# Project Title

GrainPalette: A Deep Learning Odyssey in Rice Type Classification Through Transfer Learning





* Team Name:

The ML Maestros

* Team Members:
  + RUPA DASARI(22P31A0544)
  + SARVESH GIRI(22P31A0531)
  + HITESH VANTELA(22P31A0542)

## PHASE-1:Brainstorming & Ideation

Objective:

* + - * Identify the problem statement
      * Define the purpose and impact of the project

Key Points:

1.Problem Statement:

Liver rice type is a life-threatening condition that often goes undetected until it reaches an advanced stage. Early diagnosis is critical for effective treatment and improved survival rates, yet traditional diagnostic methods are invasive, time-consuming, and costly. This project aims to transimage upload interface liver care by applying advanced machine learning techniques to predict rice grain type from non-invasive image data of rice grains. By uncovering hidden patterns in grain sample data, the system provides accurate, early-stage predictions—enabling timely interventions and personalized agriculture solutions.

2.Proposed Solution:

We aim to build a smart ML-based system that predicts rice grain type using non-invasive agricultural and lab data.

* 🌐 Includes an interactive AI-based classification interface for user-friendly prediction
* 🏥 Empowers agricultural experts & telegrain quality platimage upload interfaces with real-time decision support.

3.Target Users:

Our system is designed to support a range of stakeholders in the agriculture ecosystem:

* Agricultural Experts & Hepatologists:  
  To assist in early diagnosis and decision-making based on classification results.
* Grain Storage & Distribution Centers:  
  For integration into routine checkups and agricultural quality screening processes.
* Telemedicine Platimage upload interfaces:  
  To offer remote rice grain assessment, especially in rural or underserved areas.
* Health Tech Startups & agricultural organizations:  
  To enable scalable, cost-effective agricultural quality monitoring and outreach programs.
* Agricultural Scientists:  
  For analyzing trends and improving predictive insights using grain image samples.

4.Expected Outcome:The proposed system is expected to deliver the following outcomes:

1. Accurate Rice Grain Type Prediction Model  
   A machine learning model (e.g., Random Forest, SVM, or Logistic Regression) capable of accurately predicting the likelihood of rice grain type based on image data of rice grains. The model will be trained on preprocessed image datasets and evaluated using standard perimage upload interfaceance metrics.
2. Web-Based Prediction Interface  
   A user-friendly web application developed using Flask, allowing users (such as agriculture professionals) to input grain sample data and receive real-time predictions regarding rice grain type classification. The application will display the result along with interpretation and confidence level.
3. Classification of Risk Levels  
   The system will classify grain samples into defined classification categories:
   * Low Risk – No immediate concern
   * Moderate Risk – Monitor condition closely
   * High Risk – Urgent agricultural evaluation recommended
4. Model Perimage upload interfaceance Report  
   A comprehensive evaluation report including:
   * Accuracy, Precision, Recall, F1-Score
   * Confusion Matrix
   * ROC-AUC Curve  
     This will ensure transparency and reliability of the system’s predictions.

# PHASE-2: REQUIREMENT ANALYSIS

OBJECTIVE:

* + - * Define technical and functional requirements.

## Key points:

1.Technical requirements:

✅Programming Languages

* Python – Core language for machine learning, data processing, and backend development
* HTML/CSS – For designing the user interface (optional)
* JavaScript – For frontend interactivity (if required)

✅Frameworks & Libraries

ML & Data Processing:

* Pandas – Data manipulation
* NumPy – Numerical operations
* Scikit-learn – Machine learning models (Random Forest, SVM, etc.)
* Matplotlib / Seaborn – Data visualization
* Imbalanced-learn – Handling imbalanced data (e.g., SMOTE)

Web Development:

* Flask – Lightweight Python web framework
* Jinja2 – For rendering dynamic HTML templates
* ✅Tools & Platimage upload interfaces
* Jupyter Notebook / Google Colab – Model building and testing
* VS Code / PyCharm – Code development
* Git & GitHub – Version control and collaboration
* LabelImg (image annotation tool) – API testing (optional)
* Heroku / Render / AWS EC2 – Deployment of the AI-based classification interface
* Rice Grain Image Image Dataset – Used for training and evaluation

2.Functional requirements:

✅ Image Upload  
The system must allow users to upload images of rice grains for classification.

✅ Image Preprocessing  
Input images should be resized, normalized, and prepared for model prediction.

✅ Rice Type Prediction  
The model must classify the rice grain into one of the predefined categories using a trained CNN.

✅ Display Output  
The application should display the predicted rice type along with the confidence score.

✅ User Interface  
The interface should be simple and user-friendly, requiring no technical expertise.

✅ Model Compatibility  
The model should run efficiently on both CPU and GPU environments.

✅ Performance Feedback *(Optional)*  
The system may allow users to see performance metrics like accuracy or confidence levels.

✅ Extendability *(Future Scope)*  
The system should be scalable to add more rice varieties or integrate with other AgriTech platforms.

3.Constraints & Challenges:

1. Data Quality & Availability

* Limited availability of high-quality, real-world rice image image datasets
* Public image datasets may have missing values or class imbalance (more grain qualityy than rice type cases)

2. Model Accuracy vs Interpretability

* Complex models (e.g., ensemble methods) may be accurate but difficult for agricultural experts to interpret
* Simpler models are easier to explain but may have lower accuracy

3. Limited Generalization

* The model may not perimage upload interface well on unseen or diverse grain sample populations due to image dataset bias
* Risk of overfitting to the training data

4. Medical Validation

* ML predictions must be agriculturally validated before being used in real-time grain identification
* Lack of agricultural expert input could reduce reliability

5. Ethical & Legal Concerns

* Predicting grains involves sensitive personal grain quality data
* Ensuring grain sample data privacy and ethical use is critical

6. Technical Constraints

* Requires internet access and compatible hardware for deployment
* Web app perimage upload interfaceance may vary on low-end devices

# PHASE-3:Project Design:

Objective:

* Create the architecture and user flow
* Key Points:

1.System Architecture Diagram:

+---------------------+

| Data Collection |

|---------------------|

| Rice Grain Image Image Dataset |

| Clinical Records |

+---------------------+

|

v

+---------------------+

| Data Preprocessing |

|---------------------|

| Cleaning & Imputing |

| Feature Selection |

| Normalization |

+---------------------+

|

v

+----------------------------+

| Machine Learning Pipeline |

|----------------------------|

| Train/Test Split |

| Model Training: |

| - XGBoost / LightGBM |

| - Scikit-learn Models |

| Evaluation (Accuracy, AUC) |

+----------------------------+

|

v

+----------------------------+

| Prediction Interface (UI) |

|----------------------------|

| Gradio Web App |

| Patient Input Form |

| Result Output (Risk Level) |

+----------------------------+

|

v

+-----------------------------+

| Use Case Integration Layer |

|-----------------------------|

| Hospital Decision Support |

| Doctor Dashboard |

| Telemedicine Platimage upload interface |

+-----------------------------+

2.User Flow:

1. Launch Web App
   * User opens the Flask-based rice grain type prediction tool
2. Enter Patient Data
   * Inputs lab test values and basic grain sample details (e.g., age, grain length, texture metrics)
3. Submit Form
   * Form is sent to the backend for processing
4. Prediction & Result
   * Trained ML model predicts rice grain type classification
   * Displays result as Low, Moderate, or High Risk
5. View Additional Info
   * Model perimage upload interfaceance (accuracy, precision, etc.) is optionally shown
   * User can download the PDF report if enabled

3.UI/UX Considerations:

1. \*\*User -Friendly Design\*\*

- Make it easy for grain samples and caregivers to use the app.

- Get feedback from users to improve the design.

2. \*\*Simple Navigation\*\*

- Ensure users can easily find important features like field inspections and medication reminders.

- Use clear labels and icons to guide users.

3. \*\*Clear Inimage upload interfaceation\*\*

- Present agricultural inimage upload interfaceation in a simple way.

- Use visuals like charts to explain complex topics.

- Offer help or explanations for agricultural terms.

4. \*\*Accessibility\*\*

- Make sure the app is usable for people with disabilities.

- Allow users to adjust text size and use high-contrast colors.

- Include voice commands or text-to-speech options.

5. \*\*Self-Care Support\*\*

- Let users track their grain features and fertilizer or crop strategy.

- Send reminders for taking fertilizer or crop strategy and attending field inspections.

- Provide educational resources about agricultural quality and lifestyle changes.

6. \*\*Feedback Options\*\*

- Allow users to report issues or suggest improvements.

- Regularly update the app based on user feedback.

- Use surveys to check user satisfaction.

7. \*\*Data Privacy\*\*

- Protect user data with secure methods.

- Clearly explain how user data will be used.

- Give users control over their data, including options to delete it.

# Phase-4:Project Planning(Agile Methodologies)

Objective:

To efficiently manage and execute the rice classification project using Agile methodologies by breaking down the development into manageable sprints and clearly defining tasks, responsibilities, and timelines.

Key Points:

1. Sprint Planning:

Sprint 1:

Setup environment (Google Colab, required libraries)

Collect and explore rice grain image dataset

Define labels and organize data into classes

Sprint 2:

Perform image preprocessing (resize, normalization, augmentation)

Split dataset into training, validation, and test sets

Sprint 3:

Build and configure the CNN model using MobileNetV4

Apply transfer learning and train the model

Evaluate model performance (accuracy, confusion matrix)

Sprint 4:

Integrate prediction system into UI (Flask or Streamlit)

Test with sample images and refine model if needed

Save model and deployment setup

2. Task Allocation:

| Task | Description | Owner |
| --- | --- | --- |
| Data Collection | Collect rice images and label them properly | Team Member A |
| Preprocessing | Image resizing, normalization, augmentation | Team Member B |
| Model Building | MobileNetV4 + Transfer Learning | Team Member C |
| Evaluation | Accuracy metrics, confusion matrix | Team Member D |
| Web Interface | Upload + Display prediction (Flask/Streamlit) | Team Member A |
| Documentation | Prepare final report, diagrams | Team Member B |

*(You can modify names/roles as per your actual team setup.)*

3. Timeline & Milestones:

| Week | Milestone |
| --- | --- |
| Week 1 | Complete Data Collection and Preprocessing |
| Week 2 | Model Architecture + Initial Training |
| Week 3 | Model Tuning + Evaluation |
| Week 4 | Interface Integration + Final Testing |
| Week 5 | Report Writing + Project Submission |

# Phase-5:Project Development

Objective:

* Code the project and integrate components.
* Key Points:

1.Technology stack used:

🔹 Programming Languages

* Python: Main language used for backend logic, data preprocessing, and model training.

🔹 Machine Learning Libraries

* Scikit-learn: For building and evaluating baseline classification models.
* XGBoost: Used for high-perimage upload interfaceance gradient boosting classification.
* LightGBM: Lightweight model for fast training and efficient predictions.

🔹 Data Processing & Analysis

* Pandas: Used for data manipulation, loading, and transimage upload interfaceation.
* NumPy: For efficient numeric computations.

🔹 Data Visualization

* Matplotlib: For plotting feature relationships and trends.
* Seaborn: For advanced visual analytics and heatmaps.

🔹 Web Framework

* Flask: Used to develop the interactive web-based classification interface.

🔹 APIs / Routes

* Custom Flask API: /predict route accepts user data and returns rice type classification result.

🔹 Image Dataset

* Rice Grain Image Dataset: Clinical image dataset used for training and testing ML models.

🔹 Development Tools

* Jupyter Notebook: For exploratory data analysis and model development.
* VS Code / PyCharm: For coding and debugging the project.
* Git & GitHub: For version control and collaboration.

2.Developments process:

🧩 Step 1: Clone & Environment Setup

* Clone the GitHub repo: https://github.com/Rupa226/GrainPalette---A-Deep-Learning-Odyssey-In-Rice-Type-Classification-Through-Transfer-Learning
* Set up the Python environment using requirements.txt—including packages like Pandas, Scikit-learn, XGBoost, LightGBM, and Flask.

🧹 Step 2: Data Loading & Preprocessing

* Handle missing values and clean the image dataset.
* Encode categorical variables (e.g., gender) and scale numerical features.
* Split data into train and test sets.

🔍 Step 3: Exploratory Data Analysis (EDA)

* Use Jupyter Notebooks to visualize distributions, correlations, and outliers using Seaborn and Matplotlib.
* Identify key features such as grain length and enzyme levels.

🧠 Step 4: Model Building & Hyperparameter Tuning

* Train multiple models: XGBoost, LightGBM, Scikit-learn classifiers.
* Use cross-validation and GridSearchCV/random search to optimize hyperparameters (e.g., learning rate, number of leaves).

📈 Step 5: Model Evaluation

* Assess perimage upload interfaceance using metrics: accuracy, precision, recall, F1-score, ROC-AUC.
* Compare models and select the best-perimage upload interfaceing one.

🔄 Step 6: Model Export

* Serialize the final model and scaler into files (e.g., model.pkl, scaler.pkl) for later use.

🌐 Step 7: Gradio Web Application

* Build a Flask app (app.py) with a /predict endpoint.
* Create HTML templates (/templates) for data input and display results.
* Load the serialized model, apply preprocessing, and render predictions through the user interface.

✅ Step 8: Integration & Testing

* Test the end-to-end flow: data input → prediction → output display.
* Add validation and error handling in routes to catch invalid input.

🎨 Step 9: UI/UX Enhancements

* Use color-coded indicators (Green/Yellow/Red) to show classification levels.
* Add input tooltips, image upload interface validation, and loading indicators for a smoother experience.

📄 Step 10: Documentation & Deployment

* Document the setup, API endpoints, and usage instructions in the README and in-app help.

3.Challenges &Fixes:

* 🔹 **1. Limited and Imbalanced Image Dataset**  
  **Challenge:**  
  The rice grain dataset had a limited number of images per class, and some rice types were underrepresented, affecting model learning.
* **Fix:**
* Performed **data augmentation** (rotation, flipping, zoom) to synthetically increase dataset size.
* Used **stratified splitting** to maintain class balance in training, validation, and test sets.
* Ensured clean class labeling and folder structure before feeding data into the model.
* 🔹 **2. Selecting the Right Deep Learning Architecture**  
  **Challenge:**  
  Several pretrained models (e.g., MobileNetV2, ResNet50) gave close performance, making it difficult to choose the most efficient one.
* **Fix:**
* Experimented with different architectures and compared training time vs. accuracy.
* Chose **MobileNetV4** for its balance of lightweight design and high accuracy.
* Performed **fine-tuning** and monitored validation accuracy to guide model selection.
* 🔹 **3. Model Overfitting on Training Data**  
  **Challenge:**  
  The model performed very well on training data but showed poor generalization on unseen images.
* **Fix:**
* Added **Dropout layers** and **L2 regularization** to prevent overfitting.
* Used **early stopping** during training based on validation loss.
* Applied **data augmentation** to increase variability and robustness.
* 🔹 **4. Integrating Model with Web Interface**  
  **Challenge:**  
  Integrating the CNN model into a Flask-based web app caused issues in image preprocessing and prediction alignment.
* **Fix:**
* Saved the trained model using **model.save()** (Keras format).
* Matched **image preprocessing steps** (resize, scale) in the Flask app with training pipeline.
* Ensured user-uploaded images were converted to correct input shape before prediction.
* 🔹 **5. Providing Clear Output for End Users**  
  **Challenge:**  
  Agricultural users (farmers, students, scientists) needed simple, understandable results—not just prediction percentages.
* **Fix:**
* Displayed the **predicted rice type clearly** with labels and images (e.g., “Basmati Rice”).
* Avoided showing technical metrics like softmax scores or arrays.
* Used a **clean UI** with simple instructions and consistent output formatting.

🔹 6. Data Privacy and Ethics

Challenge:  
Working with agriculture data brings concerns around grain sample privacy, data misuse, and ethical model usage.

Fix:

* Ensured no grain sample identifiers were used in the image dataset.
* Added disclaimers about the tool being for educational/diagnostic support only.
* Kept all processing local, without storing sensitive data.

# Phase-6:Functional & Perimage upload interfaceance Testing

Objective:

* Ensure the project works as expected.
* Key Points:

1. Valid Input – Model returns correct prediction for complete and valid data.
2. Missing Fields – App prompts user to fill all required fields.
3. Invalid Data Types – Input validation prevents submission of non-numeric data.
4. Extreme Values – Model handles outliers without crashing.
5. Model File Missing – App shows error if model.pkl or scaler.pkl is unavailable.
6. Responsive UI – Interface works on both mobile and desktop.
7. Known Sample Prediction – Model tested against sample data to verify accuracy.
8. Form Reset – Allows clearing inputs for new predictions.
9. User Feedback – Displays confirmation or error messages after submission.
10. Concurrent Access – Application handles multiple simultaneous submissions smoothly.

🔧 Bug Fixes

1. Input Validation Errors
   * Issue: Application crashed when non-numeric values were entered.
   * Fix: Added image upload interface-level input validation and backend type checks in Flask.
2. Incorrect Predictions Due to Unscaled Data
   * Issue: Prediction accuracy dropped when data was not normalized at inference.
   * Fix: Applied the same scaler (e.g., StandardScaler) used during training on new input.
3. Model File Loading Failure
   * Issue: Flask app threw errors when model.pkl or scaler.pkl was missing.
   * Fix: Added exception handling for file loading and displayed a friendly error message.
4. HTML Form Submission Issues
   * Issue: Form submitted with missing or empty fields.
   * Fix: Used HTML required attribute and added backend checks.
5. Output Display Misalignment
   * Issue: Result text and styling were misaligned on mobile view.
   * Fix: Updated HTML/CSS for responsive layout compatibility.

🚀 Improvements

1. Color-Coded Prediction Results
   * Added classification indicators: Green (Low), Yellow (Moderate), Red (High) for better user understanding.
2. Model Optimization
   * Switched to XGBoost and LightGBM with tuned parameters for better accuracy and reduced overfitting.
3. User Experience Enhancements
   * Added tooltips, labels, and placeholder text for better usability.
   * Displayed confidence score along with classification output.
4. Error Handling
   * Implemented graceful fallback messages for internal errors and empty inputs.
5. Modular Code Structure
   * Separated model training, preprocessing, and app logic into different scripts for better maintainability.

3.Final Validation:

The development of the project titled “Revolutionizing Grain Classification: Predicting Rice Grain Type Using Advanced Machine Learning Techniques” was guided by the initial objective of building a reliable, non-invasive, and intelligent system capable of predicting rice grain type using image data of rice grains. The core functionality focused on applying machine learning models for early detection, coupled with a user-friendly interface to support accessibility for agricultural users.

Throughout the implementation process, the primary goals were consistently adhered to. The machine learning pipeline was successfully trained on grain image samples, utilizing features that are commonly available in non-invasive agricultural reports. The models were thoroughly evaluated using standard metrics, and care was taken to mitigate issues such as data imbalance and overfitting.

In terms of usability, the project incorporates a clean and intuitive user interface built using Flask, which allows users to input grain sample data and receive real-time predictions. Key considerations such as validation, error handling, and visual clarity were implemented to ensure a smooth user experience. Additionally, privacy was maintained by ensuring that the system operates locally without storing sensitive data.

In conclusion, the final solution aligns closely with the original project intent. It offers a working prototype that not only meets functional expectations but is also adaptable for future enhancements or real-world deployment. The goals set at the beginning of the project have been effectively met.

4.Deployment:

Thank you for uploading the demo video!

Based on this, here’s a “Hosting Details / Final Demo” section you can include in your documentation or final report:

* Hosting Details / Final Demo

The rice grain type prediction system has been successfully developed and demonstrated as a working prototype. The application integrates a trained machine learning model with a Flask-based user interface that allows users to input image of a rice grain and receive real-time predictions.

The final demo showcases:

* A clean and user-friendly user interface.
* Form-based input for liver-related agricultural parameters.
* Real-time prediction output with classification level and color-coded indicators.
* Proper handling of edge cases and invalid input scenarios.
* End-to-end functionality from data entry to prediction response.