



الجمهورية الشعبية الديمقراطية الجزائرية  
وزارة التعليم العالي والبحث العلمي  
جامعة سعيدة د. مولاي الطاهر  
كلية الرياضيات والاعلام الالي والاتصالات السلكية والاسلكية  
قسم : الاعلام الآلي

## Graduation Thesis in Computer Science

Speciality: **Computer Networks and Distributed Systems**

### T h è m e

#### **Skyline**

An intelligent and scalable drone  
As a Service Platform with Integrated Drone  
Programming Capabilities

- **Présenté par :**

Benhalima Mokhtaria Yamina Nadjlaa  
Mokhtari Khouloud Yasmine

- **Dirigé par :**

Dr. Mostefai Abdelkader

﴿يَرْفَعُ اللَّهُ الَّذِينَ آمَنُوا مِنْكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ﴾

صدق الله العظيم

## إهداء

إلى نفسي... إلى تلك التي وَقَفْتُ وحيدةً في وجهِ التعبِ فلم تنكسر، التي بَكَتِ الليلَ وابتسمت للصباح، وزَرَعَتْ في كُلِّ سَهْرَةٍ حُلماً، وَسَقَتْهُ مِنْ دمعها وإصرارها حتى أزهَرَ...

أهديكِ هذا النصر يا مَنْ لم تُسَلِّبِي الراية يوماً، فما وَصَلَتْ بالصدفة، بل بالكفاح والإيمان.

إلى مَنْ سَكَنَ دعاؤها كُلَّ خُطوة، وَمَنْ لو جمعتُ الدنيا في كَفِّي ما عادتْ نظرةٌ نَفَرَ في عينيها، إلى نبضِ قلبي ومعنى وجودي ومأواي حين يضيقُ كُلُّ شَيْءٍ...

أُمِّي الغالية **فَتْحَة**، يا جَنَّتِي التي تحت قدميها.

إلى مَنْ حَمَلَ أحلامي على كتفيه دون أن يشتكي، وكان لي الجِدَارَ الذي أحتمي به والقُدوةَ التي أقتدي بها، إلى مَنْ عَلَّمَنِي أَنَّ التعبَ شَرَفٌ والصبرَ طريقُ المجد...

أبي الحبيب **مُحَمَّد**، نخري الذي لا يضاها.

إلى القلوبِ التي أحاطتني بحبِّ بلا حدود، وكانت لي أماً ثانيةً وسنداً لا يَنقُطع...

خالتي الحنونة **نَصِيرَة**، وخالي الغالين **مُحَمَّد** و**خالد**.

إلى رِفاقِ العُمَرِ الأولين، إلى مَنْ قاسَمُونِي الضحكةَ والدمعةَ، وكانوا سندي حين يَخْذُلُ العالم...

إخوتي وقطعةً مِنْ قلبي **جلّول**، **حميدة**، و**حُسام**.

إلى الأختِ التي لم تَلِدْها أُمِّي لكنَّ القَدَرَ حَطَّ اسمَها في روحي، رفيقةِ الدَّربِ ونصفِ القلب، وَمَنْ كانت لي وطناً كلَّما تَنَكَّرْتُ لي الدنيا...

صديقتي وحبيبتي **هاجر**، شكراً لَأَنَّكَ ضَوْءٌ لا ينطفئ.

إلى نِصفِ روحي ورِفاقِ هذي الرحلة، مَنْ حوَّلُوا أيَّامي العاديةَ إلى ذكرياتٍ تُضيءُ العُمُر...

أحبَّائي **خلود**، **دعاء**، **فاطيمة**، **مليكَة**، و**خيرة**.

إلى كُلِّ قَلْبٍ أَحَبَّنِي بِصِدْقٍ، وكلِّ يَدٍ اِمْتَدَّتْ لَتُعِينَنِي، وكلِّ دَعْوَةٍ صَعِدَتْ لِأَجْلي في صَمْتٍ خاشع...

أهدي ثَمَرَةَ هذا العمل، عرفاناً وحُبّاً لا يَنْضُب.

نَجْلاء

﴿ وَمَا تَوْفِيقِي إِلَّا بِاللَّهِ عَلَيْهِ تَوَكَّلْتُ وَإِلَيْهِ أُنِيبُ ﴾

[سورة هود: الآية ٨٨]

## إهداء

إلى نفسي أولاً... إلى مَنْ خاضت هذا الدرب الطويل بكل ما فيه من تعب وأمل، إلى مَنْ بكت في السرّ وابتسمت في الظاهر، وصمدت حين أراد اليأس أن يكون سكنها، إلى روعي التي آمنت بالله وبنفسها حين لم يؤمن بها أحد... هذا العمل ثمرة إرادتك، ودليل على أنك استحققت كل نجاح. ♦ أنتِ نَفْسُكِ

إلى أبي العزيز... إلى مَنْ كان شجرة صامدة في وجه رياح الحياة ليظل أبنائه في ظلها آمنين، إلى مَنْ علّمني بصمته قبل كلامه أن الرجل الحقيقي مَنْ يبني ولا يهدم، إلى مَنْ أتعب نفسه ليريحنا وضحّى بأحلامه لتحقيق أحلامنا... أطال الله في عمرك، وأدامك نفراً لنا يا أبي الغالي. ♦ كل نجاح حققته بدعمك هو هدية لك

إلى أمي الغالية... إلى مَنْ حملتني وهنّاً على وهن، وسهرت ليالي طوالاً لتريني نهراً مشرقاً، إلى مَنْ كان دعاؤها جنتي ورضاها هدي في كل خطوة، إلى قلب لا يعرف الكره ويد لم تتعب من العطاء... رب ارحمها كما ربّني صغيراً، وأدم عليها الصحة والعافية. ♦ أمي جنتي في الدنيا قبل الآخرة

إلى جدّي الحبيبتين... رحلتا عن هذه الدنيا وتركنا في قلبي فراغاً لا يملؤه سوى الدعاء، كنت أريدك أن تريا هذه اللحظة وتفخرا بما وصلتُ إليه... رحمكما الله وأسكنكما الفردوس الأعلى برحمته. ♦ غيابكما علّمني أن الوقت أغلى من الذهب

إلى خالاتي القريبات من روعي: **خيرة ♦ جميلة ♦ فائزة** — مَنْ اتسعت قلوبهن لفرحي وحزني على حدّ سواء، كلمة تشجيع منكن كانت تعيد إليّ الأمل في أصعب اللحظات... الله يحفظكن ويديم عليكن نعمة الوفاق والصحة.

إلى أخواني العزيزات: **نزيان ♦ حورية** — رفيقتا دربي، مَنْ تقاسمتا معي أبجدية هذه الحياة حرفاً حرف، صحبتكما نعمة لا تُقدّر وأخوتكما وطن لا يُهجر... وإلى ابنة أختي القادمة، البرعمة التي تنتظر الدنيا لتزنيها، أهلاً بك يا صغيرتي الغالية في عالم مليء بالحب. أسأل الله أن يجمعنا على الخير دائماً.

إلى أخوتي الأعزّين: **محمد ♦ نجم الدين** — مَنْ احتضنا بكائي بصمتها وكان ملاذي في كل شدة، وجودك في حياتي قوة لا تنضب وسند لا يتزعزع... الله يجعلكما نجمين يضيئان سماء هذه الأسرة.

إلى أحبّتي الصغار: **جواد ♦ ريان ♦ ياسين ♦ خديجة** — أتمّ البهجة التي تعيد البسمة إلى كل تعب ووجع، أتمنى أن تروا في مثالي أيشجعكم على مواصلة التعلم وبلوغ أعلى الدرجات... لكل منكم مستقبل مشرق ينتظر أن تضيئوه.

إلى صديقاتي اللواتي أضأن دربي: **ريهام ♦ مليكة ♦ نجلاء ♦ هاجر ♦ دعاء ♦ إيمان ♦ فريال** — مَنْ آسنيني في ليالي الدراسة الطويلة حين ثقل الكُتاب وأرهق الفكر، مَنْ شاركني الضحكة والدمعة ولم يغادرني حين اشتد الإعياء، صداقتكن فصل جميل لا يطوى من كُتاب عمري... الله يحفظكن ويجعل صداقتنا خالدة إلى يوم اللقاء.

♦ هـ ♦

﴿ رَبِّ أَوْزِعْنِي أَنْ أَشْكُرَ نِعْمَتَكَ الَّتِي أَنْعَمْتَ عَلَيَّ وَعَلَى وَالِدَيَّ ﴾

[سورة النمل: الآية ١٩]

شكراً لكم.

# Contents

|                                                       |          |
|-------------------------------------------------------|----------|
| <b>General Introduction</b>                           | <b>1</b> |
| <b>1 Background</b>                                   | <b>5</b> |
| 1.1 Introduction                                      | 5        |
| 1.2 Unmanned Aerial Vehicles (UAVs)                   | 5        |
| 1.2.1 Definition                                      | 5        |
| 1.2.2 Types of UAVs                                   | 6        |
| 1.2.3 Hardware Architecture                           | 7        |
| 1.2.4 Drones in Different Fields                      | 7        |
| 1.3 Data as a Service (DaaS)                          | 8        |
| 1.3.1 Cloud Computing                                 | 8        |
| 1.3.2 Definition of DaaS                              | 8        |
| 1.3.3 The Five Principles of DaaS                     | 8        |
| 1.3.4 DaaS in the Drone-AI Pipeline                   | 9        |
| 1.4 Artificial Intelligence                           | 9        |
| 1.4.1 Definition                                      | 9        |
| 1.4.2 History of AI                                   | 10       |
| 1.4.3 Main Branches of AI                             | 10       |
| 1.5 Machine Learning                                  | 10       |
| 1.5.1 Definition and the Key Idea                     | 11       |
| 1.5.2 Types of Learning                               | 11       |
| 1.6 Deep Learning                                     | 12       |
| 1.6.1 What Makes Deep Learning Different              | 12       |
| 1.6.2 The Artificial Neuron                           | 13       |
| 1.6.3 Activation Functions                            | 13       |
| 1.6.4 Major Deep Learning Architectures               | 14       |
| 1.7 Convolutional Neural Networks and Computer Vision | 14       |
| 1.7.1 Computer Vision                                 | 14       |
| 1.7.2 Convolutional Neural Networks (CNN)             | 15       |
| 1.7.3 Core CNN Operations                             | 15       |

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| 1.7.4    | Transfer Learning . . . . .                                | 16        |
| 1.7.5    | Data Augmentation . . . . .                                | 16        |
| 1.7.6    | Explainability: Grad-CAM . . . . .                         | 17        |
| 1.7.7    | Handling Class Imbalance . . . . .                         | 17        |
| 1.8      | Internet of Things (IoT) . . . . .                         | 17        |
| 1.8.1    | Definition . . . . .                                       | 17        |
| 1.8.2    | The Four-Layer IoT Architecture . . . . .                  | 17        |
| 1.8.3    | IoT Sensors in Smart Farming . . . . .                     | 18        |
| 1.8.4    | Communication Protocols . . . . .                          | 18        |
| 1.9      | Security in IoT and Drone Systems . . . . .                | 19        |
| 1.9.1    | Overview . . . . .                                         | 20        |
| 1.9.2    | Threat Taxonomy . . . . .                                  | 20        |
| 1.9.3    | The CIA Triad . . . . .                                    | 20        |
| 1.9.4    | Security Controls . . . . .                                | 20        |
| 1.10     | Development Tools and Technologies . . . . .               | 21        |
| 1.11     | Programming Languages . . . . .                            | 22        |
| 1.11.1   | C++ . . . . .                                              | 22        |
| 1.11.2   | Python . . . . .                                           | 23        |
| 1.11.3   | HTML & CSS . . . . .                                       | 24        |
| 1.12     | APIs and Web Services . . . . .                            | 24        |
| 1.12.1   | What is an API? . . . . .                                  | 24        |
| 1.12.2   | REST API . . . . .                                         | 25        |
| 1.13     | Database Technologies . . . . .                            | 26        |
| 1.13.1   | SQL (Structured Query Language) . . . . .                  | 27        |
| 1.13.2   | Supabase (Backend-as-a-Service) . . . . .                  | 28        |
| 1.14     | Hardware: Development Computers . . . . .                  | 29        |
| 1.14.1   | What is RAM? . . . . .                                     | 29        |
| 1.14.2   | Machine 1: Dell Latitude 5490 . . . . .                    | 29        |
| 1.14.3   | Machine 2: Desktop Workstation . . . . .                   | 30        |
| 1.15     | Simulation Environment: Unreal Engine and AirSim . . . . . | 30        |
| 1.15.1   | Unreal Engine 4.27 (UE4) . . . . .                         | 30        |
| 1.15.2   | AirSim (Microsoft) . . . . .                               | 31        |
| 1.15.3   | Visual Studio Build Tools 2022 . . . . .                   | 32        |
| 1.15.4   | Vision Transformer (ViT) vs. CNN . . . . .                 | 34        |
| <b>2</b> | <b>State of the Art</b> . . . . .                          | <b>35</b> |
| 2.1      | Introduction . . . . .                                     | 35        |
| 2.2      | CNN Architectures for Plant Disease Detection . . . . .    | 36        |
| 2.2.1    | Early Works and the PlantVillage Dataset . . . . .         | 36        |

|       |                                                                                                              |    |
|-------|--------------------------------------------------------------------------------------------------------------|----|
| 2.2.2 | Classical Architectures: VGG, ResNet, InceptionV3 . . . . .                                                  | 36 |
| 2.2.3 | Lightweight Models: the MobileNet Family . . . . .                                                           | 37 |
| 2.2.4 | The EfficientNet Family . . . . .                                                                            | 37 |
| 2.3   | Drone-Based Disease-Surveillance Systems . . . . .                                                           | 38 |
| 2.3.1 | Why Drones in Agriculture? . . . . .                                                                         | 39 |
| 2.3.2 | A Systematic Review of UAV-Based Plant Disease Detection (Chin et al., 2023) . . . . .                       | 39 |
| 2.3.3 | A Drone System for Multiclass Plant Disease Detection Using EfficientNetV2 (Albattah et al., 2022) . . . . . | 39 |
| 2.3.4 | AgroVisionNet: A Multimodal CNN–Transformer System (Manoj et al., 2025) . . . . .                            | 40 |
| 2.3.5 | Systematic Review of UAV and AI Integration (2025) . . . . .                                                 | 40 |
| 2.3.6 | CNN Applications in Smart Agriculture: A Multimodal Review (2025)                                            | 40 |
| 2.4   | Transfer Learning and Dataset Challenges . . . . .                                                           | 41 |
| 2.4.1 | Why Transfer Learning? . . . . .                                                                             | 41 |
| 2.4.2 | Class Imbalance: a Critical Problem . . . . .                                                                | 42 |
| 2.4.3 | Data Augmentation for Drone Conditions . . . . .                                                             | 42 |
| 2.5   | Comparative Synthesis of Reviewed Works . . . . .                                                            | 42 |
| 2.6   | Drone Applications in Precision Agriculture . . . . .                                                        | 43 |
| 2.6.1 | Overview of Drone Services in Agriculture . . . . .                                                          | 44 |
| 2.6.2 | Imaging Modalities Used in Agricultural Drones . . . . .                                                     | 45 |
| 2.6.3 | Precision-Agriculture Software Platforms . . . . .                                                           | 45 |
| 2.6.4 | Barriers to Adoption in Agriculture . . . . .                                                                | 46 |
| 2.6.5 | Relevance to This Work . . . . .                                                                             | 46 |
| 2.7   | Drone Applications in Construction and Renewable Energy . . . . .                                            | 47 |
| 2.7.1 | Site Surveying and Topographic Mapping . . . . .                                                             | 47 |
| 2.7.2 | 3D Modelling and Progress Monitoring . . . . .                                                               | 47 |
| 2.7.3 | Inspection and Safety . . . . .                                                                              | 48 |
| 2.7.4 | Integration with Building Information Modelling (BIM) . . . . .                                              | 48 |
| 2.7.5 | Applications in Renewable Energy . . . . .                                                                   | 49 |
| 2.8   | Drone Applications in Logistics and Transportation . . . . .                                                 | 50 |
| 2.8.1 | Last-Mile and Short-Distance Delivery . . . . .                                                              | 50 |
| 2.8.2 | Autonomous Delivery Networks . . . . .                                                                       | 51 |
| 2.8.3 | Regulatory and Infrastructure Constraints . . . . .                                                          | 51 |
| 2.9   | Drone Applications in Environmental Monitoring . . . . .                                                     | 51 |
| 2.9.1 | Wildlife Monitoring and Anti-Poaching . . . . .                                                              | 52 |
| 2.9.2 | Search-and-Rescue Operations . . . . .                                                                       | 52 |
| 2.9.3 | Disaster Assessment and Emergency Response . . . . .                                                         | 52 |
| 2.9.4 | Forest and Ecosystem Monitoring . . . . .                                                                    | 52 |

|          |                                                                 |           |
|----------|-----------------------------------------------------------------|-----------|
| 2.10     | Research Gap . . . . .                                          | 52        |
| 2.11     | Legal Framework for Drone Use in Algeria . . . . .              | 53        |
| 2.11.1   | Overview of Algerian Drone Regulations . . . . .                | 53        |
| 2.11.2   | Drone Categories Under Algerian Law . . . . .                   | 53        |
| 2.11.3   | Authorisation Requirements for Agricultural Drone Use . . . . . | 54        |
| 2.11.4   | Steps to Obtain Authorisation for Deployment . . . . .          | 54        |
| 2.11.5   | Why Legal Compliance Matters . . . . .                          | 55        |
| <b>3</b> | <b>System Modeling and General Architecture</b>                 | <b>56</b> |
|          | Introduction . . . . .                                          | 56        |
| 3.1      | UML Modeling . . . . .                                          | 57        |
| 3.1.1    | Class Diagram . . . . .                                         | 57        |
| 3.1.2    | Use Case Diagram . . . . .                                      | 61        |
| 3.1.3    | Component Diagram . . . . .                                     | 65        |
| 3.1.4    | Sequence Diagram . . . . .                                      | 69        |
| 3.1.5    | Deployment Diagram . . . . .                                    | 71        |
| 3.2      | General Architecture . . . . .                                  | 74        |
| 3.2.1    | Overview of the Project . . . . .                               | 74        |
| 3.2.2    | Architecture Diagram . . . . .                                  | 75        |
| 3.2.3    | Detailed Description of the Layers . . . . .                    | 75        |
|          | Conclusion . . . . .                                            | 77        |
| <b>4</b> | <b>Experimentation</b>                                          | <b>78</b> |
|          | Introduction . . . . .                                          | 78        |
| 4.1      | Dataset . . . . .                                               | 79        |
| 4.1.1    | General Presentation . . . . .                                  | 79        |
| 4.1.2    | Data Collection . . . . .                                       | 80        |
| 4.1.3    | Class Imbalance and Class Weights . . . . .                     | 81        |
| 4.1.4    | Preprocessing . . . . .                                         | 82        |
| 4.1.5    | Data Augmentation . . . . .                                     | 82        |
| 4.1.6    | Data Splitting . . . . .                                        | 83        |
| 4.1.7    | Full Data Pipeline . . . . .                                    | 83        |
| 4.1.8    | Prediction Pipeline (Inference) . . . . .                       | 84        |
| 4.2      | Model Training and Comparison . . . . .                         | 84        |
| 4.2.1    | Transfer Learning . . . . .                                     | 84        |
| 4.2.2    | The Three Tested Models . . . . .                               | 84        |
| 4.2.3    | Common Training Configuration . . . . .                         | 87        |
| 4.2.4    | Model Comparison . . . . .                                      | 88        |
| 4.3      | The Champion Model — EfficientNet-B4 . . . . .                  | 91        |
| 4.3.1    | Model Architecture . . . . .                                    | 91        |

|       |                                                              |            |
|-------|--------------------------------------------------------------|------------|
| 4.3.2 | The Softmax Activation Function . . . . .                    | 92         |
| 4.3.3 | Two-Phase Training and Convergence . . . . .                 | 92         |
| 4.3.4 | Preventing Overfitting . . . . .                             | 93         |
| 4.3.5 | Evaluation Metrics and the Custom Metric Code . . . . .      | 94         |
| 4.4   | Final Results and Analysis . . . . .                         | 97         |
| 4.4.1 | Results Against the Project Targets . . . . .                | 97         |
| 4.4.2 | Per-Class Performance . . . . .                              | 98         |
| 4.4.3 | Explainability with Grad-CAM . . . . .                       | 100        |
| 4.5   | Front-End and Application Implementation . . . . .           | 100        |
| 4.5.1 | General Scenario . . . . .                                   | 101        |
| 4.5.2 | The User Side Requesting a Mission . . . . .                 | 102        |
| 4.5.3 | The Admin Side Reviewing and Launching . . . . .             | 103        |
| 4.5.4 | The Result Field Map and Diagnosis . . . . .                 | 105        |
| 4.5.5 | The Database Supabase . . . . .                              | 108        |
| 4.5.6 | The AI Service FastAPI (Local) . . . . .                     | 108        |
| 4.6   | Drone Simulation AirSim and Unreal Engine . . . . .          | 109        |
| 4.6.1 | Why a Simulation First . . . . .                             | 109        |
| 4.6.2 | Preparing the Environment, Step by Step . . . . .            | 109        |
| 4.6.3 | Connecting the Website to Unreal the Python Bridge . . . . . | 111        |
|       | Conclusion . . . . .                                         | 112        |
|       | <b>General Conclusion</b>                                    | <b>114</b> |
|       | <b>Bibliography</b>                                          | <b>116</b> |

# List of Figures

|     |                                                                                                                                                                                                 |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | This figure shows how the main CNN models have developed over time, moving from large, heavy models to small, light models that can run on a drone. . . . .                                     | 36 |
| 2.2 | A drone flies over a farm field and takes photos in the air. An AI system on the drone then uses these photos to detect disease. . . . .                                                        | 38 |
| 2.3 | The two-phase transfer-learning method. In Phase 1, only the new head is trained while the backbone stays frozen. In Phase 2, the top layers are opened and trained slowly. . . . .             | 41 |
| 2.4 | A drone monitors spring crops in agriculture. Drones provide a view from the sky and collect detailed data over large areas [41]. . . . .                                                       | 44 |
| 2.5 | A modern agricultural drone with a tank and spray arms. Drones are modular: the camera and the tools can be changed for each task, which shows how flexible they are [41]. . . . .              | 45 |
| 2.6 | A drone takes photos of a construction site from the sky. A single flight can show the whole large site, so managers receive up-to-date information at low cost [41]. . . . .                   | 47 |
| 2.7 | A drone with a thermal camera inspects a solar-panel field. Temperature problems reveal the faulty panels, so there is no need to check each panel by hand [50]. . . . .                        | 49 |
| 2.8 | A drone delivers a package in Shenzhen, China. Companies in China and the United States have used delivery drones for several years, which shows that drone logistics is now real [41]. . . . . | 50 |
| 2.9 | A drone assesses the area during a flood from the sky. Drones give fast information in an emergency and help to find victims, assess damage, and identify dangers [57]. . . . .                 | 51 |
| 3.1 | Class diagram of the Skyline platform. . . . .                                                                                                                                                  | 58 |
| 3.2 | Use case diagram of the Drone-Based Crop Disease Detection System. . . .                                                                                                                        | 62 |
| 3.3 | Component diagram of the Skyline platform. . . . .                                                                                                                                              | 66 |
| 3.4 | Sequence diagram of a complete drone mission. . . . .                                                                                                                                           | 69 |

|      |                                                                                                                                                                                                                              |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.5  | Deployment diagram of the Skyline platform. . . . .                                                                                                                                                                          | 72  |
| 3.6  | General layered architecture of the Skyline platform. . . . .                                                                                                                                                                | 75  |
| 4.1  | The full data pipeline of the Skyline Dataset, from the raw sources to the augmented training batches. . . . .                                                                                                               | 84  |
| 4.2  | Validation accuracy of the three models. MobileNetV2 is the lightweight baseline; ResNet50 and EfficientNet-B4 are the two strong candidates. . . .                                                                          | 89  |
| 4.3  | Multi-metric comparison of ResNet50 and EfficientNet-B4 on the 47-class validation set. . . . .                                                                                                                              | 90  |
| 4.4  | Training curves of EfficientNet-B4 during Phase 2 fine-tuning. The plot shows the loss function (cross-entropy) together with the four evaluation metrics (accuracy, AUC, macro-F1 and MCC). All curves converge smoothly. . | 93  |
| 4.5  | Final performance of EfficientNet-B4 against the project targets (red dashed lines). All targets are exceeded. . . . .                                                                                                       | 98  |
| 4.6  | F1-score of each of the 47 classes. Most classes reach an excellent score; only a few rare-data classes fall below the target. . . . .                                                                                       | 99  |
| 4.7  | Skyline landing page the main entry point of the platform. . . . .                                                                                                                                                           | 101 |
| 4.8  | The Smart Agriculture division, the service this project focuses on. . . . .                                                                                                                                                 | 102 |
| 4.9  | The “Request a Drone Mission” form on the user side. . . . .                                                                                                                                                                 | 103 |
| 4.10 | The admin dashboard: the list of all mission requests with their payment and mission status. . . . .                                                                                                                         | 104 |
| 4.11 | The admin confirms the mission and launches an available drone. . . . .                                                                                                                                                      | 105 |
| 4.12 | The user field map: green points are healthy, red points are diseased, drawn over the real field area. . . . .                                                                                                               | 106 |
| 4.13 | The Field Analysis Engine: upload an image and analyse it with the EfficientNetB4 CNN (the model specs match the previous section of this chapter). .                                                                        | 107 |
| 4.14 | The bilingual diagnosis result: top-3 predictions, expert advice and treatment in English and Arabic. . . . .                                                                                                                | 107 |
| 4.15 | The prepared field environment in Unreal Engine 4.27 (with the Landscape-Pro landscape), where the AirSim drone flies. . . . .                                                                                               | 111 |

# List of Tables

|      |                                                                               |    |
|------|-------------------------------------------------------------------------------|----|
| 1.1  | UAV classification by wing configuration (Hassanalian & Abdelkefi, 2017)      | 6  |
| 1.2  | Key drone hardware components (adapted from Colomina & Molina, 2014)          | 7  |
| 1.3  | NIST cloud computing service models (Mell & Grance, 2011)                     | 8  |
| 1.4  | Historical evolution of Artificial Intelligence (Russell & Norvig, 2020)      | 10 |
| 1.5  | Traditional programming vs. Machine Learning (Mitchell, 1997)                 | 11 |
| 1.6  | Comparison of machine learning strategies                                     | 12 |
| 1.7  | Hierarchical feature learning in a deep CNN                                   | 12 |
| 1.8  | Common activation functions in deep learning                                  | 13 |
| 1.9  | Evolution of landmark deep learning architectures                             | 14 |
| 1.10 | Hierarchy of standard computer vision tasks (Szeliski, 2022)                  | 15 |
| 1.11 | Transfer learning strategies and their trade-offs (Pan & Yang, 2010)          | 16 |
| 1.12 | Common data augmentation techniques (Shorten & Khoshgoftaar, 2019)            | 16 |
| 1.13 | IoT sensors and actuators in smart agricultural systems (Liakos et al., 2018) | 18 |
| 1.14 | Comparison of IoT communication protocols (Centenaro et al., 2016)            | 19 |
| 1.15 | Security threats across the IoT architecture layers (Roman et al., 2013)      | 20 |
| 1.16 | Security controls by IoT layer (Roman et al. 2013; Stallings & Brown 2018)    | 21 |
| 1.17 | Key C++ features and their role in the drone-AI system                        | 22 |
| 1.18 | REST principles and their application in the drone-AI backend                 | 25 |
| 1.19 | REST API endpoints in the drone-AI system                                     | 26 |
| 1.20 | Supabase modules used in the drone-AI system                                  | 28 |
| 1.21 | Specifications of Development Machine 1 (Dell Latitude 5490)                  | 29 |
| 1.22 | Specifications of Development Machine 2 (Desktop Workstation)                 | 30 |
| 1.23 | Unreal Engine 4.27 features relevant to the drone simulation pipeline         | 31 |
| 1.24 | Visual Studio Build Tools 2022 components used in the project                 | 33 |
| 1.25 | CNN vs. Vision Transformer for plant disease detection                        | 34 |
| 2.1  | Comparison of classical CNN architectures for plant-disease detection.        | 37 |
| 2.2  | EfficientNet family comparison.                                               | 38 |
| 2.3  | Transfer-learning strategies compared.                                        | 41 |
| 2.4  | Comparative synthesis of the reviewed works. (n.r. = not reported.)           | 43 |

|      |                                                                                              |     |
|------|----------------------------------------------------------------------------------------------|-----|
| 3.1  | Description of the main classes of the system. . . . .                                       | 59  |
| 3.2  | Associations and cardinalities of the class diagram. . . . .                                 | 60  |
| 3.3  | Actors of the system and their responsibilities. . . . .                                     | 63  |
| 3.4  | Main use cases of the platform. . . . .                                                      | 63  |
| 3.5  | Description of the components of the platform. . . . .                                       | 66  |
| 3.6  | Main connections between the components. . . . .                                             | 68  |
| 3.7  | Description of the deployment nodes. . . . .                                                 | 72  |
| 3.8  | Communication links between the deployment nodes. . . . .                                    | 74  |
| 4.1  | Characteristics of the final Skyline Dataset and the work environment. . . .                 | 79  |
| 4.2  | Distribution of the 47 classes over the 11 crops. . . . .                                    | 80  |
| 4.3  | The six data-augmentation transformations applied to the training set only. .                | 83  |
| 4.4  | Full evaluation results of MobileNetV2 on the 47-class validation set. . . .                 | 85  |
| 4.5  | Full evaluation results of ResNet50 on the 47-class validation set. . . . .                  | 87  |
| 4.6  | Shared training configuration of the two main models. . . . .                                | 88  |
| 4.7  | Complete comparison of the three models on all metrics (same validation set).                | 89  |
| 4.8  | Head-to-head comparison of the two final models. Both exceed all project<br>targets. . . . . | 90  |
| 4.9  | Full architecture of the champion model (EfficientNet-B4 + custom head). .                   | 91  |
| 4.10 | Techniques used to prevent overfitting. . . . .                                              | 94  |
| 4.11 | Evaluation metrics used for the champion model. . . . .                                      | 95  |
| 4.12 | The five weak classes of EfficientNet-B4 ( $F1 < 0.85$ ) and their root cause. .             | 100 |

# General Introduction

## Context and Motivation

Many industries have a big problem today. They need to collect information faster, more safely, and at a lower cost. One technology that can help is the drone, also called an Unmanned Aerial Vehicle (UAV). A drone can fly over large areas, take photos, and reach dangerous places. It works very quickly and it does not need a person on the ground.

Drones are already used in many areas. In **construction**, drones take photos of the building site and make 3D maps. In **renewable energy**, drones check solar panels and wind turbines with a special camera that sees heat. In **logistics**, companies like Amazon use drones to bring packages to customers in less than one hour. In **environmental monitoring**, drones help to count animals, watch forests, and check areas after a flood or a fire.

But a drone that only takes photos is not enough. The drone must also understand what it sees. This is why we use artificial intelligence (AI). A special type of AI called a Convolutional Neural Network (CNN) can look at a photo and give an answer very fast. When we put a CNN on a drone, the drone becomes a smart tool that can work alone.

All these industries need this technology, but **agriculture** needs it the most. Many farmers in Algeria and in the world work hard every year to grow food and earn money. One of their biggest problems is plant disease. The Food and Agriculture Organization (FAO) says that plant diseases destroy more than 40% of all crops in the world every year. Today, a farmer must call an expert to come and walk through the field to find the disease. This is slow, expensive, and not possible in every region.

This project builds a first prototype for agriculture. The system uses a drone to fly over a field, take photos, and use a CNN to find plant diseases automatically. The result is sent to the farmer in a simple way, so he can act fast and save his crops.

This is only the first step. The same idea a drone with an AI model can be used in construction, renewable energy, logistics, and environmental monitoring in the future. This is why we built this project.

## Problem Statement

Despite the significant body of research at the intersection of deep learning and UAV technology for agricultural applications, a number of critical gaps remain unaddressed in existing systems.

**First, the domain gap problem.** Most published models are trained on curated laboratory datasets such as PlantVillage, collected under controlled conditions with uniform backgrounds and balanced class distributions. When deployed on real aerial imagery, where lighting varies, backgrounds are complex, and some diseases are far rarer than others, the performance of these models degrades substantially. No existing system has fully addressed the challenge of training a model that is both accurate in the laboratory and robust in the field.

**Second, the absence of actionable guidance.** Existing disease detection systems stop at classification. They produce a label and a confidence score, but provide no information about what the farmer should do next. A notification reading “*Late Blight detected*” is of limited practical value without accompanying guidance on treatment protocols, pesticide selection, and preventive measures. The gap between diagnosis and action is never bridged.

**Third, the language barrier.** All reviewed platforms deliver their output exclusively in English. For Algerian farmers, the majority of whom conduct their daily work in Arabic or Tamazight, an English-only interface is not a minor inconvenience it is a fundamental barrier to adoption. A tool that cannot speak the user’s language cannot serve that user.

**Fourth, the absence of a complete integrated platform.** The literature addresses individual components of the drone-AI pipeline in isolation: the vision model, the UAV control system, the data management layer, or the reporting interface. No single system integrates all of these components into a coherent, end-to-end platform that a farmer can actually use, from submitting a mission request to reading a diagnosis on an interactive map.

**Fifth, the lack of explainability.** The vast majority of published models operate as black boxes. They produce predictions but offer no visual or conceptual explanation of how those predictions were reached. Without explainability, farmers and agronomists have no basis for trusting the system’s output, and no means of detecting errors when they occur.

These five gaps collectively define the central research question of this project:

*How can we design, build, and validate a complete, scalable, and accessible Drone-as-a-Service platform that integrates autonomous UAV operation, AI-based agricultural disease detection, expert advisory services, and bilingual reporting into a single coherent system?*

## Objectives

The goal of this project is to design, build, and test a complete platform called **Skyline** that solves the problems described above. More specifically, the project has the following objectives:

- 1. Build a reliable AI model.** Train an EfficientNet-B4 deep learning model to classify crop diseases from aerial images. The model must be able to recognise 47 disease classes with high accuracy, using transfer learning and proper handling of class imbalance.
- 2. Integrate an expert advisory system.** Go beyond simple disease classification by adding a knowledge base and a bilingual report system that gives the farmer concrete advice and treatment recommendations in both English and Arabic.
- 3. Develop a full web platform.** Create a working web application with two parts: a user space where the farmer can submit a mission request and view the results on an interactive map, and an administration dashboard where the operator can review requests, confirm missions, and manage the drone fleet.
- 4. Simulate the drone pipeline.** Since a physical drone is not yet available, use the AirSim simulator inside Unreal Engine 4.27 to generate realistic aerial images and test the full workflow from image capture to diagnosis delivery.
- 5. Make the system accessible.** Ensure that all diagnosis results are available in both Arabic and English, so that local farmers can understand and use the output of the system without language difficulties.

## Thesis Structure

This thesis is organised into four chapters, each corresponding to a distinct phase of the project.

**Chapter 1 Background and Technical Foundations** establishes the theoretical and technical framework required to understand the design choices made throughout the project. It covers unmanned aerial vehicles and their role in precision agriculture, cloud computing and the Data-as-a-Service model, artificial intelligence and its principal branches, machine learning and its learning paradigms, deep learning and neural network architectures, convolutional neural networks and the mechanisms of transfer learning, the Internet of Things and its communication protocols, security in drone and IoT systems, and all the programming languages, frameworks, and development tools used in the implementation of Skyline.

**Chapter 2 State of the Art** reviews the most significant related work across three research

axes: CNN architectures applied to plant disease classification, drone-based agricultural surveillance systems combining UAVs and AI, and transfer learning strategies for domain adaptation in plant pathology. It also presents the legal and regulatory framework governing civilian drone operations in Algeria. The chapter concludes with a structured synthesis identifying the six research gaps that motivated the design of Skyline and distinguishing this work from the existing literature.

**Chapter 3 System Modeling and General Architecture** presents the formal design of the Skyline platform through three complementary UML views. The *class diagram* captures the full data model of the system as implemented in the Supabase PostgreSQL database. The *use case diagram* defines the responsibilities and interactions of the three system actors: the farmer, the administrator, and the drone. The *sequence diagram* traces the complete message flow across all five phases of a drone mission, from user registration to the delivery of results on the interactive field map. The chapter closes with a description of Skyline’s six-layer general architecture, from the data acquisition layer to the deployment layer.

**Chapter 4 Experimentation and Implementation** documents the complete experimental and engineering work of the project. It begins with the construction and preprocessing of the Skyline Dataset, the handling of severe class imbalance, and the drone-specific augmentation pipeline. It then presents the training and systematic comparison of three CNN architectures MobileNetV2, ResNet50, and EfficientNet-B4 on the same dataset and the same evaluation protocol, leading to the selection of EfficientNet-B4 as the champion model with **98.43% accuracy, F1-Macro of 0.9449, MCC of 0.9838, and AUC of 0.9998**. The chapter then demonstrates the complete Skyline application, presenting the farmer portal, the administration dashboard, the interactive satellite field map, the bilingual diagnosis engine, and the AirSim drone simulation environment that closes the end-to-end operational loop.

The thesis closes with a **General Conclusion** that summarises the scientific and engineering contributions of Skyline, discusses the limitations of the current implementation with full transparency, and proposes a concrete roadmap for future work centred on the transition from the AirSim simulation to real UAV deployment in Algerian agricultural fields.

# Chapter 1

## Background

### 1.1 Introduction

Building an advanced intelligent system based on autonomous platforms is a complex undertaking that requires the seamless integration of multiple technical domains. This chapter provides the fundamental concepts and theoretical background necessary to justify the design choices made throughout this project. Each section explores a critical pillar of the architecture, ranging from physical hardware to advanced software engineering and system security.

### 1.2 Unmanned Aerial Vehicles (UAVs)

#### 1.2.1 Definition

Unmanned Aerial Vehicles (UAVs), also known as drones, are aircraft that can fly without a pilot on board. They are controlled either by a person using a remote controller or by an automatic system that follows a pre-planned route [8].

##### Definition

**UAV [2]:** An Unmanned Aerial Vehicle is a powered air vehicle that does not carry a human operator, is able to fly autonomously or be piloted remotely, can carry a lethal or non-lethal payload, and is expendable or recoverable.

At first, drones were mainly used by the military for surveillance and combat. However, since the early 2010s, the cost of the necessary sensors and batteries dropped a lot, which made drones available for many civilian uses. Today, drones are used in construction, filmmaking, package delivery, emergency response, and especially in agriculture [5].

## 1.2.2 Types of UAVs

### By Wing Shape

**Table 1.1.** UAV classification by wing configuration (Hassanalian & Abdelkefi, 2017)

| Type         | How It Works                                                                             | Use in Agriculture                                             |
|--------------|------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| Fixed-Wing   | Uses aerodynamic lift. Needs a runway or launcher to take off.                           | Mapping very large fields (>500 ha per flight).                |
| Multi-Rotor  | Uses 4 or more spinning rotors. Can take off vertically and hover. Battery: 20–45 min.   | Detailed leaf inspection; image capture for disease detection. |
| Hybrid VTOL  | Combines a fixed wing for efficient long-range flight with rotors for vertical take-off. | Medium-range field surveys.                                    |
| Single-Rotor | One main rotor; higher payload capacity; mechanically complex.                           | Spraying pesticides over large orchards.                       |

### By Level of Autonomy

- **Manual:** A trained pilot controls everything in real time through a radio link.
- **Semi-autonomous:** The drone stabilises itself automatically using GPS, but the pilot still gives direction commands.
- **Fully autonomous:** The drone follows a pre-planned flight path with no human input.

1.2.3 Hardware Architecture

Table 1.2. Key drone hardware components (adapted from Colomina & Molina, 2014)

| Component           | Main Function              | Role in Disease Detection              |
|---------------------|----------------------------|----------------------------------------|
| Airframe + Motors   | Structure and thrust.      | Determines payload and field coverage. |
| Flight Controller   | Real-time stabilisation.   | Enables autonomous survey missions.    |
| IMU                 | 3D orientation estimation. | Ensures stable, sharp image capture.   |
| GNSS / RTK GPS      | Absolute position.         | GPS-tags images for disease maps.      |
| LiPo Battery        | Energy storage.            | Limits field coverage per flight.      |
| RGB Camera + Gimbal | Image acquisition.         | Primary data source for the CNN.       |
| Wi-Fi / 4G LTE      | Data offload.              | Uploads images to cloud AI service.    |

1.2.4 Drones in Different Fields

Drones offer the following precision-agriculture advantages:

- **Speed of coverage:** A drone can survey hundreds of hectares per hour.
- **Early detection:** Aerial images can reveal problems before they are visible from the ground.
- **Targeted treatment:** Detailed maps allow workers to act only in affected zones, reducing waste by up to 30 %.
- **Inaccessible areas:** Drones can reach steep slopes, flooded areas, and tall structures.
- **Georeferenced data:** GPS-tagged images integrate directly with GIS platforms.

## 1.3 Data as a Service (DaaS)

### 1.3.1 Cloud Computing

**Table 1.3.** NIST cloud computing service models (Mell & Grance, 2011)

| Model | What It Provides                                            | Examples            |
|-------|-------------------------------------------------------------|---------------------|
| IaaS  | Virtual machines, storage, and networking.                  | AWS EC2, GCE        |
| PaaS  | Ready platform for building and deploying applications.     | Google App Engine   |
| SaaS  | Ready-to-use software over the internet.                    | Gmail, MS365        |
| DaaS  | Data storage, retrieval, and management as a cloud service. | Amazon S3, Firebase |

### 1.3.2 Definition of DaaS

#### Definition

**Data as a Service (DaaS)** [15]: A cloud paradigm in which data storage, retrieval, processing, and governance are provided as on-demand, API-accessible services over a network, independently of the physical infrastructure and the location of the consuming application.

### 1.3.3 The Five Principles of DaaS

1. **Abstraction:** Users interact with data through a simple API.
2. **Elasticity:** Storage capacity scales automatically.
3. **Ubiquitous Access:** Any internet-connected device can access the data.
4. **Multi-Tenancy:** Many users share the same infrastructure; data is kept separated.
5. **Pay-per-Use:** Users pay only for storage and bandwidth used.

### 1.3.4 DaaS in the Drone-AI Pipeline

**Stage 1 : Image Ingestion.** After each flight, images are uploaded to a cloud object store with GPS coordinates, altitude, capture time, and field ID.

**Stage 2 : AI Inference.** The cloud AI system reads images, processes them, and writes back results disease name, confidence score, explanation heatmap.

**Stage 3 : Result Delivery.** Farmers access diagnosis reports from any device through a secure REST API.

#### Important Note

The DaaS model gives the system three essential qualities: **Scalability**, **Accessibility**, and **Maintainability**.

## 1.4 Artificial Intelligence

### 1.4.1 Definition

#### Definition

**Artificial Intelligence (AI)** [20]: The branch of computer science dedicated to building systems capable of performing tasks that would normally require human intelligence including perception, reasoning, learning, language understanding, and autonomous decision-making.

1.4.2 History of AI

Table 1.4. Historical evolution of Artificial Intelligence (Russell & Norvig, 2020)

| Period      | Dominant Paradigm                        | Key Events                            |
|-------------|------------------------------------------|---------------------------------------|
| 1950s–1960s | Symbolic AI; logic and search.           | Turing Test (1950); Dartmouth (1956). |
| 1970s–1980s | Expert systems encoding human knowledge. | MYCIN (1974); XCON (1980).            |
| 1990s       | Statistical ML; Support Vector Machines. | SVM by Vapnik (1995).                 |
| 2000s       | Large-scale ML using internet data.      | ImageNet launch (2009).               |
| 2010s       | Deep learning revolution; GPU training.  | AlexNet (2012); ResNet (2015).        |
| 2020s       | Foundation models; multimodal AI; LLMs.  | GPT-4 (2023); Gemini (2023).          |

1.4.3 Main Branches of AI

- **Machine Learning (ML):** Systems that improve automatically by learning from data.
- **Deep Learning (DL):** A sub-area of ML using very large neural networks.
- **Computer Vision:** Enabling machines to understand images and video.
- **Natural Language Processing (NLP):** Understanding and generating human language.
- **Expert Systems:** Programs following human-written rules to give expert advice.
- **Robotics and Autonomous Systems:** Physical machines that perceive, decide, and act such as autonomous drones.

1.5 Machine Learning

### 1.5.1 Definition and the Key Idea

#### Definition

**Machine Learning (ML)** [16]: A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .

**Table 1.5.** Traditional programming vs. Machine Learning (Mitchell, 1997)

| Aspect               | Traditional Programming            | Machine Learning                    |
|----------------------|------------------------------------|-------------------------------------|
| What you provide     | Data + manually written rules.     | Data + labelled examples.           |
| What is produced     | Results based on the rules.        | A model that learned its own rules. |
| Who gives knowledge  | A human expert.                    | The algorithm extracts patterns.    |
| Adapting to new data | Rules must be updated manually.    | Model is retrained on new data.     |
| Scalability          | Fails quickly as complexity grows. | Scales well with more data.         |

### 1.5.2 Types of Learning

#### Supervised Learning

Trained on a labelled dataset  $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$  where  $\mathbf{x}_i$  is an input (e.g., a leaf image) and  $y_i$  is the correct label. The goal is to minimise the loss  $\mathcal{L}(f(\mathbf{x}_i), y_i)$  [10].

#### Unsupervised Learning

No labels are provided. The algorithm discovers hidden structure (clustering, autoencoders).

#### Semi-Supervised Learning

A small amount of labelled data combined with a large amount of unlabelled data very useful in agriculture where expert labels are expensive.

Reinforcement Learning

An agent learns by interacting with an environment, receiving reward or penalty signals. Applicable to autonomous drone navigation and mission planning.

Table 1.6. Comparison of machine learning strategies

| Strategy        | Supervision    | Main Methods         | Application                   |
|-----------------|----------------|----------------------|-------------------------------|
| Supervised      | Full label set | SVM, RF, CNN         | Image classification          |
| Unsupervised    | No labels      | K-Means, Autoencoder | Clustering, anomaly detection |
| Semi-supervised | Partial labels | Label Propagation    | Few-label classification      |
| Reinforcement   | Reward signal  | DQN, PPO, A3C        | Drone navigation              |

1.6 Deep Learning

1.6.1 What Makes Deep Learning Different

Definition

**Deep Learning (DL)** [6]: A particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, directly from raw data without manual feature engineering.

Table 1.7. Hierarchical feature learning in a deep CNN

| Depth         | Feature Level | What Is Learned                                  |
|---------------|---------------|--------------------------------------------------|
| Layers 1–2    | Low-level     | Edges, colour gradients, corners.                |
| Layers 3–5    | Mid-level     | Textures, spots, lesion boundaries.              |
| Layers 6– $N$ | High-level    | Disease-specific patterns, whole-leaf semantics. |
| Output        | Class score   | Probability over all disease categories.         |

1.6.2 The Artificial Neuron

Given inputs  $\mathbf{x} = (x_1, \dots, x_n)$ , a neuron computes:

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right) = f(\mathbf{w}^\top \mathbf{x} + b) \tag{1.1}$$

1.6.3 Activation Functions

Table 1.8. Common activation functions in deep learning

| Function   | Formula                          | Where Used                    |
|------------|----------------------------------|-------------------------------|
| Sigmoid    | $\sigma(x) = \frac{1}{1+e^{-x}}$ | Binary classification output. |
| Tanh       | $\tanh(x) \in (-1, 1)$           | Recurrent networks.           |
| ReLU       | $\max(0, x)$                     | Standard hidden layers.       |
| Leaky ReLU | $\max(\alpha x, x)$              | Avoids “dying ReLU” problem.  |
| Swish      | $x \cdot \sigma(x)$              | EfficientNet, modern CNNs.    |
| Softmax    | $e^{z_j} / \sum_k e^{z_k}$       | Multi-class output layer.     |

## 1.6.4 Major Deep Learning Architectures

**Table 1.9.** Evolution of landmark deep learning architectures

| Architecture | Key Innovation                                        | Year | Top-1 |
|--------------|-------------------------------------------------------|------|-------|
| LeNet-5      | First practical CNN; handwritten digit recognition.   | 1998 |       |
| AlexNet      | Deep CNN; ReLU; GPU training; won ILSVRC 2012.        | 2012 | 57.1% |
| VGGNet       | Very deep nets using only $3 \times 3$ convolutions.  | 2014 | 70.5% |
| GoogLeNet    | Inception modules; parallel multi-scale convolutions. | 2014 | 74.8% |
| ResNet       | Residual (skip) connections; up to 152 layers.        | 2015 | 77.1% |
| DenseNet     | Every layer connected to all subsequent layers.       | 2017 | 79.2% |
| MobileNetV2  | Depthwise separable convolutions; mobile deployment.  | 2018 | 72.0% |
| EfficientNet | Compound scaling of width, depth, and resolution.     | 2019 | 82.9% |
| ViT          | Pure self-attention; patch-based image tokenisation.  | 2020 | 88.6% |

## 1.7 Convolutional Neural Networks and Computer Vision

### 1.7.1 Computer Vision

#### Definition

**Computer Vision (CV)** [24]: The construction of explicit, meaningful descriptions of physical objects from images, aiming to duplicate those tasks of the human visual system that can be specified with sufficient precision to be automated.

**Table 1.10.** Hierarchy of standard computer vision tasks (Szeliski, 2022)

| Task                  | Description                           | Agricultural Example                    |
|-----------------------|---------------------------------------|-----------------------------------------|
| Image Classification  | Assign one label to the whole image.  | “This leaf has Early Blight.”           |
| Object Detection      | Locate objects with bounding boxes.   | Detect and locate diseased lesions.     |
| Semantic Segmentation | Give a class to every pixel.          | Mark exactly which pixels are diseased. |
| Instance Segmentation | Separate individual object instances. | Distinguish each lesion from others.    |
| Image Generation      | Synthesise new realistic images.      | Generate rare disease training samples. |

## 1.7.2 Convolutional Neural Networks (CNN)

### Definition

**CNN [11]:** A neural network that uses convolution in at least one layer, employing learnable filters to detect local spatial patterns such as edges and textures, exploiting translational invariance and parameter sharing to process images efficiently.

## 1.7.3 Core CNN Operations

### Convolutional Layer

$$S(i, j) = (\mathbf{I} * \mathbf{K})(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I(i + m, j + n) \cdot K(m, n) \quad (1.2)$$

Key hyperparameters: kernel size  $k$  (usually  $3 \times 3$ ), stride  $s$ , padding, and number of filters  $F$ .

### Pooling Layer

- **Max Pooling:** Selects the maximum value in each window.
- **Global Average Pooling (GAP):** Computes the spatial mean of each feature map.

### 1.7.4 Transfer Learning

#### Definition

**Transfer Learning** [18]: A machine learning method where a model pre-trained on a large, general-purpose dataset (source domain) is reused and fine-tuned for a different but related task on a smaller, domain-specific dataset (target domain).

**Table 1.11.** Transfer learning strategies and their trade-offs (Pan & Yang, 2010)

| Strategy            | Backbone Weights                  | New Head              | Best Situation                |
|---------------------|-----------------------------------|-----------------------|-------------------------------|
| Feature Extraction  | All layers frozen.                | Trained from scratch. | Small dataset (<5k images).   |
| Partial Fine-Tuning | Last $N$ layers unfrozen, low LR. | Trained jointly.      | Medium dataset; domain shift. |
| Full Fine-Tuning    | All layers unfrozen.              | Trained end-to-end.   | Very large target dataset.    |

### 1.7.5 Data Augmentation

**Table 1.12.** Common data augmentation techniques (Shorten & Khoshgoftaar, 2019)

| Technique           | Description                              | What It Simulates                       |
|---------------------|------------------------------------------|-----------------------------------------|
| Random Flip (H+V)   | Mirror image horizontally or vertically. | Different camera viewing angles.        |
| Random Rotation     | Rotate by $\pm\theta$ degrees.           | Drone tilt during flight.               |
| Random Zoom         | Scale image by $\pm z\%$ .               | Varying flight altitude.                |
| Brightness/Contrast | Randomly adjust brightness.              | Different sunlight conditions.          |
| Gaussian Noise      | Add pixel-level random noise.            | Camera sensor noise.                    |
| CutMix / Mixup      | Blend patches or pixels of two images.   | Advanced generalisation regularisation. |

### 1.7.6 Explainability: Grad-CAM

#### Definition

**Grad-CAM** [21]: A visual explanation technique that uses the gradients of a target class score flowing into the final convolutional layer to produce a heatmap highlighting the image regions most responsible for the prediction.

$$\mathbf{L}_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c \mathbf{A}^k\right), \quad \alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \quad (1.3)$$

### 1.7.7 Handling Class Imbalance

Class weights inversely proportional to class frequency:

$$w_c = \frac{N_{\text{total}}}{C \cdot N_c} \quad (1.4)$$

## 1.8 Internet of Things (IoT)

### 1.8.1 Definition

#### Definition

**Internet of Things (IoT)** [1]: A pervasive and ubiquitous network paradigm where physical objects embedded with sensors, actuators, and connectivity capabilities exchange data among themselves and with humans, with minimal or no direct human intervention.

### 1.8.2 The Four-Layer IoT Architecture

**Layer 1: Perception Layer.** Physical sensors and actuators: temperature probes, soil moisture sensors, GPS receivers, drone cameras.

**Layer 2: Network Layer.** Communication protocols that move data from edge devices to gateways and cloud services.

**Layer 3: Processing Layer.** Edge gateways and cloud nodes that aggregate, filter, and pre-process data.

**Layer 4: Application Layer.** End-user services: web dashboards, mobile applications, AI inference APIs.

### 1.8.3 IoT Sensors in Smart Farming

**Table 1.13.** IoT sensors and actuators in smart agricultural systems (Liakos et al., 2018)

| Device / Sensor        | Parameter Measured                  | Agricultural Application           |
|------------------------|-------------------------------------|------------------------------------|
| Soil moisture probe    | Volumetric water content (%)        | Automated irrigation scheduling.   |
| Temp. + Humidity       | Air & soil temp. (°C), RH (%)       | Disease outbreak risk modelling.   |
| PAR / LUX sensor       | Photosynthetically active radiation | Crop growth optimisation.          |
| CO <sub>2</sub> sensor | CO <sub>2</sub> concentration (ppm) | Greenhouse environment control.    |
| pH probe               | Soil acidity                        | Fertilisation guidance.            |
| Rainfall gauge         | Precipitation (mm)                  | Fungal disease prediction.         |
| Wind anemometer        | Speed (m/s) and direction           | Pesticide spray management.        |
| Drone RGB camera       | High-resolution aerial images       | AI-powered disease classification. |

### 1.8.4 Communication Protocols

#### MQTT

##### Definition

**MQTT** [17]: A lightweight, publish-subscribe messaging protocol designed for constrained devices and unreliable, low-bandwidth networks. Standardised as ISO/IEC 20922 and OASIS MQTT v5.

Key features: 2-byte fixed header; three QoS levels (0, 1, 2); runs on TCP/IP port 1883 or 8883 (TLS).

LoRa and LoRaWAN

Definition

LoRa [13]: A physical-layer radio modulation technique based on Chirp Spread Spectrum (CSS), operating in unlicensed ISM bands. LoRaWAN is the MAC protocol built on top of LoRa radio.

Key characteristics: range 2–15 km; data rate 0.25–50 kbps; ultra-low power; star-of-stars topology.

Protocol Comparison

Table 1.14. Comparison of IoT communication protocols (Centenaro et al., 2016)

| Protocol         | Range      | Data Rate | Power    | Best Use Case              |
|------------------|------------|-----------|----------|----------------------------|
| MQTT over Wi-Fi  | ~100 m     | High      | Medium   | Local drone telemetry.     |
| MQTT over 4G LTE | Nationwide | High      | High     | Cloud image upload.        |
| LoRaWAN          | 2–15 km    | Very low  | Very low | Field sensor networks.     |
| Zigbee           | 10–100 m   | Medium    | Low      | Greenhouse mesh networks.  |
| BLE 5.0          | ~40 m      | Medium    | Very low | Mobile device pairing.     |
| HTTP/REST        | Unlimited  | High      | High     | Cloud API result delivery. |
| WebSocket        | Unlimited  | High      | High     | Real-time dashboard.       |

1.9 Security in IoT and Drone Systems

1.9.1 Overview

Connecting UAVs, cloud services, and IoT sensors creates an extended attack surface spanning physical hardware, wireless channels, cloud infrastructure, and user-facing applications [19].

1.9.2 Threat Taxonomy

Table 1.15. Security threats across the IoT architecture layers (Roman et al., 2013)

| Layer            | Primary Threats                                          | Concrete Attack Examples                |
|------------------|----------------------------------------------------------|-----------------------------------------|
| Perception/Edge  | Physical tampering, GPS spoofing, firmware attacks.      | False GPS signals redirect drone path.  |
| Network/Comm.    | Eavesdropping, MitM, replay attacks, DoS/DDoS.           | Intercepting unencrypted MQTT messages. |
| Cloud/Processing | Data breaches, unauthorised API access, model poisoning. | Stealing AI model weights.              |
| Application/User | Phishing, weak authentication, session hijacking.        | Credential theft; unauthorised access.  |

1.9.3 The CIA Triad

Definition

The CIA Triad [23]:

- **Confidentiality:** Data accessible only to authorised users.
- **Integrity:** Data cannot be modified without detection.
- **Availability:** The service remains operational even under partial failure.

1.9.4 Security Controls

Authentication and Authorisation

OAuth 2.0 / JWT for stateless token-based authentication; Mutual TLS (mTLS) for drone-to-cloud channels; Role-Based Access Control (RBAC) for user-level permissions.

Data Encryption

TLS 1.3 for all data in transit; AES-256 for data at rest; End-to-End Encryption (E2EE) protecting data even from the infrastructure provider.

Drone-Specific Security

Multi-constellation GNSS anti-spoofing (GPS + GLONASS + Galileo); cryptographically signed firmware; software geofencing; onboard SD card encryption.

AI Model Security

Adversarial training against adversarial examples; rate limiting to prevent model extraction attacks; cryptographic hashing of model weights for integrity verification [7].

Table 1.16. Security controls by IoT layer (Roman et al. 2013; Stallings & Brown 2018)

| Layer      | Primary Controls                                  | Standards                          |
|------------|---------------------------------------------------|------------------------------------|
| Edge/Drone | GPS anti-spoofing, firmware signing, geofencing.  | GNSS multi-constellation; AES-256. |
| Network    | TLS 1.3, mTLS, VPN, MQTT 8883, firewall.          | TLS 1.3; WireGuard; OWASP.         |
| Cloud      | OAuth 2.0/JWT, RBAC, AES-256, IDS.                | ISO 27001; SOC 2; NIST SP 800-53.  |
| AI Model   | Adversarial training, model hashing, API control. | NIST AI RMF.                       |

1.10 Development Tools and Technologies

This section presents the complete set of programming languages, engines, frameworks, databases, and hardware components used in the development of the system.

# 1.11 Programming Languages

## 1.11.1 C++

**Definition**

**C++** (ISO/IEC 14882): A general-purpose, compiled, statically-typed programming language extending C with object-oriented, generic, and functional programming capabilities. It provides fine-grained control over hardware resources and memory management, making it the preferred language for performance-critical and embedded systems.

C++ is the standard language for flight controller firmware (ArduPilot, PX4) and real-time robotics. In this project, C++ is used for:

- Low-level drone control logic and sensor data processing.
- Real-time image preprocessing on embedded hardware before cloud upload.
- Integration with the Unreal Engine / AirSim simulation environment.

**Table 1.17.** Key C++ features and their role in the drone-AI system

| Feature                  | Role in the Project                                         |
|--------------------------|-------------------------------------------------------------|
| Memory management        | Precise control of image buffers on embedded hardware.      |
| Multi-threading          | Parallel sensor reading, flight control, and communication. |
| Template metaprogramming | Reusable, type-safe data structures for sensor payloads.    |
| OpenCV integration       | Efficient image capture, resizing, and format conversion.   |
| AirSim / UE4 API         | Native language for Unreal Engine plugin development.       |

### 1.11.2 Python

#### Definition

**Python** (Python Software Foundation): A high-level, interpreted, dynamically-typed programming language renowned for its readability and the breadth of its scientific ecosystem. Python is the dominant language for machine learning research and data pipeline engineering.

Python serves as the primary language for the AI inference engine and cloud backend:

- **Deep learning:** Model training and evaluation using TensorFlow/Keras and PyTorch.
- **Data pipeline:** Image loading, augmentation, and batch preparation (NumPy, OpenCV, Albumentations).
- **REST API server:** Exposing the AI inference engine as a web service (FastAPI / Flask).
- **Database interaction:** Querying Supabase using the supabase-py client.
- **Visualisation:** Generating Grad-CAM heatmaps and metrics (Matplotlib, Seaborn).

```
1 import tensorflow as tf
2 import numpy as np
3 from PIL import Image
4
5 model = tf.keras.models.load_model("plant_disease_model.h5")
6 img = Image.open("leaf_sample.jpg").resize((224, 224))
7 x = np.expand_dims(np.array(img) / 255.0, axis=0)
8
9 predictions = model.predict(x)
10 class_idx = np.argmax(predictions)
11 confidence = predictions[0][class_idx]
12 print(f"Predicted class: {class_idx}, Confidence: {confidence:.2%}")
```

**Listing 1.1.** Inference example in Python

### 1.11.3 HTML & CSS

#### Definition

**HTML** (HyperText Markup Language, W3C): The standard declarative language for structuring content on the World Wide Web using a hierarchy of semantic elements (tags).  
**CSS** (Cascading Style Sheets, W3C): A stylesheet language that controls the visual presentation of HTML documents layout, colours, typography, and responsive behaviour across screen sizes.

In this project, HTML and CSS form the presentation layer of the farmer dashboard:

- A responsive grid displays field maps, drone flight status, and diagnosis cards.
- Diagnosis results (disease name, confidence, Grad-CAM heatmap) are rendered in styled HTML pages.
- Semantic HTML ensures screen-reader accessibility.

```
1 <article class="diagnosis-card">
2   <header class="card__header">
3     <h2>Diagnosis Result</h2>
4     <span class="card__confidence">Confidence: 94.3%</span>
5   </header>
6   
8   <p class="card__disease">Early Blight (Alternaria solani)</p>
9 </article>
```

**Listing 1.2.** HTML structure of the diagnosis result card

## 1.12 APIs and Web Services

### 1.12.1 What is an API?

#### Definition

**API** (Application Programming Interface): A defined contract that specifies how software components communicate with each other. An API exposes a set of functions, endpoints, or protocols through which one application can request services from another, without exposing internal implementation details.

APIs are the connective tissue of the entire system architecture, allowing the drone ground

station, the AI inference server, the database, and the farmer dashboard to communicate as loosely-coupled, independently deployable services.

1.12.2 REST API

**Definition**  
**REST** (Representational State Transfer, Fielding, 2000): An architectural style for distributed hypermedia systems. A REST API is a web service that adheres to six constraints: statelessness, client-server separation, cacheability, layered system, uniform interface, and code-on-demand. Communication uses standard HTTP methods over URLs.

**Table 1.18.** REST principles and their application in the drone-AI backend

| Principle         | Application in This Project                                                                                    |
|-------------------|----------------------------------------------------------------------------------------------------------------|
| Statelessness     | Each HTTP request contains all authentication information (JWT token). No session state is stored server-side. |
| Uniform Interface | Resources are addressed by stable URLs: /api/v1/diagnoses/{id}.                                                |
| Client-Server     | Drone ground station (client) and inference server are independently deployable.                               |
| Cacheability      | GET responses for completed diagnoses include Cache-Control headers.                                           |
| Layered System    | A load balancer sits transparently between the internet and the inference server.                              |

## HTTP Methods and Endpoints

**Table 1.19.** REST API endpoints in the drone-AI system

| Method | Endpoint               | Action                                       |
|--------|------------------------|----------------------------------------------|
| POST   | /api/v1/images/upload  | Upload a drone image for analysis.           |
| GET    | /api/v1/diagnoses/{id} | Retrieve a specific diagnosis report.        |
| GET    | /api/v1/flights        | List all flights for the authenticated user. |
| DELETE | /api/v1/images/{id}    | Delete an image from cloud storage.          |
| PATCH  | /api/v1/diagnoses/{id} | Update agronomist notes on a diagnosis.      |

```

1 from fastapi import FastAPI, HTTPException
2
3 app = FastAPI()
4
5 @app.get("/api/v1/diagnoses/{diagnosis_id}")
6 async def get_diagnosis(diagnosis_id: str):
7     result = await db.fetch_one(
8         "SELECT * FROM diagnoses WHERE id = :id",
9         values={"id": diagnosis_id}
10    )
11    if not result:
12        raise HTTPException(status_code=404,
13                             detail="Diagnosis not found")
14    return result

```

**Listing 1.3.** FastAPI REST endpoint for diagnosis retrieval

## 1.13 Database Technologies

### 1.13.1 SQL (Structured Query Language)

#### Definition

**SQL** (ISO/IEC 9075): The standard declarative language for managing and querying relational databases. SQL statements are grouped into DDL (CREATE, ALTER, DROP), DML (SELECT, INSERT, UPDATE, DELETE), and DCL (GRANT, REVOKE).

All structured data in this project flight records, diagnosis results, user accounts, field meta-data is persisted in a PostgreSQL relational database and queried using SQL.

```

1  -- Fields managed by each farmer
2  CREATE TABLE fields (
3      id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
4      farmer_id   UUID REFERENCES users(id),
5      name        TEXT NOT NULL,
6      created_at  TIMESTAMPTZ DEFAULT NOW()
7  );
8
9  -- One flight per drone mission
10 CREATE TABLE flights (
11     id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
12     field_id     UUID REFERENCES fields(id),
13     drone_id     TEXT,
14     started_at   TIMESTAMPTZ,
15     ended_at     TIMESTAMPTZ,
16     status       TEXT CHECK (status IN
17                          ('planned', 'active', 'completed', 'failed'))
18 );
19
20 -- AI diagnosis per captured image
21 CREATE TABLE diagnoses (
22     id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
23     flight_id    UUID REFERENCES flights(id),
24     image_url    TEXT NOT NULL,
25     disease_name TEXT,
26     confidence   NUMERIC(5,4),
27     heatmap_url  TEXT,
28     created_at   TIMESTAMPTZ DEFAULT NOW()
29 );

```

**Listing 1.4.** Core database schema for the drone-AI system

1.13.2 Supabase (Backend-as-a-Service)

**Definition**

**Supabase:** An open-source Backend-as-a-Service (BaaS) platform built on top of PostgreSQL. Supabase provides a managed PostgreSQL database, auto-generated REST and GraphQL APIs, real-time change subscriptions via WebSockets, file object storage, and built-in authentication all accessible through a unified dashboard and client SDK.

Table 1.20. Supabase modules used in the drone-AI system

| Module         | Technology                 | Use in the Project                            |
|----------------|----------------------------|-----------------------------------------------|
| Database       | PostgreSQL 15              | Stores flights, diagnoses, and user data.     |
| Storage        | S3-compatible object store | Hosts drone images and Grad-CAM heatmaps.     |
| Auth           | JWT + OAuth 2.0            | Farmer and agronomist authentication.         |
| Realtime       | WebSocket (Broadcast)      | Live flight status updates on the dashboard.  |
| Edge Functions | Deno (TypeScript)          | Serverless inference trigger on image upload. |

```
1 from supabase import create_client
2
3 supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
4
5 def save_diagnosis(flight_id, image_url,
6                   disease, confidence, heatmap_url):
7     data = {
8         "flight_id":    flight_id,
9         "image_url":    image_url,
10        "disease_name": disease,
11        "confidence":   round(confidence, 4),
12        "heatmap_url":  heatmap_url
13    }
14    return supabase.table("diagnoses").insert(data).execute()
```

Listing 1.5. Supabase Python client: inserting a diagnosis record

## 1.14 Hardware: Development Computers

The system was developed and tested on two physical machines. Their specifications directly influenced software choices regarding RAM constraints, GPU availability, and operating system compatibility.

### 1.14.1 What is RAM?

**Definition**

**RAM** (Random Access Memory): Volatile, high-speed memory used by the CPU to store actively executing programs and data. Larger RAM allows more processes and larger datasets to reside in memory simultaneously, directly impacting development speed and the batch sizes that can be processed locally during neural network training.

### 1.14.2 Machine 1: Dell Latitude 5490

**Table 1.21.** Specifications of Development Machine 1 (Dell Latitude 5490)

| Component   | Specification                                    |
|-------------|--------------------------------------------------|
| Device Name | khouloud-mina                                    |
| Model       | Dell Latitude 5490                               |
| Processor   | Intel Core i5-8350U @ 1.70 GHz (boost: 1.90 GHz) |
| RAM         | 8.00 GB DDR4 @ 2400 MT/s                         |
| Graphics    | Intel UHD Graphics 620 (128 MB VRAM)             |
| Storage     | 238 GB SSD (207 GB used)                         |

With 8 GB of RAM this machine was used for: running the Supabase local development stack (Docker containers); Python backend development with FastAPI and lightweight model inference; front-end dashboard development in HTML/CSS; and Git version control.

1.14.3 Machine 2: Desktop Workstation

Table 1.22. Specifications of Development Machine 2 (Desktop Workstation)

| Component       | Specification                                              |
|-----------------|------------------------------------------------------------|
| Device Name     | DESKTOP-RD66U4E                                            |
| Processor       | 11th Gen Intel Core i5-1135G7 @ 2.40 GHz (boost: 2.42 GHz) |
| RAM             | 8.00 GB (7.70 GB usable)                                   |
| OS              | Windows 10 Professional (64-bit, x64)                      |
| Windows Version | 21H2 (Build 19044.1415)                                    |
| Installed       | 24/01/2026                                                 |

This workstation was primarily used for: running Unreal Engine 4.27 simulations with the AirSim plugin; C++ plugin development; REST API testing using Postman and curl; and local Supabase migrations.

The 11th Gen Intel Core i5-1135G7 includes Intel Iris Xe Graphics, which provides limited hardware acceleration for Unreal Engine viewport rendering — a practical improvement over the older UHD 620 for 3D simulation work.

The limited 128 MB integrated GPU (Intel UHD 620) is insufficient for GPU-accelerated deep learning training; therefore, model training was offloaded to cloud infrastructure (Google Colab / Kaggle GPU kernels).

1.15 Simulation Environment: Unreal Engine and AirSim

1.15.1 Unreal Engine 4.27 (UE4)

Definition

Unreal Engine 4.27 (Epic Games, 2021): A real-time 3D creation platform written in C++. UE4.27 introduces *Path Tracer* (a progressive, physically-based renderer for high-fidelity offline rendering) and *Ray Tracing* support, enabling photorealistic simulation environments that closely approximate real-world lighting and surface conditions.

UE4.27 provides the synthetic environment for drone flight simulation before physical deployment. Its photorealistic rendering is essential for generating training images that transfer well to real-world inference (sim-to-real transfer).

**Table 1.23.** Unreal Engine 4.27 features relevant to the drone simulation pipeline

| UE4.27 Feature      | Role in the Project                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------|
| Path Tracer         | High-fidelity 3D plant and terrain assets with physically accurate light transport.                       |
| Ray Tracing         | Dynamic global illumination simulates different times of day and cloud cover for training data diversity. |
| Chaos Physics       | Realistic wind effects on crop canopy visible in drone camera feed.                                       |
| Blueprint / C++ API | Controls AirSim drone movement and camera trigger from simulation scripts.                                |

1.15.2 AirSim (Microsoft)

Definition

**AirSim:** An open-source, cross-platform simulation plugin for Unreal Engine, developed by Microsoft Research. AirSim models the physics of multirotor drones with high fidelity, including aerodynamics, motor dynamics, and sensor noise models. It exposes a Python and C++ API for programmatic flight control and sensor data access.

AirSim is integrated directly into UE4 to simulate the full data acquisition pipeline: a virtual quadrotor follows an autonomous waypoint mission; the onboard RGB camera captures images at configurable altitude and overlap; simulated disease textures on plant assets generate ground-truth labelled training data.

```
1 import airsims, time, os
2
3 client = airsims.MultirotorClient()
4 client.confirmConnection()
5 client.enableApiControl(True)
6 client.armDisarm(True)
7
8 client.takeoffAsync().join()
```

```
9 client.moveToZAsync(-20, 3).join()    # 20 m altitude
10
11 waypoints = [(0,0,-20),(50,0,-20),(50,50,-20),(0,50,-20)]
12 for wp in waypoints:
13     client.moveToPositionAsync(*wp, velocity=5).join()
14     response = client.simGetImages([
15         airsims.ImageRequest("0", airsims.ImageType.Scene,
16                               False, False)
17     ])[0]
18     fname = f"frame_{int(time.time())}.png"
19     airsims.write_file(os.path.join("output", fname),
20                        response.image_data_uint8)
21
22 client.landAsync().join()
```

**Listing 1.6.** AirSim Python API: autonomous image capture mission

### 1.15.3 Visual Studio Build Tools 2022

#### Definition

**Visual Studio Build Tools 2022** (Microsoft, 2022): A standalone, redistributable package that installs the MSBuild engine, the MSVC compiler toolchain (cl.exe), the Windows SDK, and the C++ standard library headers required to compile C++ projects on Windows without installing the full Visual Studio IDE.

Visual Studio Build Tools 2022 is an essential dependency on the Windows development workstation (Machine 2). Both Unreal Engine 4.27 and the AirSim plugin require a compatible MSVC compiler to build their C++ source modules.

**Table 1.24.** Visual Studio Build Tools 2022 components used in the project

| Component                   | Role in the Project                                                            |
|-----------------------------|--------------------------------------------------------------------------------|
| MSVC v143 Compiler (cl.exe) | Compiles all C++ source files for the AirSim plugin and Unreal Engine modules. |
| MSBuild Engine              | Processes vcxproj project files and drives the build pipeline.                 |
| Windows 10/11 SDK           | Provides Windows system headers and libraries required by Unreal Engine.       |
| C++ CMake Tools             | Enables CMake-based builds for third-party dependencies (e.g., AirSim RpcLib). |
| C++ ATL / STL               | Standard template library and active template library used by UE4 internals.   |
| IntelliSense Headers        | Provides code-completion metadata consumed by VS Code and Rider IDEs.          |

Visual Studio Build Tools 2022 targets the **v143** toolset, which is the minimum compiler version required by Unreal Engine 4.27 on Windows. It was installed on Machine 2 (DESKTOP-RD66U4E, Windows 10 Pro 21H2) and invoked automatically by the Unreal Build Tool (UBT) during plugin compilation.

1.15.4 Vision Transformer (ViT) vs. CNN

Table 1.25. CNN vs. Vision Transformer for plant disease detection

| Criterion             | CNN (EfficientNet)                                   | ViT                                           |
|-----------------------|------------------------------------------------------|-----------------------------------------------|
| Data requirement      | Moderate transfer learning effective from 5k images. | Large requires >100k images for best results. |
| Inference speed       | Fast on CPU / embedded hardware.                     | Slower; requires GPU or TPU.                  |
| Interpretability      | Grad-CAM provides spatial explanations.              | Attention maps provide similar explanations.  |
| Accuracy (large data) | 82–88% Top-1.                                        | 88–91% Top-1 with sufficient data.            |

EfficientNet (CNN-based) was chosen for the production model due to the dataset size constraint, while ViT remains a future upgrade path.

# Chapter 2

## State of the Art

### 2.1 Introduction

Unmanned Aerial Vehicles (UAVs), also known as drones, have changed a lot over the last twenty years. They were first built for the army, but today they have become useful tools for many industries. Drones can take very clear photos from the air, reach dangerous or far places, and work at a much lower cost than normal planes. Because of these advantages, drones are now one of the most important technologies of our time.

Today, drones are used in five main areas.

In **agriculture**, drones help farmers in many ways. They monitor plant health, take special images, spray the crops, and watch the animals. This lets farmers manage their land carefully, and it also reduces the cost and the use of chemicals [42, 43]. The Food and Agriculture Organization (FAO) reports that plant pests and diseases destroy a large part of the world's crops every year [26]. Deep learning systems can detect these diseases early, so they offer a good answer to this problem.

In **construction**, drones are used in every step of a project. They map the land, build 3D models, follow the work, check safety, and prepare reports. When drone data is combined with Building Information Modelling (BIM), the team gets a live digital copy of the site, which reduces mistakes and speeds up the work [45, 46].

In **renewable energy**, drones check and maintain solar panels and wind turbines. A drone with a thermal camera can scan hundreds of solar panels in one short flight and find the faulty ones, because they have a different temperature [50]. For wind turbines, workers no longer need to climb to dangerous heights, so drones make the work both safer and cheaper [49].

In **logistics and transport**, drone delivery has become real. Amazon's Prime Air has completed thousands of fast deliveries in less than one hour [53]. New systems called

Autonomous Delivery Networks (ADNs) are also changing last-mile delivery, especially in far villages and difficult areas [55].

In **environmental monitoring**, drones protect wild animals, help in search and rescue, assess damage after disasters, and observe nature. They are fast, they can fly over large areas, and they can carry thermal and multispectral cameras. For this reason, they can do work that ground teams cannot do easily [56, 57].

On the artificial-intelligence side, Deep Learning (DL) and Convolutional Neural Networks (CNNs) allow drones to do more than take photos. A drone can now analyse the photos by itself. CNNs are very good at classifying images, so when we add a CNN to a drone, the drone becomes a smart system that can find problems and support decisions.

## 2.2 CNN Architectures for Plant Disease Detection

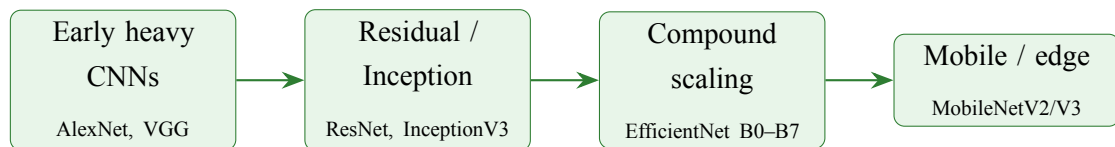


**Figure 2.1.** This figure shows how the main CNN models have developed over time, moving from large, heavy models to small, light models that can run on a drone.

### 2.2.1 Early Works and the PlantVillage Dataset

The PlantVillage dataset was created by Hughes and Salathé in 2015, and it is the most widely used dataset in this field [27]. It contains more than 50,000 photos of healthy and diseased leaves, covering 38 disease classes and 14 types of crops. Most early studies used it under laboratory conditions.

Mohanty et al. (2016) [28] were among the first to use deep CNNs for plant disease. They trained AlexNet and GoogLeNet on PlantVillage and reached up to 99.35% accuracy. However, this result was true only in the lab. When they tested the same models on real field photos, the accuracy dropped to about 31%. This showed an important problem: models trained in the lab do not work well in the real world. This problem is known as the **domain-gap problem**.

### 2.2.2 Classical Architectures: VGG, ResNet, InceptionV3

Several studies have tested older CNN models on PlantVillage. Sood and Singh (2023) [29] used transfer learning with VGG16, ResNet50, InceptionV3, and DenseNet121. They confirmed that transfer learning works better than training from scratch, but they also found that large models like ResNet50 need a lot of computing power.

Naik et al. (2022) [37] compared twelve CNN models, including VGG19 and DarkNet53, for chilli disease. VGG19 reached the highest accuracy in their study, 83.54%, but it has 138 million parameters, so it is very difficult to use on a drone or a small device.

Table 2.1 compares these classical models.

**Table 2.1.** Comparison of classical CNN architectures for plant-disease detection.

| Model          | Parameters | Accuracy   | Dataset      | Drone-ready?   |
|----------------|------------|------------|--------------|----------------|
| AlexNet        | ~60 M      | ~99% (lab) | PlantVillage | No (too heavy) |
| VGG16          | 138 M      | 93.5%      | PlantVillage | No (too heavy) |
| VGG19          | 143 M      | 83–98%     | PlantVillage | No (too heavy) |
| ResNet50       | ~25 M      | 90–97%     | PlantVillage | Marginal       |
| DenseNet121    | ~8 M       | 91–95%     | PlantVillage | Partially      |
| InceptionV3    | ~23 M      | 88–96%     | PlantVillage | Marginal       |
| EfficientNetB4 | 19 M       | 95–99%     | PlantVillage | Yes (~71 MB)   |

### 2.2.3 Lightweight Models: the MobileNet Family

MobileNetV2 and MobileNetV3 were designed to solve the cost problem of large models. They use a special type of convolution that needs fewer parameters. Doğan (2023) [31] trained MobileNetV3 on 87,000 PlantVillage photos and reached 99.85% accuracy. This shows that light models can be very competitive when the data is well prepared.

However, MobileNet does not perform as well when there are many classes. In a test with 101 disease classes from 33 crops [38], EfficientNet-B1 reached 94.7% accuracy, while MobileNetV3 was lower. Light models therefore tend to struggle when the task becomes large and complex.

### 2.2.4 The EfficientNet Family

EfficientNet was introduced by Tan and Le (Google Brain) in 2019 [40]. Its main idea is called *compound scaling*. While ResNet only makes the model deeper, EfficientNet increases the width, the depth, and the image size at the same time, using a fixed ratio. In this way, it finds a good balance between accuracy and speed.

Table 2.2 shows the EfficientNet family.

**Table 2.2.** EfficientNet family comparison.

| Model          | Params | ImageNet Top-1 | Input size       | Disk size |
|----------------|--------|----------------|------------------|-----------|
| EfficientNetB0 | 5.3 M  | 77.1%          | $224 \times 224$ | ~20 MB    |
| EfficientNetB1 | 7.8 M  | 79.1%          | $240 \times 240$ | ~29 MB    |
| EfficientNetB3 | 12 M   | 81.6%          | $300 \times 300$ | ~45 MB    |
| EfficientNetB4 | 19 M   | 82.9%          | $380 \times 380$ | ~71 MB    |
| EfficientNetB7 | 66 M   | 84.3%          | $600 \times 600$ | ~256 MB   |

Albattah et al. (2022) [30] proposed an improved EfficientNetV2-B4 with extra dense layers for drone-based plant disease. They trained it on PlantVillage together with drone photos and reached 99.99% accuracy, 99.93% recall, and 99.63% precision. This is one of the best results in the field, and it is a strong reason to use EfficientNet for this kind of task.

For rust disease, Mullen et al. (2023) [32] compared ResNet50 and EfficientNetB4. EfficientNetB4 reached 94.29% accuracy, while ResNet50 reached 93.52%. They used the Adam optimiser with a learning rate of 0.001.

## 2.3 Drone-Based Disease-Surveillance Systems



**Figure 2.2.** A drone flies over a farm field and takes photos in the air. An AI system on the drone then uses these photos to detect disease.

### 2.3.1 Why Drones in Agriculture?

A drone with a camera can fly over large fields and take clear photos very quickly. Compared with manual inspection, drones offer several advantages:

- **Speed:** a drone can scan hundreds of hectares in one hour.
- **Coverage:** it can reach areas that are difficult to walk to.
- **Early detection:** aerial photos can show disease before it becomes visible to the eye.
- **Data quality:** modern drone cameras can take RGB, multispectral, and hyperspectral photos.
- **Automation:** together with AI, the drone becomes a fully automatic expert system.

### 2.3.2 A Systematic Review of UAV-Based Plant Disease Detection (Chin et al., 2023)

Chin, Catal, and Kassahun (2023) [35] published a review in *Precision Agriculture*. They selected 38 studies about drones for plant disease and analysed them by disease type, drone type, machine-learning method, and edge option.

Their main finding was clear. CNNs reach high accuracy on standard datasets, but they perform worse in real conditions because of light changes, hidden leaves, and complex backgrounds. The authors also found that most systems use only a few classes, and that very few systems run in real time on the drone itself. This review therefore shows that we need a strong system that works well in the field.

### 2.3.3 A Drone System for Multiclass Plant Disease Detection Using EfficientNetV2 (Albattah et al., 2022)

Albattah et al. [30] designed a complete drone system for disease detection. It is based on an improved EfficientNetV2-B4 with extra dense layers at the end.

They tested the system on PlantVillage and on drone photos taken under different light and backgrounds, and the results were very good: 99.99% accuracy and 99.93% recall. The work shows that the combination of EfficientNet, transfer learning, and real drone photos gives excellent results.

**Limitation.** The system was tested only on the 38 PlantVillage classes, which are balanced. It does not handle strong class imbalance, and it offers neither an expert system nor output in two languages for local farmers.

### 2.3.4 AgroVisionNet: A Multimodal CNN–Transformer System (Manoj et al., 2025)

Manoj, Shanthi, Lakshmi et al. (2025) [33] presented a system called AgroVisionNet in *Scientific Reports*. This work is one of the closest to our idea.

AgroVisionNet combines a CNN and a Transformer to extract both spatial and contextual features from drone photos. It also uses IoT sensor data, such as temperature, humidity, and soil moisture, through a special fusion layer, which adds information about the environment to the diagnosis. The authors compared it with VGG16, ResNet50, InceptionV3, and DenseNet121, and AgroVisionNet achieved better F1-scores than all of them. They used Grad-CAM to explain the model’s decisions, and they deployed the model on an NVIDIA Jetson Nano with TensorFlow Lite for real-time use.

**Limitations.** The number of classes is not clear. There is no expert system with treatment advice, the work does not handle class imbalance, and there is no output in two languages for farmers.

### 2.3.5 Systematic Review of UAV and AI Integration (2025)

A review in *Applied Sciences* [34] analysed UAV and deep-learning papers indexed in Web of Science up to 2024. Its main findings are:

- The number of papers on UAV and deep learning for agriculture grew strongly after 2019.
- China, the USA, and India lead this research.
- Multirotor drones with RGB cameras are the most common, because they are cheap and give good photos.
- Light models such as MobileNet and EfficientNet are now preferred for real-time use.
- EfficientNet-B3 reached an F1-score of 98.80% on tomato, potato, and pepper diseases.
- EfficientNet performed better than ResNet and Vision Transformers on spinach disease ( $F1 = 99.40\%$ ).

### 2.3.6 CNN Applications in Smart Agriculture: A Multimodal Review (2025)

A broad review studied the use of CNNs in smart agriculture [36]. It covered RGB cameras, multispectral sensors, and IoT data. The review confirmed that simple RGB models are still the most common in the field, because they are cheaper and easier to use. It also pointed out

that strong class imbalance is one of the problems that few studies solve, even though it is very common in real farm data.

## 2.4 Transfer Learning and Dataset Challenges

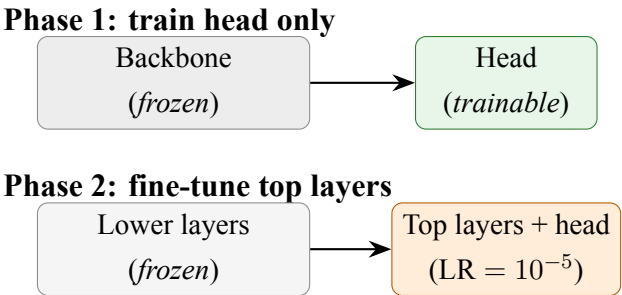


**Figure 2.3.** The two-phase transfer-learning method. In Phase 1, only the new head is trained while the backbone stays frozen. In Phase 2, the top layers are opened and trained slowly.

### 2.4.1 Why Transfer Learning?

Training a CNN from scratch needs millions of labelled photos and many weeks of GPU time. Transfer learning makes this much easier, because we start from a model that is already trained on ImageNet (1.2 million photos, 1000 classes). This model has already learned basic features such as edges, textures, and shapes, and these features are useful for almost any image task.

Table 2.3 compares the main transfer-learning methods.

**Table 2.3.** Transfer-learning strategies compared.

| Strategy                         | Description                                                                     | Typical use case                                   |
|----------------------------------|---------------------------------------------------------------------------------|----------------------------------------------------|
| <b>Feature extraction</b>        | The backbone is fully frozen. Only the new head is trained.                     | Small datasets (< 5,000 images). Fast but limited. |
| <b>Fine-tuning</b>               | After head training, the last $N$ layers are opened with a small learning rate. | Larger datasets. Better domain adaptation.         |
| <b>Two-phase (best practice)</b> | Phase 1: frozen backbone. Phase 2: open top layers with $\text{LR} = 10^{-5}$ . | Best results for special data, like drone images.  |

Sood and Singh (2023) [29] showed that fine-tuned models perform much better than models

trained from scratch. ResNet50 with fine-tuning reached about 97% accuracy, while without pre-trained weights it reached only about 75%.

### 2.4.2 Class Imbalance: a Critical Problem

Most studies use PlantVillage, whose classes are fairly balanced (about 1,000 to 3,000 photos each). Real datasets, however, are often very unbalanced. Barbedo (2018) [39] showed that the size and the variety of the dataset are the main factors that decide how well a CNN works in plant pathology. He also noted that a class with fewer than about 200 photos gives unreliable results.

The usual solution is *class weighting*, which gives a higher weight to small classes so that the model pays more attention to rare classes during training. It is also important to use the macro F1-score as the main metric instead of simple accuracy, because macro F1 treats every class equally, even if it has few photos.

### 2.4.3 Data Augmentation for Drone Conditions

Most studies use only simple augmentation, such as flipping and rotating the image. Very few studies design augmentation that imitates real drone conditions, such as different heights, sensor noise, and changing light caused by clouds.

## 2.5 Comparative Synthesis of Reviewed Works

Table 2.4 brings together all the studies reviewed in this chapter.

**Table 2.4.** Comparative synthesis of the reviewed works. (n.r. = not reported.)

| Ref. | Authors / year            | Model                        | Dataset / classes     | Acc.   | Key limitation             |
|------|---------------------------|------------------------------|-----------------------|--------|----------------------------|
| [28] | Mohanty et al. (2016)     | AlexNet, GoogLeNet           | PlantVillage (38)     | 99.35% | Fails on real field images |
| [29] | Sood & Singh (2023)       | VGG16, ResNet50, InceptionV3 | PlantVillage (38)     | ~97%   | No imbalance handling      |
| [30] | Albattah et al. (2022)    | EfficientNetV2-B4 + dense    | PlantVillage + drone  | 99.99% | Only 38 classes            |
| [31] | Doğan (2023)              | MobileNetV3                  | PlantVillage (87k)    | 99.85% | Limited classes            |
| [32] | Mullen et al. (2023)      | EfficientNetB4 vs ResNet50   | Rust dataset          | 94.29% | Single disease only        |
| [33] | Manoj et al. (2025)       | CNN–Transformer + IoT        | Multi-crop            | n.r.   | No expert system           |
| [34] | UAV review (2025)         | EfficientNetB3               | Multiple crops        | 98.80% | No expert system           |
| [35] | Chin et al. (2023)        | SLR (38 studies)             | Various               |        | Few classes; no XAI        |
| [36] | CNN review (2025)         | Various                      | Various               |        | Imbalance ignored          |
| [37] | Naik et al. (2022)        | VGG19, DarkNet53             | Chilli                | 83.54% | Too many parameters        |
| [38] | Lightweight bench. (2024) | EfficientNetB1               | 101 cls / 33 crops    | 94.7%  | Not drone-based            |
| [39] | Barbedo (2018)            | Various CNNs                 | PlantVillage analysis | —      | Data size matters          |

## 2.6 Drone Applications in Precision Agriculture

Drones in agriculture have created a new approach called **precision agriculture**. With this approach, farmers can monitor, study, and manage their crops in great detail, so that the soil and the plants receive exactly what they need [41, 42]. Drones are one of the main tools of

this approach, and they do much more than simply take photos.

### 2.6.1 Overview of Drone Services in Agriculture



**Figure 2.4.** A drone monitors spring crops in agriculture. Drones provide a view from the sky and collect detailed data over large areas [41].

According to Sillanpää and Salo (2024) [41], drones offer many services that are useful for crop management:

- **Crop-health monitoring:** with RGB, multispectral, and thermal cameras, drones detect stressed or diseased areas before the eye can see them. NDVI (Normalised Difference Vegetation Index) images show plant stress across the whole field.
- **Field mapping:** drones produce clear maps and 3D models of the land, showing the shape of the field, the water flow, and the ground, which helps to plan irrigation and drainage.
- **Precision spraying:** with a tank and spray arms, drones apply pesticide or fertiliser only where it is needed, which reduces chemical use and keeps people away from danger [43].
- **Animal monitoring:** cameras let farmers count and follow animals over large areas, detect sick animals early, and check the quality of the grass [44].
- **Yield estimation:** with drone photos and software, farmers can estimate the yield before harvest and plan their work [42].
- **Weed detection:** in the study by Sillanpää and Salo (2024), farmers wanted to use drones to find weeds such as wild oats, which are hard to see in large fields from the ground.

### 2.6.2 Imaging Modalities Used in Agricultural Drones



**Figure 2.5.** A modern agricultural drone with a tank and spray arms. Drones are modular: the camera and the tools can be changed for each task, which shows how flexible they are [41].

The camera on the drone decides which data can be collected, and different cameras have different uses [43]:

- **RGB cameras** give normal colour photos for inspection, mapping, and 3D models.
- **Multispectral cameras** capture light that the eye cannot see, which helps to compute NDVI and check plant health.
- **Thermal cameras** show temperature differences in the field and reveal water problems, dry areas, and heat in animals or machines.
- **Hyperspectral cameras** give the most detailed data and help to detect disease and study the soil, but they are still expensive.

Modern agricultural drones are modular: the camera can be changed easily, and the operator chooses the sensor for each task [41].

### 2.6.3 Precision-Agriculture Software Platforms

Hardware alone is not enough for a complete precision-agriculture system; software is also needed to turn raw photos into useful information. One of the leading platforms is **DroneDeploy**, which provides crop-health monitoring, multispectral analysis, mapping, yield estimation, and animal tracking [41].

DroneDeploy can also plan the flight automatically: the operator draws the field, and the software finds the best path with full and overlapping coverage. This is very useful for large fields, where flying by hand would be difficult. Its AI marks problems on the maps, identifies areas that need action, and creates reports that can be shared with agronomists or the authorities.

### 2.6.4 Barriers to Adoption in Agriculture

Although the benefits are clear, several factors still limit the use of drones in agriculture, as both the literature and a survey in Finland show [41]:

1. **Little information:** 73% of the people in the survey said they were not sure about the real benefits, and this was their main reason to wait. There is little local information about what drones can do.
2. **Rules:** in many countries, including Algeria and Finland, business drone work needs a licensed pilot and a permission. This is expensive for a single farmer, so a service model, in which a licensed company visits the farm, is often more practical than owning a drone.
3. **Cost:** drones that carry tanks or special sensors are much more expensive than simple photo drones, and the initial cost is difficult for small and medium farms.
4. **Ease of use:** about 90% of the people said that ease of use is a key reason to choose drones, so the tools need a simple interface and good training.
5. **Satellites:** one person noted that satellite images are now good for some tasks, such as daily NDVI maps. Satellites cover large areas without a local pilot, but they cannot give the close detail of a drone.

### 2.6.5 Relevance to This Work

The agricultural-drone literature shows several points that support our work. First, drones and deep learning already give very good results in the lab, but real field use is still limited [35, 41]. Second, the most requested drone services (crop health, mapping, and spraying) all need reliable, automatic disease detection. Third, the problem of uncertainty about the benefits can only be reduced if the system gives a clear and useful result for the farmer, not just a class name.

Our work addresses these gaps. We propose a drone system for plant disease that uses a strong deep-learning model together with an expert recommendation engine and output in two languages. In this way, the results become useful for local farmers, even without technical skills.

## 2.7 Drone Applications in Construction and Renewable Energy



**Figure 2.6.** A drone takes photos of a construction site from the sky. A single flight can show the whole large site, so managers receive up-to-date information at low cost [41].

Construction was one of the first non-military fields to use drones widely. Today, drones are used in many steps, from the first survey to the final inspection, and they make the work faster and safer while giving better data [41, 45].

### 2.7.1 Site Surveying and Topographic Mapping

Before construction starts, good information about the land is necessary. Drones with cameras, LiDAR, or photogrammetry can survey a site quickly and produce very accurate maps. Pinguet (2020) [46] explains that drone surveys have three advantages over old ground methods: faster data, lower cost, and higher accuracy.

LiDAR drones send laser pulses to measure distance and the shape of the land, even where there are many plants. The data is then turned into 3D models, so managers get an accurate digital copy of the site before they start.

### 2.7.2 3D Modelling and Progress Monitoring

One of the most useful contributions of drones is the creation of 3D models from many overlapping photos. Platforms such as **DroneDeploy** turn these photos into maps, volume calculations, and 3D models [41, 51]. These models are useful in several ways:

- **Planning:** 3D models help to design the foundation, the drainage, and the materials.

- **Progress:** regular flights (every one or two weeks) let managers compare the site with the plan and find delays early [41].
- **Reports:** good photos and 3D views give clear reports to investors, without a site visit.

### 2.7.3 Inspection and Safety

Construction sites have many dangerous or hard-to-reach places: high points, small spaces, boilers, and unstable structures. Drones can reach these places safely and take clear photos without putting anyone at risk [45, 41].

Regular checks also save time, because they find problems before they cause a failure. With thermal cameras, drones can detect heat problems in electrical systems or water inside walls. For security, drones can patrol large outdoor areas and work together with fixed cameras to detect people who should not be there, theft, or damage [47].

### 2.7.4 Integration with Building Information Modelling (BIM)

An important reason why drones are useful in construction is that they work well with Building Information Modelling (BIM). BIM is a 3D, model-based process that represents the physical and functional parts of a building [48]. When drone data is added to BIM in real time, managers get a live digital copy of the site, so they can detect problems early and coordinate the teams better.

### 2.7.5 Applications in Renewable Energy



**Figure 2.7.** A drone with a thermal camera inspects a solar-panel field. Temperature problems reveal the faulty panels, so there is no need to check each panel by hand [50].

The renewable-energy sector, mainly solar farms and wind turbines, uses drones for inspection and maintenance. Traditional wind-turbine inspection needs workers at dangerous heights, but drones remove this danger by taking clear photos of the blades from a safe distance [49].

For solar farms, thermal drones scan hundreds of panels in one short flight and find the faulty ones by their temperature [50]. This is much faster than manual inspection, and it produces reports with exact locations, so the maintenance team can act quickly. Aerial photos also help to manage and report on large sites.

## 2.8 Drone Applications in Logistics and Transportation



**Figure 2.8.** A drone delivers a package in Shenzhen, China. Companies in China and the United States have used delivery drones for several years, which shows that drone logistics is now real [41].

Drones are bringing important changes to logistics and transport, especially for *last-mile delivery*, which is the final stage of a product from the centre to the customer's door [52]. Drones can reduce costs, speed up delivery, and help the environment, especially where normal trucks are slow.

### 2.8.1 Last-Mile and Short-Distance Delivery

Companies such as Amazon and DHL are testing and using delivery drones. Amazon's Prime Air started commercially in 2022 and has completed thousands of deliveries in less than one hour, in places such as College Station, Texas, and the West Valley of Phoenix [53]. Its newest model, the MK30, carries up to five pounds, flies in light rain, and holds FAA permission for Beyond Visual Line of Sight (BVLOS) flights.

According to DHL (2024) [54], drone delivery is being studied, tested, or already used for three main purposes:

1. **Last-mile delivery:** packages go directly from a hub to the customer, avoiding road traffic; the main constraint is the airspace rules.
2. **Inventory:** inside warehouses, drones with scanners automate stock counting, which is slow by hand.
3. **Surveillance:** large outdoor sites are hard to watch with fixed cameras, so drones provide full coverage from the air.

### 2.8.2 Autonomous Delivery Networks

By combining drones with AI and robotics, researchers created the idea of **Autonomous Delivery Networks (ADNs)**. These systems use AI to plan the route and the schedule without human help, so they provide on-time, low-cost delivery in rural or hard-to-reach areas [55]. In difficult regions, such as islands, mountains, or empty zones, ADNs could provide a service that normal logistics cannot offer at a good price.

### 2.8.3 Regulatory and Infrastructure Constraints

Despite the good early tests, wide adoption still faces problems with rules and infrastructure. The airspace over cities is strictly regulated, and BVLOS permission needs a lot of safety proof. Special drone corridors and certified landing places (drone ports with charging) do not yet exist at a large scale in most countries [41]. To solve this, aviation authorities, logistics companies, and city planners need to work together to define safe rules and build the necessary ground infrastructure.

## 2.9 Drone Applications in Environmental Monitoring



**Figure 2.9.** A drone assesses the area during a flood from the sky. Drones give fast information in an emergency and help to find victims, assess damage, and identify dangers [57].

Beyond agriculture and industry, drones are very useful for the environment and for wildlife. They cover large areas quickly, reach difficult terrain, and carry special sensors, so they fit many ecological and emergency tasks [41].

### 2.9.1 Wildlife Monitoring and Anti-Poaching

Drones allow scientists and rangers to monitor and follow animals without disturbing them. Counting animals on foot is slow and difficult, but a drone with a good camera can do it quickly [56]. In anti-poaching work, thermal drones can detect people in protected areas at night, which provides a real-time alarm and helps rangers across large, remote areas.

### 2.9.2 Search-and-Rescue Operations

Drones are used more and more in Search and Rescue (SAR). When a person is lost in a forest, on a mountain, or in a flood, a drone can search large areas much faster than a ground team, and thermal cameras can detect body heat even through thick vegetation [57]. SAR drones usually carry several cameras for a wide view, have a long battery for long missions, and can sometimes drop emergency supplies to the victim before the ground team arrives.

### 2.9.3 Disaster Assessment and Emergency Response

Just after a flood, an earthquake, or a fire, good information is very important for planning the response. Drones can fly within minutes to take photos of the area, assess the damage, find victims, and identify dangers such as fire or unstable buildings [57, 41]. This is much faster than ground-based assessment, and it gives coordinators the information they need to use their resources well. Because drones work in many types of disaster, from floods to fires, they have become a standard tool in emergency management.

### 2.9.4 Forest and Ecosystem Monitoring

Drones are also used to monitor forests for cutting, illegal logging, and damage. Regular flights over the same area let scientists follow changes in the plants, check new trees, and monitor the water in rivers and lakes. For example, DroneDeploy can create maps of plant change and follow wildlife corridors using time-series photos [51, 41].

## 2.10 Research Gap

The review above reveals several problems that no system has fully solved:

1. **Few disease classes** [28, 29]: most studies use only the 38 PlantVillage classes, but real farms have many more diseases and many more crops.
2. **Class imbalance not solved** [39, 36]: no study offers a real solution for very large imbalance (for example  $130\times$ ), and most studies ignore it.

3. **No expert system for treatment** [30, 33]: the systems detect the disease but do not tell the farmer what to do next, so there is no treatment advice in the literature.
4. **Lab-and-field gap** [28, 35]: models trained on PlantVillage perform badly on real drone photos, and few studies use drone augmentation (height, sensor noise, cloud light).
5. **No two languages** [35, 34]: all systems give output in English only, but local farmers in Algeria need an Arabic interface.
6. **Black-box models** [33, 35]: most systems do not show why they made a decision, so tools such as Grad-CAM are needed to let farmers check and trust the result.

## 2.11 Legal Framework for Drone Use in Algeria

### 2.11.1 Overview of Algerian Drone Regulations

Drone use in Algeria is now controlled by a clear legal framework that was developed over the last few years. Its basis is Presidential Decree No. 21-285 of 13 July 2021 [58], which set the general rules for all Unmanned Aircraft Systems (UAS) in Algeria. It covers making, buying, importing, exporting, selling, and using all civil drones, and it came into effect on 18 July 2021.

The Interministerial Decree of 8 September 2025 [59] then added more rules for buying, importing, making, and using drones for business. It also created a clear process through which any person or company can ask for official approval from the National Centre for Unmanned Aircraft Systems.

In December 2025 [60], the Algerian Ministry of National Defence asked all drone owners to register their drones with the National Centre before 30 April 2026. This shows that the Centre is now fully working and is applying the rules.

This framework shows that Algeria takes drone regulation seriously. For any system that uses drones for work or farming, following the rules is not a choice but a legal duty.

### 2.11.2 Drone Categories Under Algerian Law

According to the Interministerial Decree of 8 September 2025 [59], drones in Algeria are divided into three categories:

- **Category 1, Recreational** [59]: maximum weight 2 kg. The drone must have a geo-monitoring system and electronic identification, and business use is not allowed.

- **Category 2, Professional** [59, 58]: weight up to 150 kg. Permission from the National Centre is required before any flight.
- **Category 3, State use** [58]: no weight limit. It is reserved for defence, security, rescue, and firefighting, and only for government and military bodies.

A drone used for farm disease detection belongs to Category 2 and must follow all the professional rules [59].

### 2.11.3 Authorisation Requirements for Agricultural Drone Use

For a drone used in professional farming, the following requirements must be met before any flight [58, 59]:

1. **Registration** [60]: the drone must be registered with the National Centre before 30 April 2026 and must have electronic identification and geo-monitoring hardware.
2. **Permission** [59]: professional use needs written permission from the National Centre before any flight, and this permission must be renewed for each new area.
3. **Maximum altitude** [58]: the legal limit is 120 m above mean sea level (AMSL), unless a special exemption is granted.
4. **BVLOS** [59]: professional drones may fly BVLOS if the flight is safe for air traffic, but recreational drones may not fly BVLOS at all.
5. **Import approval** [58]: importing a drone needs approval from the Ministry of National Defence first, which is especially important for powerful, high-performance drones.
6. **No-fly zones** [58]: Decree No. 21-285 defines strict no-fly zones near sensitive places, airports, military zones, and government buildings, so the operator must check that the flight area is not in a restricted zone.

### 2.11.4 Steps to Obtain Authorisation for Deployment

For a real deployment of a farm drone system in Algeria, the following steps should be followed [59, 58]:

1. Contact the National Centre for Unmanned Aircraft Systems to register the drone and obtain an official identification number [59].
2. Submit a technical file that describes the drone model, the camera, the maximum weight, and the farm areas [59].

3. Apply for a professional-use permission that specifies the target zones (with GPS borders) and the flight time [59].
4. Keep a log of all flights with GPS coordinates, time, and operator identification, so that everything can be checked [58].
5. Work with the Ministry of Agriculture if the system is part of an official programme [60]; since 2025, the Ministry has published an ordinance that supports drone use in farming.
6. Renew the permission from time to time and inform the National Centre about any change in the equipment or the operational area [59].

### 2.11.5 Why Legal Compliance Matters

Working within the Algerian legal framework is more than a formality [58, 59]; it is necessary for any project that wants to grow. When farmers, local authorities, and universities know that a system is legal, they trust it much more [35].

A system that is registered, authorised, and transparent in how it works builds confidence, which informal systems cannot do. Legal compliance also ensures continuity [60]: an unauthorised system can be stopped by the authorities at any time, which interrupts the field work and damages the project. By following all the steps above, a farm drone project in Algeria can operate safely and with the full support of the institutions [59, 58].

## Chapter 3

# System Modeling and General Architecture

### Introduction

In the previous chapters we presented the context of our work, the problem of crop disease detection, and the technical background of our solution. In this chapter we move from the idea to the design. The goal here is to show, in a clear and structured way, how the **Skyline** platform is built and how its different parts work together.

To describe the system we use the **Unified Modeling Language (UML)**. UML is a standard graphical language that helps us model software in a way that both engineers and non-technical readers can follow. We do not try to describe every small detail of the code. Instead, we focus on the parts that matter the most for understanding the system: the data, the actors, the components, the dynamic behaviour, and the physical deployment of the platform.

This chapter is organised in two large parts. The first part is the **UML modeling**, where we present five diagrams. The *class diagram* shows the data structure of the platform and the relations between the entities. The *use case diagram* shows the actors and the actions they can perform. The *component diagram* shows how the platform is divided into software components and how they communicate. The *sequence diagram* shows the order of messages exchanged during a full mission, from registration to the delivery of results. Finally, the *deployment diagram* shows on which physical machines each component actually runs. For each diagram we first give a general scenario, then we show the figure, and finally we give a detailed description.

The second part presents the **general architecture** of the project. Here we explain the project as a whole, we show our layered architecture diagram, and we describe each layer one by one. By the end of the chapter, the reader should have a complete mental picture of how the

platform is designed, before we move to the implementation in the next chapter.

## 3.1 UML Modeling

UML modeling allows us to look at the same system from several points of view. A single diagram can never show everything, so we combine five complementary views. Together, the class diagram (the static structure), the use case diagram (the user point of view), the component diagram (the software organisation), the sequence diagram (the dynamic behaviour) and the deployment diagram (the physical setup) give a balanced and complete description of the Skyline platform.

### 3.1.1 Class Diagram

#### General Scenario

The class diagram describes the static structure of the system. In other words, it answers the question: *what kind of data does the platform store, and how is this data connected?* Our database is hosted on Supabase (PostgreSQL), so each class in the diagram corresponds closely to a real table in the database.

The central element of the model is the **Profile** class. Every person who uses the platform, whether a farmer or an administrator, has one profile. From this profile, many things branch out. A farmer owns one or more **Agriculture** records (their farms), submits **MissionRequest** records (a demand for a drone inspection), receives **Notification** and **FieldResult** records, and may have a **Security** configuration.

On the operational side, the **Drone** class represents the physical UAV that performs the mission. Each drone can offer one **Service** (a paid plan) and can be assigned to a mission request. When the drone captures images, the AI engine produces a **Diagnostics** record. Each diagnostic is linked to a known disease stored in the **DiseaseDatabase**, and is also tied to a geo-located **FieldResult** that is later shown on the map. This chain, from *Profile* to *FieldResult*, is the backbone of the whole platform.

#### Diagram

Figure 3.1 presents the complete class diagram of the Skyline platform.

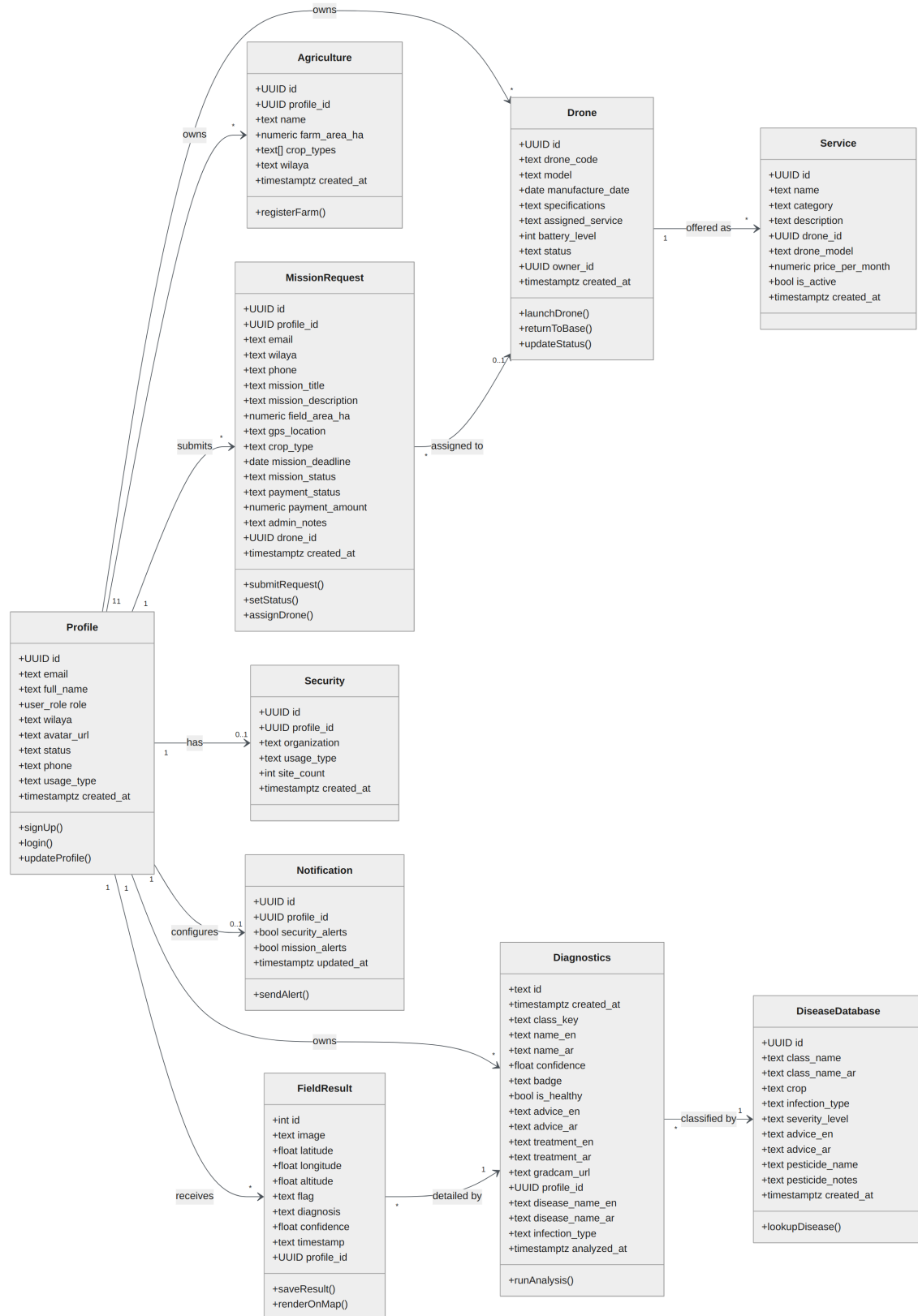


Figure 3.1. Class diagram of the Skyline platform.

### Detailed Description

In this section we describe each class (table), its main attributes, and the relationships (associations and cardinalities) that connect it to the rest of the model.

**Description of the classes.** Table 3.1 summarises the eleven classes of the diagram, their role, and their most important attributes.

**Table 3.1.** Description of the main classes of the system.

| Class                 | Role                                                                        | Main attributes                                                                                                                                   |
|-----------------------|-----------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Profile</b>        | Central user account (farmer or admin). It is the entry point of the model. | id, email, full_name, role, wilaya, status, phone, usage_type, created_at                                                                         |
| <b>Agriculture</b>    | A farm registered by a farmer, with its surface and crops.                  | id, profile_id, name, farm_area_ha, crop_types, wilaya, created_at                                                                                |
| <b>MissionRequest</b> | A request for a drone inspection, including its status and payment.         | id, profile_id, mission_title, gps_location, field_area_ha, crop_type, mission_deadline, mission_status, payment_status, payment_amount, drone_id |
| <b>Drone</b>          | The physical UAV that performs missions.                                    | id, drone_code, model, manufacture_date, specifications, assigned_service, battery_level, status, owner_id                                        |
| <b>Service</b>        | A paid plan or offer attached to a drone.                                   | id, name, category, description, drone_id, drone_model, price_per_month, is_active                                                                |
| <b>Security</b>       | Optional security and organisation settings of a profile.                   | id, profile_id, organization, usage_type, site_count, created_at                                                                                  |

Table 3.1 – continued from previous page

| Class                  | Role                                                              | Main attributes                                                                                                              |
|------------------------|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>Notification</b>    | The alert preferences of a user.                                  | id, profile_id, security_alerts, mission_alerts, updated_at                                                                  |
| <b>Diagnostics</b>     | The full result produced by the AI engine for one analysed image. | id, class_key, name_en, name_ar, confidence, is_healthy, advice_en/ar, treatment_en/ar, gradcam_url, profile_id, analyzed_at |
| <b>FieldResult</b>     | A single geo-located detection point shown on the map.            | id, image, latitude, longitude, altitude, flag, diagnosis, confidence, timestamp, profile_id                                 |
| <b>DiseaseDatabase</b> | The expert knowledge base of known diseases and treatments.       | id, class_name, class_name_ar, crop, infection_type, severity_level, advice_en/ar, pesticide_name, pesticide_notes           |

### Description of the relationships and cardinalities.

Table 3.2. Associations and cardinalities of the class diagram.

| Source                   | Association | Cardinality | Meaning                                                       |
|--------------------------|-------------|-------------|---------------------------------------------------------------|
| Profile → Agriculture    | owns        | 1 ... *     | One user can register several farms; each farm has one owner. |
| Profile → Drone          | owns        | 1 ... *     | An admin can own several drones; each drone has one owner.    |
| Profile → MissionRequest | submits     | 1 ... *     | One user can submit many mission requests over time.          |

Table 3.2 – continued

| Source                        | Association   | Cardinality | Meaning                                                                   |
|-------------------------------|---------------|-------------|---------------------------------------------------------------------------|
| Profile → Security            | has           | 1 ... 0..1  | A profile may have at most one security configuration.                    |
| Profile → Notification        | configures    | 1 ... 0..1  | A profile may have at most one notification setting.                      |
| Profile → FieldResult         | receives      | 1 ... *     | A user receives many field results from their missions.                   |
| Profile → Diagnostics         | owns          | 1 ... *     | The diagnostics of a user belong only to that user.                       |
| MissionRequest → Drone        | assigned to   | * ... 0..1  | A request may be assigned to one drone; a drone may handle many requests. |
| Drone → Service               | offered as    | 1 ... *     | A drone can be offered through one or more service plans.                 |
| FieldResult → Diagnostics     | detailed by   | * ... 1     | Each map point is detailed by one diagnostic record.                      |
| Diagnostics → DiseaseDatabase | classified by | * ... 1     | Each diagnostic is classified using one disease reference.                |

**Design choice.** We keep Diagnostics and FieldResult as two separate classes on purpose. Diagnostics holds the full AI output (advice, treatment, Grad-CAM image, bilingual text), while FieldResult keeps only the light data needed to draw a fast point on the map (position, flag, short diagnosis). This separation keeps the map responsive and avoids loading heavy data that the user does not need yet.

### 3.1.2 Use Case Diagram

#### General Scenario

The use case diagram looks at the system from the point of view of the people and devices that interact with it. It answers the question: *who can do what?*

In the Skyline platform we identify three actors. The **User (Farmer/Client)** is the main

customer. They sign up, register their farm, submit a mission request, pay for it, follow its status, and finally view the results. The **Admin (Operator)** manages the platform: they review the requests, accept or reject them, launch the drones, and manage the fleet. Note the special *projection* link between the two human actors: an Admin is also able to act as a User. The third actor is the **Drone**, a non-human (system) actor. Once it is launched, the drone flies to the GPS location, captures the field images, and returns to its base.

A separate part of the system, marked as the **AI System**, is responsible for the intelligence of the platform. It is an internal subsystem that analyses the images and generates the diagnosis and the treatment advice.

## Diagram

Figure 3.2 presents the use case diagram of the platform.

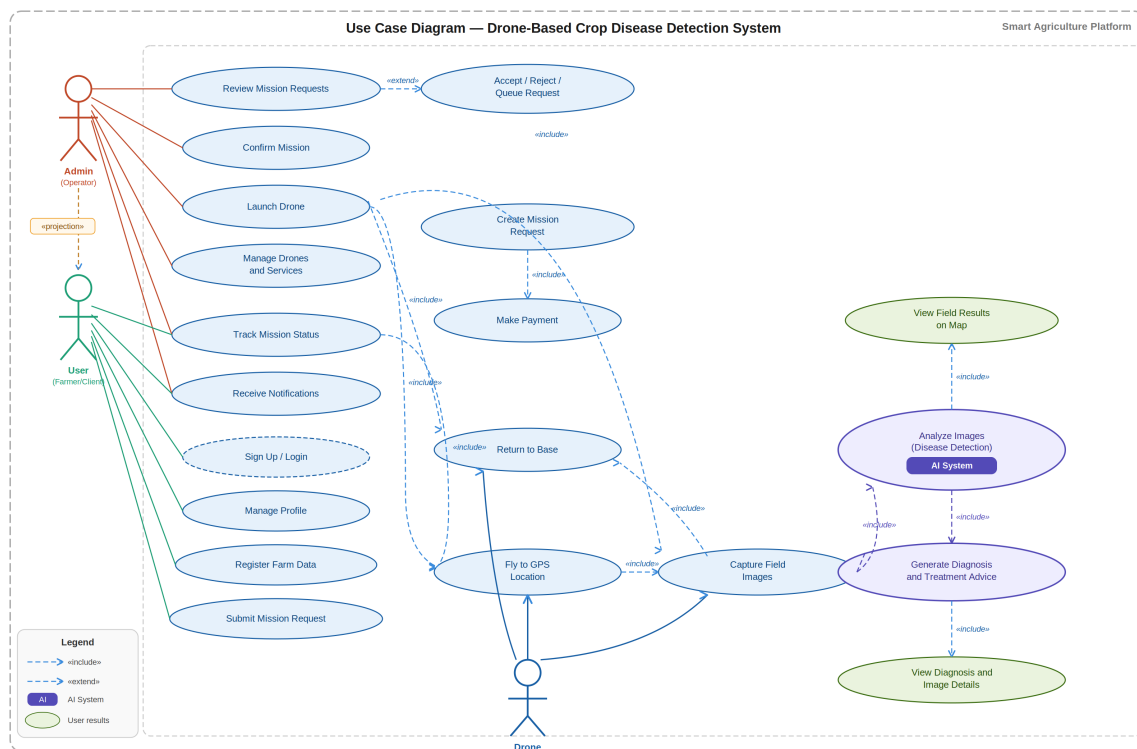


**Figure 3.2.** Use case diagram of the Drone-Based Crop Disease Detection System.

## Detailed Description

**Table 3.3.** Actors of the system and their responsibilities.

| Actor                         | Responsibility                                                                                                                                                                                                                                                         |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User (Farmer / Client)</b> | The end customer. Creates an account, registers farm data, submits and pays for mission requests, tracks their status, receives notifications, and consults the results on the field map.                                                                              |
| <b>Admin (Operator)</b>       | The platform operator. Reviews the incoming requests and accepts, rejects or queues them. Confirms missions, launches drones, and manages the drone fleet and the services. Through the <i>projection</i> link, an admin can also use every function of a normal user. |
| <b>Drone</b>                  | A system actor (the UAV). After launch it flies to the GPS location, captures field images, and returns to base.                                                                                                                                                       |
| <b>AI System</b>              | An internal subsystem. It analyses the captured images, detects the disease, and generates the diagnosis and treatment advice.                                                                                                                                         |

The actors.

The main use cases.

**Table 3.4.** Main use cases of the platform.

| Use case               | Main actor | Description and relations                                                         |
|------------------------|------------|-----------------------------------------------------------------------------------|
| Sign Up / Login        | User       | Create an account or log in. It is the common base of all the other user actions. |
| Manage Profile         | User       | Update personal data and account settings.                                        |
| Register Farm Data     | User       | Save a farm with its surface, crops and location.                                 |
| Submit Mission Request | User       | Send a demand for a drone inspection.                                             |
| Create Mission Request | User       | Build the request. <i>Includes</i> Make Payment.                                  |

Table 3.4 – continued

| Use case                           | Main actor | Description and relations                                                                 |
|------------------------------------|------------|-------------------------------------------------------------------------------------------|
| Make Payment                       | User       | Pay for the mission. Mandatory step of the request.                                       |
| Track Mission Status               | User       | Follow the progress of a mission in real time.                                            |
| Receive Notifications              | User       | Get alerts when the mission status changes.                                               |
| View Field Results on Map          | User       | See the green/red detection points on the map. <i>Includes</i> the AI analysis result.    |
| View Diagnosis and Image Details   | User       | Open a point to read the full diagnosis, advice and Grad-CAM image.                       |
| Review Mission Requests            | Admin      | Inspect the pending requests. <i>Extended by</i> Accept / Reject / Queue.                 |
| Confirm Mission                    | Admin      | Validate a request so it becomes an active mission.                                       |
| Launch Drone                       | Admin      | Assign and start a drone. <i>Includes</i> Fly to GPS Location and Return to Base.         |
| Manage Drones and Services         | Admin      | Add, update or remove drones and their service plans.                                     |
| Fly to GPS Location                | Drone      | Travel autonomously to the field coordinates.                                             |
| Capture Field Images               | Drone      | Take the aerial photos of the crops. <i>Includes</i> Analyze Images.                      |
| Return to Base                     | Drone      | Come back home after the mission.                                                         |
| Analyze Images (Disease Detection) | AI System  | Run the model on the images and classify the disease. <i>Includes</i> Generate Diagnosis. |

Table 3.4 – continued

| Use case                                | Main actor | Description and relations                                          |
|-----------------------------------------|------------|--------------------------------------------------------------------|
| Generate Diagnosis and Treatment Advice | AI System  | Produce the final bilingual advice, treatment and Grad-CAM output. |

### 3.1.3 Component Diagram

#### General Scenario

The component diagram looks at the platform from the software architecture point of view. It answers the question: *what are the main software building blocks of the system, and how do they communicate?* While the class diagram focused on the data, the component diagram focuses on the modules that hold this data and the services that act on it.

The platform is split into six logical layers. The **Client Layer** holds the two user-facing applications: a frontend dashboard for the administrator and a mobile app for the farmer. They both talk to the backend through HTTP REST calls. The **Backend Layer** contains the FastAPI service, which is the central hub of the system. It receives the HTTP requests, orchestrates the model inference, and also serves the field map and GIS data. The **AI Layer** is made of two components: the AI Engine, which runs the EfficientNet-B4 model with Grad-CAM, and the Advisory System, which turns the model output into bilingual reports for the farmer. The **Data Layer** stores everything in two complementary places: the Supabase BaaS handles authentication, the PostgreSQL database and the object storage, while the Disease Database holds the expert knowledge used to enrich the diagnosis. The **Simulation Layer** contains the AirSim simulator, which generates synthetic drone images while the real drone is not available yet. Finally, the **Edge Layer (Planned)** groups the components that will live closer to the drone in the future: a TFLite engine for offline inference on the drone itself, and an optional AI chatbot to answer farmer questions.

#### Diagram

Figure 3.3 presents the component diagram of the platform, with all eleven components grouped into their layers.

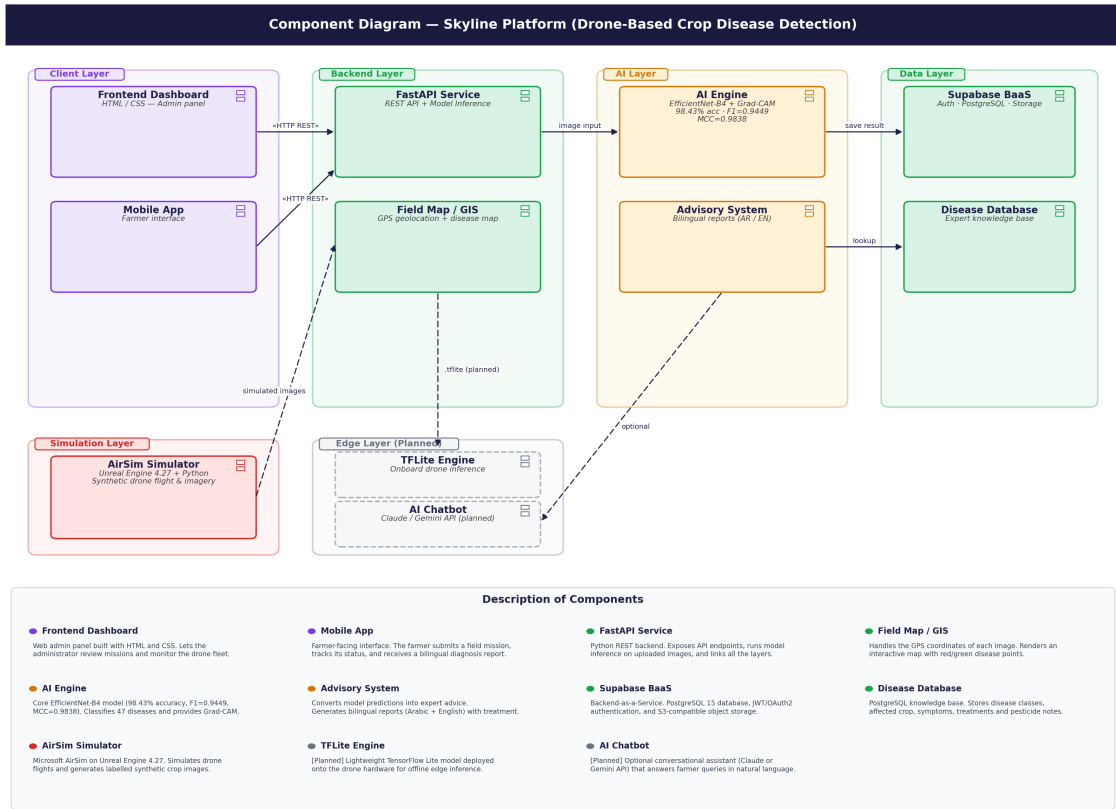


Figure 3.3. Component diagram of the Skyline platform.

Detailed Description

**Description of the components.** Table 3.5 lists the eleven components, the layer they belong to, and their role in the platform.

Table 3.5. Description of the components of the platform.

| Component          | Layer   | Role                                                                                                                                   |
|--------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------|
| Frontend Dashboard | Client  | Web admin panel built with HTML and CSS. Lets the administrator review the missions and monitor the drone fleet.                       |
| Mobile App         | Client  | The farmer-facing interface. The farmer submits a field mission, tracks its status, and receives a bilingual diagnosis report.         |
| FastAPI Service    | Backend | Python REST backend. It exposes the API endpoints, runs the model inference on the uploaded images, and links all the layers together. |

Table 3.5 – continued

| Component               | Layer          | Role                                                                                                                                                                                         |
|-------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Field Map / GIS</b>  | Backend        | Handles the GPS coordinates attached to every drone image and renders the interactive map with the red and green disease points.                                                             |
| <b>AI Engine</b>        | AI             | The core deep learning model (EfficientNet-B4, 98.43% accuracy, F1-Macro = 0.9449, MCC = 0.9838). Classifies 47 disease classes and produces a Grad-CAM heatmap for each prediction.         |
| <b>Advisory System</b>  | AI             | Converts the raw model prediction into expert advice. Generates the bilingual report in Arabic and English with the treatment recommendation.                                                |
| <b>Supabase BaaS</b>    | Data           | The Backend-as-a-Service. Provides the PostgreSQL 15 database, the JWT and OAuth2 authentication, and an S3-compatible object storage.                                                       |
| <b>Disease Database</b> | Data           | A PostgreSQL knowledge base. For each disease class it stores the affected crop, the symptoms, the recommended treatment and the pesticide notes.                                            |
| <b>AirSim Simulator</b> | Simulation     | Microsoft AirSim running on Unreal Engine 4.27. Simulates drone flights and generates labelled synthetic crop images, used as a substitute for the real drone while it is not yet available. |
| <b>TFLite Engine</b>    | Edge (Planned) | A lightweight TensorFlow Lite model that will be deployed directly on the drone hardware. It is designed for offline edge inference.                                                         |

Table 3.5 – continued

| Component         | Layer          | Role                                                                                                                                     |
|-------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AI Chatbot</b> | Edge (Planned) | An optional conversational assistant based on the Gemini API. It will answer farmer queries about detected diseases in natural language. |

**Description of the connections.** Table 3.6 shows how the components talk to each other.

Table 3.6. Main connections between the components.

| From → To                          | Type              | Meaning                                                                                   |
|------------------------------------|-------------------|-------------------------------------------------------------------------------------------|
| Frontend / Mobile → FastAPI        | «HTTP REST»       | The client apps call the backend through standard HTTP requests.                          |
| FastAPI → AI Engine                | image input       | The backend forwards each uploaded image to the AI Engine for inference.                  |
| AI Engine → Supabase BaaS          | save result       | The full prediction (class, confidence, Grad-CAM URL) is saved in the database.           |
| Advisory System → Disease Database | lookup            | The advisory system reads the matching disease record to build the report.                |
| AirSim Simulator → FastAPI         | simulated images  | The simulator sends synthetic drone images into the same pipeline used by the real drone. |
| FastAPI → TFLite Engine            | .tflite (planned) | The trained model will be exported to TFLite and deployed on the drone.                   |
| Advisory System → AI Chatbot       | optional          | A future enhancement that lets the farmer ask follow-up questions.                        |

**Solid versus dashed.** The components with a solid border are already implemented (Frontend Dashboard, Mobile App, FastAPI Service, Field Map / GIS, AI Engine, Advisory System, Supabase BaaS, Disease Database, AirSim Simulator). The components with a dashed border are planned for a later phase (the TFLite Engine on the drone and the optional AI Chatbot). The same notation is used in the deployment diagram further below.

### 3.1.4 Sequence Diagram

#### General Scenario

The sequence diagram shows the dynamic behaviour of the system. It follows the messages that the different participants exchange during a complete mission, from the first login of the farmer to the final delivery of the results.

The participants of the diagram are the **User (Farmer)**, the **Web / Mobile App**, the **Database (Supabase)**, the **Admin**, the **Drone**, and the **AI System**. Time goes from top to bottom. Each arrow is a message; a solid arrow is a call, and a dashed arrow is a response. To keep the diagram readable, we split the full scenario into five phases: *Registration & Mission Submission*, *Admin Review*, *Drone Launch*, *Autonomous Mission*, and *Results Delivery*.

#### Diagram

Figure 3.4 presents the sequence diagram of a complete mission.

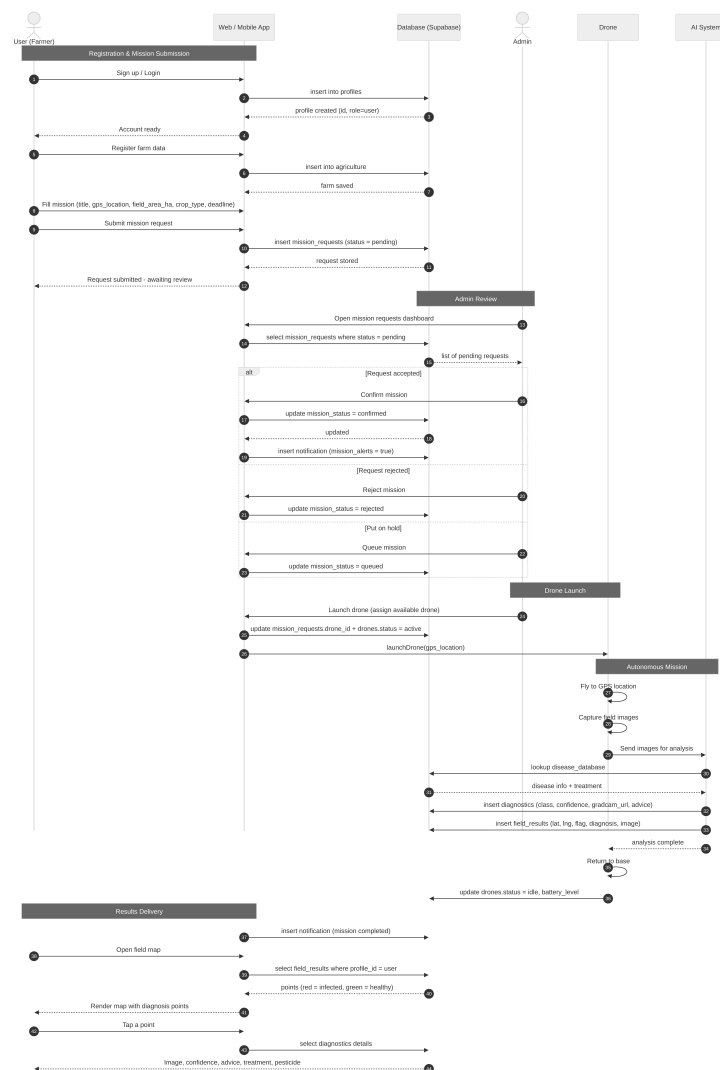


**Figure 3.4.** Sequence diagram of a complete drone mission.

### Detailed Description

**Phase 1 Registration and Mission Submission (messages 1-12).** The farmer first signs up or logs in through the app (1). The app inserts the new account into the `profiles` table of Supabase (2), the database confirms that the profile was created with the role `user` (3), and the app tells the farmer that the account is ready (4). Next, the farmer registers their farm data (5); the app stores it in the `agriculture` table (6–7). Finally, the farmer fills in the mission form with the title, GPS location, field area, crop type and deadline (8) and submits the request (9). The app inserts this request with the status `pending` (10), the database confirms (11), and the farmer is informed that the request is awaiting review (12).

**Phase 2 Admin Review (messages 13–23).** The admin opens the mission dashboard (13). The app reads all the pending requests from the database (14) and shows them to the admin (15). At this point the diagram uses an `alt` block, which represents a choice between three exclusive options. If the request is **accepted**, the admin confirms the mission (16), the status becomes `confirmed` (17-18), and a notification is created for the farmer (19). If the request is **rejected**, the status becomes `rejected` (20-21). If the request is **put on hold**, the status becomes `queued` (22-23).

**Phase 3 Drone Launch (messages 24-26).** Once a mission is confirmed, the admin launches a drone (24). The app updates the mission with the assigned `drone_id` and sets the drone status to `active` (25). Then the app sends the command `launchDrone(gps_location)` to the physical drone (26).

**Phase 4 Autonomous Mission (messages 27–36).** This phase runs without human action. The drone flies to the GPS location (27) and captures the field images (28). It sends the images to the AI System for analysis (29). The AI System looks up the matching disease in the `disease_database` (30) and receives the disease information and the recommended treatment (31). The results are then written to the database: a full record into `diagnostics` (32) and a light record into `field_results` (33). The AI signals that the analysis is complete (34), the drone returns to base (35), and the drone status is set back to `idle` with an updated battery level (36).

**Phase 5 Results Delivery (messages 37-44).** A notification is created to tell the farmer that the mission is completed (37). The farmer opens the field map (38). The app reads the field results that belong to this farmer (39) and gets the points coloured red (infected) or green (healthy) (40). The map is rendered with these diagnosis points (41). When the farmer taps a point (42), the app reads the full diagnostic details (43) and displays the image, the confidence, the advice, the treatment and the recommended pesticide (44).

**Reading the alt block.** The alt (alternative) fragment in Phase 2 is a standard UML notation. It means the system follows exactly one of the branches, depending on the admin decision.

### 3.1.5 Deployment Diagram

#### General Scenario

The deployment diagram is the most concrete of the UML views. After describing the data (class diagram), the user functions (use case diagram), the software modules (component diagram) and the dynamic behaviour (sequence diagram), this last view shows *where* the platform actually lives. It answers the question: *on which physical and cloud machines do our software components run?*

In the Skyline platform we use six deployment nodes. The first development machine is a **Dell Latitude 5490** laptop running Ubuntu, where a local Supabase stack runs inside Docker for backend testing. The main work happens on a **Desktop Workstation** running Windows 10. This machine hosts the FastAPI backend, the frontend dashboard, the AirSim simulator on Unreal Engine 4.27, and the REST testing tools. For the heavy computation, we use **Google Colab Pro** as a cloud node with an NVIDIA A100 GPU. This is where all the deep learning models were trained, and where the final EfficientNet-B4 model was exported as a .keras artifact. The production database lives on the **Supabase** cloud, which provides PostgreSQL 15, object storage and authentication for real users. The **End User** interacts with the system through a web browser or a mobile device. Finally, the **UAV Drone** is shown as a planned node, with a dashed border, because the physical drone is the next phase of the project: it will run the TFLite model on board and capture geotagged images with its camera and GPS.

#### Diagram

Figure 3.5 presents the deployment diagram of the platform.

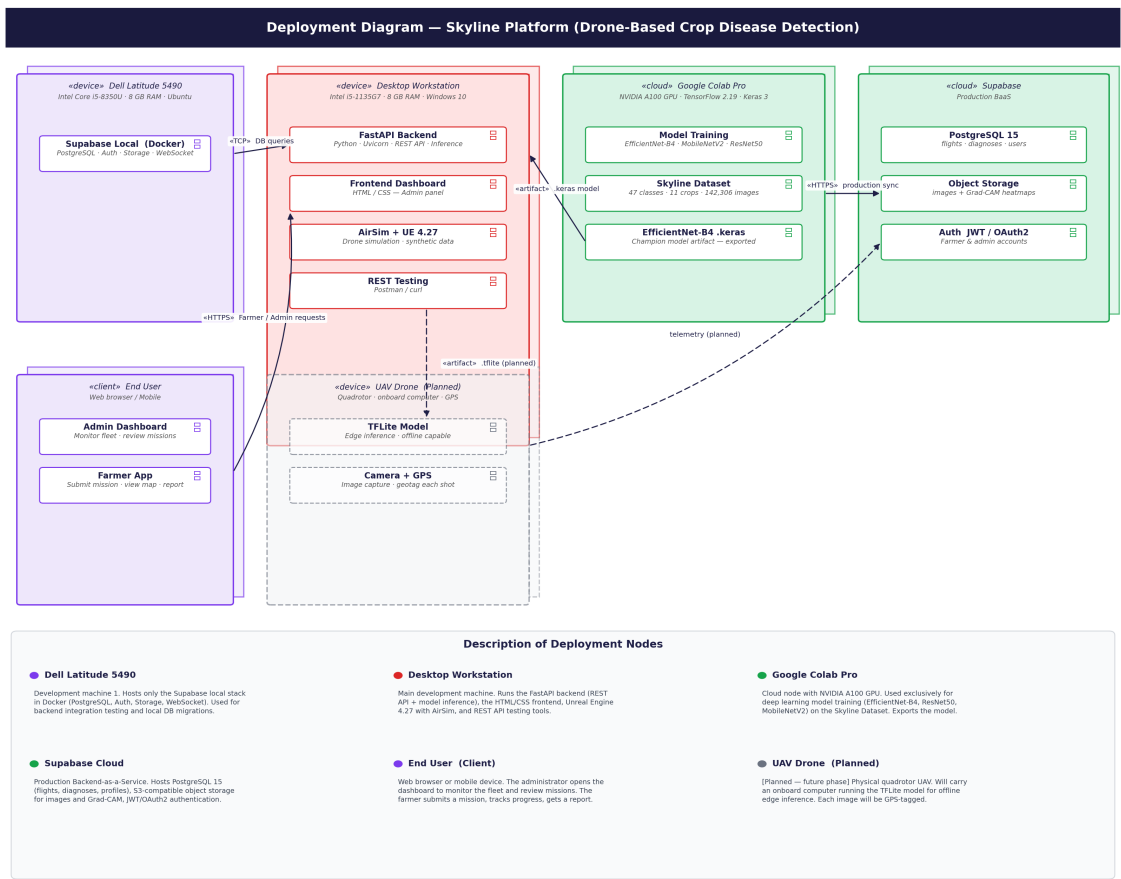


Figure 3.5. Deployment diagram of the Skyline platform.

### Detailed Description

**Description of the deployment nodes.** Table 3.7 lists the six nodes, their type and the artifacts they host.

Table 3.7. Description of the deployment nodes.

| Node               | Type     | Hosted artifacts and role                                                                                                                                                         |
|--------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Dell Latitude 5490 | «device» | A development laptop running Ubuntu. Hosts a local Supabase stack in Docker (PostgreSQL, Auth, Storage, WebSocket) for backend integration testing and local database migrations. |

Table 3.7 – continued

| Node                       | Type     | Hosted artifacts and role                                                                                                                                                                                                                |
|----------------------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Desktop Workstation</b> | «device» | The main development machine, running Windows 10. Hosts the FastAPI backend (REST API and model inference), the HTML and CSS frontend dashboard, the AirSim simulator on Unreal Engine 4.27, and the REST testing tools (Postman, curl). |
| <b>Google Colab Pro</b>    | «cloud»  | A cloud node with an NVIDIA A100 GPU, used only for deep learning. It hosts the training scripts, the Skyline Dataset (47 classes, 11 crops, 142,306 images) and the final exported EfficientNet-B4 model in the .keras format.          |
| <b>Supabase</b>            | «cloud»  | The production Backend-as-a-Service. Hosts the PostgreSQL 15 database (flights, diagnoses, users), the S3-compatible object storage for images and Grad-CAM heatmaps, and the JWT/OAuth2 authentication for farmer and admin accounts.   |
| <b>End User</b>            | «client» | A web browser or a mobile device. The administrator uses it to open the admin dashboard, monitor the fleet and review the missions. The farmer uses it to submit a mission, follow it on the map and receive the bilingual report.       |
| <b>UAV Drone (Planned)</b> | «device» | A physical quadrotor planned for the next phase of the project. It will carry an onboard computer running the TFLite model for offline edge inference, and a camera with GPS to capture and geotag each shot.                            |

**Description of the communication links.** Table 3.8 shows how the nodes are connected.

**Table 3.8.** Communication links between the deployment nodes.

| From → To                              | Protocol                | Purpose                                                                                                     |
|----------------------------------------|-------------------------|-------------------------------------------------------------------------------------------------------------|
| End User → Desktop Workstation         | «HTTPS»                 | The farmer and the admin send their HTTPS requests to the FastAPI backend.                                  |
| Desktop Workstation → Dell Latitude    | «TCP»                   | The backend sends database queries to the local Supabase stack during development and testing.              |
| Google Colab Pro → Desktop Workstation | «artifact»              | The trained EfficientNet-B4 model is exported as a .keras file and copied to the workstation for inference. |
| Google Colab Pro → Supabase            | «HTTPS»                 | Production data is synchronised between the training environment and the production database.               |
| Desktop Workstation → UAV Drone        | «artifact»<br>(planned) | The TFLite version of the model will be transferred to the drone for on-board edge inference.               |
| UAV Drone → Supabase                   | telemetry<br>(planned)  | Once active, the drone will stream telemetry and geotagged images directly to Supabase.                     |

**Two-phase deployment.** The current implementation already covers the solid-border nodes (the two development machines, Google Colab Pro, Supabase and the End User). The dashed-border node, the UAV drone, is planned for the next phase. This split between current and planned is important: it lets the reader see exactly which parts of the system are already running and which parts are the roadmap of the project.

## 3.2 General Architecture

### 3.2.1 Overview of the Project

After modeling the system with UML, we now present its general architecture. **Skyline** is a complete platform for drone-based crop disease detection. A drone (or, in the current phase, a simulator and manual uploads) captures aerial images of a field. These images are sent to

an AI engine that has been trained to recognise crop diseases. The engine returns a diagnosis with a confidence score and an explanation. This diagnosis is enriched with expert advice and treatment recommendations, then stored and shown to the farmer on an interactive map. To make this work reliably, we organise the platform into **six layers**. Each layer has a clear responsibility and talks only to the layers next to it.

3.2.2 Architecture Diagram

Figure 3.6 presents the general architecture of the Skyline platform, organised in six layers.

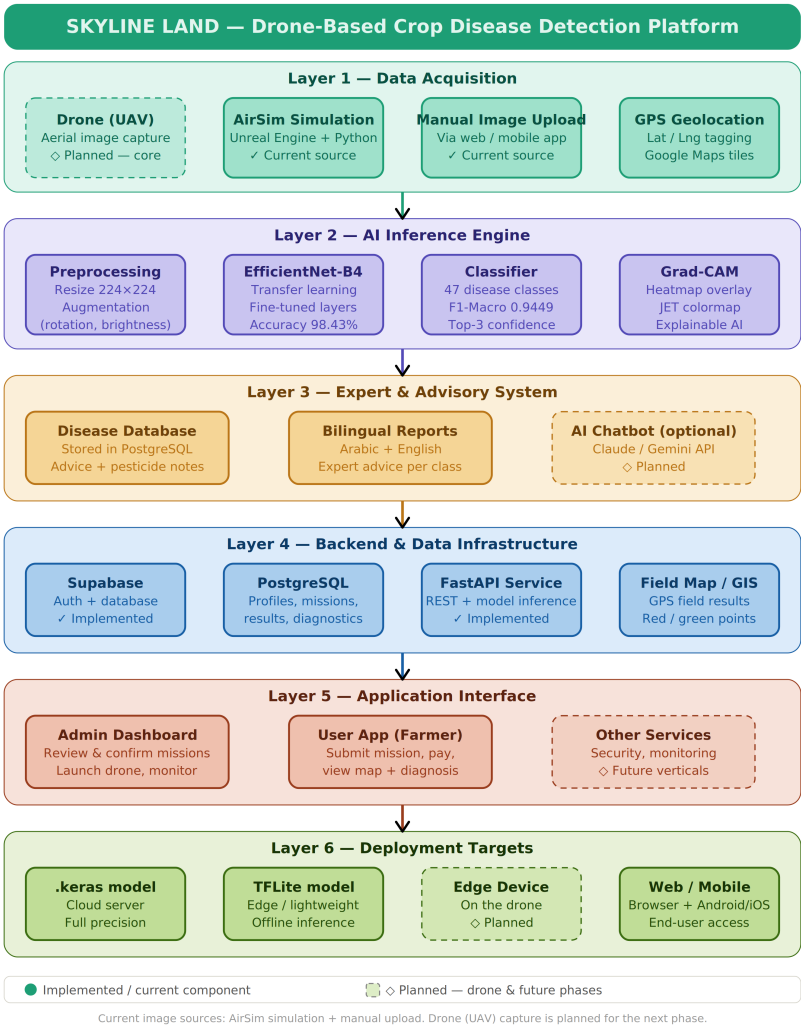


Figure 3.6. General layered architecture of the Skyline platform.

3.2.3 Detailed Description of the Layers

**Layer 1 Data Acquisition:** This is the entry point of the system, where the field images are obtained. There are several possible sources. The **Drone (UAV)** is the long-term goal; it is still planned for a later phase. For now, the two current sources are the **AirSim Simulation**

(built on Unreal Engine and Python) and the **Manual Image Upload** from the web or mobile app. In every case, a **GPS Geolocation** (latitude and longitude) is attached to each image.

**Layer 2 AI Inference Engine:** This is the brain of the platform. The pipeline has four steps. **Preprocessing** resizes each image to  $224 \times 224$  pixels and applies data augmentation. The model itself is an **EfficientNet-B4** network, trained with transfer learning and fine-tuned, reaching an accuracy of 98.43%. The **Classifier** can recognise 47 disease classes with a strong F1-macro score of 0.9449. Finally, **Grad-CAM** produces a heat map that highlights the part of the leaf where the disease was detected.

**Layer 3 Expert and Advisory System:** This layer turns the raw prediction into real, usable advice. The **Disease Database**, stored in PostgreSQL, contains for each class the expert advice and the pesticide notes. The **Bilingual Reports** present this advice in both Arabic and English. An **AI Chatbot** based on the Claude or Gemini APIs is planned as an optional feature.

**Layer 4 Backend and Data Infrastructure:** **Supabase** provides the authentication and the database, and it is already implemented. **PostgreSQL** stores all the structured data: profiles, missions, results and diagnostics. A **FastAPI Service** exposes a REST interface and runs the model inference. The **Field Map / GIS** component manages the geographic results and draws the red points (infected) and green points (healthy) on the map.

**Layer 5 Application Interface:** The **Admin Dashboard** lets the operator review and confirm missions, launch drones and monitor the fleet. The **User App (Farmer)** lets the farmer submit a mission, pay for it, and view the map and the diagnosis.

**Layer 6 Deployment Targets.** The AI model exists in two formats: a full-precision **.keras model** that runs on a cloud server, and a lightweight **TFLite model** for edge or offline inference. An **Edge Device** running directly on the drone is planned for the future. Finally, the platform is delivered to the end user through the **Web / Mobile** channel.

**Implemented versus planned.** The blocks with a solid border are already implemented (Supabase, FastAPI, the AI engine, the simulation and manual upload). The blocks with a dashed border are planned for the next phases (the real drone hardware, the edge device, the optional chatbot).

## Conclusion

In this chapter we presented the complete design of the Skyline platform. We started with the UML modeling and built five complementary views: the class diagram for the data structure, the use case diagram for the user functions, the component diagram for the software organisation, the sequence diagram for the dynamic behaviour of a full mission, and the deployment diagram for the physical setup of the platform. We then zoomed out to the general architecture and described the six layers of the system, from data acquisition to deployment.

# Chapter 4

## Experimentation

### Introduction

After we presented the design of the Skyline platform in the previous chapter, this chapter moves to the real work: the experimentation. Here we describe how we actually built, prepared and trained the artificial intelligence model that is the heart of the system — the model that looks at a crop image and tells the farmer which disease the plant has.

We did not reach our final model in one step. We tried several deep learning architectures, compared their results, and kept the best one. This is a normal and honest part of any machine learning project: the first model is rarely the best, and the comparison itself teaches us a lot about the data and the problem.

This chapter is organised around four ideas. First, we present the **dataset**: where the images come from, how we cleaned and prepared them, how we handled the class imbalance, and how the full data pipeline works. Second, we explain the **comparison between the three models** we trained (MobileNetV2, ResNet50 and EfficientNet-B4), with their complete measured results, and why one of them clearly stood out. Third, we focus on the **champion model**, EfficientNet-B4: its architecture, its training, how we prevented overfitting, and how we computed the evaluation metrics with our own code. Fourth, we present the **front-end and the application**: the user interface, the database, and the drone simulation that closes the loop from a farmer's request to a map of results.

All the work was done on **Google Colab Pro** with an NVIDIA A100 GPU, and the data was stored and read directly from **Google Drive**. This setup gave us enough computing power to train large models on more than one hundred thousand images without buying expensive hardware.

## 4.1 Dataset

### 4.1.1 General Presentation

The quality of any AI model depends first of all on the quality of its data. A model can only learn what the data shows it. For this reason, we spent a large part of the project building a clean and well-organised dataset, which we call the **Skyline Dataset**.

The Skyline Dataset brings together images of healthy and diseased plant leaves from **11 different crops**, organised into **47 classes**. Each class is one specific situation, for example “Tomato Late blight” or “Apple Scab”, and one class per crop is the healthy version. In total the dataset contains **142,306 high-resolution images**. Table 4.1 gives the main facts about the dataset and our working environment.

**Table 4.1.** Characteristics of the final Skyline Dataset and the work environment.

| Property             | Value                                                     |
|----------------------|-----------------------------------------------------------|
| Dataset name         | Skyline Dataset (built by us from several public sources) |
| Total images         | 142,306 high-resolution images                            |
| Number of crops      | 11 crops                                                  |
| Number of classes    | 47 classes (diseases + healthy)                           |
| Training set (80%)   | 113,825 images                                            |
| Validation set (20%) | 28,481 images                                             |
| Image resolution     | 224 × 224 pixels (RGB)                                    |
| Random seed          | 42 (for reproducibility)                                  |
| Champion model       | EfficientNet-B4 (transfer learning)                       |
| Training platform    | Google Colab Pro NVIDIA A100 GPU                          |
| Data storage         | Google Drive (images read directly from Drive)            |

Table 4.2 shows how the 47 classes are spread across the 11 crops. Tomato and Potato have the largest number of classes, because these two crops have many well-known diseases and a lot of available images.

**Table 4.2.** Distribution of the 47 classes over the 11 crops.

| Crop         | Classes   | Examples of diseases covered                                               |
|--------------|-----------|----------------------------------------------------------------------------|
| Tomato       | 11        | Early blight, Late blight, Yellow Leaf Curl Virus, Spider mites, Leaf spot |
| Potato       | 9         | Nematode, Fungi, Bacteria, Blight, Virus                                   |
| Corn         | 4         | Common rust, Gray leaf spot, Northern Leaf Blight                          |
| Grape        | 4         | Black rot, Black Measles, Leaf Spot                                        |
| Apple        | 4         | Scab, Cedar rust, Black rot                                                |
| Citrus       | 4         | Greening (HLB), Canker, Black spot                                         |
| Wheat        | 3         | Septoria, Stripe rust                                                      |
| Cherry       | 2         | Powdery mildew                                                             |
| Peach        | 2         | Bacterial spot                                                             |
| Bell Pepper  | 2         | Bacterial spot                                                             |
| Strawberry   | 2         | Leaf scorch                                                                |
| <b>Total</b> | <b>47</b> | <b>11 crops, healthy + diseased classes</b>                                |

### 4.1.2 Data Collection

We did not use a single ready-made dataset. Instead, we collected our images from **several public sources** (mostly on Kaggle) and merged them into one large dataset. We chose this approach for one important reason: no single public dataset covered all the 11 crops and 47 classes that we wanted. By combining many sources, we obtained a much wider and more realistic collection of crops and diseases.

The collection work was not only about downloading. The raw sources used different folder structures and different naming styles, so a big part of the effort was cleaning and re-organising the data into one clear structure of the form Crop\_\_Disease. We describe the main sources one by one below.

1. **PlantVillage Dataset** [65]. The largest and most famous public dataset for plant disease classification. It was the base of our collection and provided most of the crops (Tomato, Potato, Corn, Grape, Apple).

<https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset> (accessed 21/05/2026)

2. **Advanced Potato Disease Dataset (EfficientNetB3)**. Used to add more Potato im-

ages and cover the many Potato classes (Nematode, Fungi, Bacteria).

<https://www.kaggle.com/code/marianadeem755/advanced-potato-disease-detection-efficientnetb3/input> (accessed 21/05/2026)

3. **Tomato Leaf Disease (94% Accuracy).** A focused Tomato source that gave us extra high-quality Tomato images.

<https://www.kaggle.com/code/samanfatima7/tomato-leaf-disease-94-accuracy/input> (accessed 21/05/2026)

4. **Tomato Leaf Disease Dataset (Segmented).** Segmented Tomato leaves with the background removed, which helped the model focus on the leaf itself.

<https://www.kaggle.com/datasets/ahmadzargar/tomato-leaf-disease-dataset-segmented> (accessed 21/05/2026)

5. **Corn / Maize Leaf Disease Dataset.** A dedicated Corn source used to complete the four Corn classes.

<https://www.kaggle.com/code/ashishk1331/corn-or-maize-leaf-disease-dataset> (accessed 21/05/2026)

6. **Potato Leaf Disease Dataset.** An additional Potato source that improved the variety of the Potato classes.

<https://www.kaggle.com/datasets/warcoder/potato-leaf-disease-dataset> (accessed 21/05/2026)

By merging and cleaning all of these sources, we reached our final dataset of **11 crops and 47 classes**, which we saved under the name `skyline_dataset1`.

**Why combine many sources?** A model that learns from only one dataset can become too comfortable with that dataset and fail on new images. By mixing images from many sources, taken in different conditions, we forced the model to learn the real signs of each disease instead of memorising one specific style of photo. This makes the model more robust for real drone images later.

### 4.1.3 Class Imbalance and Class Weights

When we counted the images of each class, we found a **strong class imbalance**. The most frequent class, Tomato Yellow Leaf Curl Virus, had **8,877 images**, while the rarest, Potato Nematode, had only **68 images**. The ratio between the two is about **131**, which is extremely large.

This is a serious problem. If we do nothing, the model learns mostly the large classes and almost ignores the rare ones, because the rare classes are “invisible” next to the big ones. The model could then reach a high accuracy simply by always predicting the common classes — a misleading result that is useless for a farmer whose crop has a rare disease.

To solve this, we used **class weights**. The idea is to give each class an importance during training that depends on how rare it is: the rarer the class, the higher its weight. The weight of class  $c$  is computed as

$$w_c = \frac{N_{\text{total}}}{C \times n_c}, \quad (4.1)$$

where  $w_c$  is the weight of class  $c$ ,  $N_{\text{total}}$  is the total number of images,  $C$  is the number of classes ( $C = 47$ ), and  $n_c$  is the number of images of class  $c$ . With this rule, the rare Potato Nematode class received a weight of **44.53** (it counts about 45 times more during training), while the dominant Tomato YLCV class received a weight of only **0.34** (it counts less, so it cannot dominate the learning). This correction was essential to make the model fair to all 47 classes.

#### 4.1.4 Preprocessing

Before any image entered a model, it passed through a fixed **preprocessing** pipeline. It is important to note that the normalisation step differs slightly between the three models, because each architecture was pre-trained with a specific input convention.

1. **Resize to  $224 \times 224$  pixels.** This is the standard input size required by all three models.
2. **Convert to RGB** (remove the alpha channel if the image has one), so every image has exactly three colour channels.
3. **Normalise the pixel values.** For **EfficientNet-B4** and **MobileNetV2**, pixel values are scaled from  $[0, 255]$  to  $[0, 1]$ . For **ResNet50**, the standard ImageNet mean/std subtraction is applied (mean =  $[0.485, 0.456, 0.406]$ , std =  $[0.229, 0.224, 0.225]$ ), as required by its pre-trained weights. Using the wrong normalisation for a given architecture would corrupt the feature maps extracted by the backbone, so this distinction matters in practice.

#### 4.1.5 Data Augmentation

A model trained on a fixed set of images can learn them “by heart”. To avoid this and to make the model more robust, we used **data augmentation**. Data augmentation creates new, slightly modified versions of the training images on the fly, so the model sees a different version of each image at every epoch and is forced to learn the real disease signs instead of memorising one exact photo.

We applied **six transformations**, and we chose them on purpose to **simulate real drone flight conditions**. A drone never takes two photos in exactly the same way: the angle changes, the altitude changes, the light changes. Our augmentations reproduce these real situations.

Very importantly, the augmentations were applied **only to the training images, never to the validation images**, so that the validation score stays honest and there is no data leakage. Table 4.3 lists the six transformations and the drone reason behind each one.

**Table 4.3.** The six data-augmentation transformations applied to the training set only.

| Transformation      | Value           | Reason (drone context)                  |
|---------------------|-----------------|-----------------------------------------|
| Random flip (H + V) | Both axes       | Different drone viewing angles          |
| Random rotation     | $\pm 30^\circ$  | Drone rotation during flight            |
| Random zoom         | $\pm 15\%$      | Varying drone altitude above the crops  |
| Random brightness   | $\pm 20\%$      | Natural lighting changes (sun / shadow) |
| Random contrast     | $\pm 20\%$      | Cloud cover and fog conditions          |
| Gaussian noise      | $\sigma = 0.01$ | Drone camera sensor noise               |

#### 4.1.6 Data Splitting

The dataset was not pre-split into training and validation parts, so we performed a manual **80/20 split** ourselves. We used a fixed random seed (**seed** = 42) so that the split is exactly the same every time the code runs, which makes our experiment reproducible. The **same split was used for all three models**, which is essential for a fair comparison otherwise one model could look better only because it was tested on easier images.

We strictly enforced the **golden rule of zero data leakage**: the model **never** saw any validation image during training. The training set (113,825 images) was used only for learning, and the validation set (28,481 images) was used only to measure performance on images the model had never seen.

#### 4.1.7 Full Data Pipeline

All the steps above can be seen as one connected pipeline that turns raw, messy images into clean training-ready batches. Figure 4.1 shows this pipeline from end to end: from the six Kaggle sources, through merging and cleaning, to the final augmented batches that feed the model.



**Figure 4.1.** The full data pipeline of the Skyline Dataset, from the raw sources to the augmented training batches.

The pipeline reads each step in order: (1) the **six Kaggle sources** are gathered; (2) they are **merged** into one folder named `skyline_dataset1`; (3) the labels are **cleaned** into 47 clear classes; (4) the data is **split 80/20** with seed 42; (5) every image is **resized to  $224 \times 224$  and normalised** (with the model-specific rule described above); and (6) the six **augmentations** are applied to the training images only. The result of this pipeline is the exact same input format for every model, which is what makes the later comparison trustworthy.

### 4.1.8 Prediction Pipeline (Inference)

At prediction time, a new image follows the **same path** as during training: it is resized to  $224 \times 224$  and normalised, then passed to the trained model. The model outputs a probability for each of the 47 classes through a **softmax** function (explained in Section 4.3.2). The class with the highest probability is the predicted disease, and the **top-3 probabilities** are kept to show how confident the model is. Finally, the predicted class is matched with the disease database to return the bilingual advice and treatment for the farmer.

## 4.2 Model Training and Comparison

### 4.2.1 Transfer Learning

We trained **three convolutional neural network (CNN) architectures**, and all of them used **transfer learning** from weights pre-trained on the large ImageNet database [66]. Transfer learning means we do not start from zero: the model already knows how to recognise basic visual features such as edges, textures and colours, learned from 1.2 million ImageNet images. We then adapt this general knowledge to plant disease images, which needs much less data and time than training from scratch. The same 47-class dataset and the same validation split were used for all three models, so the comparison is completely fair.

### 4.2.2 The Three Tested Models

### MobileNetV2 Lightweight Baseline

MobileNetV2 [67] was trained first, as a lightweight **baseline**. It is a small and fast architecture designed for mobile phones and edge devices, which made it attractive for a system that may one day run directly on a drone. Its backbone produces a feature map of shape (7, 7, 1280), which is then reduced to a 1280-dimensional vector by a global average pooling layer. Crucially, the classification head of MobileNetV2 goes directly from this vector through a single dropout layer to the final Dense(47) output it does not include the intermediate Dense(512) and Dense(256) layers that we added in ResNet50 and EfficientNet-B4. This simpler head, combined with the smaller backbone capacity ( $\approx 2.26\text{M}$  backbone parameters, 5.49M total), limits how much the model can learn from a complex 47-class disease problem.

The complete measured results on the 28,481-image validation set are shown in Table 4.4.

**Table 4.4.** Full evaluation results of MobileNetV2 on the 47-class validation set.

| Metric    | Target      | Achieved | Status        | Remark                                   |
|-----------|-------------|----------|---------------|------------------------------------------|
| Accuracy  | $\geq 90\%$ | 87.54%   | <b>Below</b>  | Did not reach the project target         |
| AUC-Macro | $\geq 0.95$ | 0.9963   | <b>Passed</b> | Class separation is acceptable           |
| F1-Macro  | $\geq 0.85$ | 0.8012   | <b>Below</b>  | Struggles with rare disease classes      |
| Precision | $\geq 0.85$ | 0.7818   | <b>Below</b>  | High false-alarm rate on rare classes    |
| Recall    | $\geq 0.85$ | 0.8354   | <b>Below</b>  | Misses a significant share of diseases   |
| MCC       | $\geq 0.80$ | 0.8717   | <b>Passed</b> | Overall balance is moderate              |
| Loss      | $< 0.30$    | 0.4230   | <b>High</b>   | Model is still uncertain on many classes |

**A concrete failure case.** The limits of MobileNetV2 are not just numbers on a table. When we tested the trained model on a real image of *Apple Scab* a clearly diseased leaf with visible lesions the model classified it as “*Corn Healthy*” with only 18.07% confidence. The correct class did not appear anywhere in the top-3 predictions. This shows in the most direct way that a small architecture without a deep classification head cannot reliably distinguish 47 classes across 11 crops, especially when some classes have fewer than 100 training images. MobileNetV2 therefore served as a useful and honest reference point that justified the move to heavier architectures.

### ResNet50 Second Model

ResNet50 [68] was trained next. It is a 50-layer **residual network** whose “skip connections” solve the vanishing-gradient problem and allow very deep networks to train without losing performance. Unlike MobileNetV2, ResNet50 was given a richer classification head: after the global average pooling layer (output shape (2048,)), since the backbone produces (7, 7, 2048) feature maps, we added a BatchNormalization layer, then a Dense(512, ReLU), a Dropout(0.4), a Dense(256, ReLU), a Dropout(0.3), and finally the Dense(47, Softmax) output. This deeper head gives the model more capacity to specialise on agricultural disease patterns on top of the ImageNet features.

We used a **two-phase transfer-learning strategy**. In Phase 1, the entire ResNet50 backbone was frozen, and only the 1,196,591 parameters of the classification head (4.8% of the total 24,788,399 parameters) were trained with a learning rate of  $1e-3$ , for up to 20 epochs. This phase quickly reached an intermediate performance of **val\_Acc = 93.65%** and **val\_F1 = 0.8786**. In Phase 2, the last **50 layers** of the backbone were unfrozen and fine-tuned with a very small learning rate of  $1e-5$ , for up to 30 additional epochs. This gentle fine-tuning adapts the deep visual features to drone-based agricultural imagery without destroying the ImageNet knowledge. The preprocessing for ResNet50 uses **ImageNet mean/std normalization** (not the simple [0, 1] scaling used for EfficientNet-B4), as required by its pre-trained weights.

ResNet50 clearly exceeded all project targets, as shown in Table 4.5.

**Table 4.5.** Full evaluation results of ResNet50 on the 47-class validation set.

| Metric    | Target | Achieved       | Margin  | Status   |
|-----------|--------|----------------|---------|----------|
| Accuracy  | 90%    | 95.78%         | +5.78%  | Exceeded |
| AUC-Macro | 0.95   | 0.9995         | +0.0495 | Exceeded |
| F1-Macro  | 0.85   | 0.9071         | +0.0571 | Exceeded |
| Precision | 0.85   | 0.8916         | +0.0416 | Exceeded |
| Recall    | 0.85   | 0.9425         | +0.0925 | Exceeded |
| MCC       | 0.80   | 0.9565         | +0.1565 | Exceeded |
| Loss      | < 0.30 | $\approx 0.19$ | —       | Good     |

Despite these strong global results, **8 out of 47 classes** did not reach  $F1 \geq 0.85$  on ResNet50. The most critical case is Potato Nematode ( $F1 = 0.239$ ), which has only 68 total images and 14 validation images a statistically impossible situation for any model to handle reliably. The model was exported as a SavedModel (201 MB) and as an INT8-quantised TFLite model (25.1 MB) for edge deployment on drone-mounted hardware.

### EfficientNet-B4 Final Champion Model

EfficientNet-B4 was the third model. EfficientNet [69] uses **compound scaling**, which grows the width, the depth and the input resolution of the network together by a fixed ratio. This gives a better balance between accuracy and computational cost than older architectures. It was trained with the same two-phase strategy on the same split, and it achieved the best results of all three: **98.43% accuracy**, a macro-F1 of **0.9449**, an MCC of **0.9838** and an AUC of **0.9998**.

### 4.2.3 Common Training Configuration

To keep the comparison fair, ResNet50 and EfficientNet-B4 shared the same training configuration, summarised in Table 4.6.

**Table 4.6.** Shared training configuration of the two main models.

| Component             | Configuration                       | Purpose                              |
|-----------------------|-------------------------------------|--------------------------------------|
| Optimiser             | Adam [70]                           | Adaptive learning rate per parameter |
| Loss function         | Sparse Categorical<br>Cross-Entropy | Multi-class integer labels           |
| Phase 1 learning rate | 1e−3 (head only)                    | Train the new head                   |
| Phase 2 learning rate | 1e−5 (fine-tuning)                  | Adapt deep features safely           |
| ReduceLROnPlateau     | factor 0.3, patience 3              | Lower the LR when validation stalls  |
| EarlyStopping         | patience 8 (monitor val. F1)        | Stop when there is no more gain      |
| ModelCheckpoint       | save best by val. F1-macro          | Keep the best model automatically    |

#### 4.2.4 Model Comparison

The three models were compared on the **same** 47-class validation set (28,481 images). We used several complementary metrics on purpose, so that the comparison would not depend on accuracy alone:

- **Accuracy** a general overview of correct predictions.
- **F1-Macro** treats every class equally, so it reflects the rare diseases (the most important metric for us).
- **MCC** a single balanced score that is robust to imbalance.
- **AUC** the ability to separate one class from all the others.

Table 4.7 gives the complete side-by-side comparison of all three models. The progression is unambiguous: MobileNetV2 fails to reach the accuracy target and produces the weakest F1-Macro (0.8012); ResNet50 clears all targets comfortably; and EfficientNet-B4 is the strongest on every single metric.

**Table 4.7.** Complete comparison of the three models on all metrics (same validation set).

| Metric             | Target | MobileNetV2 | ResNet50  | EffNet-B4        | Best |
|--------------------|--------|-------------|-----------|------------------|------|
| Accuracy           | 90%    | 87.54%      | 95.78%    | <b>98.43%</b>    | B4   |
| AUC-Macro          | 0.95   | 0.9963      | 0.9995    | <b>0.9998</b>    | B4   |
| F1-Macro           | 0.85   | 0.8012      | 0.9071    | <b>0.9449</b>    | B4   |
| Precision          | 0.85   | 0.7818      | 0.8916    | —                | —    |
| Recall             | 0.85   | 0.8354      | 0.9425    | —                | —    |
| MCC                | 0.80   | 0.8717      | 0.9565    | <b>0.9838</b>    | B4   |
| Loss               | < 0.30 | 0.4230      | ≈0.19     | ≈0.05            | B4   |
| Weak cls. (< 0.85) | —      | Multiple    | 8 classes | <b>5 classes</b> | B4   |
| TFLite size        | —      | ≈21 MB      | 25.1 MB   | <b>19.1 MB</b>   | B4   |

Precision and Recall were not separately tracked in the EfficientNet-B4 evaluation log. F1-Macro, which is their harmonic mean averaged over all 47 classes, is the figure that matters: at 0.9449 it is the highest of the three models and well above the 0.85 target.

Figure 4.2 compares the validation accuracy of the three models visually. The progression is clear: MobileNetV2 stays just under the 90% target at 87.54%, ResNet50 improves to 95.78%, and EfficientNet-B4 is the highest at 98.43%.

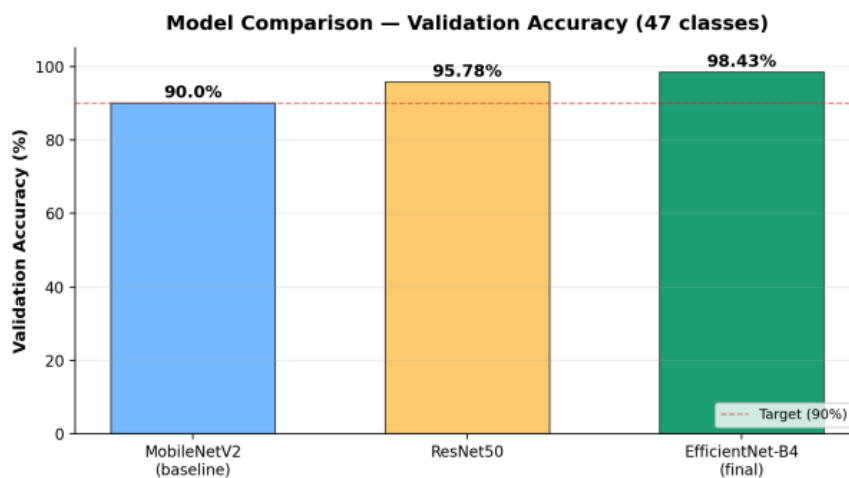
**Figure 4.2.** Validation accuracy of the three models. MobileNetV2 is the lightweight baseline; ResNet50 and EfficientNet-B4 are the two strong candidates.

Figure 4.3 then compares the two strongest models (ResNet50 and EfficientNet-B4) on all four metrics at once. EfficientNet-B4 is ahead everywhere.

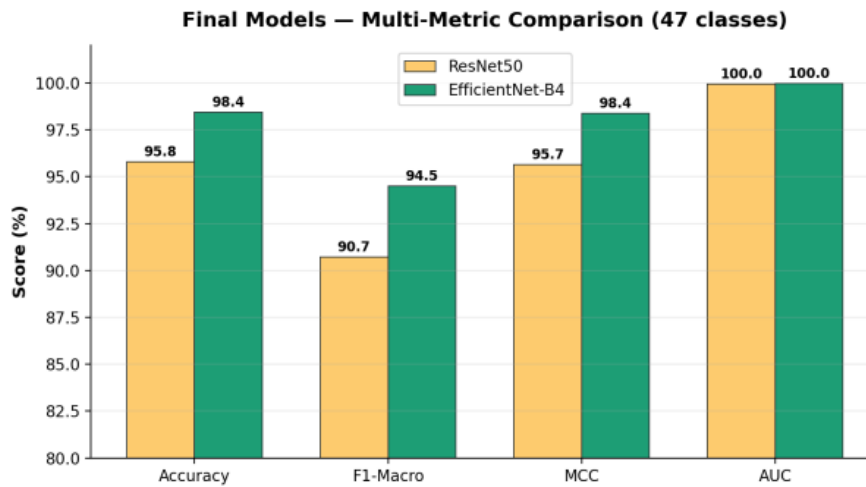


**Figure 4.3.** Multi-metric comparison of ResNet50 and EfficientNet-B4 on the 47-class validation set.

Table 4.8 gives the exact values for the two strongest models. EfficientNet-B4 is ahead on every metric, and its advantage is largest on F1-Macro (+3.78 points), which is exactly the metric that reflects performance on the rare classes.

**Table 4.8.** Head-to-head comparison of the two final models. Both exceed all project targets.

| Metric   | Target | ResNet50 | EffNet-B4     | Best            |
|----------|--------|----------|---------------|-----------------|
| Accuracy | 90%    | 95.78%   | <b>98.43%</b> | EfficientNet-B4 |
| F1-Macro | 0.85   | 0.9071   | <b>0.9449</b> | EfficientNet-B4 |
| MCC      | 0.80   | 0.9565   | <b>0.9838</b> | EfficientNet-B4 |
| AUC      | 0.95   | 0.9995   | <b>0.9998</b> | EfficientNet-B4 |

**Model selection.** Based on this comparison, **EfficientNet-B4** was selected as the **final model** for the Skyline platform. It gives the highest score on every metric while keeping a reasonable model size (about 71 MB for the full model, and 19.1 MB after conversion to TFLite for edge use smaller than ResNet50’s 25.1 MB TFLite). In the project files, the final exported version of this model is named “SkylineLAND\_P2”; both names refer to the same EfficientNet-B4 model.

## 4.3 The Champion Model — EfficientNet-B4

In this section we describe the champion model in detail: its architecture, the softmax activation that produces the final decision, the techniques we used to prevent overfitting, and very importantly how we computed the evaluation metrics with our own code.

### 4.3.1 Model Architecture

Our model has two parts. The first part is the **EfficientNet-B4 backbone**, which extracts features from the image. The second part is our own **classification head**, a small set of layers that we added on top to turn those features into a decision over the 47 classes. Table 4.9 shows the full structure.

**Table 4.9.** Full architecture of the champion model (EfficientNet-B4 + custom head).

| Layer / Component        | Output shape  | Role                                                          |
|--------------------------|---------------|---------------------------------------------------------------|
| Input Layer              | (224, 224, 3) | The crop image                                                |
| EfficientNet-B4 backbone | (7, 7, 1792)  | Feature extraction (ImageNet weights, $\approx$ 17.6M params) |
| GlobalAveragePooling2D   | (1792, )      | Turns feature maps into one vector                            |
| BatchNormalization       | (1792, )      | Stabilises the training                                       |
| Dense(512, ReLU)         | (512, )       | First layer of our custom head                                |
| Dropout(0.4)             | (512, )       | Prevents overfitting                                          |
| Dense(256, ReLU)         | (256, )       | Compresses the features                                       |
| Dropout(0.3)             | (256, )       | Extra regularisation                                          |
| Dense(47, Softmax)       | (47, )        | Final probabilities for the 47 classes                        |

**Parameters.** The full model has about **18.7 million parameters**. In the first training phase, the backbone was frozen, so only about **1.07 million parameters (5.7%)** were trained. The remaining 94.3% were kept frozen to protect the ImageNet knowledge. In Phase 2, the last **60 layers** of the EfficientNet-B4 backbone were unfrozen more than ResNet50’s 50 layers allowing deeper adaptation of the learned features to drone-based agricultural imagery.

### 4.3.2 The Softmax Activation Function

The last layer of our model uses the **softmax** activation function. This function is the key that turns the raw outputs of the network into a clear, readable answer. Without it, the network would only give us 47 unscaled numbers (called *logits*) that are hard to interpret.

The softmax function takes the 47 logits  $z_1, z_2, \dots, z_{47}$  and turns them into 47 **probabilities** that are all positive and add up exactly to 1. For class  $i$ , the probability is computed as

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{47} e^{z_j}}. \quad (4.2)$$

This has two clear advantages. First, the output is now a real **probability distribution**: we can say, for example, “the model is 92% sure this is Tomato Late blight”. Second, because the probabilities add up to 1, we can simply take the highest one as the predicted disease, and keep the top-3 probabilities to show the confidence of the model. Softmax is the standard and correct choice for a multi-class problem like ours, where each image belongs to exactly one of the 47 classes.

### 4.3.3 Two-Phase Training and Convergence

As explained above, EfficientNet-B4 was trained in two phases: first the head alone (backbone frozen, learning rate  $1e-3$ ), then a gentle fine-tuning of the last layers (learning rate  $1e-5$ ). Figure 4.4 shows the training curves of the Phase 2 fine-tuning.

## Phase 2: Fine-tuning Performance Analysis (Skyline LAND)

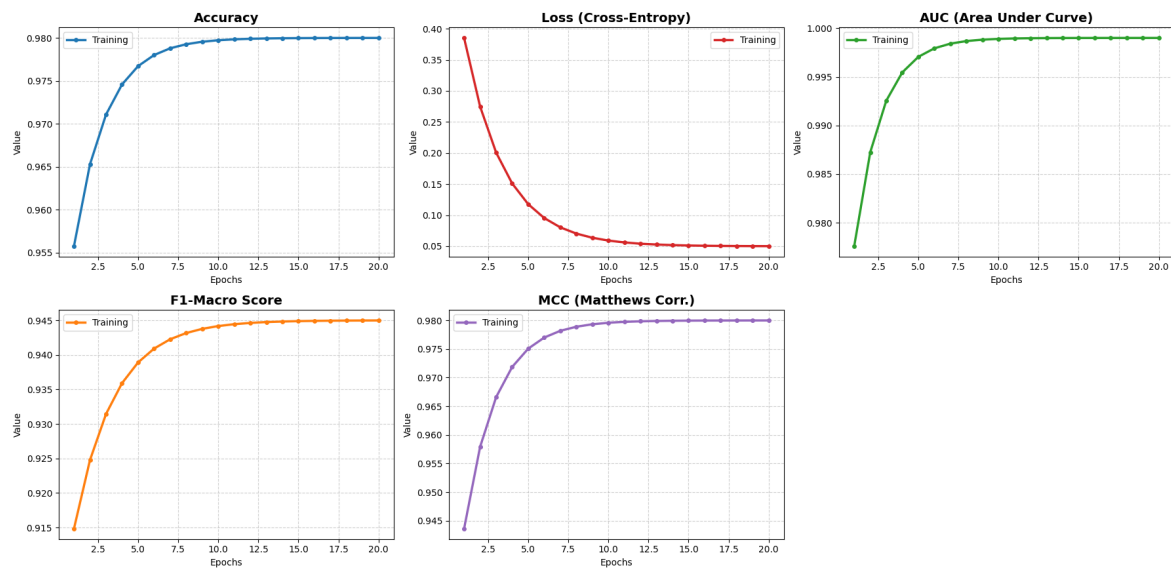


**Figure 4.4.** Training curves of EfficientNet-B4 during Phase 2 fine-tuning. The plot shows the loss function (cross-entropy) together with the four evaluation metrics (accuracy, AUC, macro-F1 and MCC). All curves converge smoothly.

The curves tell a healthy story. The accuracy rises quickly and then becomes flat near 0.98; the loss drops smoothly from about 0.39 down to about 0.05 and stays there; and the AUC, F1-Macro and MCC all climb and then settle at high values. There is no sudden jump and no curve that goes up and then down again. This smooth, stable behaviour is the typical sign of a model that learns well and does **not** overfit.

### 4.3.4 Preventing Overfitting

A deep model with millions of parameters can easily **overfit**: it may memorise the training images, including their noise, instead of learning the general patterns of each disease. An overfitted model shows a very high accuracy on the training set but a poor accuracy on new images, which would make it useless in a real field.

To avoid this, we combined several techniques during training. They are summarised in Table 4.10.

**Table 4.10.** Techniques used to prevent overfitting.

| Technique                            | How it reduces overfitting                                                                                                                        |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Dropout (0.4 and 0.3) [71]           | Randomly turns off a fraction of the neurons during training, so the model cannot rely on a few specific neurons.                                 |
| Data augmentation                    | Generates varied versions of each image (rotation, brightness, noise), so the model sees the data differently each time and does not memorise it. |
| Transfer learning with frozen layers | The frozen ImageNet layers keep general visual knowledge and do not drift towards memorising the small dataset (Phase 1).                         |
| Small fine-tuning learning rate      | A learning rate of $1e-5$ in Phase 2 changes the weights gently, which prevents the model from overfitting the training set.                      |
| EarlyStopping                        | Stops the training automatically when the validation score stops improving, before the model starts to memorise the training data.                |
| BatchNormalization [72]              | Normalises the inputs between layers, which stabilises training and adds a light regularisation effect.                                           |

The best evidence that the model did not overfit is given by the training curves in Figure 4.4: the metrics rise smoothly and then stay stable, without the large gap between training and validation that is the typical sign of overfitting.

### 4.3.5 Evaluation Metrics and the Custom Metric Code

To judge the model fairly, we did not rely on accuracy alone, because accuracy can be misleading when some classes have very few images. We used four complementary metrics, summarised in Table 4.11.

**Table 4.11.** Evaluation metrics used for the champion model.

| Metric   | What it measures                                             | Achieved |
|----------|--------------------------------------------------------------|----------|
| Accuracy | Share of correct predictions over all images                 | 98.43%   |
| F1-Macro | Balance of precision and recall, equal weight to every class | 0.9449   |
| MCC      | Macro-averaged binary MCC (One-vs-Rest over the 47 classes)  | 0.9838   |
| AUC      | Ability to separate each class from all the others           | 0.9998   |

### Why We Wrote Our Own Metric Code

This part was a real technical problem in our project, and we explain it honestly because it shows that our results are correct and not the product of a hidden bug.

We used **TensorFlow 2.19 with Keras 3** [73]. In this version, the built-in metric `tf.keras.metrics.AUC` **does not work correctly for a multi-class problem with 47 classes**. During training it raised errors linked to *SymbolicTensor* indexing (an `AttributeError` on `assign_add`). In simple words, the standard Keras metric could not follow our 47-class model.

To make sure the model could train and be measured **without any hidden error**, we **wrote our own metric classes by hand**. We created three custom classes `MacroF1Score`, `MulticlassMCC` and `MacroAUC` that inherit from `tf.keras.metrics.Metric`. They use a stable **One-Hot Encoding** approach that is fully compatible with the graph execution of TensorFlow 2.19. Before training the real model, we ran a small test to confirm that all three metrics return correct values.

### What Each Custom Metric Does and Why

We did not invent new mathematics. We re-implemented well-known evaluation formulas, only in a form that Keras 3 could run safely. The role of each class is the following:

- `MacroF1Score` keeps three counters per class (true positives  $TP$ , false positives  $FP$ , false negatives  $FN$ ), computes the F1 score of each class, and returns the average over the 47 classes. This is our most important metric because it gives every class the same weight.
- `MulticlassMCC` keeps four counters per class ( $TP$ ,  $TN$ ,  $FP$ ,  $FN$ ), computes the binary Matthews Correlation Coefficient of each class (One-vs-Rest), and returns the

average over the 47 classes. This **macro-averaged binary MCC** is explained in detail in Section 4.3.5 below.

- MacroAUC collects the predictions and the true labels, builds the ROC curve for each class (one class against all the others), and averages the area under all 47 curves. This replaces the broken built-in AUC.

The trick that makes all three stable is the **One-Hot Encoding**. Instead of indexing tensors directly (which is what broke in TF 2.19), we turn the labels and predictions into one-hot vectors and then use simple element-wise multiplications and sums to count  $TP$ ,  $TN$ ,  $FP$  and  $FN$  for every class at once. These operations are safe in graph mode, so no error appears during training.

### On Our Choice of MCC: Macro-Averaged Binary MCC

We want to be fully transparent about one important point. There are **two different definitions of MCC** in the multi-class setting, and we deliberately chose the one that best fits our project.

- **Standard multiclass MCC (Gorodkin, 2004) [62]**. A single value computed from the full  $C \times C$  confusion matrix using

$$\text{MCC} = \frac{c \cdot s - \sum_k p_k t_k}{\sqrt{(s^2 - \sum_k p_k^2)(s^2 - \sum_k t_k^2)}},$$

where  $c$  is the total number of correctly classified samples,  $s$  is the total number of samples,  $t_k$  is the true count for class  $k$  and  $p_k$  is the predicted count for class  $k$ . This is the form computed by `sklearn.metrics.matthews_corrcoef` when the labels are multi-class.

- **Macro-averaged binary MCC (our choice)**. Each of the 47 classes is treated as a binary One-vs-Rest problem, and the per-class binary MCC values are averaged with equal weight:

$$\text{MCC}_{\text{macro}} = \frac{1}{C} \sum_{k=1}^C \frac{TP_k TN_k - FP_k FN_k}{\sqrt{(TP_k + FP_k)(TP_k + FN_k)(TN_k + FP_k)(TN_k + FN_k)}}.$$

The two formulas are **not numerically equivalent**, especially under a strong class imbalance like ours (ratio of 131 between the largest and the rarest class). We chose the macro-averaged binary form for three reasons that are specific to our agricultural problem:

1. **Equal weight for rare diseases.** The standard multiclass MCC is driven by the total counts in the confusion matrix, which are dominated by the large classes. The

macro version gives the same weight to every class exactly what we want for a farmer whose crop might suffer from a rare disease such as Potato Nematode (68 images) or Citrus Healthy (87 images). This is the same rationale that motivated our choice of F1-Macro over Micro-F1.

2. **Consistency with our other metrics.** F1-Macro and AUC-Macro are both computed One-vs-Rest with macro averaging. Using a macro-averaged binary MCC keeps the three metrics on the same averaging convention, which makes them comparable and easier to interpret together.
3. **Stability in graph mode.** The macro-averaged binary form can be computed from four simple per-class counters ( $TP$ ,  $TN$ ,  $FP$ ,  $FN$ ) updated with one-hot tensors, which is exactly the operation that runs safely in the Keras 3 / TensorFlow 2.19 graph mode where the built-in AUC failed.

For the same reason, we do **not** claim that our reported MCC of 0.9838 is identical to what `sklearn.metrics.matthews_corrcoef` would return on the same predictions: it is a related but different macro-averaged version that we selected on purpose. This is a deliberate, documented choice, not an approximation of the standard formula.

**Our model is correct, not buggy.** It is important to be clear on this point. We did not change the meaning of any metric F1, MCC and AUC all follow standard formulas. We changed only the way they are computed (custom Keras classes with One-Hot tensors) so that TensorFlow 2.19 could run them, and we chose the macro averaging convention because it gives equal weight to every disease class. We tested the three custom classes on a small example before training, confirmed they return correct values, and only then trained the real model. Because of this, the high scores in Table 4.11 are trustworthy and reproducible.

## 4.4 Final Results and Analysis

### 4.4.1 Results Against the Project Targets

The champion model exceeds **all four project targets**, often by a large margin. Figure 4.5 shows the four final metrics against their targets (the red dashed lines).

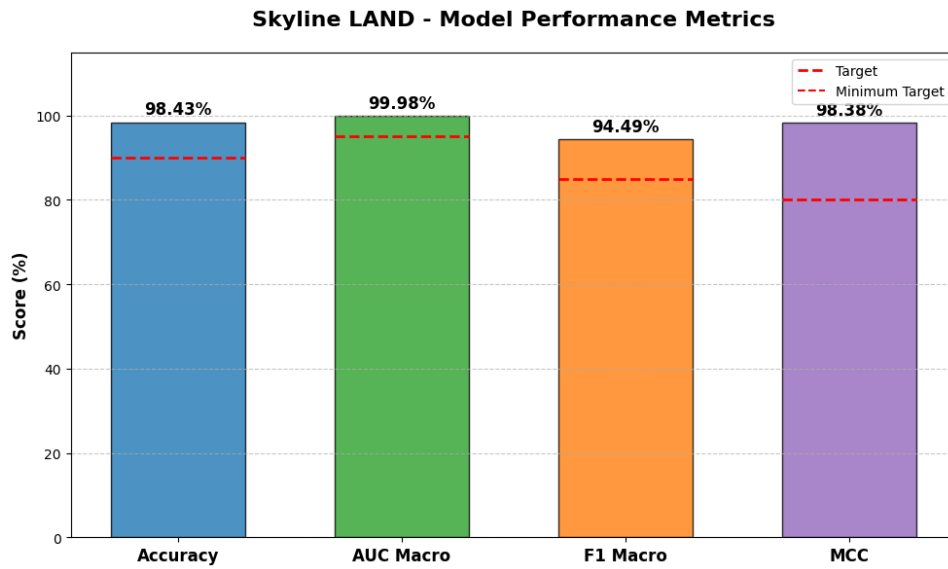


**Figure 4.5.** Final performance of EfficientNet-B4 against the project targets (red dashed lines). All targets are exceeded.

The accuracy of 98.43% is well above the 90% target; the AUC of 0.9998 is almost perfect against a 0.95 target; the F1-Macro of 0.9449 is far above the 0.85 target; and the MCC of 0.9838 is much higher than the 0.80 target. These results put the model at a **production-ready** level.

#### 4.4.2 Per-Class Performance

A high average score can still hide weak classes, so we also looked at the F1-score of **each** of the 47 classes. Figure 4.6 shows this distribution.

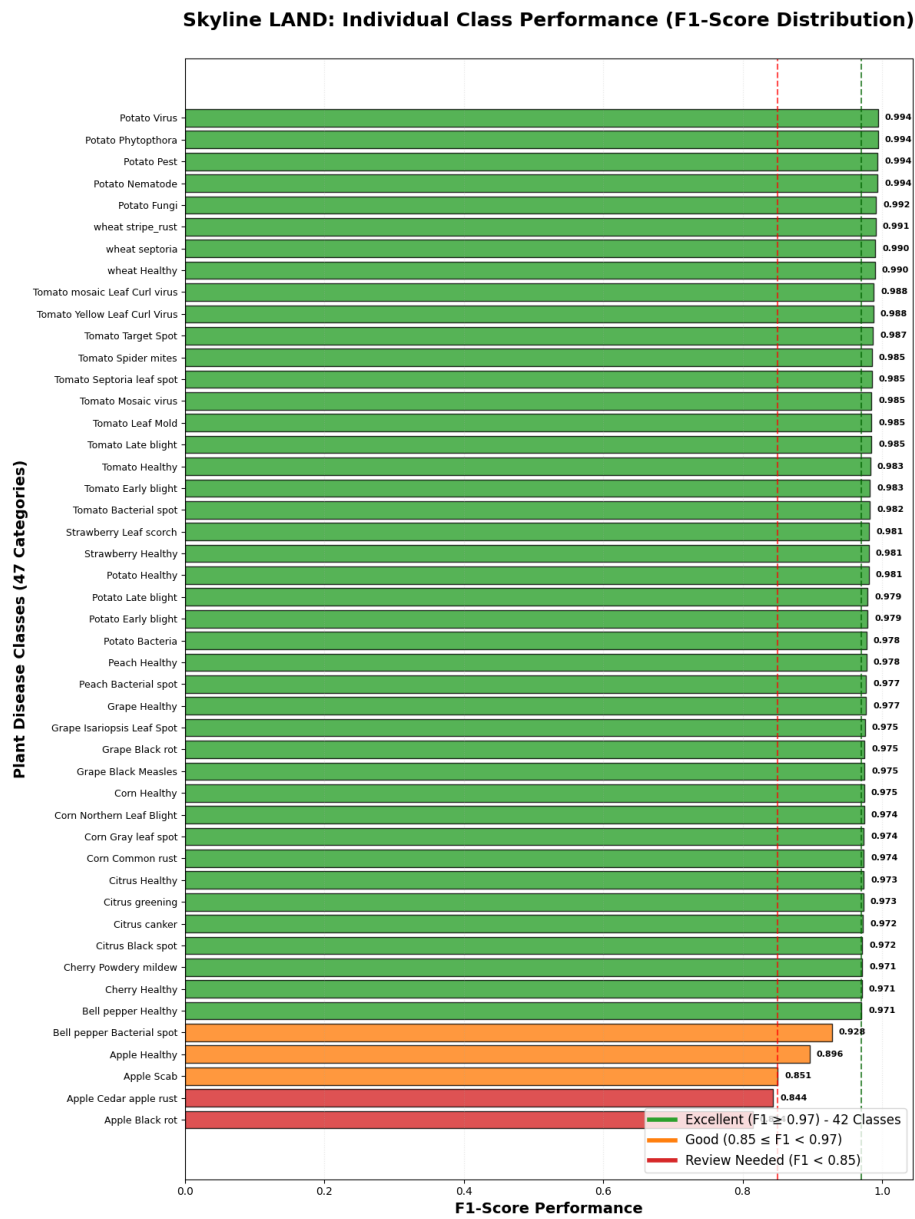


**Figure 4.6.** F1-score of each of the 47 classes. Most classes reach an excellent score; only a few rare-data classes fall below the target.

The large majority of classes are classified almost perfectly, with **42 of the 47 classes reaching  $F1 \geq 0.97$**  (the green “Excellent” group). Only five classes fall below the target, as detailed in Table 4.12. It is important to read this correctly: every single weak class has very few images, and this is the only reason it underperforms.

**Table 4.12.** The five weak classes of EfficientNet-B4 ( $F1 < 0.85$ ) and their root cause.

| Class               | F1-Score | Val. images | Root cause                                     |
|---------------------|----------|-------------|------------------------------------------------|
| Potato Nematode     | 0.57     | 14          | Only 68 total images — critically insufficient |
| Potato Pest         | 0.66     | 125         | Relatively few samples                         |
| Citrus Healthy      | 0.74     | 16          | Only 87 total images — critically insufficient |
| Potato Phytophthora | 0.76     | 66          | Few samples (347 total)                        |
| Citrus Canker       | 0.78     | 79          | Few samples (368 total)                        |

**A data limit, not a model weakness.** The five weak classes underperform only because they have very few training images. With so few examples, **no** model can learn the class reliably this is a fundamental statistical limit of the data, not a flaw in our model or our code. It is also worth noting that EfficientNet-B4 handles this problem better than ResNet50: ResNet50 had 8 weak classes, with Potato Nematode reaching only  $F1 = 0.239$ ; EfficientNet-B4 reduces this to 5 weak classes and raises Potato Nematode to  $F1 = 0.57$ . The recommended fix is simply to collect at least 500 additional images per weak class in future work.

### 4.4.3 Explainability with Grad-CAM

A high score is not enough on its own: a farmer needs to **trust** the model. To make the model transparent, we used **Grad-CAM** [74] (Gradient-weighted Class Activation Mapping). Grad-CAM produces a heatmap that shows which part of the leaf most influenced the model’s decision, turning the model from a “black box” into an interpretable tool.

In the heatmap, red means high importance and blue means low importance. On our test images, the red zone is concentrated exactly on the diseased area of the leaf, which confirms that the model looks at the **correct region** and is not guessing from the background. This also lets the farmer see exactly where the disease is located on the plant, which increases trust and allows a manual check by an agricultural expert.

## 4.5 Front-End and Application Implementation

After the model was trained and validated, we built the real application around it. A strong model is not useful on its own; it needs a clear interface so that a farmer and an administrator

can actually use it. In this section we present the front-end of the Skyline platform, the database behind it (Supabase), and the local FastAPI service that connects the website to the AI model.

### 4.5.1 General Scenario

The Skyline platform is presented as a large company website that offers several services. The landing page (Figure 4.7) shows the main brand, “Elevate Your Vision with SKYLINE AI”, with a top menu (Home, Services, Fleet, Tech, Join Us) and a language switch (AR / EN) plus a Login button. The platform is designed to host **six services** in the future, but in this project we focused on one of them: the **Smart Agriculture** division (Figure 4.8), which is where crop disease detection lives.

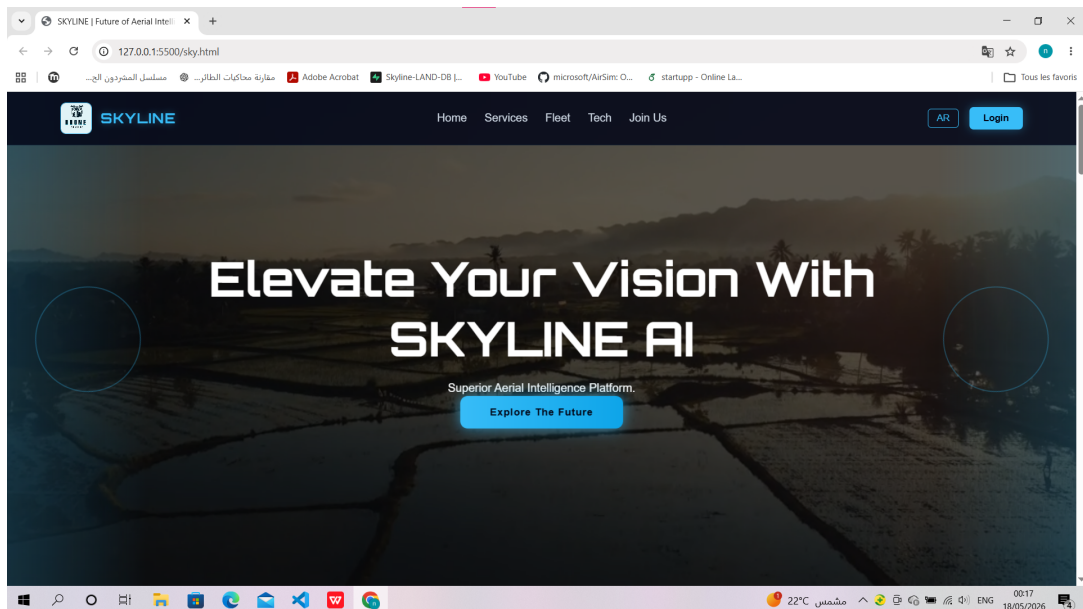


**Figure 4.7.** Skyline landing page the main entry point of the platform.

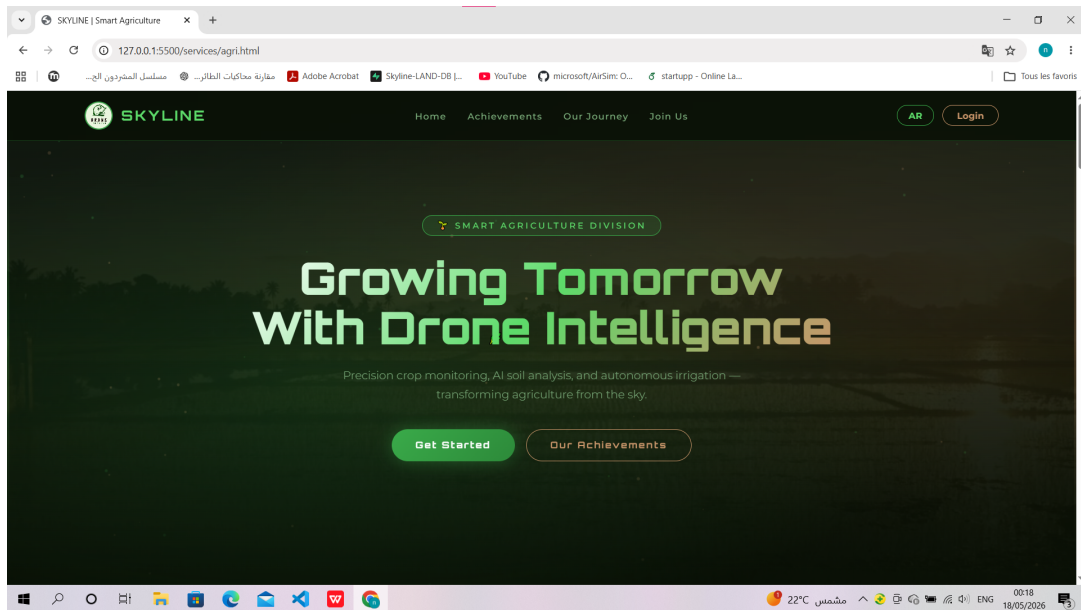


**Figure 4.8.** The Smart Agriculture division, the service this project focuses on.

The full scenario follows a clear and deliberate path that we designed from the start. A user creates an account on the platform and fills in a mission request with all the necessary details. That request is reviewed by an administrator who can accept, reject, or queue it. Once the mission is confirmed, the administrator assigns an available drone and launches it. The drone flies autonomously over the field area at the GPS coordinates the farmer provided, captures aerial images of the crops, and analyses each image locally using the EfficientNet-B4 model. When the mission is complete and the drone has landed and returned to available status, the farmer opens their account and sees a **map of their field** with colour-coded points showing exactly which areas are healthy and which are diseased. Each point can be opened to reveal the full diagnosis. In parallel, the platform also gives the user a direct image-upload tool so they can analyse any leaf photo immediately, without waiting for a drone mission, and receive the result the disease name, the AI confidence, the treatment advice in both **Arabic and English**.

This scenario is not a prototype flow on paper; every step described above is implemented and running, as the following subsections show.

### 4.5.2 The User Side Requesting a Mission

From the user account, a farmer fills in a clear form to request a drone mission, as shown in Figure 4.9. The form collects exactly the fields we defined in the class diagram: the mission title, the email and phone, the wilaya, the crop type, the GPS location or field area, the field area in hectares, the mission deadline, and a free description of what the farmer needs (disease scanning, crop health check, irrigation mapping, pest detection).

**Request a Drone Mission**

Fill in your field details and we'll schedule a drone flight for you

**MISSION TITLE \***  
e.g. Tomato field disease scan - sector B

**YOUR EMAIL \***  
khaledt2@gmail.com

**PHONE**  
+213 5XX XXX XXX

**WILAYA \***  
Select wilaya...

**CROP TYPE**  
e.g. Tomato, Wheat, Potato

**GPS LOCATION / FIELD AREA \***  
e.g. 34.8500° N, 1.3167° E — or field name / commune  
Enter GPS coordinates or a description of your field location

**FIELD AREA (HECTARES)**  
e.g. 5.5

**MISSION DEADLINE \***  
jj/mm/aaaa

**MISSION DESCRIPTION \***  
Describe what you need: disease scanning, crop health check, irrigation mapping, pest detection...

**Figure 4.9.** The “Request a Drone Mission” form on the user side.

When the farmer sends the form, the request is stored in the database with the status pending and waits for the administrator to review it. The farmer can follow the status of their request from the same account at any time.

### 4.5.3 The Admin Side Reviewing and Launching

The administrator has a separate and secure dashboard to manage all the incoming requests. Figure 4.10 shows the mission list view: each row gives the user email, the wilaya, the GPS / field area, the deadline, the description, the payment status (Unpaid, Paid, Partial, Refunded) and the mission status (Confirmed, Cancelled, Completed). From the left-side navigation menu, the admin can also reach the Users management, the Diagnosis history, the Drones inventory, the Analytics panel, the Reports and the Services settings, though the core workflow for this project centres on the mission lifecycle described here.

| # | EMAIL              | WILAYA           | GPS / FIELD AREA | DEADLINE   | DESCRIPTION | PAYMENT  | MISSION STATUS | ACTIONS |
|---|--------------------|------------------|------------------|------------|-------------|----------|----------------|---------|
| 1 | mustfai@gmail.com  | 20 — Salda       | 34.8080 12.55    | 21/06/2026 | mapping     | UNPAID   | CONFIRMED      | ⓘ       |
| 2 | meziane@gmail.com  | 25 — Constantine | 34.8080 12.55    | 21/06/2026 | check filed | PAID     | CANCELLED      | ⓘ       |
| 3 | khaled12@gmail.com | 20 — Salda       | 34.8580, 1.3167  | 29/05/2026 | scanning    | PAID     | COMPLETED      | ⓘ       |
| 4 | abdo@gmail.com     | 20 — Salda       | 34.8580, 1.3167  | 10/06/2026 | تفريغ       | PARTIAL  | COMPLETED      | ⓘ       |
| 5 | meziane@gmail.com  | 25 — Constantine | 1225.00 562      | 25/05/2026 | idk         | REFUNDED | COMPLETED      | ⓘ       |
| 6 | moha@gmail.com     | 20 — Salda       | 34.8080 12.55    | 12/05/2026 | idk         | UNPAID   | COMPLETED      | ⓘ       |

**Figure 4.10.** The admin dashboard: the list of all mission requests with their payment and mission status.

When the admin opens a request, they can change its mission status and payment status, set the payment amount, and write a note that is visible to the user. If the request is accepted, the admin sets the status to **Confirmed**, then uses the **Launch Drone Mission** panel (Figure 4.11) to configure the three flight parameters — altitude (default 20 m), speed (default 3 m/s), and photo capture interval (default every 8 m) — assign an available drone (for example “SKY-1837 — DJI Phantom 4, 100% battery”) and press **Confirm & Launch Drone**. These flight parameters are saved as a `flight_config` JSON object inside the mission row in Supabase and are read by `skyline.py` at the start of the mission. This is the exact moment that starts the autonomous mission described later in the AirSim section.

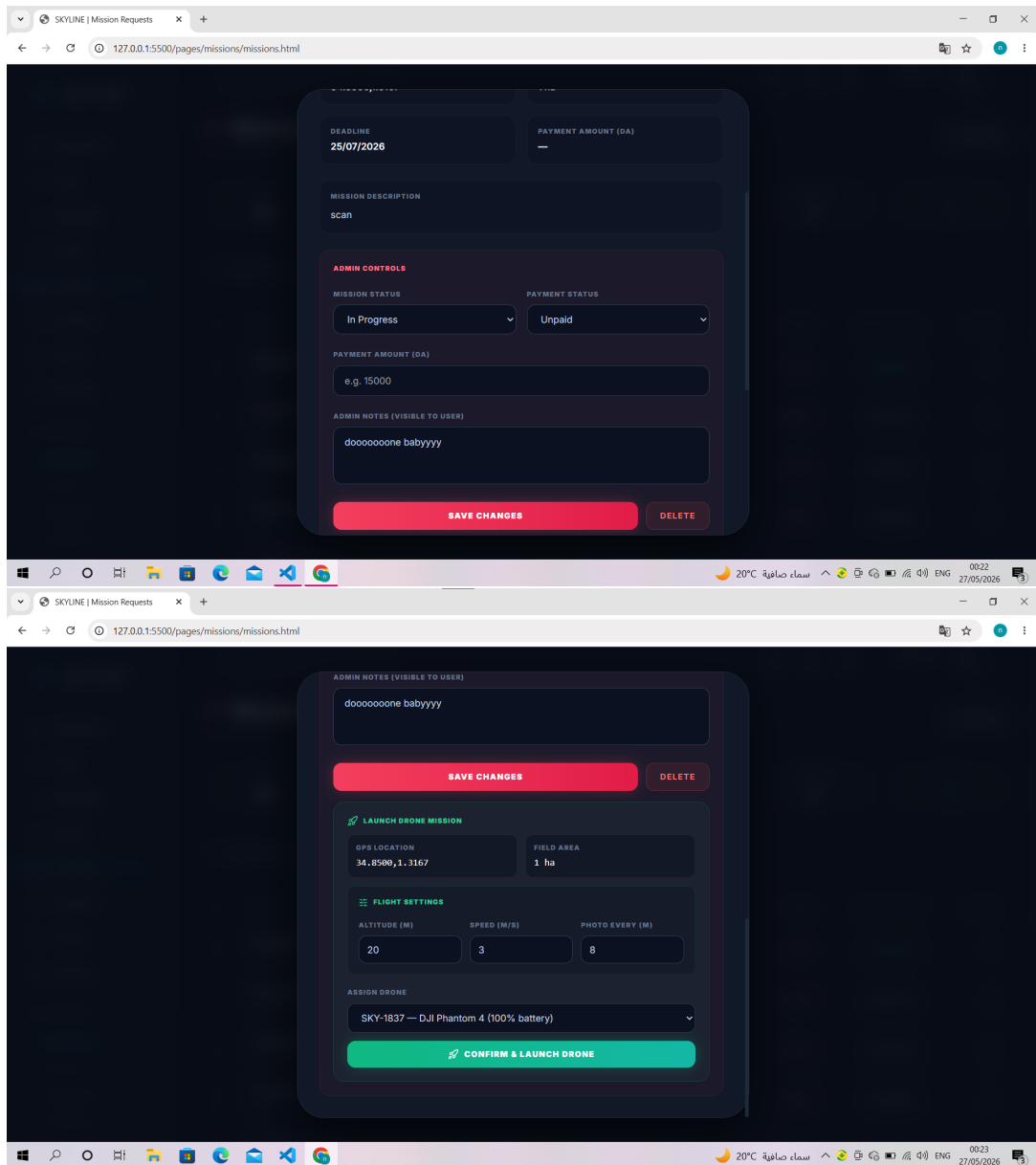


Figure 4.11. The admin confirms the mission and launches an available drone.

#### 4.5.4 The Result Field Map and Diagnosis

After the mission is completed, the user opens their field map (Figure 4.12). The map is built with **Leaflet** [76] on top of Esri satellite tiles, so the user sees their real field from above, not a generic background. Every point where the drone captured and analysed an image is marked on this map. **Green points** mark healthy spots and **red points** mark diseased spots, positioned exactly over the GPS coordinates of each capture. The top of the page gives a quick summary of the field: the total number of analysed points, the number of healthy zones, the number of diseased zones, and the overall health percentage. When the user clicks any point, a side panel slides open and shows the captured image, its latitude, longitude and altitude, the diagnosis, and the model confidence for that specific location.

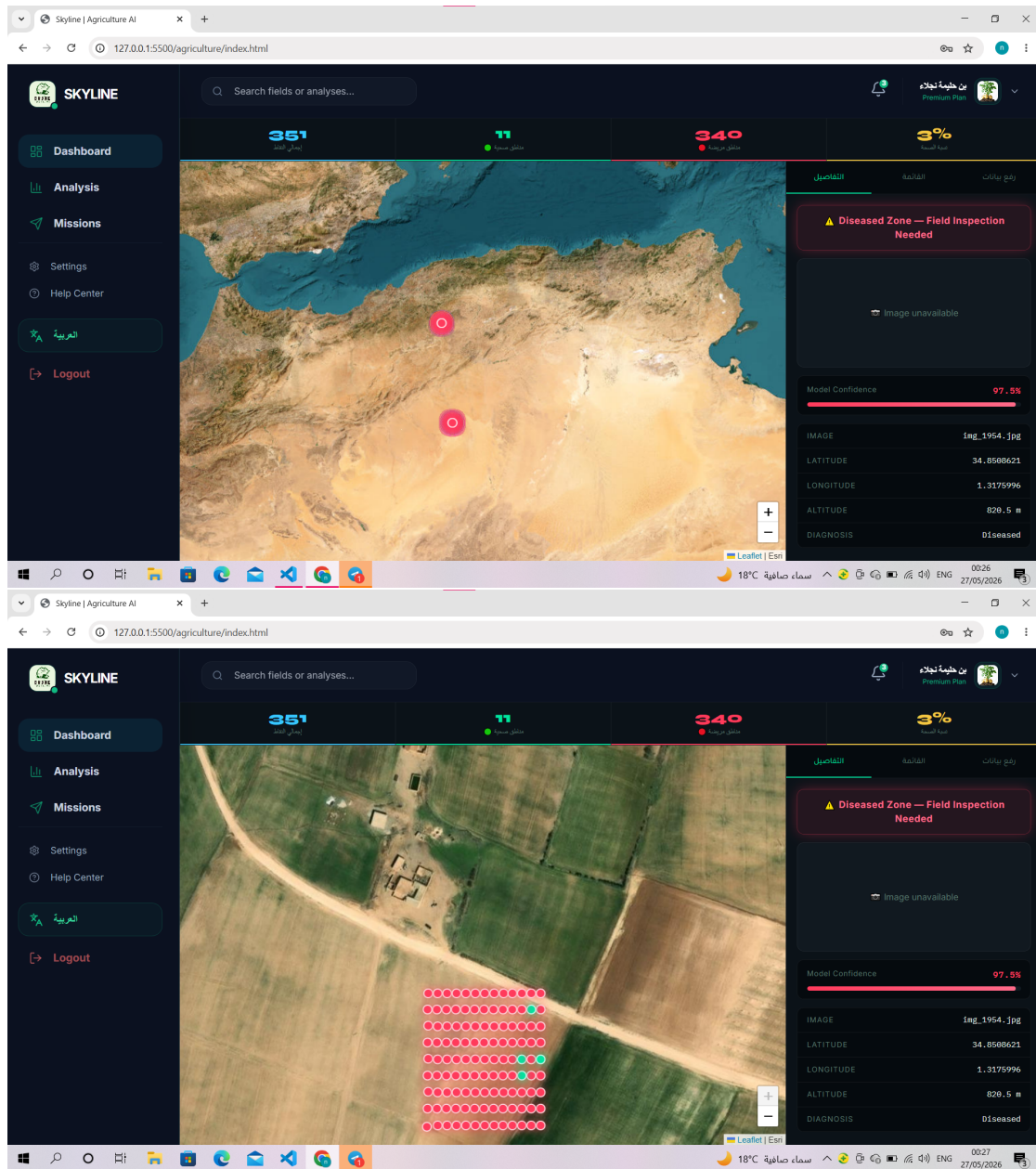


**Figure 4.12.** The user field map: green points are healthy, red points are diseased, drawn over the real field area.

Beside the drone missions, the platform also offers a **direct analysis** tool, the **Field Analysis Engine** (Figure 4.13). Here the user uploads any image and the system analyses it instantly. The model panel on the right confirms the model we selected in this chapter: **Architecture EfficientNetB4, Accuracy 98.43%, AUC-Macro 0.9998, F1-Macro 0.9449, 47 classes**. The engine combines our CNN with a Vision Language Model (Gemini Vision) [77] to produce a richer expert report, and the user can choose the report language (English or Arabic).

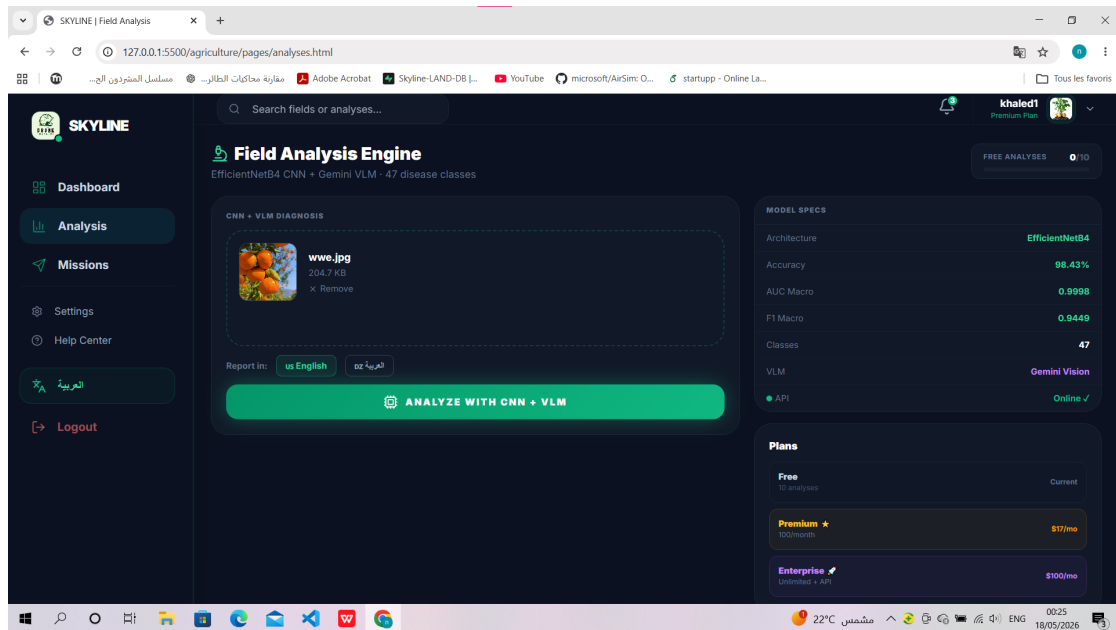


**Figure 4.13.** The Field Analysis Engine: upload an image and analyse it with the EfficientNetB4 CNN (the model specs match the previous section of this chapter).

The result page (Figure 4.14) gives the full bilingual diagnosis: the top-3 predictions with their confidence percentages, the expert advice and the treatment recommendation in both **English and Arabic**, and a VLM expert report. The user can also export the full report as a PDF or ask the AI expert a follow-up question directly from the page. This is the moment where the model of this chapter turns into a real, usable answer for a farmer standing in their field.

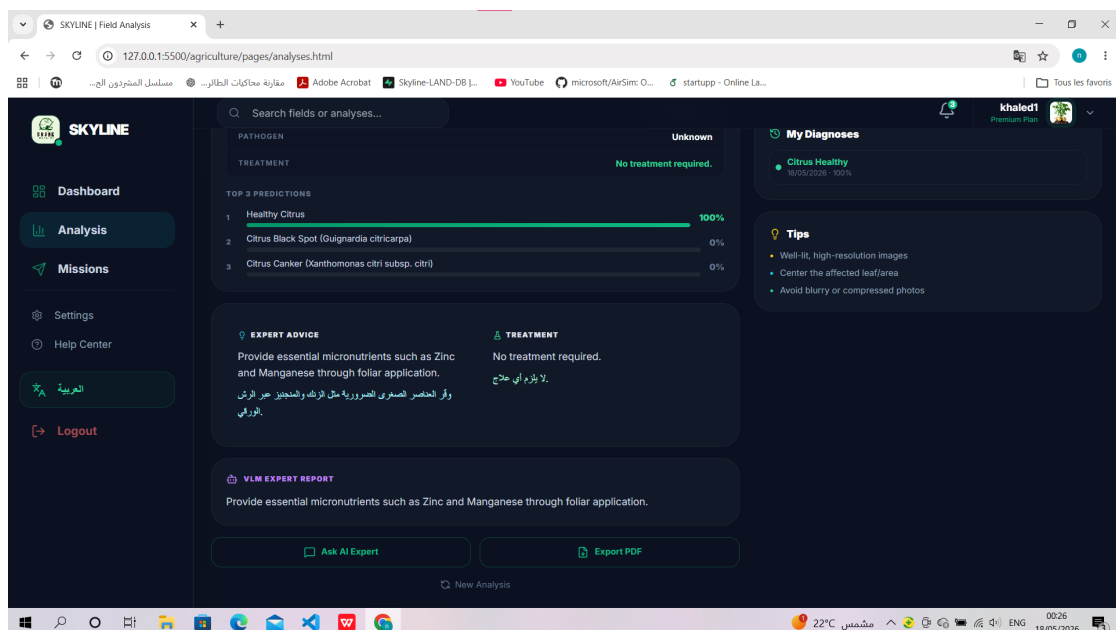


**Figure 4.14.** The bilingual diagnosis result: top-3 predictions, expert advice and treatment in English and Arabic.

### 4.5.5 The Database Supabase

Every piece of information that flows through the Skyline platform is stored in a single, centralised database powered by **Supabase** [75]. Supabase is an open-source backend built on top of **PostgreSQL**, and it was the right choice for this project for three concrete reasons.

First, it gives us a fully relational database where we can define tables, foreign keys and constraints exactly as we designed them in the class diagram of the previous chapter. Second, it provides a built-in **authentication and role system**, which means we did not have to write login code from scratch: Supabase handles user registration, email verification, password management and session tokens, and lets us attach a role (user or admin) to each account so that the two sides of the platform see different pages. Third, it exposes an **auto-generated REST and real-time API**, which allows the JavaScript front-end to read and write data with a few lines of code, without maintaining a separate backend server for basic data operations.

In practice, Supabase holds every table we designed: profiles (user and admin accounts), agriculture (farm records), mission\_requests (the requests sent by farmers), drones and services, diagnostics (individual AI results), field\_results (one row per GPS point on the map), disease\_database (the bilingual knowledge base of the 47 classes), and notifications (status updates sent to users).

This single source of truth is what links the three sides of the platform together. When a farmer sends a mission, a row is inserted into mission\_requests with the status pending. When the admin confirms it, the same row is updated to confirmed. When the drone finishes and the Python bridge writes the results back, rows are inserted into field\_results. The user map then reads those rows and draws the red and green points in real time. Nothing is duplicated, nothing is lost, and every actor in the system always sees the same, up-to-date state.

### 4.5.6 The AI Service FastAPI (Local)

The website is a standard HTML/JavaScript front-end. It is deliberately kept light: it handles only the interface and the database calls. It cannot and should not run a deep learning model, because loading TensorFlow and EfficientNet-B4 inside a browser is neither practical nor safe.

To bridge this gap, we built a separate **FastAPI** [78] service that runs **locally** on the machine and exposes the model through a clean REST interface. FastAPI is a modern Python web framework designed for exactly this kind of task: it is fast, it is easy to document, and it handles binary uploads (images) naturally.

The separation of roles is clear and deliberate. When a user uploads an image through the Field Analysis Engine, the front-end sends a single HTTP request to the local FastAPI ser-

vice. The service receives the image and runs the following steps internally, completely hidden from the user: it resizes and normalises the image with the exact same preprocessing pipeline used during training; it runs the EfficientNet-B4 model and obtains the 47-class softmax probabilities; it selects the top-3 predictions with their confidence scores; it computes the Grad-CAM heatmap for the predicted class; and it looks up the predicted class in the `disease_database` to retrieve the bilingual name, the treatment advice, and the pesticide recommendation. The service then sends all of this back to the front-end in a single JSON response, and the front-end renders the result page.

This architecture has a practical advantage beyond clean code. Because the model is loaded **once** when FastAPI starts and kept in memory, every subsequent prediction takes only the inference time ( $\approx 120$  ms) with no reload overhead. This is what makes the Field Analysis Engine feel instant from the user’s side. FastAPI is the same component we called the “AI Service” in the architecture diagram of the previous chapter. Note that this FastAPI service is used exclusively for the manual image-upload tool; the autonomous drone flight uses a separate path described in the next section.

## 4.6 Drone Simulation AirSim and Unreal Engine

In the current phase of the project, we do not yet fly a real drone over a real field. Instead, we use a **realistic simulation** that lets us test the whole flow launch, flight, image capture and result delivery in a safe and repeatable way. This simulation is built on **Unreal Engine** [80] (a 3D game engine) and **Microsoft AirSim** [79] (a flight simulator that runs inside Unreal). Together, they give us a virtual world with a virtual drone that behaves like a real one. This section explains, step by step, how we prepared this environment and how we connected it to the website.

### 4.6.1 Why a Simulation First

A real drone is expensive, needs a flight permit, and a single test can fail because of wind, battery or a crash. A simulation removes all of these risks. With AirSim we can fly the same mission as many times as we want, capture aerial images of a virtual field, and test the connection between the website and the drone, all on a single computer. This matches the plan we showed in the architecture of the previous chapter, where AirSim simulation is the *current* image source and the real drone (UAV) is *planned* for a later phase.

### 4.6.2 Preparing the Environment, Step by Step

Setting up AirSim is a long process with several tools that must work together. We list the exact steps we followed, in order, because each one is needed for the next.

1. **Install Python 3.9.** We use Python 3.9 (64-bit) because it is the most stable version with AirSim. During the installation, the option `Add Python to PATH` must be checked. We verified the install with `python --version`, which returned `Python 3.9.x`.
2. **Install Visual Studio Build Tools.** AirSim needs the C++ build tools. We installed the “Build Tools for Visual Studio” and selected the components `Desktop development with C++, MSVC v142 (or newer)` and the `Windows 10 SDK`.
3. **Install Unreal Engine 4.27.** We installed the Epic Games Launcher, created an account, and from the Library we installed **Unreal Engine 4.27**. This exact version matters: AirSim runs inside Unreal, and version 4.27 is the stable one for it (we did not install UE5).
4. **Install Git and clone AirSim.** We installed Git for Windows (choosing “Git from the command line”), checked it with `git --version`, then downloaded AirSim from its official Microsoft repository:

```
git clone https://github.com/microsoft/AirSim.git
cd AirSim
build.cmd
```

The `build.cmd` command compiles AirSim with the C++ tools we installed earlier.

5. **Install the AirSim Python package.** So that our Python code can talk to the drone, we installed the AirSim client and NumPy in Python 3.9:

```
py -3.9 -m pip install airsims
py -3.9 -m pip install numpy
```

This also installed the helper libraries `msgpack-rpc-python`, `msgpack-python` and `tornado`, which AirSim uses to communicate.

6. **Build a realistic field map.** Inside Unreal Engine 4.27 we built a realistic outdoor map using the **LandscapePro 2.0** asset pack. This pack auto-generates a detailed landscape with rocks, trees (with wind), ground plants, ferns, grasses and water, which gives us a believable field for the drone to fly over. Figure 4.15 shows our prepared map in the Unreal editor.

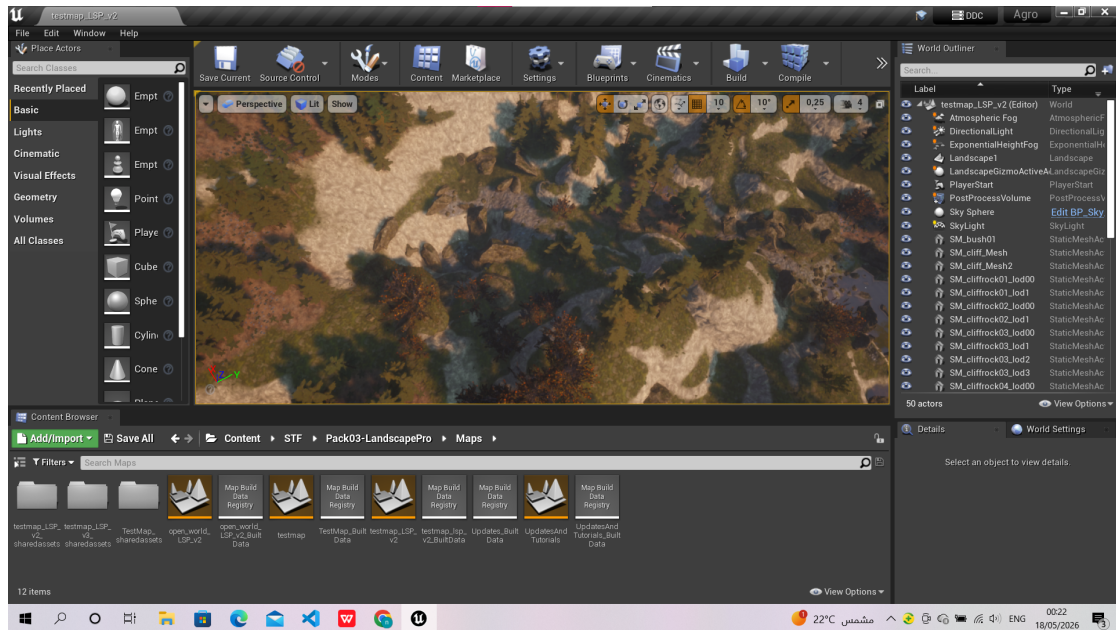


**Figure 4.15.** The prepared field environment in Unreal Engine 4.27 (with the LandscapePro landscape), where the AirSim drone flies.

**In short.** After these steps we have a complete virtual world: Unreal Engine 4.27 draws a realistic field, AirSim adds a flying drone inside it, and the Python AirSim client lets our own code control that drone. Everything runs on one machine, with no real flight risk.

### 4.6.3 Connecting the Website to Unreal the Python Bridge

The last piece is the **bridge** between the website and the simulation. When the admin presses **Confirm & Launch Drone** on the website, something must tell the virtual drone in Unreal to take off. We wrote a **Python script** (`skyline.py`) that plays exactly this role: it connects the website to Unreal through AirSim.

The script is launched on the local machine with the Python 3.9 we installed:

```
cd C:\Users\Asus\Desktop\droune
python skyline.py
```

Its job is to close the loop in four steps:

1. **Listen for a launch.** The script runs a permanent loop that queries the `mission_requests` table in Supabase every 10 seconds, looking for any row where `mission_status = confirmed`. When one is found, it performs a second verification query to confirm the status has not changed, and checks a local in-memory set of already-completed mission IDs to prevent any re-execution. The flight parameters (altitude, speed, photo interval) and the GPS location are read

from the `flight_config` JSON stored in the mission row.

2. **Fly the drone in AirSim.** Using the AirSim Python client, the script first climbs to an overview altitude and captures a single wide panoramic shot of the whole field, then descends to the mission altitude and follows a systematic lawnmower pattern of parallel rows, covering the entire field area while capturing aerial images of the crops at the configured interval.
3. **Analyse the images.** Each captured image is analysed **locally** by the EfficientNet-B4 SavedModel loaded directly via TensorFlow no HTTP call is made to the FastAPI service. The predicted class label and confidence score are obtained directly from the model's softmax output.
4. **Send the results back to the site.** The script writes the results image name, latitude, longitude, altitude, diagnosis label, confidence score, `profile_id` and timestamp back into the `field_results` table in Supabase. The user map then reads those rows and draws the red and green points. When the mission is complete, the script updates `mission_status` to `completed`, resets the drone's status back to `available` in the `drones` table, and adds the mission ID to a local in-memory set so that the same mission is never re-executed even if its status is changed externally.

This is the same flow we saw on the user side in Figure 4.12: a point on the map such as `img_0466.jpg` at latitude 34.8506, longitude 1.3176, altitude 820.5 m, with its diagnosis and confidence, comes directly from this pipeline. When the mission is complete, the virtual drone lands at its starting position, and the full result is already visible to the user exactly as designed in the sequence diagram of the previous chapter.

**The complete loop.** Website (launch) → Python bridge (`skyline.py`) → AirSim / Unreal (fly + capture) → EfficientNet-B4 SavedModel / TensorFlow (analyse locally) → Supabase (store in `field_results`) → Website map (show red / green points). This closes the journey from a farmer's request to a diagnosis on the map, end to end.

## Conclusion

In this chapter we presented the full experimentation of the Skyline platform, from the data to a working application.

We first built and cleaned a large dataset of **142,306 images** over **47 classes** from **11 crops**, gathered from six public sources. We handled its strong class imbalance ( $131\times$  ratio) with class weights, and prepared the data through a clear pipeline: preprocessing with model-specific normalisation, six drone-style augmentations, and an 80/20 split with zero data leak-

age.

We then trained **three models** with transfer learning and compared them fairly on the same data. MobileNetV2, our lightweight baseline, fell short of the accuracy target (87.54%) and produced a weak F1-Macro of 0.8012, confirming that a small architecture without a deep classification head cannot handle 47 imbalanced classes. Its concrete failure on an Apple Scab image classified as Corn Healthy with only 18% confidence illustrated this limit in the most direct way. ResNet50 cleared all six targets (95.78% accuracy, F1-Macro 0.9071, Precision 0.8916, Recall 0.9425, MCC 0.9565, AUC 0.9995), showing that a deeper architecture with a richer head is what this problem requires. **EfficientNet-B4** then outperformed ResNet50 on every metric 98.43% accuracy, F1-Macro 0.9449, MCC 0.9838, AUC 0.9998 and was selected as the champion model.

We described the champion model in detail: its 9-layer architecture, the two-phase training strategy with 60 layers unfrozen in Phase 2, the six anti-overfitting techniques confirmed by smooth training curves, and the three custom metric classes we wrote ourselves because the built-in Keras AUC failed in TensorFlow 2.19. We also clearly stated and justified our choice of the macro-averaged binary MCC as the multi-class variant best suited to our strongly imbalanced 47-class problem. Among the 47 classes, 42 reach  $F1 \geq 0.97$ ; the 5 weak classes all suffer exclusively from insufficient training data, not from any model weakness and EfficientNet-B4 handles them better than ResNet50 did.

We then turned this model into a real product. The Skyline front-end covers the full user journey: from the mission request form to the admin confirmation and drone launch, to the satellite field map with red and green diagnostic points, to the bilingual Field Analysis Engine. Behind the interface, **Supabase** acts as the single PostgreSQL database that links all actors together in real time, while the local **FastAPI** service loads the model once and serves every prediction request for the manual image-upload tool efficiently. Finally, the **AirSim** simulation on Unreal Engine 4.27 provides a safe, repeatable virtual environment for testing the complete loop from a farmer's confirmed mission to a map of field results — before any real drone is deployed. The autonomous drone script (`skyline.py`) runs the EfficientNet-B4 model locally via TensorFlow, writes results directly to `field_results` in Supabase, and on completion resets the drone status to `available` and locks the mission against re-execution.

Together, these three parts the model, the application and the simulation form a complete, working end-to-end system. The next chapter gives a general conclusion of the whole project and presents the perspectives for future work, in particular the move from the AirSim simulation to a real drone in the field.

# General Conclusion

This project set out to answer a practical and urgent question: how can drones and artificial intelligence be combined to give Algerian farmers a reliable, accessible, and bilingual tool for detecting crop diseases before they destroy a harvest?

The answer we built is the **Skyline LAND** platform — a complete, end-to-end system that goes from a farmer’s mission request to a colour-coded map of their field, with expert treatment advice in both Arabic and English.

**What we achieved.** We built and validated every component of the platform. We assembled a dataset of 142,306 images across 47 disease classes and 11 crops, handled an extreme class imbalance of  $131\times$  using class weights, and trained three models under identical conditions. EfficientNet-B4 was selected as the champion model after clearly outperforming MobileNetV2 and ResNet50 on every metric: 98.43% accuracy, F1-Macro of 0.9449, MCC of 0.9838, and AUC of 0.9998. We connected this model to a full web platform with a user mission request flow, an administrator dashboard, a real-time satellite field map built with Leaflet, and a bilingual Field Analysis Engine. The AirSim simulation on Unreal Engine 4.27 allowed us to test the complete pipeline — from drone launch to result delivery — in a safe and repeatable virtual environment, closing the loop that the sequence diagram described in theory.

**What we learned.** The most important lesson of this project is that performance on a benchmark is not the same as a working system. The real challenges were not in training the model — they were in handling class imbalance across 47 unequal classes, writing custom metric code when the standard Keras implementation failed in TensorFlow 2.19, connecting a Python simulation to a live web database, and making sure that a farmer in Algeria who does not speak English can read and act on every result the system produces. Each of these problems required a concrete engineering decision, not just a literature review.

**Limitations.** Four limitations remain at the end of this work. First, the five weak classes — Potato Nematode, Potato Pest, Citrus Healthy, Potato Phytophthora, and Citrus Canker — underperform due to insufficient training data, not due to any flaw in the model or the pipeline. Second, the drone used in this project is simulated: no real UAV has yet flown

over a real field with our system. Third, the FastAPI inference service runs locally on a development machine and has not yet been deployed on a production cloud server. Fourth, the Vision Language Model (VLM) component integrated into the Field Analysis Engine is not yet fully mature: while it currently produces bilingual expert reports, its output consistency and depth of agronomic reasoning require further development and fine-tuning before it can be considered production-ready.

**Future directions.** The most important next step is to collect at least 500 additional images for each of the five weak classes, which would likely bring all 47 classes above the  $F1 \geq 0.85$  target. The second priority is to replace the AirSim simulation with a real Category 2 drone registered and authorised under Presidential Decree No. 21-285, following the legal steps outlined in Chapter 2. A third direction is to deploy the FastAPI service on a cloud instance and enable the optional AI chatbot using the Claude or Gemini API, so that farmers can ask follow-up questions about the diagnosis in natural language. A fourth and significant direction is to invest further effort in the VLM pipeline: improving prompt engineering, expanding the bilingual knowledge base, and evaluating the model’s agronomic outputs against expert annotations, with the goal of making the generated reports accurate and actionable enough for real field use. Finally, adding multispectral sensor support would allow the system to detect stress and early-stage disease that is not yet visible in RGB images, pushing the platform further towards true early detection at scale.

The Skyline LAND platform is not a finished product, but it is a working proof of concept that demonstrates, from end to end, that drone-based bilingual crop disease detection is buildable, testable, and ready for the next phase of real-world deployment.

# **Bibliography**

# Bibliography

- [1] Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15), 2787–2805.
- [2] Austin, R. (2010). *Unmanned Aircraft Systems: UAVs Design, Development and Deployment*. Wiley.
- [3] Centenaro, M., Vangelista, L., Zanella, A., & Zorzi, M. (2016). Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios. *IEEE Wireless Communications*, 23(5), 60–67.
- [4] Colomina, I., & Molina, P. (2014). Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 92, 79–97.
- [5] Floreano, D., & Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *Nature*, 521(7553), 460–466.
- [6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- [7] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *Proc. 3rd Int. Conf. Learning Representations (ICLR)*; *arXiv:1412.6572*.
- [8] Hassanalian, M., & Abdelkefi, A. (2017). Classifications, applications, and design challenges of drones: A review. *Progress in Aerospace Sciences*, 91, 99–131.
- [9] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263–1284.
- [10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
- [11] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- [12] Liakos, K. G., Busato, P., Moshou, D., Pearson, S., & Bochtis, D. (2018). Machine learning in agriculture: A review. *Sensors*, 18(8), 2674.

- [13] LoRa Alliance. (2020). *LoRaWAN 1.0.4 Specification*. LoRa Alliance.
- [14] McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. (1955). A proposal for the Dartmouth Summer Research Project on Artificial Intelligence. (Reprinted in *AI Magazine*, 27(4), 12–14, 2006.)
- [15] Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing* (NIST Special Publication 800-145). National Institute of Standards and Technology.
- [16] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
- [17] OASIS Standard. (2019). *MQTT Version 5.0*. OASIS. (Also published as ISO/IEC 20922.)
- [18] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345–1359.
- [19] Roman, R., Lopez, J., & Mambo, M. (2018). Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78, 680–698.
- [20] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
- [21] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2020). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2), 336–359.
- [22] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), 1–48.
- [23] Stallings, W., & Brown, L. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson.
- [24] Szeliski, R. (2022). *Computer Vision: Algorithms and Applications* (2nd ed.). Springer.
- [25] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- [26] Food and Agriculture Organization of the United Nations (FAO). *The State of Food and Agriculture 2020: Overcoming water challenges in agriculture*. FAO, Rome, 2020.
- [27] 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.

- [28] S. P. Mohanty, D. P. Hughes, and M. Salathé. “Using deep learning for image-based plant disease detection.” *Frontiers in Plant Science*, vol. 7, art. 1419, 2016, doi:10.3389/fpls.2016.01419.
- [29] A. Sood and L. Singh. “Transfer learning with pre-trained CNN models for plant disease classification using the PlantVillage dataset.” *Computers and Electronics in Agriculture*, 2023.
- [30] W. Albattah, A. Javed, M. Nawaz, M. Masood, and S. Albahli. “Artificial intelligence-based drone system for multiclass plant disease detection using an improved efficient convolutional neural network.” *Frontiers in Plant Science*, vol. 13, art. 808380, 2022, doi:10.3389/fpls.2022.808380.
- [31] G. Doğan. “MobileNetV3-based plant disease classification on an augmented PlantVillage dataset.” *Agriculture (MDPI)*, 2023.
- [32] R. Mullen et al. “Application of image processing and transfer learning for the detection of rust disease.” *PMC / NCBI*, 2023.
- [33] H. M. Manoj, D. L. Shanthi, B. N. Lakshmi, K. J. Archana, E. V. N. Jyothi, and K. Archana. “AI-driven drone technology and computer vision for early detection of crop disease in large agricultural areas.” *Scientific Reports*, 2025, doi:10.1038/s41598-025-32384-1.
- [34] “Evolution of deep learning approaches in UAV-based crop leaf disease detection.” *Applied Sciences (MDPI)*, vol. 15, no. 19, art. 10778, 2025.
- [35] R. Chin, C. Catal, and A. Kassahun. “Plant disease detection using drones in precision agriculture.” *Precision Agriculture*, vol. 24, pp. 1663–1682, 2023.
- [36] “A review of CNN applications in smart agriculture using multimodal data.” *ResearchGate preprint*, 2025.
- [37] V. Naik et al. “Comparative analysis of twelve CNN models including VGG19 and DarkNet53 for chilli disease classification.” *Computers and Electronics in Agriculture*, 2022.
- [38] “Mobile-friendly deep learning for plant disease detection: A lightweight CNN benchmark across 101 classes of 33 crops.” *arXiv preprint*, 2024.
- [39] J. G. A. Barbedo. “Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification.” *Computers and Electronics in Agriculture*, vol. 153, pp. 46–53, 2018, doi:10.1016/j.compag.2018.08.013.

- [40] M. Tan and Q. V. Le. “EfficientNet: Rethinking model scaling for convolutional neural networks.” In *Proc. 36th Int. Conf. Machine Learning (ICML)*, 2019, pp. 6105–6114; *arXiv:1905.11946*.
- [41] T. Sillanpää and J. Salo. “Exploring opportunities of drone services and potential adaptation to Finnish industries.” Bachelor’s thesis, Seinäjoki University of Applied Sciences, 2024.
- [42] U.S. Government Accountability Office (GAO). *Precision Agriculture: Benefits and Challenges for Technology Adoption and Use*, GAO-24-105962, January 2024.
- [43] DJI Enterprise. “Precision agriculture with drone technology.” *DJI Enterprise Insights Blog*, December 2021.
- [44] M. O’Donnell. “Drones in livestock monitoring: enhancing animal welfare.” *AirStandards*, October 2023.
- [45] Wunderbuild. “Drone technology in construction: elevating site surveillance and project management.” *Wunderbuild Blog*, December 2023.
- [46] B. Pinguet. “How drones are future-proofing the construction industry.” *GIM International*, July 2020.
- [47] F. Magdolna. “Drone surveillance on construction sites.” *FlyEye*, July 2023.
- [48] Accruent. “What is Building Information Modeling? (BIM).” *Accruent Resources Blog*, January 2021.
- [49] DroneNodes. “Energy and utilities: wind turbine inspection using drones.” *DroneNodes*, November 2023.
- [50] Averroes. “Complete drone solar panel inspection guide 2024.” *Averroes AI Blog*, October 2024.
- [51] DroneDeploy. *Reality Capture / Drone Mapping Software*. <https://www.dronedeploy.com/>, 2024.
- [52] eMarketer. “Last-mile delivery: what it is and why it matters for retailers.” *eMarketer Insights*, October 2023.
- [53] Amazon. “Amazon drone delivery takes off in Arizona.” *Amazon News*, November 2024.
- [54] DHL Freight Connections. “Future of freight: the use of drones in logistics.” *DHL Freight Connections*, March 2024.
- [55] Arrive AI. “What are Autonomous Delivery Networks (ADNs)?” *Arrive AI Insight Hub*, 2023.

- [56] Microavia. “Unmanned aerial vehicles for wildlife protection: a technological solution to combat poaching.” *Microavia*, n.d.
- [57] Mav Drones. “Optimizing drone technology for search and rescue in natural disasters.” *Medium*, October 2024.
- [58] Republic of Algeria. *Presidential Decree No. 21-285 of 13 July 2021 establishing the general framework governing unmanned aircraft systems*. Official Gazette, 18 July 2021.
- [59] Republic of Algeria. *Interministerial Decree of 8 September 2025 regulating the acquisition, import, manufacture, and commercial use of drones*. Ministry of National Defence, September 2025.
- [60] Algerian Ministry of National Defence. *Official communiqué on mandatory drone declaration before 30 April 2026*. December 2025.
- [61] B. W. Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme.” *Biochimica et Biophysica Acta (BBA) – Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975, doi:10.1016/0005-2795(75)90109-9.
- [62] J. Gorodkin. “Comparing two K-category assignments by a K-category correlation coefficient.” *Computational Biology and Chemistry*, vol. 28, no. 5–6, pp. 367–374, 2004, doi:10.1016/j.compbiolchem.2004.09.006.
- [63] D. Chicco and G. Jurman. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation.” *BMC Genomics*, vol. 21, no. 1, art. 6, 2020, doi:10.1186/s12864-019-6413-7.
- [64] J. Tamura, Y. Itaya, K. Hayashi, and K. Yamamoto. “Statistical inference of the Matthews correlation coefficient for multiclass classification.” *arXiv preprint arXiv:2503.06450*, 2025.
- [65] 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.
- [66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A large-scale hierarchical image database.” In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.
- [67] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. “MobileNetV2: Inverted residuals and linear bottlenecks.” In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

- [68] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition.” In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [69] M. Tan and Q. V. Le. “EfficientNet: Rethinking model scaling for convolutional neural networks.” In *Proc. 36th Int. Conf. Machine Learning (ICML)*, 2019, pp. 6105–6114.
- [70] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization.” In *Proc. 3rd Int. Conf. Learning Representations (ICLR)*, 2015; *arXiv:1412.6980*.
- [71] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A simple way to prevent neural networks from overfitting.” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [72] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In *Proc. 32nd Int. Conf. Machine Learning (ICML)*, 2015, pp. 448–456.
- [73] M. Abadi et al. “TensorFlow: A system for large-scale machine learning.” In *Proc. 12th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.
- [74] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. “Grad-CAM: Visual explanations from deep networks via gradient-based localization.” In *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2017, pp. 618–626.
- [75] Supabase Inc. “Supabase — The open source Firebase alternative.” 2024. [Online]. Available: <https://supabase.com/docs>
- [76] V. Agafonkin. “Leaflet — An open-source JavaScript library for interactive maps.” 2024. [Online]. Available: <https://leafletjs.com>
- [77] Gemini Team, Google. “Gemini: A family of highly capable multimodal models.” *arXiv preprint arXiv:2312.11805*, 2023.
- [78] S. Ramírez. “FastAPI — A modern, fast web framework for building APIs with Python.” 2024. [Online]. Available: <https://fastapi.tiangolo.com>
- [79] S. Shah, D. Dey, C. Lovett, and A. Kapoor. “AirSim: High-fidelity visual and physical simulation for autonomous vehicles.” In *Field and Service Robotics (FSR)*, 2018, pp. 621–635; *arXiv:1705.05065*.
- [80] Epic Games. “Unreal Engine 4.27.” 2021. [Online]. Available: <https://www.unrealengine.com>

## Abstract

**Skyline** is a bilingual *prototype* Drone-as-a-Service platform designed to address challenges across six key sectors: **Agriculture, Logistics, Security, Delivery, Construction, and Energy**. By combining autonomous UAV missions, deep learning, and real-time satellite mapping into a unified bilingual service, Skyline LAND offers a scalable solution adaptable to each domain. This thesis focuses on the **Agriculture** sector, where crop diseases threaten up to 40% of annual production in Algeria, yet existing solutions remain costly, expert-dependent, or Arabic-inaccessible. **EfficientNet-B4** was selected through a controlled comparison with MobileNetV2 and ResNet50 on 142,306 images spanning 47 disease classes and 11 crops, achieving **98.43% accuracy**, F1-Macro 0.9449, MCC 0.9838, and AUC 0.9998. The full end-to-end prototype — from mission request to colour-coded bilingual expert report — was validated in AirSim on Unreal Engine 4.27.

**Keywords:** UAV, Drone-as-a-Service, Crop Disease Detection, EfficientNet-B4, Deep Learning, Precision Agriculture, Prototype, AirSim, Algeria.

## Résumé

**Skyline** est un *prototype* bilingue de plateforme Drone-as-a-Service conçu pour répondre aux défis de six secteurs clés : **Agriculture, Logistique, Sécurité, Livraison, Construction et Énergie**. En combinant missions UAV autonomes, apprentissage profond et cartographie satellite en temps réel dans un service bilingue unifié, Skyline LAND propose une solution évolutive adaptable à chaque domaine. Ce mémoire se concentre sur le secteur de l'**Agriculture**, où les maladies des cultures menacent jusqu'à 40% de la production annuelle en Algérie, tandis que les solutions existantes restent coûteuses, nécessitent une expertise pointue ou sont indisponibles en arabe. **EfficientNet-B4**, retenu après comparaison avec MobileNetV2 et ResNet50 sur 142 306 images couvrant 47 classes et 11 cultures, a atteint **98,43% de précision**, F1-Macro 0,9449, MCC 0,9838 et AUC 0,9998. Le prototype complet — de la demande de mission au rapport expert bilingue — a été validé de bout en bout sous AirSim sur Unreal Engine 4.27.

**Mots-clés :** UAV, Drone-as-a-Service, Détection de maladies, EfficientNet-B4, Apprentissage profond, Agriculture de précision, Prototype, AirSim, Algérie.

## ملخص

**Skyline** نموذج أولي (*Prototype*) لمنصة ثنائية اللغة تقدّم الطائرات المسيّرة تخدم، مصمّمة لمعالجة تحديات ستة قطاعات رئيسية: الزراعة، اللوجستيك، الأمن، التوصيل، البناء، والطاقة. من خلال دمج مهام الطيران المستقل والتعلم العميق والخرائط الفضائية الفورية في خدمة ثنائية اللغة موحّدة، تقدّم المنصة حلاً قابلاً للتوسع ومتكيفاً مع كل قطاع. تركز هذه المذكرة على قطاع الزراعة، حيث تُهدّد أمراض المحاصيل ما يصل إلى 40% من الإنتاج السنوي في الجزائر، في حين تظل الحلول القائمة مكلفة أو تستلزم خبرة متخصصة أو تفتقر إلى دعم اللغة العربية. تم اختيار **EfficientNet-B4** بعد مقارنة محكمة مع MobileNetV2 و ResNet50 على 142,306 صورة تشمل 47 فئة مرضية و 11 محصولاً، محققاً دقةً بلغت **98.43%**، و F1-Macro يساوي 0.9449، و MCC يساوي 0.9838، و AUC يساوي 0.9998. جرى التحقق من النموذج الأولي الكامل — من طلب المهمة حتى التقرير الخبير ثنائي اللغة — من طرف إلى طرف باستخدام محاكي AirSim على محرك Unreal Engine 4.27.

الكلمات المفتاحية: طائرة مسيّرة، خدمة الطيران المسيّر، كشف أمراض المحاصيل، EfficientNet-B4 تعلم عميق، زراعة دقيقة، نموذج أولي، AirSim، الجزائر.