FarmIQ: An Integrated Smart Farming And AI-Based Agricultural Decision Support System

Rajvardhan Bhosale[1], Pratik Gurav[2], Prem Nemade[3], Prasad Sarvade[4], Prof. M. B. Chougule[5]
[1,2,3,4]*Department of Computer Science and Engineering, MIT College of Railway Engineering and Research (MITCORER), Barshi, Maharashtra, India*
[5]*Project Guide, Department of Computer Science and Engineering, MIT College of Railway Engineering and Research (MITCORER), Barshi, Maharashtra, India*

***Abstract*—Farm record keeping and agronomic decision support are usually delivered by separate tools, which leaves smallholder farmers with fragmented information. This paper reports the design and implementation of FarmIQ, a Flask and SQLite web application that combines role-based farm-record management (crops, expenses, harvests, analytics, PDF reports and an administrator console with an audit log) with three decision-support modules of different kinds: a Random Forest Crop Advisor that uses seven numeric inputs; a rule-based Regional Crop Recommendation Engine for 35 Maharashtra districts, 22 crops and three seasons that ranks crops by a weighted score of historical suitability, climate, water and yield (weights 0.45, 0.25, 0.20 and 0.10); and a Plant Disease Detection module that wraps a third-party pre-trained image classifier without retraining. The Regional Engine is not machine learning, and its weights are implementation-defined. Evaluation is limited to software testing and a re-execution of the supplied training script: 19 unittest cases passed in 3.419 s, and the Crop Advisor reached 99.55% accuracy on a random 20% hold-out split of its 2,200-row table, which is not evidence of field accuracy. The Regional Engine and the disease module have not been validated, and no farmer field trials were conducted. The contribution is an integrated, documented engineering system together with an explicit account of the evidence that does and does not exist.**

***Index Terms*—smart agriculture, decision support system, Random Forest, crop recommendation, plant disease detection, role-based access control, Flask**

I. INTRODUCTION AND RELATED WORK

Digital technologies are reshaping agriculture, from sensing and data analytics to decision support delivered through web and mobile software [1], [2]. For an individual farmer two needs recur: a dependable record of what has been sown, spent and harvested, and timely guidance on what to grow and how to protect a crop. In practice these needs are served by unrelated tools. Record-keeping applications seldom offer agronomic guidance, whereas advisory tools usually ask for inputs afresh and keep no history of the farm. A team that wants to combine them must decide how independent, differently built components can coexist behind one authenticated interface.

Wolfert *et al.* reviewed big data in smart farming and identified data integration and ownership as open challenges [1], and Zhai *et al.* surveyed decision support systems for agriculture 4.0 and the difficulty of joining data collection, analysis and farmer-facing recommendations [2]. Liakos *et al.* classified machine-learning applications in agriculture [3], and van Klompenburg *et al.* reviewed crop-yield prediction, noting that results depend strongly on data availability and quality [4]. Pudumalar *et al.* proposed a crop-recommendation system over soil attributes that combines several learners [5], and Jeong *et al.* applied random forests to crop-yield prediction [6]. Random forests were introduced by Breiman [7] and are available in scikit-learn [8]; FarmIQ's Crop Advisor belongs to this family but is a small classifier over seven inputs. For image-based disease classification, Mohanty *et al.* reported 99.35% accuracy on held-out PlantVillage images [9], [10] while noting a marked drop on images taken under other conditions, Ferentinos reported accuracies above 99% on a larger multi-plant collection [11], and Barbedo showed that dataset size and variety strongly affect generalisation [12]. These are the published

results of other models and are not FarmIQ results. Regional crop ranking can use learned models, expert rules or scoring schemes against local evidence, and reanalysis products such as NASA POWER supply district-level climate series for the latter [13]. In the selective, non-systematic review reported here, no source was found that documents a scoring scheme identical to that of the Regional Engine; no claim of novelty is made.

II. PROBLEM STATEMENT AND CONTRIBUTIONS

The work addresses four engineering problems. (i) *Fragmented tools:* records and advice are rarely available in one place. (ii) *Record integrity:* crops, expenses and harvests must be isolated per farmer, remain consistent when a crop is deleted, and be summarised for reporting. (iii) *Regional suitability:* a single national-level classifier cannot reflect district-level history, climate and water conditions. (iv) *Honest integration:* a trained classifier, a rule-based scoring engine and a third-party neural network need separated interfaces and accurate labelling so that they are not mistaken for one another. The gap is therefore an engineering one; the paper does not assert that no comparable system exists.

The contributions are: (a) a description of a six-table, role-aware farm-record architecture with farmer-scoped identifiers and an audit log; (b) a transparent formulation of the Regional Engine's scoring, including its implementation-defined weights and an undocumented asymmetry; (c) an account of how three methodologically different decision modules are integrated without being conflated; and (d) a statement of the evidence that exists (19 automated software tests and one re-executed hold-out evaluation) and of what has not been evaluated. FarmIQ is presented as an integrated system and implementation study, not as a new machine-learning algorithm.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

A. Architecture and Data Model

FarmIQ is a single Flask application with 35 route declarations (3 public, 23 login-protected and 9 administrator-only) that render 28 Jinja2 templates and share one SQLite database through Flask-SQLAlchemy (Fig. 1). Requests pass through Flask-Login role checks and global CSRF protection (Flask-WTF). Two roles exist. Registration is public and always creates a Farmer, who manages only their own crops, expenses and harvests and uses analytics, reports and the decision-support modules. An Administrator additionally views all records, manages farmer accounts, and opens the financial overview, audit log and system-health pages. The database has six tables (user, crop, expense, harvest, farmer_sequence and audit_log; Fig. 2); expenses and harvests may refer to a crop through a nullable crop_id, which is set to NULL when the crop is deleted.

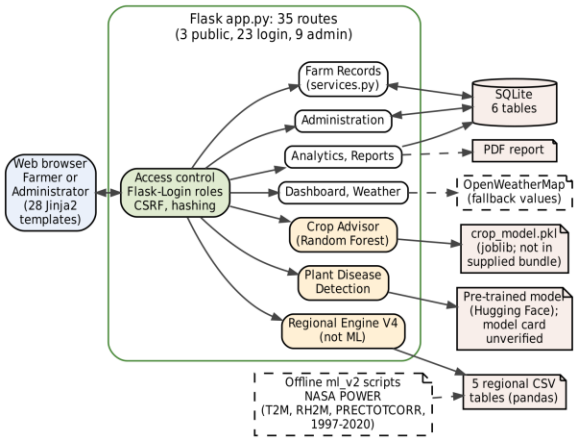


Fig. 1. Architecture of FarmIQ as implemented, verified against the source code.

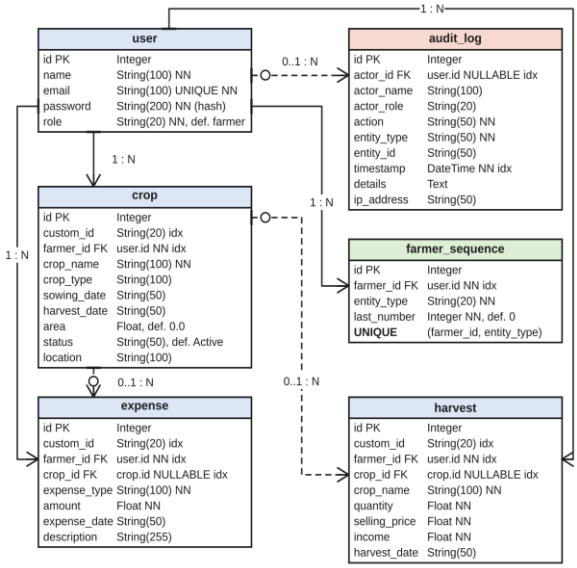


Fig. 2. Entity-relationship diagram of the six tables (consistent with models.py).

The decision-support routes do not read the farm-record tables: the modules are integrated at the interface and authentication level, not at the data level. The main software components are Flask 3.1.3, SQLAlchemy 2.0.50, scikit-learn 1.9.1, ReportLab, and PyTorch with Hugging Face Transformers for image inference, as pinned in requirements.txt; that file also lists several packages (Django, psycopg2, gunicorn, whitenoise, python-dotenv and tablib) that the application never imports.

B. Farm Management, Access Control and Reporting

Record routes call ownership helpers: a missing record gives HTTP 404 and a record owned by another Farmer gives HTTP 403 unless the user is an Administrator. Harvest income is computed on the server as quantity × selling price. Display identifiers (CROP-001, EXP-001, HAR-001) are allocated per farmer by a function that locks the farmer_sequence row for the (farmer, entity) pair, increments its counter and formats a three-digit suffix; it does not commit, so the counter and the new record are saved in one transaction and numbers are not reused. Whether the row lock protects against simultaneous requests on SQLite has not been tested. Passwords are stored as Werkzeug hashes, but registration applies no password-strength rule. Audit entries are written by a log_audit() helper for logins, registrations, record changes, report exports and account changes. The analytics page draws four Chart.js charts, and PDF reports are generated with ReportLab and scoped to the signed-in Farmer. The dashboard requests current weather from the OpenWeatherMap API with a 3 s timeout and falls back to fixed values on failure. The API key is hard-coded in app.py; it is deliberately not reproduced here and should be revoked and read from the environment.

C. Crop Advisor (Random Forest)

The Crop Advisor takes seven numeric inputs: nitrogen, phosphorus, potassium, temperature, humidity, pH and rainfall. The training script reads Crop_recommendation.csv (2,200 rows, 22 crop labels, 100 rows per label), makes an 80/20 stratified split with random_state 42 and fits RandomForestClassifier with 100 trees [7], [8]. The predicted class is the one with the highest-class probability averaged over the trees, and the route displays the three highest averaged probabilities. These values are the forest's mean class probabilities and are not calibrated measures of correctness. Input ranges are enforced only by the browser. The serialised model (crop_model.pkl) was not part of the supplied materials, and the project does not document the origin or licence of the dataset.

D. Regional Crop Recommendation Engine

The Regional Engine (version V4) is a weighted scoring engine with no trained model and no link to the Crop Advisor. The user supplies a location, an optional district and taluka, a season (Kharif, Rabi or Summer) and a water-availability level (High, Medium, Low or None). A location resolver normalises the text and matches it against a 44,810-row village lookup, returning success, ambiguous, needs-clarification or not-found; the district is then mapped to the name used in the scoring tables (Fig. 3). District-season climate is derived offline from NASA POWER monthly series (T2M, RH2M and PRECTOTCORR, 1997–2020) [13]. A crop is scored only if a profile exists for its district, crop and season with at least 10 historical records (MIN_LOCAL_RECORDS = 10) and a climate row exists; the tables cover 35 districts, 22 crops and three seasons.

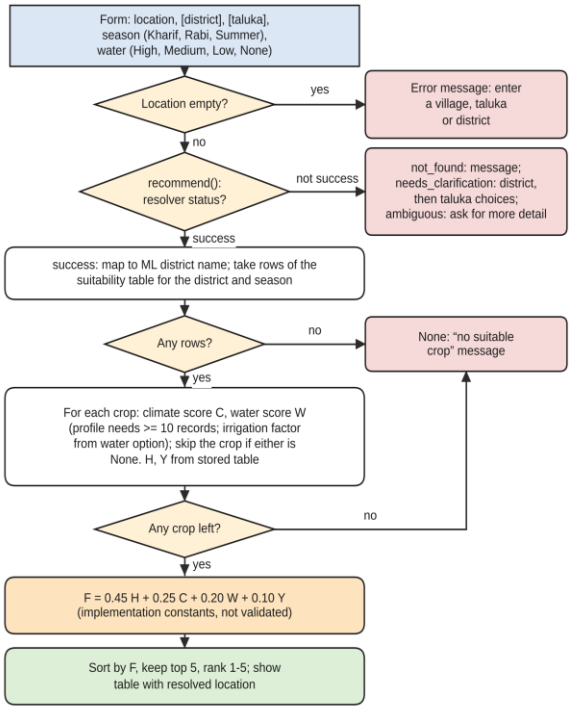


Fig. 3. Regional Crop Recommendation workflow: location resolution, candidate selection and scoring.

The final score of each candidate is

$F = 0.45\,H + 0.25\,C + 0.20\,W + 0.10\,Y$ (1)

where H, C, W and Y lie on a 0–100 scale (Table I). The weights, thresholds and factors are implementation-defined constants; they were not optimised or validated against any benchmark. H and Y are read from a stored table (crop_suitability_scores.csv) whose generating script is not in the supplied materials, so FarmIQ is described as reading these values, not computing them.

TABLE I Regional engine score components

Symbol	Weight	Component	Source in the implementation
H	0.45	Historical suitability	Stored suitability score for the district, crop and season
C	0.25	Climate	Temperature and rainfall against the crop's local P_{25}, median and P_{75}
W	0.20	Water	Rainfall band and irrigation factor of the chosen water option
Y	0.10	Yield	Stored yield score × 100

Let x be the district-season mean temperature or seasonal rainfall, P_{25}, P_{50} and P_{75} the crop's local profile percentiles, $w = \max(P_{75} - P_{25}, 0.01)$ and d the distance from the nearest band edge.

For x inside the band $P_{25} \le x \le P_{75}$, each climate sub-score is

$s = \max(80, 100 - 20|x - P_{50}|/w)$ (2)

and outside the band

$s = \max(0, 80 - k\,d/w)$ (3)

$C = 0.5\,s_T + 0.5\,s_R$ (4)

with $k = 80$ for temperature on both sides and for rainfall below P_{25}, and $k = 50$ for rainfall above P_{75}. The code gives no reason for this asymmetry. The water score uses the same seasonal rainfall and percentiles. Inside the band

$W = \max(80, 100 - 20|x - P_{50}|/w)$ (5)

and outside it

$W = \max(0, 100 - a \cdot \min(100, 50 + 50\,d/w))$ (6)

where a is the irrigation factor of the water option (None 1.00, Low 0.85, Medium 0.60, High 0.30) for rainfall below P_{25}, and $a = 1$ for rainfall above P_{75}, so the water-availability choice affects only the shortfall branch. Candidates are sorted by F and the top five are shown. Season boundaries are inconsistent across the project: the climate-aggregation script read by the engine uses Kharif June–October, Rabi November–February and Summer March–May, an earlier preprocessing script uses Rabi November–March and Summer April–May, and the user interface labels Rabi as October–March and Summer as March–June. The aggregation script governs the climate component and is treated here as the implementation; the discrepancy is unresolved.

E. Plant Disease Detection

Plant Disease Detection loads a pre-trained third-party image classifier from the Hugging Face hub (identifier mesabo/agri-plant-disease-resnet50) at start-up and uses it in evaluation mode. The FarmIQ team did not train or fine-tune it, and its architecture, training data, licence and accuracy are not documented in the project and are treated as unverified. The user selects one of six crops (Tomato, Apple, Grape, Corn, Pepper or Potato) and uploads a leaf image, which is converted to RGB, resized to 256, centre-cropped to 224 and normalised with ImageNet statistics. Only the logits of classes whose labels begin with the selected crop's prefix are retained before the softmax, so the reported confidence is relative to that crop's classes and is not an absolute probability that the image is a leaf of that crop. Up to three ranked labels are returned. A top-label confidence of at least 80 is reported as high and 60 to below 80 as moderate; below 60 a disease label is replaced by an "unable to confidently identify" message without advice. Advice text comes from a dictionary keyed by label. The code has no file-type or size check and does not reject non-leaf images.

IV. IMPLEMENTATION AND EXPERIMENTAL SETUP

The application was run locally from the project folder with python app.py, which creates the SQLite tables, seeds two demonstration accounts and starts the Flask development server with debug enabled. This is a development configuration and the system should not be regarded as production-ready; it has not been deployed to a server. Table II lists the test environment. The terminal capture in Fig. 4 shows a Windows PowerShell session but not the operating-

system version, Python version or date, so those entries are author-supplied and were not independently verified; installed package versions were not recorded.

TABLE II Test environment (author-supplied) and automated test result

Item	Value
Laptop	ASUS TUF Gaming F16 FX608JMR
Processor	Intel(R) Core (TM) i7-14650HX
Memory	16 GB RAM
Operating system	Windows 10 (author-supplied)
Python	3.13.7 (author-supplied)
Browser	Google Chrome (author-supplied)
Test date	18 November 2026 (author-supplied)
Test cases run	19
Passed; failed	19; 0
Execution time	3.419 s
Final status	OK

Testing used Python's unit test framework with the Flask test client. One class, FarmIQ Comprehensive Test Suite, holds 19 test methods run with python test_platform.py. The fixture copies instance/database. dB to a temporary file, enables testing mode, disables CSRF checking and sets the database URI to the copy; the tests therefore depend on the contents of that database (an Administrator and at least two Farmers), and CSRF protection is not exercised. The tests assert authentication and role behaviour; HTTP 403 for a Farmer on administrator URLs; isolation of one Farmer's crops, expenses and harvests from another's; per-farmer identifier allocation; administrator views; PDF generation scoped to the signed-in Farmer; the audit helper; HTTP 404 handling; page loads for dashboards, analytics and decision-support pages (the Crop Advisor POST returns a recommendation and the Regional form keeps its input, but the disease page is tested by page load only); and ledger sorting and navigation links. The Crop Advisor hold-out evaluation was obtained by re-executing the supplied train_model.py, with the model-saving step omitted, on the supplied dataset. This was done during manuscript preparation in a Linux environment (Python 3.12.3, scikit-learn 1.9.1, pandas 3.0.6, NumPy 2.5.4), not on the laptop in Table II.

V. RESULTS AND DISCUSSION

Software tests. The suite executed 19 test cases in 3.419 s and the final status was OK, so all 19 passed (Fig. 4, Table II). This is software-quality evidence only. It says nothing about the prediction accuracy of the Random Forest, the correctness of the Regional Engine's rankings or the accuracy of the disease classifier.



Fig. 4. Terminal output of python test_platform.py (path line cropped): "Ran 19 tests in 3.419s", "OK".

Crop Advisor. Re-executing the training procedure gave an accuracy of 99.55% on the 440 held-out rows (438 correct, two misclassified; precision and recall at least 0.95 for every crop). This figure should be read cautiously. It comes from one random split of a balanced table whose provenance is undocumented, with no cross-validation and no spatial or temporal separation, and the serialised model deployed in the application was not available to confirm that it is this model. It does not show performance on Maharashtra farm conditions.

Regional Engine and disease module. No accuracy, ranking-validity or sensitivity result exists for either module, and none is claimed. Form behaviour of the Regional Engine and page loading of the disease module were exercised by tests, and the modules return ranked outputs in the running application, but single outputs are illustrative and not accuracy evidence. Table III separates the evidence available for each component from what is not established.

TABLE III Evidence available for each component

Component and method	Evidence available	Not established
Records and access control (engineering)	19 passing tests incl. role separation and farmer-level isolation	Concurrency, load, security hardening, CSRF-enabled tests

Component and method	Evidence available	Not established
Crop Advisor (supervised ML, Random Forest)	Single hold-out accuracy 99.55% (re-executed); POST behaviour tested	Cross-validation, spatial split, field accuracy; deployed model file inspected
Regional Engine (rule-based weighted scoring, not ML)	Formulation documented; form behaviour tested	Weight sensitivity, back-testing, expert or farmer validation
Disease detection (third-party pre-trained model)	Page load tested only	Model documentation and accuracy; non-leaf rejection

Discussion. Placing the modules behind one authentication layer gives a single account, a common interface and a shared audit trail, and the farmer-scoped identifiers and ownership helpers keep record data separated, which the tests assert. Keeping the decision modules independent of the record tables limits coupling and makes each easy to replace, although it also means that no module uses a farmer's own history. The one-laptop, one-file deployment lowers the barrier to reproduction but limits scalability. High accuracies published for PlantVillage-trained models [9], [11] come from different models and datasets, and Barbedo's findings [12] caution that they do not carry over to field images; they must not be attributed to FarmIQ. Its distinguishing feature is the integration and the explicit separation of three differently grounded modules, not predictive performance.

VI. LIMITATIONS AND FUTURE WORK

Limitations.

(1) FarmIQ has not been deployed, validated in farmer field trials or reviewed by agronomy experts. (2) The Crop Advisor result is a single random-split evaluation, and the dataset's provenance and licence, the deployed model file and server-side input range checks are all missing. (3) The Regional Engine's weights, percentile bands, irrigation factors and rainfall asymmetry are implementation-defined and untested for sensitivity, the origin of the stored suitability table is undocumented, and season definitions are inconsistent (Section III-D). (4) The disease model is third-party and undocumented in the project, and the module has no non-leaf rejection or upload validation. (5) The decision modules do not use farmer records. (6) The configuration is not production-ready: it has a hard-coded weather API key, seeded demonstration accounts, debug mode and no password-strength rule. (7) Two further issues are suspected from code inspection but not confirmed by a controlled run: audit rows written after a route's own commit may not persist because log_audit() does not commit, and the tests may write to the real database because the URI is changed after the extension is initialised. (8) The system-health page labels the weather service Open-Meteo whereas the code calls OpenWeatherMap; dates are stored as text; and no unique constraint covers (farmer_id, custom_id).

Future work.

Priorities are to (i) evaluate the Crop Advisor with cross-validation and spatially separated data and report standard metrics; (ii) run a weight-sensitivity analysis of the Regional Engine and back-test it against historical outcomes; (iii) assess the disease model on field images or replace it with a documented, evaluated model; (iv) fix the audit-commit ordering and test isolation and add coverage and performance measurements; (v) move secrets to environment variables, harden authentication and deploy behind a production server; (vi) feed farmer records into the recommendation modules; and (vii) conduct a usability study with farmers.

VII. CONCLUSION

This paper described FarmIQ, a Flask and SQLite system that integrates role-based farm-record management with a Random Forest Crop Advisor, a rule-based Regional Crop Recommendation Engine for Maharashtra and a third-party plant-disease classifier. The six-table design was checked against the source code, the 19-test suite passed in 3.419 s, and the supplied Crop Advisor training procedure gave 99.55% on a single random hold-out split. The contribution is a documented engineering artefact with its limits stated, not a validated agricultural predictor: field accuracy, Regional Engine validity and disease-

detection accuracy remain to be established by the validation programme outlined above.

REFERENCES

[1] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, "Big data in smart farming – A review," *Agric. Syst.*, vol. 153, pp. 69–80, 2017, doi: 10.1016/j.agsy.2017.01.023.

[2] Z. Zhai, J. F. Martínez, V. Beltran, and N. L. Martínez, "Decision support systems for agriculture 4.0: Survey and challenges," *Comput. Electron. Agric.*, vol. 170, Art. no. 105256, 2020, doi: 10.1016/j.compag.2020.105256.

[3] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine learning in agriculture: A review," *Sensors*, vol. 18, no. 8, Art. no. 2674, 2018, doi: 10.3390/s18082674.

[4] T. van Klompenburg, A. Kassahun, and C. Catal, "Crop yield prediction using machine learning: A systematic literature review," *Comput. Electron. Agric.*, vol. 177, Art. no. 105709, 2020, doi: 10.1016/j.compag.2020.105709.

[5] S. Pudumalar, E. Ramanujam, R. H. Rajashree, C. Kavya, T. Kiruthika, and J. Nisha, "Crop recommendation system for precision agriculture," in *Proc. 8th Int. Conf. Adv. Comput. (ICoAC)*, Chennai, India, 2017, pp. 32–36, doi: 10.1109/ICoAC.2017.7951740.

[6] J. H. Jeong *et al*., "Random forests for global and regional crop yield predictions," *PLoS ONE*, vol. 11, no. 6, Art. no. e0156571, 2016, doi: 10.1371/journal.pone.0156571.

[7] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[8] F. Pedregosa *et al*., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[9] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Front. Plant Sci.*, vol. 7, Art. no. 1419, 2016, doi: 10.3389/fpls.2016.01419.

[10] 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 [cs.CY], 2015, doi: 10.48550/arXiv.1511.08060.

[11] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Comput. Electron. Agric.*, vol. 145, pp. 311–318, 2018, doi: 10.1016/j.compag.2018.01.009.

[12] J. G. A. Barbedo, "Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification," *Comput. Electron. Agric.*, vol. 153, pp. 46–53, 2018, doi: 10.1016/j.compag.2018.08.013.

[13] NASA Langley Research Center, "Prediction of Worldwide Energy Resources (POWER) Project," [Online]. Available: https://power.larc.nasa.gov/. [Accessed: date to be added].