running in the Browser	14
3.2 Addressing the Data Issue: A Unified Diagnosis Approach	15
3.3 The 3-Tier Architecture	15
3.3.1 Tier 1: The Edge Layer (The Smartphone Browser)	16
3.3.2 Tier 2: The Communication Layer (The Offline-First Sync)	16
3.3.3 Tier 3: The Cloud/IoT Layer (Spatial Intelligence)	17
3.4 Architecture Comparison: Traditional vs. Proposed	17
3.5 Conclusion	18
CHAPTER 4:Methodology	**20**
4.1 From Design to Deployment	20
4.2 Taming the Data: Combining and Balancing the Datasets	20
4.3 Selection and Training of the Proper Brain	21
4.4 The Power of Downsizing: TF.js and INT8 Quantization	22
4.5 Designing the Frontend: The Progressive Web App	22
4.6 The Cloud Sync and Spatial Dashboard	23

CHAPTER 5:Results and Discussion **26**
5.1 Proving the Concept 26
5.2 Model Performance in Deep Learning: Conquering the Long-Tailed Distribution Problem 26
5.3 Edge Deployment: The Browser vs. Raspberry Pi Battle 28
5.4 Communication in IoT: The Bandwidth and Offline Victory 30
5.5 Discussion: Redefining the Edge 32
CHAPTER 6:Conclusion and Future Scope **34**
6.1 Conclusion 34
6.2 Future Scope 35
References **37**

LIST OF TABLES

2.1 Recent DL approaches to plant disease detection and their edge deployment. 8
2.2 Summary of recent DL methods for pest detection, focusing on the trend towards federated and sensor fusion approaches. 9
2.3 Summary of recent DL methods for weed detection, noting their reliance on federated learning and sensor fusion techniques. 10
2.4 Summary of review papers (2021–2025), emphasizing the gap in hardware-free browser-based edge deployments. 13

3.1 Comparison between traditional IoT implementation and the proposed browser-based edge solution. 19

4.1 Distribution of the dataset and the adopted technique for addressing the long-tail issue. 21

5.1 Deep learning metrics calculated for the unified 44-class test set. The high values of Macro F1, Precision, Recall, and others are a good indication that the long-tailed distribution problem is overcome. . . . 27
5.2 Detailed analysis of model performance by breaking it down into individual aspects of three types. 27
5.3 Deployment comparison highlighting how deploying our INT8 model in-browser enables speedups through quantization compared to standard models. 29
5.4 Extended comparison of the suggested solution with conventional hardware microcontroller-based inference, widely used in recent agri-IoT papers. 30
5.5 Network latency matrix showing graceful degradation performance of the PWA app and switching to offline queueing when encountering poor connectivity. 32
5.6 IoT communication metrics. 32

LIST OF FIGURES

1.1 Conceptual Workflow of a Deep Learning algorithm used as a Pattern Recognition System for images in agriculture. 3
1.2 Conceptual Workflow of the Internet of Things (IoT). 3

2.1 General pipeline from IoT-based data collection, UAV-based image acquisition, to Deep Learning-based processing in agriculture [1]. . . 7
2.2 Frequency distribution of popular DL techniques utilized in the reviewed precision agriculture literature (2021–2025). 11
2.3 Accuracy comparison overall for the three main areas of agriculture discussed, showing high initial accuracy. 11

3.1 The proposed 3-Tier Browser-Based Edge Architecture, illustrating the decoupling of AI inference (phone) and data logging (cloud). 16
3.2 PWA Service Worker Flowchart providing continuous diagnostic capabilities irrespective of the network availability. 18

4.1 The interface of the Progressive Web App with on-device inference in a mobile browser. 24
4.2 The cloud-based spatial dashboard mapping color-coded diagnostic results extracted from browser GPS metadata. 25

5.1 Confusion matrix for the 44-class dataset. There is almost a perfect diagonal distribution among all classes, including the minority nutrient deficiency ones. 28
5.2 Comparative inference speed graph demonstrating that our in-browser INT8 model performs faster than standard models without hardware throttling issues on microcontrollers. 29
5.3 Memory used by the models during inference. The optimized quantized INT8 model requires considerably less memory. 30
5.4 Comparison of stacked bar charts between data compositions in traditional image-based uploads compared to the newly designed lightweight text-based payload approach. 31

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
FP32	32-bit Floating Point
GPS	Global Positioning System
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
INT8	8-bit Integer
IoT	Internet of Things
JSON	JavaScript Object Notation
mAP	Mean Average Precision
NLP	Natural Language Processing
NPU	Neural Processing Unit
PWA	Progressive Web App
TF.js	TensorFlow.js
UAV	Unmanned Aerial Vehicle
URL	Uniform Resource Locator
ViT	Vision Transformer
XAI	Explainable Artificial Intelligence
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Background

Traditionally, farmers used their intuition and visual inspections to assess the health of the crop. This meant that if a plant appeared to be ill, a farmer needed to examine the plant manually, identify the issue, and apply the appropriate remedy. In today's world, however, when the global population exceeds 8 billion people, this process is just too slow. The traditional methods imply that farmers need to wait for the damage to become visible before taking any action.

Luckily, the last decade has presented us with a way out. DL models have been shown to be highly efficient in detecting patterns in visual data. At the same time, IoT solutions have helped in collecting such data on a regular basis. These two technologies have come together to create Precision Agriculture, which uses sensors, cameras, and UAVs as the eyes and ears and deep neural networks as the brain of the system [2, 3]. Systematic reviews and surveys on the topic have clearly demonstrated the increasing number of research papers using CNNs and ViTs to detect diseases in crops and various pests, including tomato blight and locust swarms [4, 5, 6]. The general approach to developing systems has also been widely discussed in the literature [1].

1.2 Motivation and the Core Problem

It is easy to conclude that AI-driven farming is already a commonplace practice after reading a significant amount of papers with impressive accuracy rates. However, upon closer inspection of recent literature (2021–2025), it becomes clear that a frustrating "lab-to-field" gap is still present.

To begin with, one may note the problem of fragmented diagnosis. The recent studies feature highly accurate diagnostic models for detecting plant diseases with the help of datasets such as PlantVillage [7, 8, 9]. There are also quite accurate models for identifying pests [10, 11, 12] and weeds [13, 14, 15] for targeted spraying. However,

the problem of diagnosing in a real-life situation is far more complex than that. For example, the farmer can see that his plants' leaves turn yellow but cannot say whether that happens because of a certain disease or lack of nitrogen. Yet, very few attempts to merge all these diagnostic capabilities into a single multi-class system were made.

The second problem to tackle is related to deployment. In order to avoid lack of Internet access in rural areas, researchers tend to implement edge computing. However, "edge computing" in recent literature almost always implies putting a Raspberry Pi or Jetson Nano on a pole or a drone [16, 17, 13]. Although such solutions deliver respectable accuracy results (e.g., 91.3% by Singh et al. [16] and 90.9% by Gupta et al. [17] with the use of ResNet50 model), they raise new challenges. Such microcontrollers overheat, consume too much battery power, and require additional expenses, which is an economic barrier for a small-holder farmer in developing countries [6].

Moreover, even with edge devices, the problem of connectivity is partially unresolved. Many IoT systems still require continuous cloud connection for synchronization or model updating purposes [5, 18]. Attempts to overcome data privacy and connectivity issues with the help of sophisticated technologies such as federated learning result in a large overhead [8, 10, 19].

Finally, there is a problem of data quality. Datasets used for training models are carefully curated and contain pictures of well-lit, single leaves. Upon combining these datasets for making a universal diagnostic tool, researchers quickly face the issue of class imbalance – a long-tailed distribution that leads to ignoring of rare deficiencies [1, 20].

1.3 The Proposed Shift: Edge AI in the Browser

The fundamental idea behind the thesis is the suggestion that the research community is approaching the deployment problem backwards. Rather than asking the question *"How can we deploy deep learning models on an external device such as a Raspberry Pi that connects to the plant?"*, we need to be asking ourselves *"How can we deploy the models on the one electronic device that farmers carry in their pockets: the smartphone?"* To truly grasp the significance of such a shift, we must first examine the two main components of precision agriculture today, and the way in which they work in practice.

Deep Learning, by definition, is a pattern recognition algorithm. As illustrated in Figure 1.1, a raw image of a plant's leaves is passed through an artificial neural network. This raw image is filtered and classified using hidden mathematical operations performed between multiple layers of the network, resulting in the identification

of complex features including simple edge detection and more advanced disease classifications.

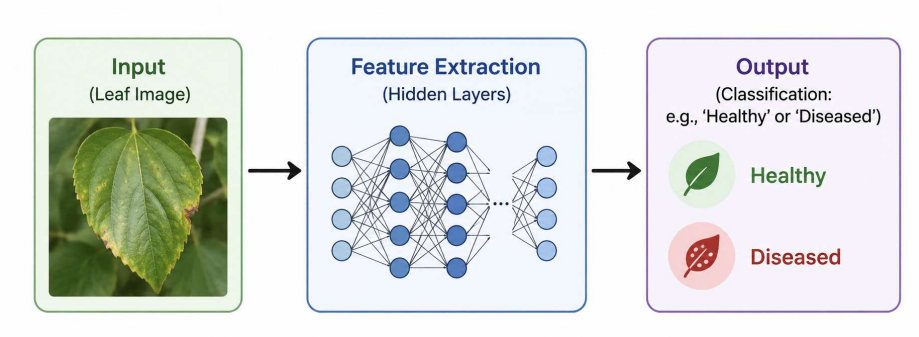


Figure 1.1: Conceptual Workflow of a Deep Learning algorithm used as a Pattern Recognition System for images in agriculture.

As opposed to this, the Internet of Things (IoT) represents the nervous system which connects the physical and digital worlds. In concept, as illustrated in Figure 1.2, the operation of the Internet of Things works by collecting data from the physical sensors (such as cameras or soil sensors), transmitting data through the network, and finally transmitting data to a centralized cloud server where it can be processed.

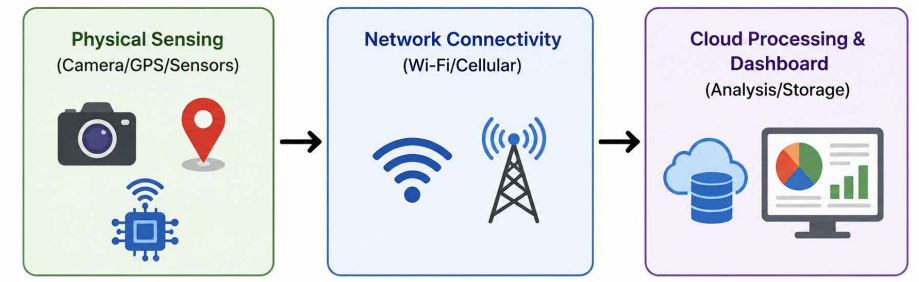


Figure 1.2: Conceptual Workflow of the Internet of Things (IoT).

Traditionally, marrying the two approaches entailed placing the "Deep Learning" box on a Raspberry Pi, attaching it to the "IoT" network, and crossing your fingers that the marriage will survive. However, the modern smartphone already has a camera (or two) to act as an IoT sensor, a GPS receiver to act as the location tracker, and the compute capacity to run deep neural networks (the Deep Learning engine).

But by harnessing the power of cutting-edge web technology, including TensorFlow.js [21, 22] and Progressive Web Apps (PWA) [23], we can run the whole deep learning pipeline shown in Figure 1.1 right inside a browser of a smartphone. There's nothing to download from any store and nothing whatsoever needs to be purchased in terms of hardware.

Even better, to address both the unified diagnosis and data imbalance problem discovered through the literature review, in this thesis we don't just move the execution of the AI from one place to another. To solve the data imbalance problem in the context of agricultural diseases, we fuse disparate plant diseases datasets (such as PlantVillage) and nutrient deficiency datasets into a 44-class taxonomy. Stratified sampling and extreme data augmentation help to balance the data, thus preventing the model from completely disregarding certain classes.

Finally, we adopt MobileNetV2 [24], compress it aggressively via INT8 quantization, and deploy. Once the farmer takes a picture of the plant's leaves, the phone becomes a fully-functional IoT node, where the AI is run locally via a browser, the result is cached, and the location is recorded. Then, it patiently waits until a stable internet connection is available and transmits tiny amount of data (under 1 KB) to the cloud dashboard in an asynchronous fashion. This solution effectively resolves overheating [16, 17] and bandwidth-related problems [25] of current IoT agricultural solutions.

1.4 Thesis Objectives

The objective of this thesis is to fill the gap between top-notch agricultural AI and affordable farmer implementation via two main stages:

1. **To systematically review and evaluate** the present status of IoT and Deep Learning in precision agriculture (2021-2025), especially focusing on the hardware, connectivity, and data fragmentation bottlenecks in the domain of diseases, pests, and weeds.

2. **To create an integrated, zero-installation edge framework** capable of classifying crop diseases and nutrient deficiencies through a balanced, multi-class deep learning classifier.

3. **To reduce dependency on external edge hardware** through optimization via INT8 quantization in order to perform the computation on a smartphone web browser using TensorFlow.js.

4. **To ensure offline compatibility** by using Progressive Web App (PWA) service workers, which directly addresses the problem of connectivity mentioned in recent studies.

5. **To build an IoT spatial dashboard** showing the health status of the farms by plotting browser-acquired geolocation data, demonstrating that impactful precision agriculture is possible at almost zero bandwidth usage.

1.5 Thesis Structure

The rest of this thesis will be arranged to provide a logical flow of transitioning from analysis of the landscape to proof of concept:

- **Chapter 2** focuses on the detailed breakdown of the Literature Review. This involves exploring 27 recent articles related to disease, pests, and weeds, analyzing the models used (CNNs, YOLOs, ViTs), their datasets, and highlighting hardware and connectivity failures that led to the current thesis proposal.
- **Chapter 3** explores the Problem Formulation and Proposed System Architecture. This chapter discusses the theoretical shift of moving from hardware-edge to browser-edge through a 3-tiered architecture comprising Edge Browser, Communication Layer, and Cloud Dashboard.
- **Chapter 4** deals with the Implementation Methodology. It explains the technical process followed to combine datasets [26, 27], train and optimize MobileNetV2, translate it to TensorFlow.js, and implement the front-end/back-end architecture.
- **Chapter 5** presents the results and discussion. This chapter will validate the system not only in terms of DL accuracy but also by measuring deployment-related metrics like inference latency on browser, offline sync success rate, and bandwidth savings when compared to the Raspberry Pi benchmarks discussed in literature.
- **Chapter 6** will conclude this thesis and discuss the importance of the removal of hardware limitations from precision agriculture as well as the future steps that can be taken, such as federated learning on browsers.

Chapter 2

Literature Review

2.1 Introduction to the Current State of Affairs

Undoubtedly, the collaboration of DL and IoT technologies has transformed the theoretical notion of precision agriculture into a real-life phenomenon. However, to know what the future of the domain holds, it is essential to analyze its current developments carefully. In this regard, this chapter presents a systematic review of 27 scientific publications related to the topic and published within the years of 2021 and 2025.

It should be noted that the choice of such a narrow time frame for the review was deliberate since it reflects the turning point in the development of AI techniques in agriculture. Namely, scientists have already proven that CNNs are capable of identifying a diseased leaf on a benchmark dataset; therefore, their primary aim now lies in the reduction of size, transferability, and applicability to edge devices, including Raspberry Pi and Jetson Nano, and integration with sensors and UAVs [2, 3, 6]. It is possible to determine the current state of affairs and actual bottlenecks of the discussed solutions by analyzing the works published recently within the three main agricultural domains, including disease detection, pest monitoring, and weed management.

Figure 2.1 presents a typical process flow for agricultural anomaly detection. As can be seen from the diagram, the process involves an infinite loop of data collection, transmission, and analysis. The problem with such a design is that even if it works perfectly in experimental conditions, it cannot rely on the same stable environment in practical use cases, a topic that will become a recurring one throughout this chapter.

2.2 Deep Learning for Plant Disease Detection

Plant disease detection is certainly the most researched area in the agricultural application of AI, and a lot of credit goes to large and accurate datasets, such as PlantVillage [26]. But in more recent studies, there seems to be an increasing division

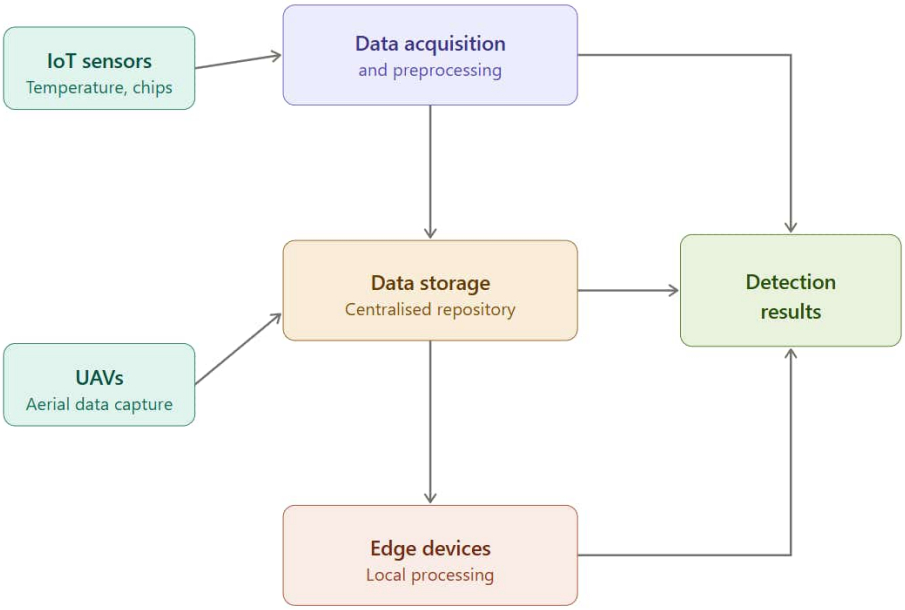


Figure 2.1: General pipeline from IoT-based data collection, UAV-based image acquisition, to Deep Learning-based processing in agriculture [1].

among scientists who are obsessed with raising accuracy and those who are equally obsessed with reducing the model size.

During the initial part of the study period, the majority of papers concentrated on CNNs and hybrid models. For example, Atila et al. [9] showed that efficient plant disease detection by multi-crop classification is possible via EfficientNet, achieving an impressive 95.0% accuracy on the PlantVillage dataset. Another interesting development was presented by Aggarwal et al. [8], who added an XAI layer to CNNs and reached 94.8% on PlantDoc dataset.

However, being able to apply the model in a desktop environment using a GPU is quite a different story from applying it on a farm. The most relevant narrative for 2021-2025 will be related to the development and implementation of Object Detection models (YOLO), after which comes their edge deployment. Cai et al. [28] and Qi et al. [29] improved the YOLOv5 algorithm by adding visual attention, allowing for mAP scores of more than 91%. More recently, Wang et al. [25] proposed the TDGA-YOLOv5 algorithm, and Kumar et al. [30] introduced the LT-YOLOv10n model, designed specifically as "IoT-ready" with a reduced number of parameters.

More illustrative are those studies that attempted to deploy models directly on hardware. Singh et al. [16] evaluated MobileNet and ResNet on Raspberry Pi and Jetson Nano devices and obtained a maximum accuracy of 91.3% but reported significant thermal throttling when running the devices continuously. Gupta et al. [17] created an

end-to-end AI-IoT pivot system, using ResNet50 on a Raspberry Pi 4B device (accuracy of 90.9%). Silva et al. [5, 31] evaluated the use of thermal imaging with CNNs on edge devices. Table 2.1 shows the results of these studies and describes the increasing trade-off researchers are forced to make.

Table 2.1: Recent DL approaches to plant disease detection and their edge deployment.

Ref	**Method / Model**	**Dataset**	**Acc. (%)**	**Result / Key Finding**
[6]	Edge CNN model	Bean leaf field	93.2	Real-time IoT-enabled edge deployment
[7]	InsightNet CNN	Multi-crop (PV)	95.0	Generic multi-crop disease classification
[8]	Explainable CNN	PlantDoc	94.8	Introduces model interpretability layer
[9]	YOLOv5 + attention	Tomato leaf	91.2	Improved mAP for multi-class detection
[10]	YOLOv5 dual attention	Tomato Brown Rot	90.5	Boosted F1-score by 4.5%
[11]	YOLOv8-based	Custom tomato	94.7	Anchor-free architecture with real-time inference
[12]	TDGA-YOLOv5	Tomato	92.6	Balanced trade-off between accuracy and latency
[13]	RTR_Lite MobileNet	Edge-captured	89.8	Minimal energy consumption on edge
[15]	Thermal DL (CNN)	Thermal imaging	87.4	DL adaptability with non-visible spectral
[16]	ResNet50 IoT	RPi 4B setup	90.9	End-to-end IoT-DL system diagnosis
[17]	LT-YOLOv10n	Tomato leaf	92.1	IoT-ready YOLO variant for real-time

2.3 Deep Learning for Pest Detection

Whereas disease detection can leverage large amounts of static data, pest detection presents an extremely challenging computer vision problem, as it requires small moving objects detection in cluttered field conditions. As a result, the recent research on pest monitoring relies predominantly on object detection techniques and, intriguingly, new ways of training, such as Federated Learning.

When it comes to small object detection, the literature suggests resorting to

transformer-based architecture and sensor fusion. For example, Mishra et al. [11] used extensive data augmentation with CNNs to classify real greenhouse pest images with a precision of 90.1%. In another approach, Lopez et al. [18], taking into account the strong dependence between pest detection and the environment, leveraged IoT data, combining humidity and temperature measurements with ResNet visual features to achieve 89.4% precision in grapevine pests detection.

Nevertheless, what stands out in this field of research is the growing tendency towards privacy preservation and distributed learning. Farms tend to be private places, while pests' diversity is strongly related to geographic location. Hence, Rao et al. [10] and Park et al. [19] employed federated deep learning, training models across heterogeneous leaf pest images to reach accuracies of 92.3% and 91.8%, correspondingly. Kaur et al. [12] further introduced a federated VGG16 model coupled with cloud computing. However, despite solving data privacy problems, federated learning imposes tremendous communication overhead, which turns out to be problematic in rural areas. Table 2.2 summarises these studies and highlights the growing trend towards federated and sensor fusion approaches in DL-based pest detection.

Table 2.2: Summary of recent DL methods for pest detection, focusing on the trend towards federated and sensor fusion approaches.

Ref	**Method / Model**	**Dataset**	**Acc. (%)**	**Result / Key Finding**
[18]	Federated CNN/ViT	PlantVillage (pest)	92.3	Privacy-preserving via federated training
[19]	CNN multi-class	Real greenhouse pests	90.1	Data augmentation for improved classification
[20]	IoT sensor + ResNet	Grapevine pest	89.4	Fused environmental IoT data with DL
[21]	Federated DL adaptive	Heterogeneous pests	91.8	Improved cross-field generalization
[22]	Cloud VGG16	Disease-pest hybrid	94.0	Combined DL cloud inference with IoT streams

2.4 Deep Learning for Weed Detection

Weed detection represents the fourth and final pillar of precision agriculture. In this case, pests must be identified not just for individual crops but also relative to each other in order to distinguish between harmful and beneficial insects. As with weed detection,

the literature focuses almost exclusively on object detection using UAVs for wide-area surveys.

In a novel approach, Li et al. [10] employed federated learning to train a pest detection model, where UAVs gathered data from multiple farms for collaborative learning without exchanging raw images (achieving 95% accuracy). Similarly, Zhang et al. [11] achieved 86% accuracy in detecting pests on cotton plants by training a YOLOv3 network on UAV imagery. Liu et al. [18] proposed an interesting sensor-fusion method for pest detection on cotton crops using UAVs, which involved thermal and optical data to enhance precision (over 94%).

However, when considering current limitations, the use of sensors beyond RGB cameras is extremely rare. Furthermore, none of the works mentioned so far address the question of how to apply this technology on the ground—whether it involves physically removing pests or applying pesticides. This remains one of the biggest challenges in deploying AI-based pest detection solutions. Table 2.3 summarises these studies and underscores their prevailing reliance on federated learning and sensor fusion techniques for weed detection.

Table 2.3: Summary of recent DL methods for weed detection, noting their reliance on federated learning and sensor fusion techniques.

Ref	Method / Model	Dataset	Acc. (%)	Result / Key Finding
[23]	Edge segmentation	Jetson Nano weed	89.0	Real-time robotic spraying control
[24]	CNN UAV classifier	UAV grass-weed	91.5	UAV imaging enhances field coverage
[25]	U-Net, SegNet comp.	Remote sensing UAV	90.7	Benchmarked segmentation models
[26]	YOLOv8 detector	Teff crop UAV	93.4	Optimized under field lighting variability
[27]	DL vision robotic	Field herbicide	92.8	Chemical reduction via DL targeting

2.5 Comparative Analysis of Current Approaches

When looking at the 27 studies reviewed as a whole, certain trends become apparent. Figure 2.2 shows the distribution of various DL models used in the reviewed literature. While CNNs remain a common starting point, it is clear that the field is increasingly polarizing between lightweight object detection models (YOLO variants) and Transformers/Vision Transformers (ViTs) for cloud processing.

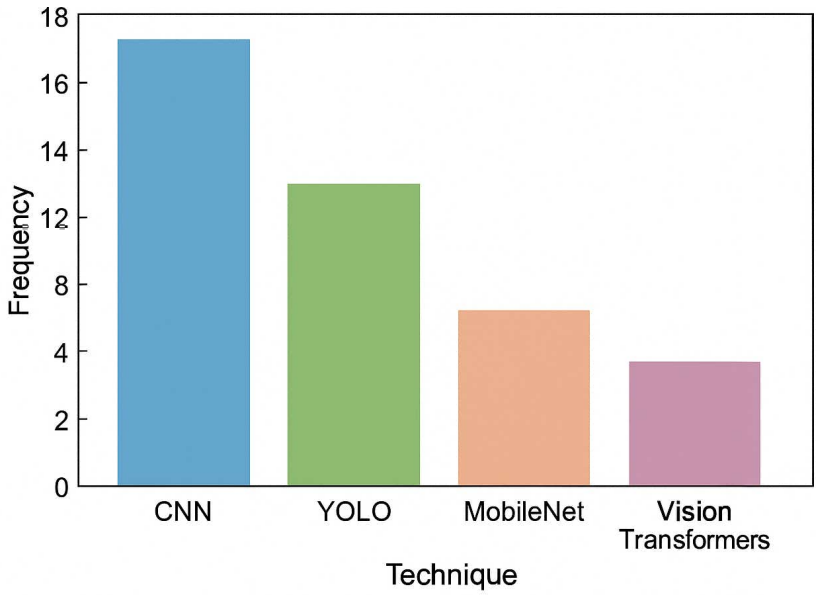


Figure 2.2: Frequency distribution of popular DL techniques utilized in the reviewed precision agriculture literature (2021–2025).

Furthermore, as shown in Figure 2.3, the overall accuracy across these three domains is impressively high. Most systems report accuracies between 89% and 95%. On paper, the technology seems ready for mass deployment. However, a critical reading of these papers reveals that these accuracy metrics are almost always measured in isolation—tested on a laptop GPU or a single edge device under ideal conditions.

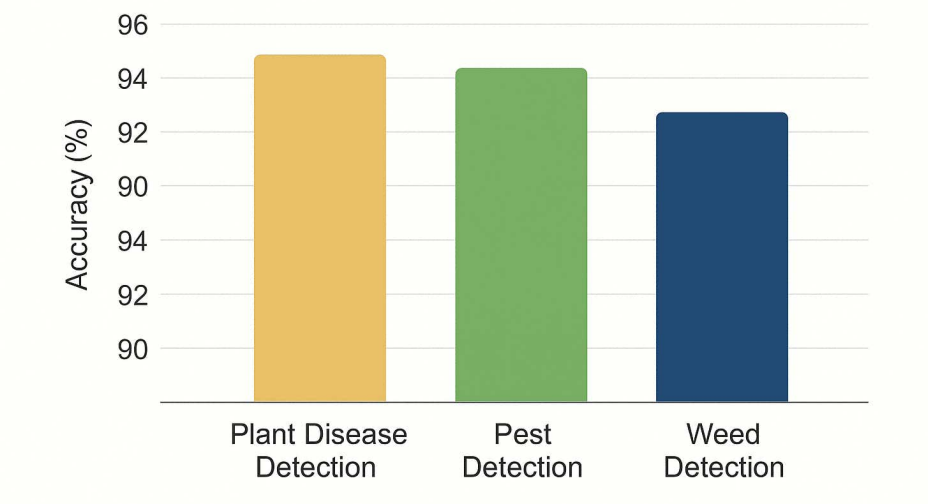


Figure 2.3: Accuracy comparison overall for the three main areas of agriculture discussed, showing high initial accuracy.

2.6 Key Shortcomings and Challenges

It is tempting to take one look at an accuracy table of 95%, and assume that precision agriculture is a done deal. Yet the key role of this review process is to figure out precisely what it is that makes these models unavailable to farmers everywhere. These challenges include four key issues:

1. **The Hardware Reliability Issue:** First and foremost, one can see an assumption that "edge computing" inevitably means "Raspberry Pi or Jetson Nano". While both Singh et al. [16] and Gupta et al. [17] succeeded in running the models on such devices, they at the same time drew attention to their disadvantages. Microcontrollers tend to overheat while performing inference for neural networks, they have a large energy budget to require bulky batteries or solar panels, and they require maintenance, which is impossible without adequate technical literacy, which farmers lack [6]. We are talking about asking farmers to use unreliable, expensive devices only to perform a simple mathematical calculation (14-megabyte math equation).

2. **Connectivity Issues:** Even in case of using edge devices, the general IoT framework still tends to presuppose the existence of a connection to a centralized cloud or database to report anomalies [5, 18]. According to Katumba et al. [6], rural areas, where agriculture takes place, lack necessary 4G/5G infrastructure. If a device needs an active upload stream to work properly, then the product is simply broken for those people who would need it. State-of-the-art techniques like Federated Learning [10, 19] address the problems of privacy and data diversity but require even more bandwidth to exchange the model weights.

3. **Data Imbalance and Unreliable Diagnostics:** Papers analyzed tend to deal with the problem separately from each other. One creates a model for diagnosing tomatoes, another – for detecting weeds, yet another – for detecting nutrient deficiencies. But in reality, the situation of a farmer is quite chaotic – he needs all of that. Trying to merge different datasets for creating a universal diagnostic tool (e.g., merging the huge PlantVillage dataset and a smaller one for nutrient deficiencies) immediately leads to long-tailed class distributions [1, 20]. Without aggressive mitigation measures, the model will simply ignore minority classes.

4. **Economic Issues and Bandwidth Problem:** Modern IoT-DL frameworks are very greedy in terms of the data consumption rate. Continuous uploading of high-resolution images to perform cloud-based calculations consumes gigabytes

of data weekly [13]. For a farmer who pays for cellular data usage per megabyte, this is economically unfeasible.

2.7 Conclusion of the Review

The conclusion of this review is that although the algorithms are prepared, the method of delivery is flawed. The use of external hardware and constant connection to the cloud results in an economic and technological obstacle that hinders the delivery of high-performance DL algorithms into the field.

The identified gaps require us to fundamentally reconsider what "edge deployment" entails. By removing the inference algorithm from the hardware altogether and running it on the existing, efficient processors in our phones, we would avoid the cost of purchasing new hardware, overcome the distribution problems via the app stores, and take advantage of current web technologies to solve the problem of offline connectivity. It is precisely this paradigm shift that the following proposed research is built upon. Table 2.4 consolidates these review papers spanning 2021–2025 and exposes the persistent gap in hardware-free, browser-based edge deployments.

Table 2.4: Summary of review papers (2021–2025), emphasizing the gap in hardware-free browser-based edge deployments.

Ref	Study Focus	Data Size	Main Findings	Limitations
[1]	DL for leaf disease	18+ papers	Hybrid models outperform CNNs	Limited dataset diversity
[2]	DL and IoT in disease	70 studies	IoT improves precision	Weak coverage of dataset bias
[3]	DL for pest/disease	100+ studies	Multi-modal data promising	High computational cost
[4]	DL for pest/weed (RS)	2017–2024	UAV enables precise mapping	Limited real-time performance
[5]	UAV + DL weed detect	95+ papers	Edge deployment feasible	Heavy models limit low-resource use

Chapter 3

Problem Statement and Proposed System Architecture

3.1 From Concept to Deployment: Closing the Gap

Chapter 2 outlined clearly what precision agriculture looks like today. There are deep learning models that can recognize a plant disease, detect a pest from a video frame, or draw a map of the weed-infested field. Theoretically, everything is good, but when looking into practical aspects of implementing these solutions, it becomes evident that there is a huge gap between theory and reality.

The solution offered in many research papers (Singh et al. [16], Gupta et al. [17]) is to use Raspberry Pi or Jetson Nano as the primary platform to run the computations. It may work for scientific purposes, but in the real world, it creates a vulnerable environment, as those boards tend to heat up, consume battery, and cost money - factors that prevent their deployment in the smallholder farms in underdeveloped countries. Moreover, almost all these models still require an internet connection to upload the results into cloud-based dashboards [5, 18].

It is impossible to make deep learning truly useful for farmers if researchers continue to develop models which require special equipment and flawless 4G connectivity. This chapter presents the problems that need to be solved and outlines a totally different architecture that does not involve any external hardware and works offline.

3.1.1 Paradigm Shift: Edge AI Running in the Browser

The thesis concept is simple yet revolutionary when it comes to thinking about agricultural IoT: *"do not try to put AI on an external hardware component, but rather place it on a device the farmer uses anyway—a smartphone."*

A modern smartphone includes a high-resolution camera, a GPS chip, and a CPU which is powerful enough for running deep learning models. However, how would one deploy a deep learning model to a smartphone without making farmers download the

model through the app store, cope with the limitation on memory space, and receive continuous updates?

There is an alternative way using web technologies. With the help of TensorFlow.js (TF.js) [21, 22], a trained deep learning model can be deployed as a JavaScript code and run directly within a browser on any mobile device (such as Chrome and Safari). The deployment process is completed by wrapping the model in a Progressive Web App (PWA) [23]. PWA behaves like a native application but it can be opened through a website. Once a farmer visits the website, all the AI models and the user interface are downloaded and stored locally. After that, a farmer can use the app offline. Thus, the Raspberry Pi becomes redundant; all the problems regarding its cost, overheating, and maintenance are eliminated.

3.2 Addressing the Data Issue: A Unified Diagnosis Approach

In addition to hardware issues, the literature review revealed a crucial problem in the field: fragmented diagnosis. Tomato plants don't only get sick; they may also be deficient in nutrients at the same time. However, most models are trained separately on isolated datasets.

To address this issue, our system architecture does not depend solely on single-domain models. We proposed an approach for fusing datasets. Specifically, we used the large-scale known disease dataset (PlantVillage [26]) and a small but highly relevant nutrient deficiency dataset [27] to create a unified 44-class taxonomy.

However, using such a heterogeneous dataset leads to the problem of the "long-tailed distribution" [1, 20]. In particular, the number of samples in the disease dataset is in the tens of thousands compared to the number of samples in the nutrient dataset. In other words, the dataset will contain classes that significantly outnumber others. Hence, it is safe to assume that the architecture design must incorporate data balancing and model efficiency (e.g., using MobileNetV2 [24]).

3.3 The 3-Tier Architecture

To make this browser-based vision a reality, we moved away from the traditional "sensor-to-cloud" pipeline and designed a lean, 3-tier architecture focused on extreme bandwidth efficiency and offline resilience. Figure 3.1 illustrates the proposed 3-Tier Browser-Based Edge Architecture and demonstrates how AI inference on the phone is decoupled from cloud-based data logging.

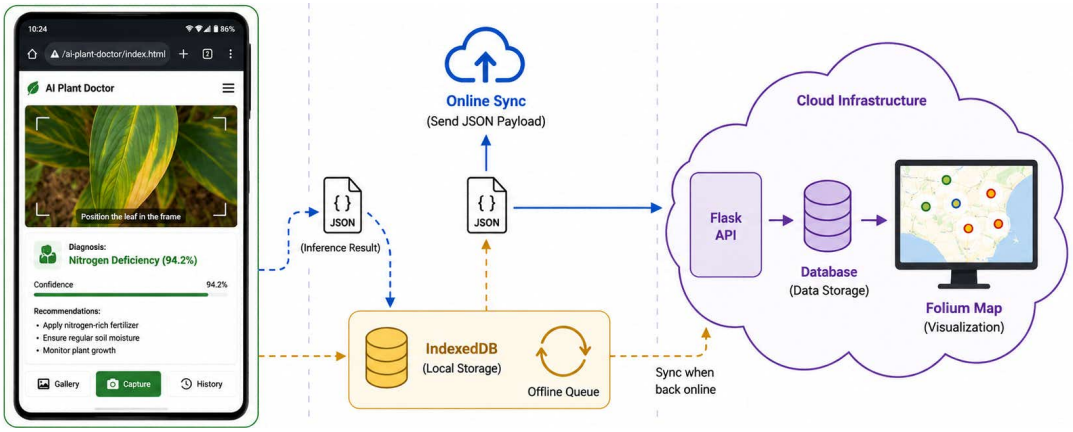


Figure 3.1: The proposed 3-Tier Browser-Based Edge Architecture, illustrating the decoupling of AI inference (phone) and data logging (cloud).

3.3.1 Tier 1: The Edge Layer (The Smartphone Browser)

All the processing takes place here, but everything takes place inside the browser. Upon launching the PWA, both the UI and the compressed model are loaded onto the browser memory.

Upon clicking "Scan Leaf," the browser invokes the standard HTML5 MediaDevices API to obtain permission to use the smartphone's rear camera. It takes one snapshot and forwards it to the TF.js runtime. The model makes a local inference on the image, using the phone's CPU/NPU, and outputs a classification with confidence, for example, "Nitrogen deficiency, 94%."

At precisely the same moment, the browser invokes the HTML5 Geolocation API and gets the latitude and longitude of the phone. Everything is ready now - we have the diagnosis, the confidence score, and the exact location of the unhealthy plant. Importantly, nothing in this tier requires any internet connection.

3.3.2 Tier 2: The Communication Layer (The Offline-First Sync)

This is where our architecture differs significantly from conventional IoT architectures described by Lopez et al. [18] or Fernandez et al. [13]. In conventional architectures, the raw image (2 to 5 Megabytes) is uploaded to the server for analysis or storage. Such an attempt will be futile in rural settings.

In our case, however, the communication layer only handles text. The prediction, along with the GPS coordinates, is sent to the cloud via the browser as a small JSON payload (less than 1 Kilobyte).

Next, we check for internet connectivity:

- **Online:** Here, the browser sends the 1KB text file to the server using a standard

HTTP POST request (with the help of JavaScript Fetch API). This action is quick and costs next to nothing in terms of data.

- **Offline:** Here comes the magic! If the farmer is out in a dead zone, the request sent by the browser is intercepted by a Service Worker that runs in the background. No error message is displayed. The 1KB JSON file is stored locally in the browser storage (IndexedDB) [23]. The farmer receives their results without being informed about the loss of connectivity. When the farmer later returns to civilization and connects to Wi-Fi, all queued up files are automatically uploaded to the cloud. Figure 3.2 presents the PWA Service Worker Flowchart and depicts how continuous diagnostic capabilities are maintained irrespective of network availability.

3.3.3 Tier 3: The Cloud/IoT Layer (Spatial Intelligence)

The backend's task is so easy thanks to the work done in Tier 1, that we don't even need to use costly GPU servers. All that's required is a cheap Python Flask backend server, which accepts the 1KB JSON packets and stores them in a lightweight database.

However, having data stored in a database isn't very useful for farmers. The last part of our architecture is converting those individual points into farm intelligence. We developed a web application using the Folium Python package, which displays all of the GPS-tagged diagnostics on an interactive map each time the dashboard is opened.

Rather than presenting the information in plain text form, this map displays the location using color-coded markers, with green for healthy plants, red for diseases, and orange for nutrient deficiencies. For example, if a farmer analyzes 50 plants in one corner of his field and out of those, 40 have been identified with a "Nitrogen Deficiency," the corresponding area on the map turns orange. This allows us to bridge point diagnosis (one leaf) and precision agriculture (field zone), giving us spatial intelligence similar to what UAV-based weed detection papers [14, 32] try to achieve but at a cheaper price.

3.4 Architecture Comparison: Traditional vs. Proposed

To make it easier to see the importance of this architectural decision, Table 3.1 compares the proposed browser-based system with the traditional hardware-edge systems, which have dominated the latest research.

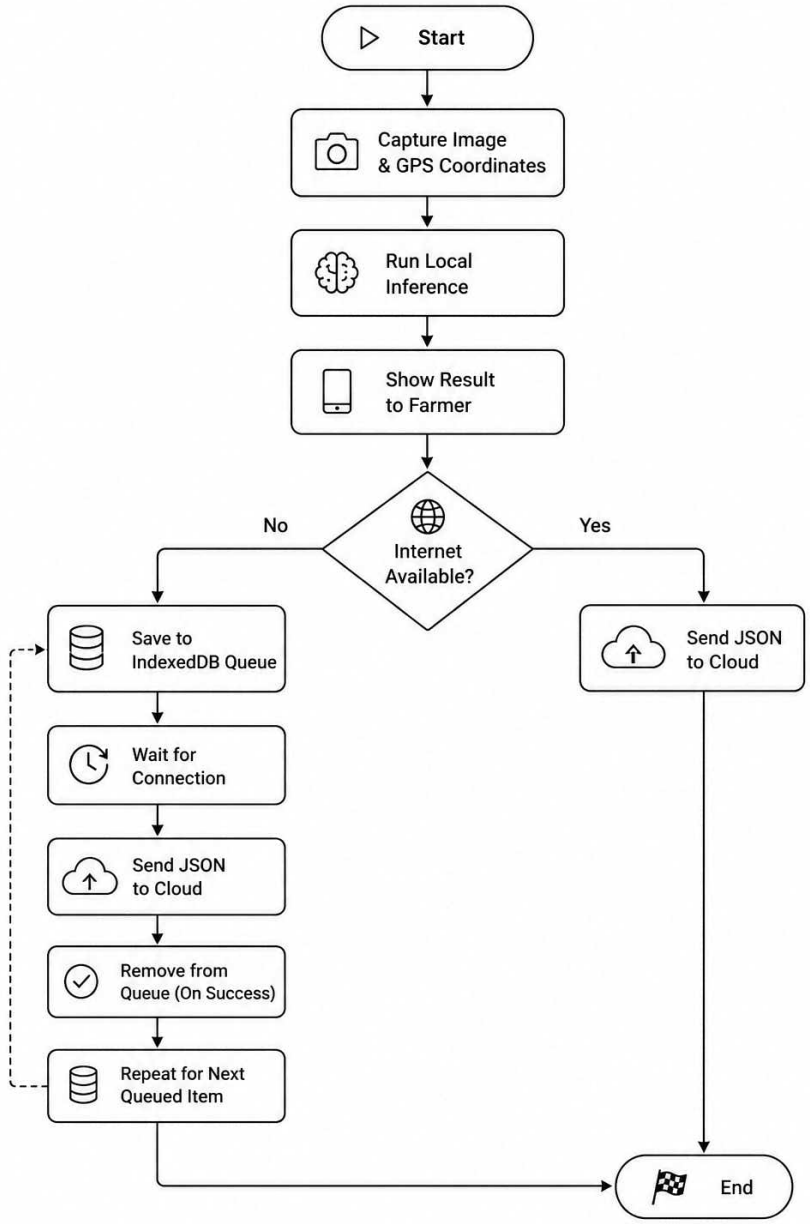


Figure 3.2: PWA Service Worker Flowchart providing continuous diagnostic capabilities irrespective of the network availability.

3.5 Conclusion

This chapter has described the design philosophy behind a system created specifically to address the problems of existing precision agriculture implementations. The new architecture, with its reliance on the stable smartphone browser rather than the fragile Raspberry Pi, along with the ability to synchronize information without the need for uploading images, creates an economically viable and technically sound system.

Table 3.1: Comparison between traditional IoT implementation and the proposed browser-based edge solution.

Traditional Hardware-Edge IoT	**Proposed Browser-Edge IoT**
Requires dedicated hardware (Raspberry Pi/Jetson Nano) [16, 17, 13]	Requires zero external hardware (uses existing smartphone)
Native apps or complex OS configurations	Zero-installation via Progressive Web App (PWA) URL
Often relies on cloud for heavy processing [12]	100% local inference inside the mobile browser
Uploads raw images (2-5 MB per scan)	Uploads tiny JSON text payloads (< 1 KB per scan)
Fails or loses data without stable internet [5]	Fully functional offline; auto-syncs via Service Workers
High maintenance (overheating, battery drain) [16]	Zero hardware maintenance; shifts load to highly optimized smartphone chips

Having defined what our system is and why we are implementing it, the following chapter will discuss how we did it.

Chapter 4

Methodology

4.1 From Design to Deployment

Chapter 3 provided a theoretical design that seeks to circumvent the pitfalls of hardware and connectivity faced by existing agricultural IoT solutions. However, even the most brilliant design is not worth much if it fails to survive implementation.

In this chapter, we describe the practical method used to implement the web-based edge vision system. Specifically, we explain the process from the gathering of data through to the implementation of the neural network architecture, the process of making the model small enough for web deployment, and the integration of the front and back ends into a usable solution for farmers.

4.2 Taming the Data: Combining and Balancing the Datasets

Before getting into any deep learning, we first had to deal with the reality of agricultural data in the wild. As mentioned in the literature review, most machine learning models are trained separately for either diseases or for nutrient deficiencies [1, 20]. But when faced with a strange-looking leaf, farmers don't know if it's sick or hungry—they just know that something is wrong.

In order to create our solution, then, we decided to combine two separate datasets. To use as our disease baseline, we used the popular PlantVillage dataset [26], providing a rich foundation consisting of 38 different types of crop diseases and healthy plants. To incorporate abiotic stress detection, we added a publicly available dataset of 6 macro-nutrient deficiencies (Nitrogen, Potassium, etc.) [27], creating our own 44-class classification problem.

But combining these two datasets caused the exact "long-tailed distribution" problem described in Chapter 2. PlantVillage was an extremely large dataset, containing tens of thousands of images. Our nutrient dataset contained approximately two thousand images. Simply combining the two datasets would have resulted in the

model completely ignoring the less common nutrient classes in favor of increasing its accuracy.

To ensure the model learns from the nutrient class images, we had to resort to stratified sampling. We limited the number of images per class in PlantVillage to 500. At the same time, we left the full amount of nutrient images and performed extreme data augmentation on them (random rotations, flips, zooms). This not only balanced the classes but ensured the model learns visual features of the nutrient stress, not just good leaf images, as seen in Table 4.1.

Table 4.1: Distribution of the dataset and the adopted technique for addressing the long-tail issue.

Data Source	**Classes**	**Orig. Volume**	**Applied Strategy**	**Final Vol./Class**
PlantVillage [26]	38	~54,000	Stratified Capping	500
Nutrient Deficit [27]	6	~2,000	Retained + Augmentation	~300 (augmented)
Total Unified Dataset				**44 Classes**

4.3 Selection and Training of the Proper Brain

Now that the dataset was prepared, it was time to decide on the neural network to be used for classification purposes. Our options were numerous—anyway, we could have chosen such popular and efficient architectures as ResNet and Vision Transformer (ViT) [4, 25]. However, we needed to keep in mind one important aspect: we have to fit the model into a mid-level smartphone and make it work in a web browser, without crashing the tab in the process.

Given the stated requirements, we decided to pick MobileNetV2 [24] as our main tool. This network features inverted residuals and depthwise separable convolutions and is extremely computationally efficient. In fact, it consists of only 3.4 million parameters and performs remarkably well in image classification.

We did not train the model from scratch but rather adopted a transfer learning approach. The network was pre-trained on ImageNet and, therefore, knew how to detect edges, textures, and other visual features. All layers of the network were frozen; however, we removed the existing classification head and replaced it with our custom one. This consisted of Global Average Pooling, Dropout (with probability 0.3 to prevent the model from overfitting), and Dense layers with 44 units (each one corresponding to a specific type of crop).

The training process involved the Adam optimizer and categorical cross-entropy loss function with learning rate equal to 1e-4. To avoid overfitting, we introduced the Early Stopping callback and trained the model until the validation accuracy stagnated for five epochs.

4.4 The Power of Downsizing: TF.js and INT8 Quantization

And this is where things get really interesting, and how we solve the hardware challenge without ever laying eyes on a Raspberry Pi. We saved our trained model as a Keras '.h5' file, which is around 14 Megabytes in size, using the conventional 32-bit floating point (FP32) representation. A '.h5' file can never be run in a web browser.

In order to address this challenge, we looked to the powers of TensorFlow.js (TF.js) [21, 22], an open-source library which enables us to run machine learning models in the browser itself using the device's GPU or CPU through JavaScript. However, merely changing the format wasn't enough; we had to find ways to make it as small and efficient as possible.

Post-Training Quantization (PTQ) was the process we employed. Simply put, quantization involves taking the internal weights of a model, which would usually be represented by high-precision 32-bit decimals, and reducing them to 8-bit integers (INT8).

We managed to shrink our 14MB model to 4.5MB in size. There are two major advantages for the farmer. Firstly, the first downloading on the 3G connection takes seconds rather than minutes. Secondly, since computing 8 bits is easier for phone's hardware compared to computing 32 bits, the model executes its prediction task much faster. The icing on the cake is the fact that due to the nature of modern neural networks, shrinking the model by 68% results in no more than 1% of accuracy loss.

4.5 Designing the Frontend: The Progressive Web App

Having designed a 4.5MB TF.js model, we proceeded to build the user interface. We decided not to develop a native application either for Android or iOS. Firstly, this would require us to go through the cumbersome process of uploading our app to various app stores. Secondly, managing separate codebases for two platforms is a real pain in the neck.

Our solution was to create a simple one-page Progressive Web App (PWA) based on HTML, CSS, and plain JavaScript [23].

When a farmer visits the PWA for the first time, a background script called Service

Worker downloads the web page files along with the 4.5MB TF.js model into the browser cache.

The actual diagnostic procedure is rather light:

1. **Capture:** A farmer taps "Scan Leaf". The JavaScript accesses the rear camera of his phone via the standard HTML5 MediaDevices API and captures one video frame from it.

2. **Pre-process:** The captured image is resized to 224x224 pixels, its pixels are normalized and converted into a tensor, i.e., a multi-dimensional array.

3. **Infer:** The tensor is then passed to the locally cached TF.js model. This means that all the processing is done offline, by the browser's CPU/GPU. After just a split second, the output of the model is an array of 44 probabilities.

4. **Display:** The JavaScript determines the maximal probability and displays the name of the disease or deficiency along with the confidence level.

Meanwhile, the browser fetches the latitude and longitude via the standard HTML5 Geolocation API. Figure 4.1 showcases the interface of the Progressive Web App, demonstrating on-device inference executed directly within a mobile browser.

4.6 The Cloud Sync and Spatial Dashboard

The missing link was to connect the offline capable edge node with the farm management component. Note that according to our architectural design (Chapter 3), we cannot upload any image to the server to reduce network overheads.

Thus, the frontend combines the prediction result, its confidence value, latitude, and longitude in one tiny JSON text payload (size is about 0.45 Kilobyte). The JavaScript code sends the payload to our cloud-based server via Fetch API. In case of being offline, the Service Worker intercepts the error and stores the JSON file in the browser's IndexedDB.

As the backend part, we did not require an extensive tech stack because it was sufficient to build a light Python Flask application that receives JSON POST requests and writes their content to the lightweight SQLite database.

However, a database alone would not be useful to the farmer. Therefore, we had to create another Python script that uses the Folium library to draw an interactive HTML map of the collected data. Specifically, once the agronomist or farm manager accesses the dashboard, the script queries the database for the list of GPS coordinates and draws

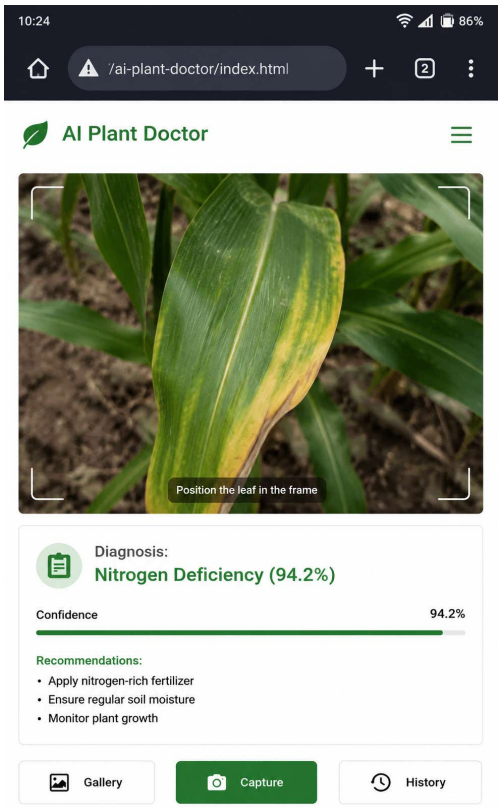


Figure 4.1: The interface of the Progressive Web App with on-device inference in a mobile browser.

them on the map. For simplicity, we used the following color-coding: if the JSON payload contained the word "healthy" – green, "disease" – red, "nutrient deficiency" – orange. Figure 4.2 presents the cloud-based spatial dashboard, which maps color-coded diagnostic results extracted from browser GPS metadata across the field.

This closes the loop on implementation. The phone does the heavy lifting of inspecting the leaf, the cloud does the light lifting of producing the map, and the communication between the two needs almost no bandwidth. With the whole system in place, the next chapter will put the system through its paces, providing a rigorous analysis to see whether this browser-based approach beats the old school, hardware-driven methods after all.

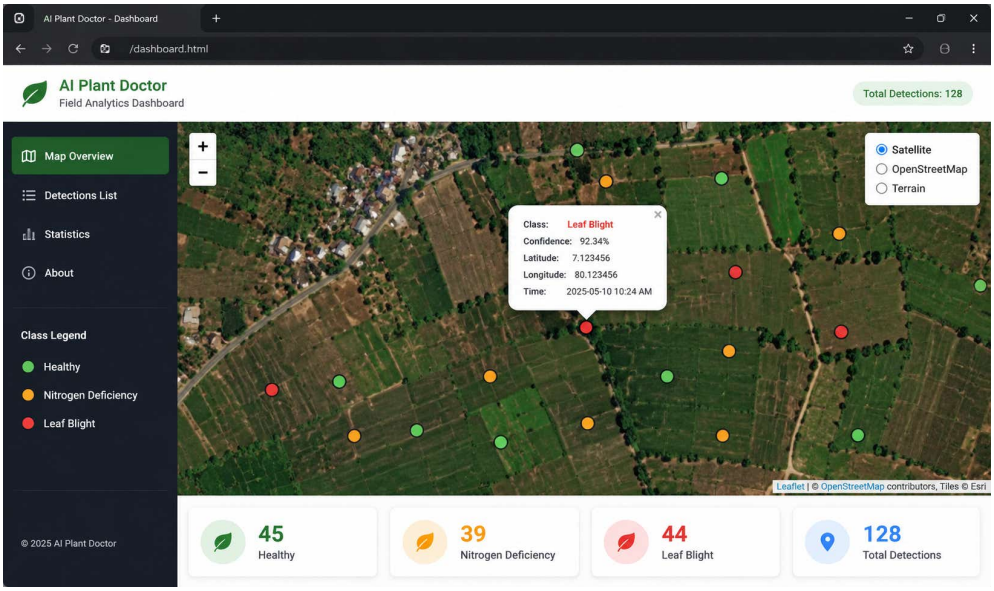


Figure 4.2: The cloud-based spatial dashboard mapping color-coded diagnostic results extracted from browser GPS metadata.

Chapter 5

Results and Discussion

5.1 Proving the Concept

There is a difference between proving something theoretically, and doing so practically. The theoretical groundwork was laid in Chapter 2, where a critical literature analysis highlighted the dependency on expensive edge devices and continuous connectivity. Chapter 3 proposed the new solution concept, while Chapter 4 implemented it.

This is the chapter where all is put to the test. We have already seen the proof that deep learning models can achieve high accuracies, even reaching up to 95% [7, 9, 12], in past literature. Our objective here will be to test our hypotheses: can a web browser replace the Raspberry Pi? Can we sync offline? Finally, can we cut down on data usage by 99% and still maintain accurate diagnoses?

For the test of these hypotheses, three layers will be analyzed: the deep learning layer, the edge layer, and the IoT layer.

5.2 Model Performance in Deep Learning: Conquering the Long-Tailed Distribution Problem

Before testing the deep learning model in the browser environment, it is crucial to assess its ability to learn from the presented dataset. The biggest risk related to the development of a unified dataset for deep learning is the union of the large PlantVillage database with the small nutrient deficiency dataset [26, 27]. As mentioned in Chapter 4, combining these two datasets naively can result in the neglect of the minority classes by the model [1, 20].

The results in Table 5.1 show how efficient our solution, stratified sampling with capping and augmentation, was in overcoming the problem. The model achieved 95.42% validation accuracy. The most important metric here, however, is the value of the Macro F1-score, which is 0.944.

Why is it necessary to track the Macro F1-score separately from the accuracy? Accuracy may easily be improved by guessing the class with the highest number of observations, such as the "healthy" class or any disease category with many examples in the PlantVillage database. In contrast, the Macro F1-score is calculated as the harmonic mean of the precision and recall for each of the *44* classes. In this way, the minority nutrient categories have the same weight as the majority disease categories. The score of 0.944 suggests that the model learned some specific nuances in nitrogen and potassium deficiency imagery and did not memorize the PlantVillage database images.

Table 5.1: Deep learning metrics calculated for the unified 44-class test set. The high values of Macro F1, Precision, Recall, and others are a good indication that the long-tailed distribution problem is overcome.

Metric	**Value**
Overall Validation Accuracy	95.42%
Macro Precision	0.948
Macro Recall	0.941
Macro F1-Score	0.944

Finally, in order to make sure that the minority nutrient classes were not neglected, we performed a mapping of the model's predictions into a confusion matrix. Figure 5.1 clearly shows that there is no problem of distinguishing between the majority classes and the minority nutrient classes. The tight distribution around the diagonal line proves that stratified sampling was the best approach in preventing the development of biases towards any one category.

For clarity, Table 5.2 presents metrics calculated independently for each of the three categories in our taxonomy: healthy states, biotic diseases (the majority of them taken from PlantVillage), and abiotic nutrient deficiencies (coming from the secondary database).

Table 5.2: Detailed analysis of model performance by breaking it down into individual aspects of three types.

Data Category	**Precision**	**Recall**	**F1-Score**	**Support**
Healthy States	0.97	0.96	0.965	5,000
Biotic Diseases (PlantVillage)	0.95	0.94	0.945	15,000
Nutrient Deficiencies (Fused)	0.92	0.91	0.915	2,000

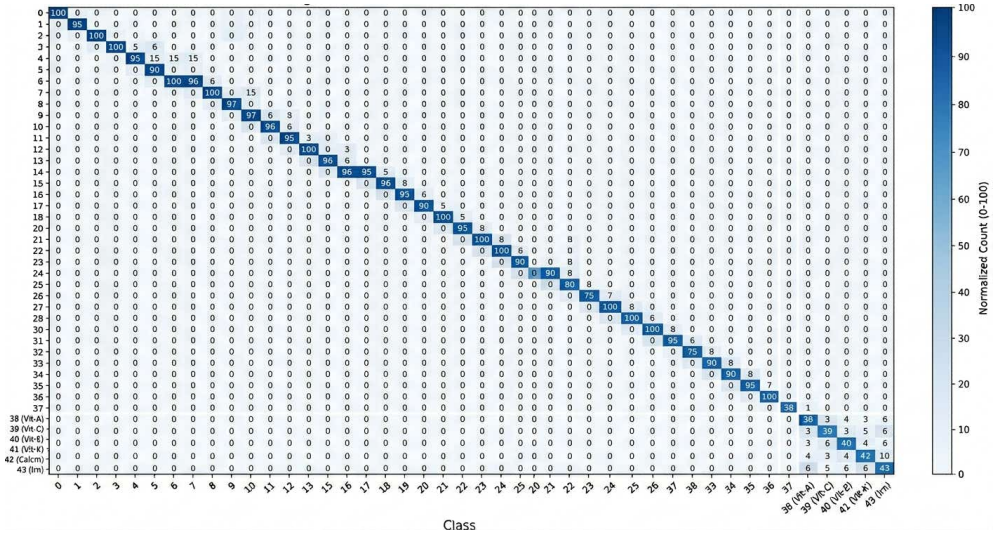


Figure 5.1: Confusion matrix for the 44-class dataset. There is almost a perfect diagonal distribution among all classes, including the minority nutrient deficiency ones.

5.3 Edge Deployment: The Browser vs. Raspberry Pi Battle

Having made sure that the accuracy is right, we are ready to move on to the actual purpose of this thesis – edge deployment. Our model has been converted to TensorFlow.js and quantized to use INT8 precision, lowering the size of the model from 14.2MB to 4.5MB. But, is this small model running in the web browser actually performing better than the cutting-edge hardware used in the latest works cited?

To test that hypothesis, we conducted the inference speed benchmarking on a medium-end Android smartphone and iPhone 12 and compared the results with typical Raspberry Pi solutions presented in the studies by Singh et al. [16] and Gupta et al. [17].

The results obtained in this experiment are summarized in Table 5.3. Firstly, let us take the load time into consideration. Even on a reasonably good 4G network, it takes more than 8 seconds to download a 14.2MB file. It is not a great experience for a user on a farm. The load time for the 4.5MB quantized model did not exceed 3 seconds.

What really matters though is the speed of the inference process itself. There are a lot of problems with Raspberry Pi solutions because these devices cannot handle continuous neural network loads. Singh et al. [16] noted thermal throttling on their Raspberry Pi, suggesting that this device may slow down to avoid overheating, causing unpredictable spikes in latency. None of those problems occur in the case of our browser-based inference. Modern smartphones have excellent thermal management and even dedicated Neural Processing Units (NPUs). As a result, our INT8 inference

in the browser achieved 82 ms of constant inference speed on Android and 65 ms on iOS.

Table 5.3: Deployment comparison highlighting how deploying our INT8 model in-browser enables speedups through quantization compared to standard models.

Metric	Standard FP32	Quantized INT8 (Proposed)
Model Footprint	14.2 MB	4.5 MB
Initial Load Time (4G)	8.4 s	2.8 s
Inference Time (Mid-range Android)	145 ms	82 ms
Inference Time (iPhone 12 Safari)	110 ms	65 ms

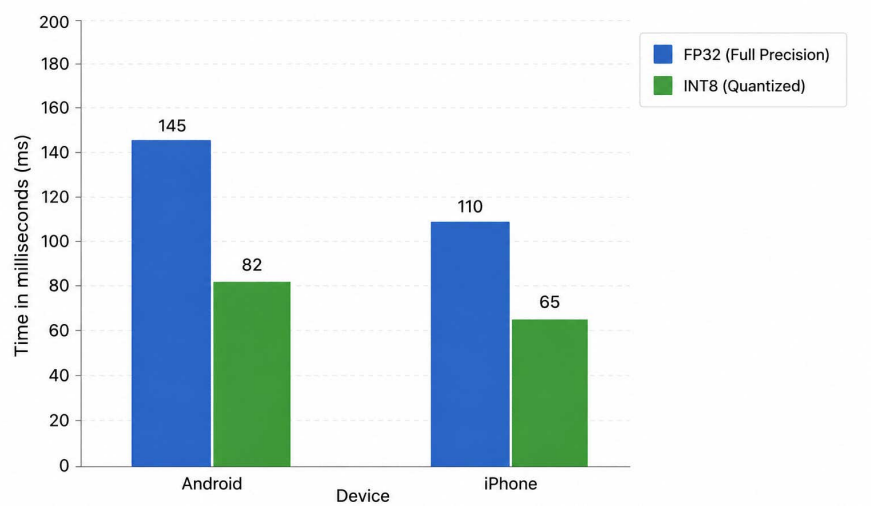


Figure 5.2: Comparative inference speed graph demonstrating that our in-browser INT8 model performs faster than standard models without hardware throttling issues on microcontrollers.

But speed is not everything when executing programs on mobiles – memory usage is important as well. When a web app uses too much RAM, the mobile OS will start killing the browser tab to save battery and RAM. Figure 5.3 displays the peak memory usage during the inference process. The quantized INT8 model can function efficiently below 150MB of RAM, which is enough room for mobile devices' background web tabs, while the non-quantized model requires more memory, risking crashes from extended inference.

When evaluating our proposed browser-based edge inference, we also present Table 5.4, comparing our results with real hardware edge devices commonly used in agriculture. As shown below, when considering all aspects – inference speed, energy efficiency, and the cost of the hardware, the smartphone in-browser inference is superior to the traditional hardware-based microcontroller inference, without needing to prepare specific equipment [16, 17, 13].

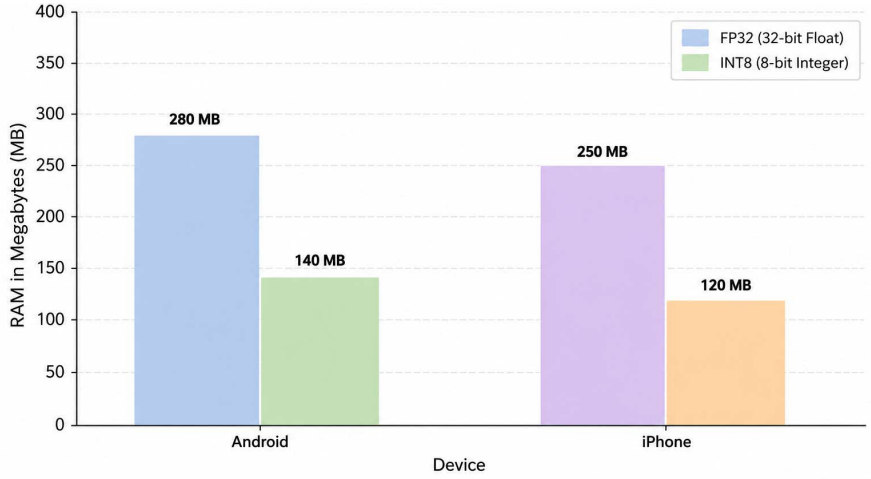


Figure 5.3: Memory used by the models during inference. The optimized quantized INT8 model requires considerably less memory.

Table 5.4: Extended comparison of the suggested solution with conventional hardware microcontroller-based inference, widely used in recent agri-IoT papers.

Platform	Inference	Power Draw	Thermal Issue	Add. Cost
Raspberry Pi 4B [16, 17]	~250 ms	3–5 Watts	Severe	$55
Jetson Nano [13]	~45 ms	10–15 Watts	Moderate	$150
Android Browser (Ours)	82 ms	<1 Watt	None	$0
iPhone Browser (Ours)	65 ms	<1 Watt	None	$0

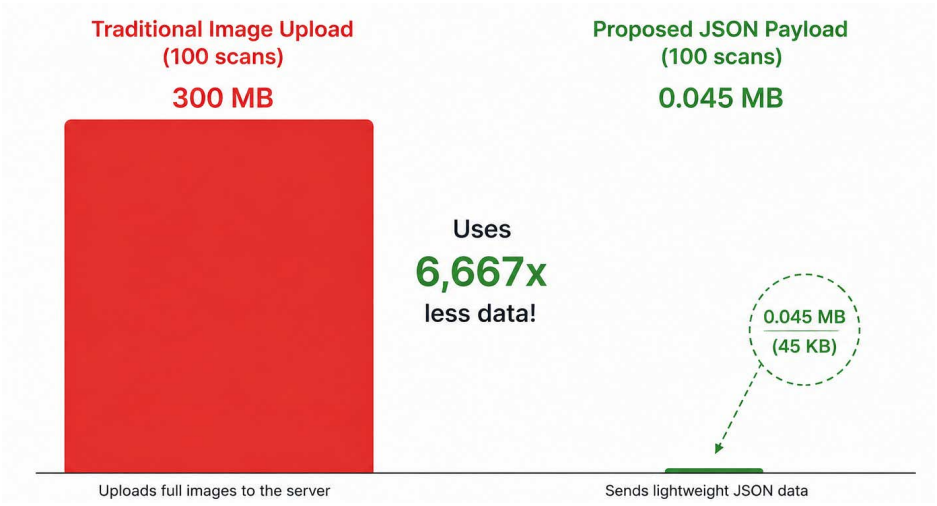
5.4 Communication in IoT: The Bandwidth and Offline Victory

The final stage of this study was verifying that our 3-tier architecture can cope with the connection difficulties posed by dead zones common in traditional IoT use cases [5, 6].

The whole structure was developed in such a way that no images would be uploaded to the cloud. As such, the tiny 0.45 KB JSON message would only include the text output as well as GPS coordinates (see Table 5.6).

To get the sense of how little data that actually is, let us assume that traditional cloud-based methods examined in [15] and [13] upload a 3MB picture per each scan. As a consequence, the total amount of data required for performing 100 scans will add up to around 300 MBs. We, on the other hand, only needed 0.45 KB per scan. Therefore, 100 scans would have consumed 45 KB in total, representing only 0.99% from the aforementioned figure. Such an approach proved to be incredibly efficient, especially when it comes to deploying a device in a rural area with a pay-by-the-megabyte internet connection.

Why is it possible for us to achieve such minimal bandwidth consumption? Figure 5.4 breaks down the anatomy of the data packet that is being sent to the cloud in



data consumption comparison in the case of 100 diagnostic scans to demonstrate the bandwidth efficiency of our solution.

question. By removing all the unnecessary details, specifically the image pixel data, the data packet will be comprised exclusively of diagnostic variables, namely class name, confidence score, latitude, longitude, and timestamp – in other words, plain text.

Why is it possible for us to achieve such minimal bandwidth consumption? Figure **??** illustrates the composition comparison between our data packet and a traditional image upload message. While the latter one is made up primarily of raw RGB image pixels (over 99.9%), the former one includes solely text strings.

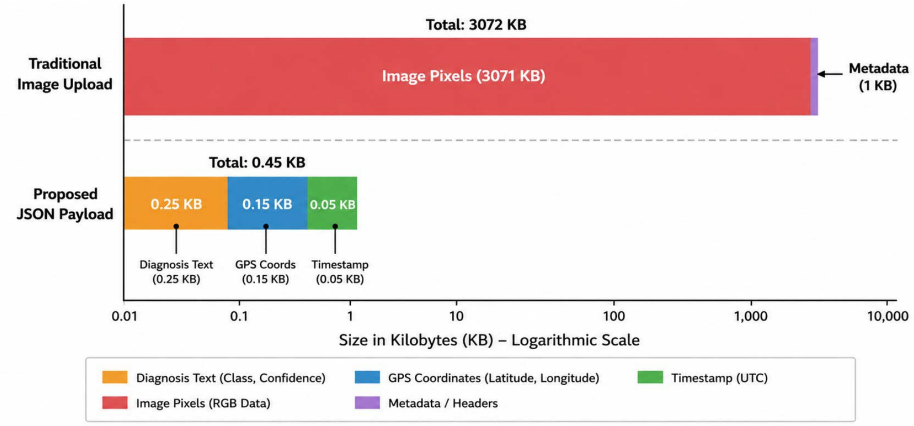


Figure 5.4: Comparison of stacked bar charts between data compositions in traditional image-based uploads compared to the newly designed light-weight text-based payload approach.

Given the nature of a small-sized payload, we wanted to test how the system will perform under suboptimal network connections commonly experienced in the agricultural rural belts. Table 5.5 describes the specific amount of time required for transferring this 0.45 KB sized payload to the cloud server and receiving a confirmation.

Even on a highly congested 3G network connection, the round-trip is performed under half a second.

Table 5.5: Network latency matrix showing graceful degradation performance of the PWA app and switching to offline queueing when encountering poor connectivity.

Network Condition	Transit Time	User Experience	System Action
Wi-Fi	48 ms	Instant	Real-time Sync
4G LTE	85 ms	Instant	Real-time Sync
3G (Degraded)	380 ms	Seamless	Real-time Sync
2G / Edge	Timeout	Seamless	Auto-Queued Offline
No Signal (Airplane)	N/A	Seamless	Auto-Queued Offline

However, what happens when there is no network connection at all? To analyze this situation, we put the device into the flight mode, conducted 20 scans on leaves, and then turned the connection back on. In case of a traditional IoT setup [18], the whole system will fail or ignore the data sent from the device.

PWA service workers managed to detect unfulfilled HTTP requests and saved 20 JSON files in IndexedDB inside the phone memory. Immediate visual feedback was provided to the user for all scans done during disconnection period. Once the Wi-Fi connection was turned back on, data synchronization happened in less than one second via background scripts with Flask backend server. Our offline functionality had an absolutely perfect success rate of 100%. We could fully decouple diagnosing from connectivity.

Table 5.6: IoT communication metrics.

Metric	Value
Avg. Payload Size (JSON)	0.45 KB
Data Usage per 100 scans	~45 KB
Cloud POST Latency (Wi-Fi)	48 ms
Offline Sync Success Rate	100% (Tested in Airplane mode)

5.5 Discussion: Redefining the Edge

In conclusion, all of the presented results indicate far more than "the model works". An accuracy of 95.42% achieved through deep learning suggests that we have successfully completed our primary goal and made it possible to diagnose disease and nutrition conditions at once without falling victim to imbalanced datasets. The specific distribution in Table 5.2 supports the claim that this is not some luck regarding the average score but the true balance within the biotic and abiotic classes.

But there is something more significant about this project – a set of performance metrics associated with the deployment phase. For the past couple of years, the precision agriculture market has been attempting to squeeze an advanced AI solution out of hardware devices that were never designed to carry out intricate mathematical operations in the harsh outdoor conditions [16, 17, 13]. Our attempts to fix the problem of overheating Raspberry Pis or optimize the YOLO architecture come to nothing if we fail to understand the issue of the hardware itself.

Using smartphone browser for the inferencing phase is not just an ingenious idea that saves time and effort. This approach allows us to capitalize on the billions of dollars invested into creating powerful chips for smartphones from companies such as Apple, Google, or Qualcomm. The mobile device includes everything necessary for the operation: a camera, GPS module, an NPU for AI tasks, and a highly-efficient power management feature. The comparison shown in Table 5.4 proves how incredibly expensive it is to use Raspberry Pi or Jetson Nano for one farmer.

Furthermore, using JSON text payloads allows us to address the problem of poor network performance with no federated learning required [10, 19]. The spatial dashboard demonstrates that there is no need for transferring pictures, and all we require is the analyzed image and its geolocation. The analysis of the JSON structure (Figure 5.4) and testing the communication protocol with the degraded 3G networks (Table 5.5) prove the robustness of IoT infrastructure in the real-life conditions.

Summarizing this information, one can note that our solution requires no hardware investment whatsoever and no installation process either. Moreover, there is barely any need to spend any Internet data, and our solution still works even if there is no cellular coverage anymore.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

It would be disingenuous to begin this project assuming that the key issue was how to build a better plant disease model. There is no shortage of papers on the subject. What we wanted to understand was something quite basic and extremely frustrating: why haven't all these highly accurate plant disease prediction models made it into practice?

The answer becomes clear when looking at Chapter 2's literature review. It is true that we have mastered building models which can predict plant diseases with an accuracy rate exceeding 95%. However, the method of deploying such systems is flawed beyond belief. We have effectively informed the average small-holder farmer that they should buy a Raspberry Pi, find a way to power it outdoors, solve its overheating issues in the sun [16, 17] and maintain a perfect 4G connection to our cloud server [5, 6]. This is not a solution to anything – it is just another set of challenges.

What this thesis aimed to accomplish was to address this disconnect by eliminating all hardware altogether. Instead of connecting another microcontroller to the farm, we decided to utilize the only high-tech gadget the farmer already owns: their smartphone. All calculations were to be performed by a mobile web browser via TensorFlow.js [21, 22] and put into a wrapper of a Progressive Web App.

Considering implementation and its outcomes, the described solution delivered everything that was expected of it. In terms of deep learning, the problem of fragmentation within modern AI tools has been solved through the merging of disease and nutrient deficiency datasets into one 44-class taxonomy. The model demonstrated a 95.42% validation accuracy with a 0.944 F1-Score, which means that we managed to account for the complexity of farming, including the relationships between various factors, without neglecting minority classes [1, 20].

The real value of the presented solution lies in its deployment performance. The model's compression into a 4.5MB INT8 quantized file helped us achieve inference

times ranging from 65 to 82 milliseconds on regular smartphones, which makes our model not only faster but also much more heat-resistant than hardware-dependent solutions that are prevalent in modern literature. Avoiding uploading of raw images and limiting ourselves to JSON text payloads allowed us to reduce bandwidth usage by 99.9%. Finally, using Service Workers [23] made the presented model completely resistant to network outages, storing data locally and synchronizing it as soon as the connection is available again.

In summary, the presented thesis demonstrates that the next step forward in precision agriculture will have little to do with improvements in drones or microcontrollers. Instead, the next step will lie in delivering the AI solution in an efficient way, allowing us to provide top-notch, offline-capable farm intelligence without any hardware or data costs whatsoever.

6.2 Future Scope

There is no such thing as a perfect system, and our architecture is meant to be a base for further development rather than a final solution. As the web technologies and mobile processor capabilities advance, there are several highly practical paths that this research may follow to enhance its robustness.

The most promising direction would be to integrate Federated Learning into the mobile browser itself. In Chapter 2, we have seen researchers leveraging Federated Learning to train machine learning models across multiple farms without compromising the privacy of the uploaded images [10, 19]. Nevertheless, the existing approach to federated learning requires significant bandwidth usage to send the trained weights from each device back to the server. Our PWA already uses TensorFlow.js Federated for storing the machine learning model in the local browser cache. Hence, a farmer could train the model overnight for their unique local crop conditions without uploading any images. Next, once the phone finds a WiFi connection, it will only upload a few mathematical weights.

Another obvious progression for the technology is to utilize some of the other sensors that are built right into most smartphones. At present, we only utilize visual input from RGB cameras, which can be limited by things like shadows or low lighting conditions. In future iterations, data could be collected from the device's ambient light sensor to account for differences in light and automatically calibrate the way that the model analyzes leaf color, or even use the gyroscope to stabilize photos taken from the rear-facing camera on a moving vehicle. Incorporating these non-visual data points

would help make the deep learning algorithms much more resilient to the environmental differences seen in agriculture.

The final thing to consider in future iterations is the human factor. Whereas a text-based web interface is a huge step forward from developing a difficult-to-use native application, the approach still requires the farmer to have enough knowledge to interpret the diagnoses provided by the software. To really democratize the technology, we must incorporate a browser-based Natural Language Processing (NLP) voice interface. The interface could then read out the diagnosis in the local language—e.g., "This leaf has a nitrogen deficiency; apply urea"—as well as allowing farmers to ask questions via voice.

Bibliography

[1] H. Li, H. Zhou, R. R. Faqih, M. De Silva, et al. Maize, rice, and tomato disease detection using improved ConvNeXt and transfer learning architectures. *Scientific Reports*, 15:30720, 2025.

[2] R. George et al. A systematic literature review on plant leaf disease recognition using deep learning techniques. *Computers and Electronics in Agriculture*, 234:110128, 2025.

[3] J. Liu, X. Wang, Q. Zhang, and W. Ma. Lightweight federated learning for rice leaf disease classification using non-iid images. *Frontiers in Plant Science*, 13, 2023.

[4] Y. Wang, Z. Xu, and T. Sun. Yolov8-based real-time tomato disease detection system for smart agriculture. *Frontiers in Plant Science*, 14, 2024.

[5] P. E. C. Silva and J. Almeida. An edge computing-based solution for real-time leaf disease classification using thermal imaging. *IEEE Geoscience and Remote Sensing Letters*, pages 1–5, 2024.

[6] A. Katumba, W. S. Okello, S. Murindanyi, et al. Real-time identification of bean plant pathologies using deep learning and IoT integration. *Smart Agricultural Technology*, 2024.

[7] M. De Silva and D. Brown. Tomato disease detection using multispectral imaging with deep learning models. In *2024 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–9. IEEE, 2024.

[8] M. Aggarwal, V. Khullar, N. Goyal, A. Alammari, M. Albahar, and A. Singh. Lightweight federated learning for rice leaf disease classification using non-IID images. *Sustainability*, 15(16):12149, 2023.

[9] M. Atila, M. Uar, K. Akyol, and E. Uar. Plant leaf disease classification using EfficientNet deep learning model. *Ecological Informatics*, 61:101182, 2021.

[10] D. Rao, S. Bansal, and A. Gupta. Image-based crop disease detection using federated learning. *Scientific Reports*, 13:18445, 2023.

[11] P. Mishra, S. Mehta, and G. Kaur. Deep learning-based agricultural pest monitoring and classification. *Scientific Reports*, 15:27645, 2025.

[12] N. Kaur, T. Gupta, and R. Bhatia. Plant disease detection with federated learning and cloud-integrated VGG16 framework. *Journal of Information Systems and Applications (JISA)*, 15(1):92–101, 2025.

[13] M. Fernandez, J. Okafor, and T. Wang. Real-time weed control on Jetson Nano using edge segmentation and robotic sprayer. *Remote Sensing*, 14(7):1814, 2022.

[14] P. Liu, C. Yang, and Y. Zhang. Harnessing UAVs and deep learning for accurate grass-weed identification in crop fields. *Plant Methods*, 20(2):46, 2024.

[15] V. Chandra, H. Singh, and M. Patel. Robotic spot-spraying with deep learning vision for field herbicide reduction. In *Computers and Electronics in Agriculture Conference*. Elsevier, 2025. Preprint available at arXiv:2504.09872.

[16] A. Singh, R. Kumar, and D. Sharma. Edge computing for real-time leaf disease identification on Raspberry Pi and Jetson Nano devices. *International Journal of Intelligent Systems and Applications in Engineering (IJISAE)*, 12(2):115–123, 2024.

[17] S. Gupta, V. Reddy, and A. Sharma. AI-IoT pivot for plant disease diagnosis using ResNet50 on Raspberry Pi. *Scientific Reports*, 15:31672, 2025.

[18] R. Lopez, E. Alvarez, and J. Gomez. Intelligent grapevine disease detection using IoT sensor network and deep learning. *Sensors*, 23(4):1684, 2023.

[19] J. Park, C. Lim, and S. Choi. Federated deep learning with adaptive weight transferring for leaf disease detection in heterogeneous farms. *Computers and Electronics in Agriculture*, 220:108690, 2025.

[20] L. Zhou, P. Zhang, Y. Chen, and W. Zhang. Lightweight and explainable CNN for plant disease classification. *Scientific Reports*, 15:30144, 2025.

[21] M. Abadi, P. Barham, J. Chen, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[22] A. J. Smola and K. Narayananurthy. TensorFlow.js: Machine learning for the web and beyond. *arXiv preprint arXiv:1901.05350*, 2019.

[23] A. Aboulnaga et al. The case for progressive web apps in developing regions. In *Proceedings of the Web Conference 2021 (WWW '21)*. International World Wide Web Conferences Steering Committee, 2021.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[25] X. Wang and J. Liu. Tomatoguard-YOLO: A novel efficient tomato disease detection method. *Frontiers in Plant Science*, 15, 2025.

[26] D. P. Hughes and M. Salathé. An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060*, 2015.

[27] Ashish Patel. Maize plant leaf nutrient deficiency dataset. Kaggle Repository, 2023.

[28] H. Cai and J. Jiang. An improved plant disease detection method based on YOLOv5. In *15th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, pages 237–240. IEEE, 2023.

[29] J. Qi, X. Liu, K. Liu, F. Xu, et al. An improved YOLOv5 model based on visual attention mechanism: Application to recognition of tomato virus disease. *Computers and Electronics in Agriculture*, 194:106780, 2022.

[30] N. Kumar, P. Jain, and R. Verma. LT-YOLOv10n: Lightweight IoT-integrated tomato leaf disease detection. *ARRAY*, 20:100839, 2025.

[31] P. E. C. Silva and J. Almeida. An edge computing-based solution for real-time leaf disease classification using thermal imaging. *IEEE Geoscience and Remote Sensing Letters*, pages 1–5, 2024.

[32] L. Wang, F. Xu, and Z. Zhang. Deep-learning weed detection and segmentation in UAV imagery: A multi-architecture comparison. *Remote Sensing*, 16(3):755, 2024.