AI Crop Monitoring & Agricultural Intelligence Platform – Full Implementation Guide

Generated by Grok

September 03, 2025

Contents

Introduction

The AI Crop Monitoring & Agricultural Intelligence Platform is a professional-grade ecosystem for precision agriculture. It integrates computer vision, time-series forecasting, market analytics, explainable AI, and microfinance scoring into a single, modular, and scalable system.

It is designed to serve:

- Farmers: simple mobile/web interface.
- Cooperatives/NGOs: dashboards for group management.
- Enterprises: large-scale deployments with monitoring, compliance, and integration into supply chains.

The system combines edge computing (offline models on phones, Raspberry Pi, or drones) with cloud-based retraining (Google Colab, Kaggle, or free-tier cloud). This makes it both low-cost and scalable.

Project Structure

You will maintain the following directory structure (already adapted for professional projects):

```
C:\Users\ADEGOKE\Desktop\Crop Monitor
|-- LICENSE
|-- README.md
|-- Dockerfile
|-- docker-compose.yml
|-- docker-compose.prod.yml
|-- docker-compose.mlflow.yml
|-- pyproject.toml
|-- requirements.txt
|-- run_pipeline.py
|-- setup.py
|-- main.py
|-- .gitignore
|-- .github/workflows/ci-cd.yml
|
|-- notebooks/ (experiments, EDA, training)
|-- monitoring/ (Prometheus, Grafana configs)
|-- docs/ (guides, architecture, API docs)
|-- deployment/ (Terraform, Kubernetes, configs)
|-- infrastructure/ (Docker, Helm, Terraform, manifests)
|-- src/ (recommendation_engine, frontend, crop_monitor)
|-- backend/ (workers, APIs)
|-- ai/ (training, models, explainability)
|-- tests/ (unit, integration, performance)
```

When & How You'll Use It

- Use notebooks/ for experiments in Google Colab or Jupyter. This includes exploratory data analysis (EDA), model prototyping, and initial training scripts.
- Use src/ for production-ready code. This directory houses modular components like recommendation engines and core business logic.
- Use ai/ for training pipelines and model definitions. Includes scripts for model training, evaluation, and export to formats like ONNX or TFLite.
- Use docs/ to store guides for farmers, administrators, and deployment engineers. Utilize Sphinx or MkDocs for generating documentation from Markdown files.
- Use tests/ to ensure everything works before deployment. Implement unit tests with pytest, integration tests for API endpoints, and performance tests for model inference times.

Features & Capabilities

Category	**Feature**	**Why It Matters**	**When & How You Use It**
Data Sources	Multi-modal (images, soil, weather, satellite, drone)	Improves prediction accuracy	Download from Kaggle (PlantVillage/PlantDoc), FAO SoilGrids, Sentinel-2 (Google Earth Engine or alternatives), OpenWeatherMap API. Use DVC to track versions and ensure reproducibility.
Camera Inputs	Smartphone, webcam, drone, IP camera	Enables scalable real-time capture	Farmers upload via Streamlit/Next.js UI. Developers test with OpenCV and webcams. Process images using cv2.imread for local testing.
Explainable AI	SHAP, LIME, Grad-CAM	Builds trust with farmers	Use Grad-CAM in notebooks to show heatmaps; use SHAP on structured models (e.g., credit risk). Integrate into dashboards for visual explanations.
Recommendations	Pesticide, fertilizer, irrigation	Provides actionable insights	Define mapping in YAML/JSON configs (src/config/treatments.yaml). Load configs using PyYAML and generate recommendations based on model outputs.

Continuous Learning	Feedback loops + active learning	Models evolve over time	Collect farmer feedback into SQLite (offline) and sync to MongoDB Atlas; retrain models monthly using Colab free GPUs. Use active learning strategies like uncertainty sampling.
Offline Mode	Edge inference	Enables remote farm usage	Export models to TensorFlow Lite / ONNX and deploy on mobile/edge devices. Use tflite_runtime for inference on Android/iOS, store results in SQLite.
Predictive Analytics	Yield forecasting, disease spread	Improves crop planning	Use Prophet or LSTM in ai/models/yield_forecasting/. Train in Colab, deploy via FastAPI endpoints for API access, store results in MongoDB Atlas.
Market Insights	Dynamic pricing, supply chain	Maximizes farmer profit	Collect price data via scraping (BeautifulSoup/Selenium); apply regression models like LinearRegression from scikit-learn. Store in MongoDB Atlas market_prices collection. Schedule with Celery.
Digital Finance	Loan recommendations	Provides financial access	Train logistic regression or boosting models; deploy via backend/api/recommendations.py. Store in MongoDB Atlas loan_requests collection.
Notifications	WhatsApp, Email, SMS	Ensures timely actions	Use Twilio sandbox for SMS/WhatsApp and Gmail API for email. Store logs in MongoDB Atlas notifications collection. Integrate with Celery for asynchronous sending.
Scalability	Multi-farm dashboards	Enterprise readiness	Build dashboards in Streamlit (src/frontend/streamlit_app/dashboard.py). Fetch data from MongoDB Atlas for multi-farm views. Use Kubernetes for scaling.

Compliance	GDPR-style logging	Trust & legal safety	Encrypt farmer logs in MongoDB Atlas crop_monitoring collection; allow farmers to request deletion. Use PyMongo for data access and implement audit logs.

Next-Level Differentiators

- **Edge AI & On-Farm Inference**: Deploy compressed models (ONNX, TFLite) on smartphones, Raspberry Pi, or drones. Use ONNX Runtime for cross-platform inference.
- **Adaptive Treatment Optimization**: Store historical actions in MongoDB Atlas → retrain dosage optimization model with new outcomes using reinforcement learning techniques if advanced.
- **AI Pest/Disease Forecasting**: Use weather API + LSTM model to forecast outbreaks. Implement in ai/models/forecasting.py with time-series data stored in MongoDB Atlas yield_forecasts collection.
- **Satellite + Drone Fusion**: Process Sentinel-2 via Google Earth Engine or alternatives; fuse with drone images in fusion_pipeline.py using libraries like rasterio for alignment. Store metadata in MongoDB Atlas images_metadata collection.
- **Farm Health Score**: Weighted index (disease % + soil + rainfall). Compute in backend/services/health_score.py and store in MongoDB Atlas farms collection for dashboard display.
- **AI Marketplace Integration**: Web scrape suppliers using Selenium and store in MongoDB Atlas market_prices collection for integration into recommendations.
- **Gamification & Community**: Reward farmers for consistent use (Streamlit/Gradio leaderboards). Track points in MongoDB Atlas farmer_feedback collection and display top users.
- **Climate Analytics**: Show sustainability metrics (CO , fertilizer usage) using open APIs like Carbon Interface API. Store results in MongoDB Atlas for analytics.
- **Voice Interfaces**: Use Whisper (offline STT) or Vosk for local languages (Yoruba, Hausa, Igbo). Store voice command logs in MongoDB Atlas for analysis.
- **AR Overlays**: Use AR.js/ARCore to guide farmers visually on infected leaf regions. Store AR metadata in MongoDB Atlas images_metadata collection.
- **CI/CD for Models**: GitHub Actions checks accuracy before pushing models live. Store model metadata in MongoDB Atlas for tracking.

Governance, Compliance & Reliability

Licensing & Provenance

When: Every time you add or update datasets.

How:

- Maintain manifest.csv with source, license, date, sha256 checksum. Use pandas to generate and validate, store metadata in MongoDB Atlas crop_monitoring collection.
- Use DVC for dataset versioning. Commands: dvc add data/, dvc push.
- Require farmer consent forms for uploads. Store digitally in MongoDB Atlas users collection with timestamps.

Annotation & Label QA

When: Before training any supervised model.

How:

- Use CVAT or Roboflow for annotations. Export in COCO or YOLO formats, store in MongoDB Atlas images_metadata collection.
- Set clear guidelines: resolution (min 1024x1024), label taxonomy (e.g., disease classes from PlantVillage).
- Ensure inter-annotator agreement 0.75 (Cohen's Kappa). Compute using scikit-learn's cohen_kappa_score.

Data Validation

When: Before training and deployment.

How:

- Add Great Expectations tests: correct file counts, schema checks. Define expectations in great_expectations/expectations/. Validate MongoDB Atlas data.
- Run pytest in CI/CD pipelines. Integrate with GitHub Actions for automated validation.

Model Governance & Cards

When: At each model release.

How:

- Store model card (model-card.md) with intended use, limitations, metrics (accuracy, F1-score). Save in MongoDB Atlas crop_monitoring collection.
- Track in MLflow registry. Use mlflow.log_artifact for cards, sync metadata to MongoDB Atlas for multi-user access.

Drift Detection & Retraining

When: Continuously after deployment.

How:

- Use Evidently AI to detect distribution drifts. Generate reports in monitoring/evidently/, store in MongoDB Atlas system_logs collection.
- Retrain if macro-F1 drops below baseline. Automate with cron jobs or Airflow, update models in MongoDB Atlas.

Security, Privacy & Compliance

When: Always in production.

How:

- Store secrets in Vault/GCP Secret Manager. Access via API in code.
- Encrypt data (AES-256 at rest, TLS in transit). Use MongoDB Atlas client-side field-level encryption.
- Implement data retention policies (max 2 years). Use MongoDB Atlas queries to purge old data.

Observability & SLOs

When: During production monitoring.

How:

- Set latency p95 <200ms, uptime 99.5%.
- Monitor with Prometheus, Grafana, OpenTelemetry. Store metrics in MongoDB Atlas system_logs collection.

CI/CD & Testing Gates

When: Before merging to production.

How:

- GitHub Actions → Run tests (pytest, lint with black/flake8).
- Block deployment if model accuracy < baseline. Use conditional steps in workflows.
- Canary rollout with rollback triggers. Implement using Kubernetes deployments.

Reproducibility & Backups

When: Every training run & release.

How:

- Pin seeds, dependencies (requirements.txt, pyproject.toml). Use random.seed(42) in code.
- Version datasets with hashes. Use DVC or Git LFS, store metadata in MongoDB Atlas.
- Backup MongoDB Atlas weekly (RPO=1 day, RTO=4h). Use mongodump for exports, automated via cron.

Knowledge & Skills Roadmap

Mathematics & Statistics

- Linear Algebra, Calculus → For CNN/LSTM backpropagation. (Resource: Gilbert Strang, MIT OCW lectures on YouTube or website.)
- Probability & Statistics → For risk scoring & uncertainty. (Resource: ”Introduction to Statistical Learning” by James et al., free PDF available.)

- Information Theory → For model optimization (entropy, KL). (Resource: ”Elements of Information Theory” by Cover & Thomas.)

Machine Learning

- Trees, Random Forests → Market pricing, yield risk. (scikit-learn docs: https://scikit-learn.org/stable/modules/ensemble.html)
- Boosting (XGBoost, LightGBM) → Credit scoring, yield models. (Kaggle tutorials: Search for XGBoost on Kaggle kernels.)
- Clustering → Group farms by conditions. (Use KMeans from scikit-learn.)

Deep Learning

- CNNs (ResNet, EfficientNet) → Crop disease detection. (PyTorch/TensorFlow tutorials: Official docs on transfer learning.)
- Transformers (ViT) → High-performance image classification. (Hugging Face Transformers library.)
- LSTM/GRU → Weather & yield forecasting. (Keras sequential models.)

MLOps

- MLflow → Experiment tracking. (Docs: https://mlflow.org/docs/latest/index.html)
- Docker/Kubernetes → Deployment. (Official tutorials for containerization.)
- Prometheus/Grafana → Monitoring. (Setup guides on official sites.)

Edge AI

- ONNX, TFLite → Compress & deploy on phones. (Conversion scripts in TensorFlow/PyTorch.)
- Pruning, Quantization → Reduce model size. (Use torch.nn.utils.prune.)

Compliance

- GDPR basics → Data privacy. (Read EU GDPR documentation.)
- OAuth2/JWT → Secure farmer login. (Use Authlib or FastAPI Users.)

Frontend Details

The frontend provides user-friendly interfaces for different stakeholders. It is built to be responsive, accessible, and integrated with backend APIs.

Technologies and Frameworks

- **Streamlit**: For rapid development of dashboards and prototypes. Ideal for data scientists to create interactive apps quickly.

- **Next.js with React**: For production web applications. Supports server-side rendering, API routes, and static generation for performance.
- **React Native**: For cross-platform mobile apps (iOS/Android). Enables native camera access and offline capabilities.
- **UI Components**: Material-UI (MUI) for consistent design, or Tailwind CSS for custom styling.
- **Visualization Libraries**: Chart.js for charts, Leaflet.js or Mapbox for maps displaying farm locations and satellite overlays.
- **State Management**: Redux or MobX for complex states in Next.js/React Native.
- **AR and Voice**: ARCore/ARKit for mobile AR, Web Speech API or Vosk.js for voice in web.

Key Components and Pages

- **Login/Register**: OAuth2 integration with email/phone verification.
- **Farmer Dashboard**: Image upload form, real-time recommendations, notifications feed, farm health score visualization.
- **Cooperative Management**: Multi-user views, aggregated analytics, group leaderboards for gamification.
- **Enterprise Console**: Compliance reports, scalability metrics, integration settings for supply chains.
- **Mobile Specific**: Camera integration for direct captures, offline mode with local storage (SQLite via Expo SQLite), sync when online to MongoDB Atlas.

Code Structure

Located in src/frontend/.

- streamlit_app/: dashboard.py for main app, pages/ for multi-page setup.
- nextjs_app/: pages/ for routes (e.g., index.js for home), components/ for reusable UI (e.g., ImageUploader.js).
- reactnative_app/: App.js entry, screens/ for views (e.g., DashboardScreen.js), services/ for API calls.

Example Code Snippet (Streamlit Upload)

```
import streamlit as st
import requests

st.title("Crop␣Image␣Upload")
uploaded_file = st.file_uploader("Choose␣an␣image...", type=["jpg", "png"])
if uploaded_file is not None:
    response = requests.post("http://backend/api/inference", files={"file":
         uploaded_file})
    st.write(response.json()["recommendation"])
```

Deployment and Testing

- Deploy Streamlit on Heroku or AWS; Next.js on Vercel; React Native via Expo or direct builds.
- Testing: Jest for unit tests, Cypress for E2E tests.
- Accessibility: Ensure WCAG compliance with ARIA labels.

Backend Architecture (Atlas Integrated)

The backend manages business logic, API endpoints, and background processing, with MongoDB Atlas as the primary database.

Technologies and Frameworks

- **FastAPI**: Asynchronous API framework for high performance.
- **Celery**: Task queue for asynchronous jobs like model training or scraping.
- **Database Connections**: MongoDB Atlas via pymongo for centralized storage.
- **Workers**: Dedicated scripts for compute-intensive tasks (e.g., image processing with OpenCV).
- **Authentication**: FastAPI Security with JWT, dependencies for role-based access (farmer, admin).

MongoDB Atlas Collections

- **crop_monitoring**
  - images_metadata: Stores image paths, labels, GPS, drone info.
  - farmer_feedback: User feedback for retraining and recommendation refinement.
  - market_prices: Scraped or API-based market data.
  - loan_requests: Microfinance and credit scoring data.
  - notifications: SMS/email/WhatsApp logs.
  - yield_forecasts: Predicted crop yields.
  - system_logs: Monitoring and drift detection logs.

Use Cases

- Query images /

  $_{m}etadatafordashboarddisplay.Collectfarmer_{f}eedbackinreal - timeforactivelearning.$

- Update market$_{p}ricesfordynamicpricingmodels.Trackloan_{r}equestshistoryandrepayment.$
- Monitor recommendation usage via notifications.

Key Endpoints and Services

- **/api/auth**: Login, register, token refresh.

- **/api/upload**: Image upload, trigger inference, store metadata in MongoDB Atlas images_metadata.
- **/api/recommendations**: Fetch personalized suggestions from MongoDB Atlas based on user ID.
- **/api/forecast**: POST with farm data, store results in MongoDB Atlas yield_forecasts.
- **/api/notifications**: Send alerts, log in MongoDB Atlas notifications collection.
- **/api/market**: Get current prices from MongoDB Atlas market_prices, historical trends.
- **Background Services**: Scraper worker for market data, inference worker for models.

Code Structure

Located in backend/.

- api/: main.py for app, routers/ for endpoints (e.g., inference.py).
- workers/: tasks.py for Celery tasks.
- services/: Business logic (e.g., recommendation_service.py).
- models/: Pydantic models for request/response validation.

Example Code Snippet (FastAPI with MongoDB Atlas)

```
from fastapi import FastAPI, UploadFile, File
from pymongo import MongoClient
import tensorflow as tf

app = FastAPI()
client = MongoClient("mongodb+srv://<user>:<pass>@cluster0.mongodb.net/")
db = client.crop_monitoring

@app.post("/api/inference")
async def inference(file: UploadFile = File(...)):
    content = await file.read()
    img = tf.image.decode_image(content)
    prediction = model.predict(img)
    result = {"disease": prediction.argmax(), "confidence": float(
       prediction.max())}
    db.images_metadata.insert_one({"image": file.filename, "result": result
       })
    return result
```

Deployment and Testing

- Run locally with uvicorn backend.api.main:app –reload.
- Scale with Gunicorn, deploy in Docker containers.
- Testing: Pytest for APIs, Locust for load testing.

Edge & Offline Devices

This section details the strategy for edge computing and offline functionality, ensuring usability in remote areas.

Edge Models

- **TFLite / ONNX**: Deploy compressed models on smartphones, Raspberry Pi, or drones for local inference.
- **Implementation**: Use tflite_runtime or ONNX Runtime for inference. Convert models in notebooks/export.py.

Caching

- **SQLite**: Local storage on edge devices for offline data (images, predictions, recommendations).
- **JSON**: Alternative lightweight format for simple metadata if SQLite is too heavy.
- **Sync to MongoDB Atlas**: When network is available, push SQLite/JSON data to Atlas collections (images_metadata, farmer_feedback).

Offline Recommendations

- Deliver immediate recommendations using local TFLite/ONNX models.
- Store results in SQLite, sync to MongoDB Atlas when online.

Workflow Example

- Farmer captures image → save to SQLite with metadata.
- Run inference locally → store result in SQLite.
- On network availability → sync SQLite to MongoDB Atlas, clear synced rows.

Observability & CI/CD

This section ensures robust monitoring and deployment pipelines.

Monitoring

- **Prometheus + Grafana**: Track API latency, model performance, and system health. Store metrics in MongoDB Atlas system_logs.
- **OpenTelemetry**: For distributed tracing across services.

Model Tracking

- **MLflow**: Track experiments locally, sync metadata to MongoDB Atlas crop_monitoring for multi-user setups.
- **Model Cards**: Store in MongoDB Atlas with metrics and limitations.

CI/CD

- **GitHub Actions**: Run tests, linting, and model accuracy checks before deployment.
- **Workflows**: Defined in .github/workflows/ci-cd.yml. Block if accuracy < baseline.
- **Canary Rollouts**: Use Kubernetes for staged deployments with rollback triggers.

Alerts & Logs

- Store alerts in MongoDB Atlas notifications collection.
- System logs (errors, drifts) in MongoDB Atlas system_logs collection.

Implementation Roadmap

MVP (0–3 months)

- Dataset setup (PlantVillage). Download from Kaggle, organize in data/, store metadata in MongoDB Atlas.
- Train CNN (ResNet50). Use transfer learning in notebooks/train_cnn.ipynb.
- Build Streamlit app for farmers. Implement upload and inference in streamlit_app.py, query MongoDB Atlas.
- Feedback collection in SQLite (offline), sync to MongoDB Atlas farmer_feedback.

Phase 2 (3–6 months)

- Add yield forecasting (Prophet/LSTM). Script in ai/yield_forecast.py, store in MongoDB Atlas yield_forecasts.
- Add market pricing model. Scraper in backend/scrapers/market.py, store in MongoDB Atlas market_prices.
- Add microfinance scoring (logistic regression). Model in ai/finance/, store in MongoDB Atlas loan_requests.

Phase 3 (6–12 months)

- Integrate satellite + drone pipelines. Fusion script with GDAL/rasterio, store in MongoDB Atlas images_metadata.
- Launch cooperative dashboards. Advanced Streamlit or Next.js views fetching from MongoDB Atlas.
- Add gamification, voice interfaces. Leaderboards in frontend, Vosk integration, logs in MongoDB Atlas.

Enterprise Rollout (12+ months)

- Deploy full CI/CD. Complete workflows in .github/.
- Implement drift detection + retraining automation. Store metrics in MongoDB Atlas system_logs.

- Launch with NGOs & government programs. Prepare demo and documentation, all data centralized in MongoDB Atlas.

Free-Tier Friendly Notes

- **MongoDB Atlas Free-Tier (M0)**: 512MB storage, sufficient for MVP data and multi-farm prototype. Scales automatically for small datasets.
- **Google Colab / Kaggle**: Free GPUs for model training. Use notebooks/ for experiments.
- **Edge Devices + SQLite**: Offline-first with local storage, sync to MongoDB Atlas when online.
- **Cost Efficiency**: No payment or ATM verification needed for MVP; leverages free tiers of AWS, MongoDB Atlas, and open APIs.

Database & Offline-Edge Strategy

MongoDB Atlas (Central Database)

Purpose: Centralized cloud database for multi-user access, dashboards, analytics, and persistent storage.

Collections:

- images_metadata: Stores image paths, labels, GPS, drone info.
- farmer_feedback: User feedback for retraining and recommendation refinement.
- market_prices: Scraped or API-based market data.
- loan_requests: Microfinance and credit scoring data.
- yield_forecasts: Predicted crop yields.
- notifications: SMS/email/WhatsApp logs.
- system_logs: Monitoring, drift detection, and system metrics.

Usage: All multi-user, cross-device, or long-term analytics stored in MongoDB Atlas. Access via pymongo with connection string in .env.

SQLite (Local / Edge Cache)

Purpose: Temporary, local storage on mobile/edge devices (smartphones, Raspberry Pi, drones).

Why: Supports offline-first operations; avoids blocking farm use when there is no internet.

Example Data:

- Uploaded images before syncing to MongoDB Atlas.
- Local predictions and recommendations.
- Sensor readings or drone metadata.

Edge Workflow:

- Farmer captures image offline → save metadata in SQLite.
- Device computes local inference using TFLite/ONNX model → store in SQLite.
- When internet is available → sync SQLite entries to MongoDB Atlas (images_metadata, farmer_feedback).
- Clean SQLite after successful sync to manage storage.

Notes: MongoDB Atlas is the single source of truth; SQLite is a temporary offline buffer.

Combined Workflow

- **Edge Device / Offline Mode**:
  - Image + Metadata → SQLite.
  - Local TFLite/ONNX inference.
  - Recommendation / Feedback → SQLite.
- **Sync Process (when online)**:
  - SQLite rows → MongoDB Atlas.
  - Confirm success → Clear synced SQLite rows.
  - Dashboards / Backend read from MongoDB Atlas.

Benefits:

- Ensures offline-first usability for farmers.
- Maintains centralized data in MongoDB Atlas for analytics, dashboards, and retraining.
- Low-cost, free-tier compatible with MongoDB Atlas M0.

Satellite Data Alternatives & Local Simulation (Sentinel-2)

Options for Free Sentinel-2 Data Without Google Cloud Registration

Since Google Earth Engine registration is not possible, use these alternatives:

- **AWS Open Data (registry.opendata.aws/sentinel-2/)**: Public S3 buckets for Sentinel-2 data. Access via boto3: s3 = boto3.client('s3'); s3.download_file('sentinel-s2-l1c', path, local).
- **Copernicus Open Access Hub (scihub.copernicus.eu)**: Official ESA portal for free Sentinel downloads. Register (free), use API with sentinelsat: from sentinelsat import SentinelAPI; api.query(...); api.download().
- **USGS EarthExplorer (earthexplorer.usgs.gov)**: Free access to Sentinel-2 and Landsat datasets. Bulk download tools available.
- **Sentinel Hub (www.sentinel-hub.com)**: Free tier for browsing, analyzing, and downloading Sentinel-2 data. Use EO Browser or sentinelhub-py library.
- **SkyWatch EXPLORE**: Aggregates multiple sources, free for certain datasets.

Notes: These sources provide .SAFE or GeoTIFF formats without requiring cloud payments.

Terrestrial Data Alternatives

Local Simulation for MVP

If live Sentinel-2 access is unavailable:

- **Raster Images**:
  - Use sample .tif or .png bands from AWS/Copernicus datasets.
  - Store in data/satellite/ for local processing.
- **CSV Feature Extraction**:
  - Precompute indices like NDVI, EVI using rasterio or GDAL.
  - Save as satellite_features.csv for ML model input.
- **Edge Device Testing**:
  - Use sample CSV + TFLite model for offline yield prediction or disease risk scoring.
  - Ensures functionality without cloud connectivity.

Recommended MVP Strategy

- Start offline-first using SQLite + sample Sentinel-2 rasters or CSVs.
- Sync all offline outputs to MongoDB Atlas when network is available.
- Replace simulation with real satellite feeds when Google Earth Engine or other APIs become accessible.

APIs and External Integrations

All APIs and services required for data ingestion and notifications.

Core APIs

- **OpenWeatherMap API**: For real-time and forecast weather. Sign up for free API key, limit to 1,000 calls/day. Usage: requests.get(f"https://api.openweathermap.org/data/2.5/weather?lat=la = key"). Store in MongoDB Atlas yield_forecasts.
- **Twilio API**: SMS/WhatsApp. Free sandbox for testing, paid for production. Client: from twilio.rest import Client; client.messages.create(...). Log in MongoDB Atlas notifications.
- **Gmail API**: Email notifications. Use OAuth2 with google-auth library, store logs in MongoDB Atlas.
- **BeautifulSoup/Selenium**: For scraping market prices from agricultural sites. Store in MongoDB Atlas market_prices.
- **FAO APIs**: For soil data (SoilGrids). Free access via REST endpoints, store in MongoDB Atlas.
- **PubChem API**: For chemical info on pesticides/fertilizers, integrate into recommendations.

All Other Requirements

Libraries and Dependencies

Full requirements.txt:

```
fastapi==0.95.0
uvicorn==0.21.0
streamlit==1.20.0
tensorflow==2.11.0
torch==1.13.0
scikit-learn==1.2.0
xgboost==1.7.0
lightgbm==3.3.0
prophet==1.1.0
opencv-python==4.7.0
shap==0.40.0
lime==0.2.0
mlflow==2.1.0
dvc==2.45.0
great-expectations==0.15.0
evidently==0.2.0
prometheus-client==0.16.0
pymongo==4.3.0
celery==5.2.0
twilio==7.16.0
beautifulsoup4==4.11.0
selenium==4.8.0
rasterio==1.3.0
geopandas==0.12.0
onnx==1.13.0
tensorflow-lite-runtime==2.11.0
vosk==0.3.0
requests==2.28.0
pandas==1.5.0
numpy==1.24.0
matplotlib==3.7.0
sentinelhub==3.9.0
sentinelsat==1.1.0
boto3==1.26.0
```

Hardware and Infrastructure

- **Edge Devices**: Raspberry Pi 4 (for on-farm inference), drones like DJI with cameras.
- **Cloud**: AWS Free Tier (EC2 t2.micro, S3), MongoDB Atlas M0, alternatives like DigitalOcean.
- **Tools**: Git for version control, Docker for containerization, Kubernetes/Helm for orchestration, Terraform for IaC.

Testing and Quality Assurance

- Unit Tests: Pytest for functions, models.

- Integration: Test API chains with MongoDB Atlas.
- Performance: Measure inference time, scalability.
- Security: OWASP scans, dependency checks with pip-audit.

Additional Considerations

- **Cost Management**: Use free tiers of MongoDB Atlas, AWS, and open APIs.
- **Localization**: Support Yoruba, Hausa, Igbo in UI (React Intl).
- **Sustainability**: Optimize models for low energy on edge devices.

Summary

- **Training Pipeline**: Local images + CSV manifests, processed in Colab/Kaggle.
- **Backend & Multi-User Functionality**: MongoDB Atlas for centralized storage and analytics.
- **Edge & Offline**: TFLite/ONNX models + SQLite, synced to MongoDB Atlas.
- **Dashboards & Analytics**: FastAPI/Streamlit reading from MongoDB Atlas.
- **MVP Compatibility**: Fully free-tier compatible without payment or ATM verification.