PLANT DISEASE DETECTION & RECOMMENDATION SYSTEM

(7. Deployment)

1) DEFINE SCOPE
2) COLLECT DATA
3) PREPROCESS DATA
4) CHOOSE MODEL
5) MODEL TRAINING
6) EVALUATE THE MODEL
7) DEPLOYMENT
8) FARMER USUABILITY
9) GATHER FEEDBACK

1. How Applications Work Offline (General Principles)

Offline functionality relies on storing necessary data and logic locally on the device. Here are the main strategies:

- **Local Data Storage:** The application stores data (e.g., databases, files, images) on the device's storage. This allows the app to access the data even without an internet connection.
- **Caching:** Frequently accessed data is stored in a temporary storage (cache) on the device. When the user requests the data again, the app first checks the cache. If the data is present (a "cache hit"), it's retrieved from the cache (fast access) instead of fetching it from the network.
- **Local Logic/Processing:** The application's code and logic are designed to perform operations locally. Instead of sending requests to a server, the app processes data directly on the device.
- **Synchronization:** When the device comes back online, the app can synchronize any locally made changes with the server, ensuring data consistency.

2. How to make a model work offline?

For machine learning models, offline operation usually involves:

- **Model Deployment:** The trained model is deployed to the device. This might involve converting the model to a format suitable for the device (e.g., using **TensorFlow Lite** for mobile devices).
- **Local Inference:** The application uses the deployed model to make predictions locally on the device. This requires a runtime environment or library that can execute the model (e.g., **TensorFlow Lite interpreter**).

3. How to make some specific features work offline?

It's absolutely possible to make only some features of an application work offline. This is a common design pattern. Here's how:

- **Modular Design:** Design your application in a modular way, separating the features that require network access from those that can work offline.
- **Feature Flags:** Use feature flags (or feature toggles) to control which features are enabled or disabled. You can use these flags to dynamically switch between online and offline modes for specific features.
- **Conditional Logic:** Implement conditional logic in your code to check for network connectivity. If the network is available, use the online path for a feature. If not, use the offline path.

4. What are the tools and technologies helps to make an application run in offline?

Mobile Development (Android/iOS):

- **SQLite:** A lightweight database commonly used for local storage in mobile apps.
- **Room (Android):** A persistence library for Android that provides an abstraction layer over SQLite.
- **Core Data (iOS):** Apple's framework for managing persistent data in iOS apps.
- **TensorFlow Lite:** For deploying and running machine learning models on mobile devices.
- **ONNX (Open Neural Network Exchange):** A format for representing machine learning models, which can be used with various inference engines

Backend/Cloud:

- **Cloud Databases (Cloud Firestore, Realm):** Offer offline synchronization capabilities.

General Purpose:

- **JSON:** A common format for serializing and storing data.
- **Protocol Buffers:** Another format for serializing structured data, often used for performance reasons.

5. What are the steps to achieve offline mobile app for our project (CNN)?

Fo the following specifications (CNN, potentially SVM/LSTM, FastAPI API, mobile app) with a combined approach:

1. Model Training and Conversion:

- **Train your models (CNN, SVM, LSTM):** Train your models using your dataset of plant images. Make sure you evaluate them thoroughly using appropriate metrics.
- **Model Conversion (TensorFlow Lite):** Convert the best-performing model to TensorFlow Lite format (.tflite). TensorFlow Lite is optimized for mobile and embedded devices. This process can be done using the TFLiteConverter. If you're using SVM or LSTM, you might need to find suitable mobile-compatible libraries or frameworks, as TensorFlow Lite primarily focuses on neural networks. There are options like Core ML (for iOS) or other specialized libraries.

- **Model Size Optimization (Important):** Mobile devices have limited resources. Try to optimize your model size:
    - **Quantization:** Reduce the precision of the model's weights (e.g., from 32-bit floating point to 8-bit integers). TensorFlow Lite supports various quantization techniques.
    - **Pruning:** Remove less important connections in the neural network.
    - **Knowledge Distillation:** Train a smaller "student" model to mimic the behavior of a larger "teacher" model.

2. Mobile Application Development:

- **Choose a Platform (Android/iOS/Cross-Platform):** Decide which mobile platform you're targeting. If you want to support both, consider cross-platform frameworks like Flutter or React Native.
- **Integrate TensorFlow Lite Interpreter:** Include the TensorFlow Lite interpreter in your mobile app. This interpreter will load and run your .tflite model.
- **Image Preprocessing:** Implement the same image preprocessing steps (resizing, normalization, etc.) in your mobile app that you used during training. This is crucial for consistent results.
- **Model Inference:** Use the TensorFlow Lite interpreter to load the .tflite model and perform inference on the preprocessed image from the user.
- **Disease/Crop Identification and Recommendations:**
    - The model's output will be the predicted disease (and potentially the crop).
    - Store a database (or use a local file) of disease/crop combinations and their corresponding fertilizer/pesticide recommendations within your app.
    - Based on the model's prediction, retrieve and display the appropriate recommendations.
- **Offline Data Storage:** Store the disease/crop/recommendation database locally in your app (e.g., using SQLite on Android, Core Data on iOS, or similar mechanisms in cross-platform frameworks).
- **User Interface:** Design a user-friendly interface for image upload, displaying predictions, and showing recommendations.

3. FastAPI API (Online Features - Optional):

- **API Endpoints:** Your FastAPI API can still be useful for optional online features:
    - **Model Updates:** You could potentially have a mechanism to update the .tflite model on the device periodically (if you release a new version).
    - **Data Collection/Feedback:** Allow users to provide feedback on the recommendations, which you can use to improve your model.
    - **Advanced Features:** Any features that are too resource-intensive for the mobile device (e.g., complex image analysis, accessing large datasets) can be handled by the API.

4. Offline Operation:

- **No Network Check:** Your mobile app should be able to function fully even when there's no network connection. All the model inference and recommendation lookups should happen locally.
- **Data Synchronization (If needed):** If you have any data that needs to be synchronized (e.g., user feedback), implement this as an optional feature when the device comes back online.

Technology Choices (Examples):

- **Android:** Java/Kotlin, Android Studio, TensorFlow Lite, SQLite, Room
- **iOS:** Swift/Objective-C, Xcode, Core ML (or TensorFlow Lite), Core Data
- **Cross-Platform:** Flutter (Dart), React Native (JavaScript), TensorFlow Lite, SQLite (or similar)

Simplified Workflow (Offline):

1. User uploads image to mobile app.
2. App preprocesses the image.
3. App loads the .tflite model (if not already loaded).
4. App runs inference using the model.
5. App retrieves recommendations from the local database based on the prediction.
6. App displays the results and recommendations to the user.

Key Challenges:

- **Model Size:** Optimizing the model size for mobile is crucial.
- **Performance:** Mobile devices have less processing power. Ensure your model and app are performant.
- **Data Storage:** Managing the local database of recommendations efficiently.
- **Platform-Specific Development:** Dealing with the intricacies of mobile development (especially if you're targeting multiple platforms).

This detailed breakdown should help you structure your project for offline functionality. Remember to focus on model optimization, mobile app development best practices, and thorough testing. Start with a minimal prototype and gradually add features.

Reference:

- Caching Best Practices | Amazon Web Services
- What is Offline Application and How Does it Work?
- Introduction to Offline Databases: What they are? What they are NOT?
- Simplify Offline Data Collection With Offline Mobile Forms

6. **Keys things to note:**
    - Synchronization mechanisms
    - Local storage
    - Cache essential data
    - Unbroken access to crucial functions
    - Service Workers and Progressive Web Apps (PWAs)

- Edge ML-optimized models and lightweight variants

7. **Useful resources:**
    - **Keras 3 API documentation**
    - **Keras: Deep Learning for humans**
8. **What are the features to provide in the application?**
    - Soil nutrient deficiency detection: By sampling plant tissues at critical growth stages
    - Forecast of pest/disease: Based on factors such as RH, temperature and other agronomic parameters, warnings are provided.
    - Advisory based upon Weather parameters: Wind direction, Temperature, Relative humidity, Rainfall prediction, Cloud cover.
    - Identification of pests and disease infestation (& analysis of nutrient deficiency).
    - Recommendations on what & how much to apply.
    - Scientific crop calendar
    - Satellite-based soil moisture estimation.
    - Irrigation advisory: Schedule for irrigation management.
    - Weather Forecast.
    - Crop health monitoring: by getting indices like NDVI
    - Land Surface Water Index (LSWI): 0 means soil is dry and 1 means soil is saturated.
    - Multilingual support.
    - Sharing with Experts.
    - Inventory management & Cost control.
    - Motion alerts.
    - Product Deals.
    - Enforce Climate-smart Agriculture Practices
    - Track and Report Sustainability Goals
    - Assess Cover Crops with Cover Crop Monitoring Models
    - Yield Prediction
    - Commodity Prices
    - Market Trends
    - Buyer/Seller Connections
    - Educational Materials
    - Communication Tools
    - Data Sharing
    - Automatic weeding and harvesting
    - Agrobot – LLM support for farmers

Referenced Applications:

- Sat2Farm_Brochure
- Agricultural machinery management via App | xFarm
- Plantix | #1 FREE app for crop diagnosis and treatments
- Cropin Cloud for Sustainability
- Normalized difference vegetation index - Wikipedia
- Revolutionizing Agricultural Marketing: Farmonaut's Guide to Smart Farming and Real-Time Market Data in India –
- AI in Agriculture and Farming: Revolutionizing Crop Growth - Intellias

9. What are the features to be in offline?

- Priority:
  - Soil nutrient deficiency detection
  - Identification of pests and disease infestation.
  - Recommendations on what & how much to apply.
- Secondary:
  - Scientific crop calendar.
  - Irrigation advisory: Schedule for irrigation management.
  - Inventory management & Cost control.

Reference:

- Use Google Calendar offline - Android - Google Calendar Help
- ResearchGate

10. What are the data to be in local for the offline features?

- Model for the crops – in a dedicated directory.
- Data about
  - pests/disease/nutrient deficiency caused in that crop,
  - remedies for each disease.
  - General nutrients needed for the crop and amount of nutrients.
  - General info about soil, soil types & its nutrients, recommended crops.
  - Nutrient deficiency, its severity, what nutrient it is, what fertilizer should be applied, how much to apply based on severity.
  - pathogen name, severity, what pesticide must be applied, how much to apply based on severity.
  - Crop name, how much water it needs, irrigation type, recommended schedule, user's schedule, applied or not.
  - Inventory bought, cost of the inventory, inventory type, budget, date of bought.
  - Crop name, disease name, disease causes, cause type.
  - If other type, causes, remedies.

11. Structure of Data in Database:

Data Storage Structure:

- **Model Files (for image recognition):**
  - Store your trained machine learning models (e.g., TensorFlow Lite models) in a dedicated directory. These will be used for pest/disease/nutrient deficiency identification from images.
- **Databases:**
  - Use a lightweight, mobile-friendly database like SQLite. This is excellent for structured data and allows for efficient querying. You can have multiple tables within the database.
- **Crops Table:**
  - crop_id (INTEGER PRIMARY KEY)
  - crop_name (TEXT)
  - water_needs (TEXT) (e.g., "High," "Medium," "Low")
  - irrigation_type (TEXT) (e.g., "Drip," "Sprinkler," "Flood")
  - general_nutrients (TEXT) (JSON format to store multiple nutrient needs)
  - recommended_schedule (TEXT) (JSON format for schedule)
- **Diseases Table:**
  - disease_id (INTEGER PRIMARY KEY)
  - disease_name (TEXT)
  - cause_type (TEXT) (e.g., "Pest", "Nutrient", "Environmental")
  - disease_details_id (INTEGER) (Foreign Key)
- **NutrientDeficiencies Table:**
  - disease_details_id (INTEGER PRIMARY KEY)
  - crop_id (INTEGER, FOREIGN KEY referencing Crops)
  - nutrient (TEXT) (e.g., "Nitrogen," "Phosphorus")
  - symptoms (TEXT)
  - fertilizer_recommendation (TEXT) (JSON format for multiple fertilizers)
  - application_amount (TEXT) (JSON format for different severity levels)
- **PestsAndDiseases Table:**
  - disease_details_id (INTEGER PRIMARY KEY)
  - crop_id (INTEGER, FOREIGN KEY referencing Crops)
  - pathogen_name (TEXT)
  - symptoms (TEXT)
  - causes (TEXT) (JSON format for multiple causes)
  - remedies (TEXT) (JSON format for multiple remedies)
  - pesticide_recommendation (TEXT) (JSON format for multiple pesticides)
  - application_amount (TEXT) (JSON format for different severity levels)
- **EnvironmentalIssues Table:**
  - issue_id (INTEGER PRIMARY KEY)
  - issue_name (TEXT) (e.g., "Water Stress", "Heat Stress", "Frost")

    - details (TEXT) (Description of the environmental issue)
    - remedies (TEXT) (JSON format for multiple remedies)
- **Inventory Table:**
    - inventory_id (INTEGER PRIMARY KEY)
    - item_name (TEXT)
    - item_type (TEXT) (e.g., "Fertilizer," "Pesticide", “Equipment”)
    - quantity_bought (REAL)
    - cost (REAL)
    - purchase_date (TEXT) (ISO 8601 format: YYYY-MM-DD)
- **IrrigationSchedule Table:**
    - schedule_id (INTEGER PRIMARY KEY)
    - crop_id (INTEGER, FOREIGN KEY referencing Crops)
    - user_schedule (TEXT) (JSON format)
    - applied_dates (TEXT) (JSON format)
- **Files Table:**
    - file_id (INTEGER PRIMARY KEY)
    - file_name (TEXT)
    - file_path (TEXT) (JSON format)
    - created_date (TEXT) (JSON format) (ISO 8601 format: YYYY-MM-DD)

Data Formats (Recommendations):

- **JSON:** Use JSON (JavaScript Object Notation) to store complex data like lists of nutrients, fertilizer recommendations, pesticide recommendations, causes, remedies, schedules, etc., within the database fields. This allows you to store structured data within a single database cell.
- **ISO 8601 Dates:** Use the ISO 8601 date format (YYYY-MM-DD) for storing dates. This is a standard and unambiguous format.

12. What database to choose for local storage and what to consider when choosing?

SQLite is the best choice.

What to Consider When Choosing:

- **Data Size:** For the amount of data, (tables, images, models), SQLite will be perfectly fine. It can handle databases of considerable size.
- **Concurrency:** If your app anticipates extremely high concurrent access to the database (many users making changes simultaneously), SQLite might have limitations. However, for a typical agricultural app used by individual farmers, concurrency is unlikely to be a major concern.
- **Complex Queries:** While SQLite supports standard SQL queries, very complex or large joins might be slower compared to more powerful database systems. However, for the kind of queries you've described (lookups based on disease name, crop, etc.), SQLite should perform well.

- **Full-Text Search:** If you need advanced full-text search capabilities (searching within large text fields), SQLite has some built-in options but might not be as powerful as dedicated search engines. Consider whether full-text search is a critical requirement.
- **Platform Compatibility:** SQLite is very well-supported on both Android and iOS, so you won't have any platform-specific issues.
- **Model Storage:** SQLite can store your models as BLOBs. However, consider if loading the models directly from the database will be the most performant approach, or whether it might be more efficient to store them as separate files on the device's file system and just store the file paths in the database.

13. What is SQL lite?

SQLite

- a lightweight, embedded database that is widely used in mobile application development.
- stores data in the form of a tabular, which means rows and columns.
- compatible with mobile platforms such as Android and iOS.

Key Features

- Zero Configuration
- Cross-Platform Support
- ACID compliance
- SQL compatibility (SQL92 standard)
- Low memory requirement
- High Performance

Reference:

- Top 7 Databases for Mobile App Development in 2025 - GeeksforGeeks

14. What can be the limitation of storage in mobile app?

Let's say you have:

- 2 quantized models (5 MB each) = 10 MB
- 1000 images (compressed to an average of 50 KB each) = 50 MB
- Database (estimated) = 5 MB

Total = 65 MB (which is a good target size).

Reference:

- Maximum App Size Limits: A Comprehensive Guide

15. What are the ways to optimize the data storage in mobile device?

Reference:

- Effective Methods for Managing Data Storage Limits in Mobile Apps | MoldStud

16. How to store the model in the user's local storage?

- Store Model Files: Place your .tflite model files in a dedicated directory within your app's assets or data directory.
- Store File Paths in Database: In your database (e.g., in a "Models" table), store the *paths* to these model files.
- Load Models: When your app needs to use a model, retrieve the file path from the database and then load the model from that path.

17. How to integrate SQL lite with PostgreSQL?

18. How to sync once online?

19. How to store the new images?

- Process & compress image Files: Resize, in SVG format, ...
- Store image files: Place image files in a dedicated directory within your app's assets or data directory.
- Store File Paths in Database: In your database, store the paths to these model files.
- Load Models: When your app needs to use a model, retrieve the file path from the database and then load the model from that path.

20. How to trigger training once 10k images are stored in the cloud?

21. How to train with new images?

22. How to compare the model?

23. How to overwrite the existing model?

24. What is the efficient way for storing and processing, (cloud or server or both)?

25. How to integrate cloud and server?