Final report — Feedback Flow (RAG + Agent + Crop & Disease pipelines)

Author: M Ahmed Imtiaz
Project: Feedback Flow — Hybrid RAG + Tools Agent for crop & disease advice
Date: November 2025

1. Problem statement

Smallholder farmers and agronomists need rapid access to reliable agricultural guidance: what to plant given soil/weather constraints, and how to detect and treat plant diseases from images and symptoms. This project builds an agentic system that routes user queries to purpose-built tools: (1) a crop recommendation model (tabular ML), (2) a disease detector (CNN), and (3) a Retrieval-Augmented Generation (RAG) knowledge base for agricultural advice. An agent (simple rule-based + explainability) selects the tool, runs it, and returns structured outputs including reasoning for tool selection.

Goals:

- Provide a single entry point (API/CLI) that accepts natural-language queries and optional images.
- Route queries to appropriate tool(s) with explicit reasoning and deliver structured results.
- Evaluate each component and the agent routing, produce a 4–6 page report and reproducible repo.

2. Datasets

- Crop dataset (synthetic/small sample for demo): tabular features (soil texture proxies, rainfall, planting stage, pH) with crop labels (wheat, maize, rice, potato). For demo, the repo contains `data/crop_test.csv` (test examples).
- Disease dataset: labeled leaf images for several disease categories + healthy. For demo, there are model artifacts in `models/` and small test paths in `data/disease_test.csv`.
- RAG knowledge base: domain documentation, FAQs, and curated agricultural text converted to chunks; precomputed embeddings and FAISS index stored in `models/rag_embeddings.npy` and `models/rag_faiss.index`. `rag_chunks_meta.jsonl` lists chunk metadata.
- Agent test set: 40–60 hand-labeled queries (`data/agent_test.csv`) where each query has a `gold_tool` label.

(When training at scale, use public datasets such as PlantVillage for disease images and national/crop-agency soil datasets for crop recommendation. Training notebooks included in `notebooks/`.)

3. Preprocessing & modeling

Crop predictor (tabular)

- Preprocessing: feature scaling (StandardScaler), categorical encoding where needed, feature selection (use `feature_columns` metadata). Saved model artifact `models/crop_predictor.pkl` includes `{ 'model': estimator, 'scaler': scaler, 'feature_columns': [...] }`.
- Model: XGBoost/RandomForest classifier saved with joblib and wrapped in a small predict wrapper. Output: recommended crop and confidence score.

Disease detector (CNN)

- Base architecture: EfficientNet (transfer learning).
- Preprocessing: images resized to 224×224, normalized with ImageNet statistics, minimal augmentation during inference. Model artifacts: `disease_efficientnet_final.h5` (weights), checkpoint variants for head/fine-tune present in `models/`.
- Inference: returns predicted class, confidence, and optionally class probabilities.

RAG (FAISS + transformer generator)

- Chunks: documents split to 200–400 token chunks, stored in `rag_chunks_meta.jsonl`.
- Embeddings: SentenceTransformers used to precompute `rag_embeddings.npy`.
- Retrieval: FAISS index `rag_faiss.index` for top-K retrieval.
- Generator: local small LLM fallback (distilgpt2/gpt2) used if OpenAI not available. The RAG pipeline concatenates retrieved chunks into context and prompts the LLM with the question.

Agent logic

- Simple rule-based router:
  - If query mentions `leaf`, `plant`, `disease` or an image is attached → `disease_detector_tool`.
  - If query asks "what to plant", mentions `soil`, `weather`, or crop selection → `crop_predictor_tool`.
  - Else → `rag_qa_tool`.
- For ambiguous cases, agent logs `reasoning_text` with the keyword matches and confidence.
- Output schema for the agent: `{ tool_used, tool_input, tool_output, reasoning_text }`.

4. Evaluation methodology & results

Metrics computed

- Crop model: accuracy, precision, recall, F1, confusion matrix, feature importances.
- Disease model: training/validation curves (loss/accuracy), final test accuracy, per-class precision/recall, inference time per image (ms).
- RAG: Hit rate (how often the gold-relevant chunk was in top-K), MRR (mean reciprocal rank), example human inspection, optional BLEU for short-response comparison.
- Agent: routing accuracy measured on a 40–60 query test set where each query has `gold_tool`. Also confusion matrix for routing errors and qualitative failure cases.

Sample numeric results (demo)

NOTE: replace these numbers with your actual outputs when generating the PDF.

- Crop predictor (test set):
  - Accuracy: 0.86
  - Precision (macro): 0.84
  - Recall (macro): 0.83
  - F1 (macro): 0.835
  - Top features: rainfall, pH, soil_texture_index
- Disease detector:
  - Final test accuracy: 0.92
  - Per-class precision: [healthy:0.94, early_blight:0.90, late_blight:0.88]
  - Avg inference time/image: 45 ms (CPU) — measure with `time.time()` around `model.predict`.
- RAG retrieval:
  - Hit rate @5: 0.78
  - MRR@5: 0.63
  - Qualitative: retrieved chunk example succinctly contained the recommended pesticide mix.
- Agent routing:
  - Routing accuracy on agent_test.csv (N=50): 0.88
  - Confusion matrix: show counts (crop vs disease vs rag)
  - Typical error: ambiguous queries referencing both “leaf” and “fertilizer” — in these cases we prefer to call disease_detector_tool by current rules.

Figures/tables to include (attach to the PDF)

- Crop confusion matrix (table + heatmap).
- Disease training/validation curves (accuracy & loss).
- RAG hit-rate vs topK plot (1..10).
- Agent routing confusion matrix & 6 qualitative example queries with gold vs predicted and reasoning_text.

5. Agentic flow diagram (text + diagram instructions)

Flow (brief):

1. Receive request (question + optional file).
2. Preprocess: if file attached, store temp and mark `has_file=True`.
3. Agent router: keyword-scan or rule-based classifier → selects tool.
4. Tool runs (disease_detector_tool / crop_predictor_tool / rag_qa_tool).
5. Response composed: include `tool_used`, `tool_input`, `tool_output`, `reasoning_text`.
6. Logging: store request, selected tool, tool output, timestamp (for future human-in-the-loop tuning).

(You can create a small flow diagram in PowerPoint/Diagrams.net: Request → Router → Tool branches → Response.)

6. Limitations & future work

- Data limitations: demo datasets are small; production requires larger, geographically diverse datasets (PlantVillage + local field data).
- Model performance: disease detector may be domain-limited (lighting, background). Consider segmentation + higher-res images.
- RAG hallucinations: local LLM can hallucinate. Use conservative reply templates and include source citations from retrieved chunks.
- Agent routing: rule-based routing is brittle; upgrade to a learned classifier that uses embeddings of the query to route better.
- Deployment: for production, move large models to a model server (TorchServe/TensorFlow Serving) and use GPU inference.
- Privacy / ethics: avoid storing user images without consent; add opt-in/opt-out and data deletion instructions.

7. How to reproduce & run

- See README.md for environment install, sample run commands, and how to run the `eval/` scripts and `main.py`.
- To reproduce evaluations run:
- `python -m venv venv`
- `.\venv\Scripts\Activate.ps1`
- `pip install -r requirements.txt`
- `python -m uvicorn main:app --reload`
- `python eval/eval_crop.py --test data/crop_test.csv --out reports/eval/crop`
- `python eval/eval_disease.py --test data/disease_test.csv --out reports/eval/disease`
- `python eval/eval_rag.py --test data/rag_test.csv --topk 5 --out reports/eval/rag`
- `python eval/eval_agent.py --test data/agent_test.csv --venv .\venv\Scripts\python.exe --out reports/eval/agent`

8. Deliverables included in repo

- Code: `src/` , `main.py`, `eval/` scripts.
- Notebooks: training notebooks and Kaggle notebook link (in `notebooks/`).
- Models: `models/` (or download links if too big).
- Reports: `reports/` including RAG examples, evaluation output, and the final PDF (this file).