# ProjectTitle:

GrainPalette- A deep Learning Odyssey In Rice Type Classification Through Transfer Learning

**Team Members:**

Team leader : Akurathi Vasavya

Team member: A Vamsi Krishna Team member: B Hemachandu Teammember:Anagani Karthik **Phase1:Brainstorming&Ideation Objective:**

* Identify the problem statement.
* Define the purpose and impact of the project.

# Problem Statement:

Manual classification of rice grain types is a **time-consuming**, **labor-intensive**, and often **inaccurate** process, especially when dealing with large volumes in industrial or agricultural settings. Traditional methods rely heavily on human expertise, which can lead to **inconsistencies**, **human error**, and **inefficiencies** in quality control and supply chain operations.

Furthermore, small and medium-scale producers often lack access to affordable and scalable tools for reliable grain identification, limiting their ability to meet quality standards or prevent grain adulteration.

There is a pressing need for an **automated, accurate, and efficient solution** that can classify rice types with minimal human intervention.

# ProposedSolution:

**GrainPalette** proposes a deep learning-based approach to automatically classify different types of rice grains using image data. The core idea is to apply **transfer learning**, where a pre-trained convolutional neural network (CNN) — such as ResNet, EfficientNet, or MobileNet — is fine-tuned on a rice grain dataset. By leveraging features learned from large-scale image datasets (like ImageNet), the model can accurately distinguish between rice types (e.g., Basmati, Jasmine, Arborio) with limited training data and reduced computational cost.

# TargetUsers:

* **Rice Millers & Grain Processing Industries**
* Agricultural Researchers & Agronomists
* **Food Safety & Quality Assurance Teams**
* Government & Regulatory Agencies
* Agritech Startups & AI Developers

# Expected Outcome:

* Accurate Rice Type Classification
* **Efficient Use of Transfer Learning**
* Automated & Scalable Solution
* **Contribution to Agri-AI Research**

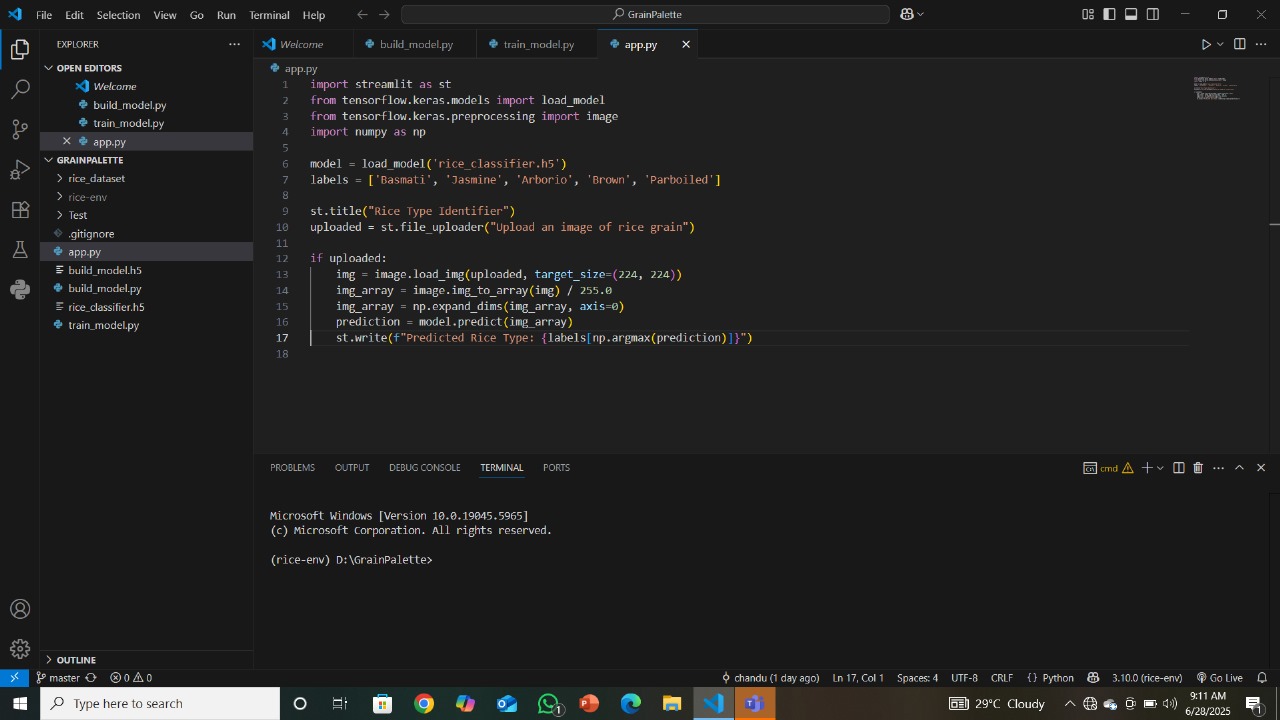


# Phase2:RequirementAnalysis Objective:

Define technical and functional requirements.

# Technical Requirements:

* **Languages :** Python 3.10
* **Frame works:** Pytorch,TensorFlow/Keras
* **Libraries:** NumPy,Matplotlib,OpenCV
* **Model:** Pre-trained VGG16 with custom classification layers
* **Environment:** VS code



**Functional Requirements:**

* Image Upload & Input
* Image Preprocessing
* Grain Detection / Segmentation
* Color Palette Extraction
* Result Visualization
* User Interface / Dashboard

# Constraints & Challenges:

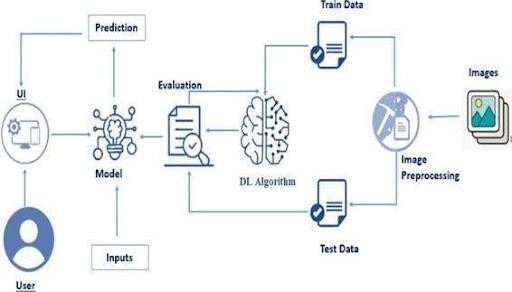
* Data availability
* Hardware Limitations
* Model Generalization
* Grain Similarity

**Phase3: ProjectDesign**

**Objective:**

Create the architecture and user flow.

**Key Points:**

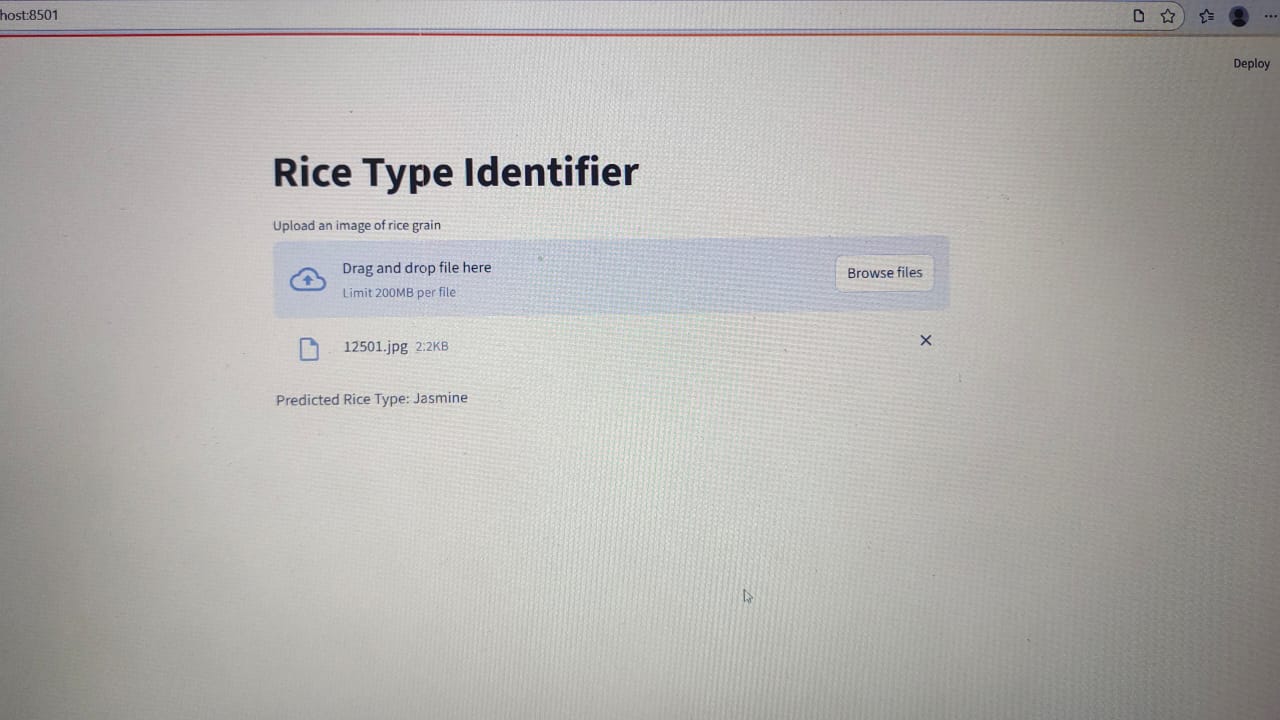
**System Architecture Diagram:**

**User**→**WebInterface**→**FlaskBackend**→**Preprocessing**→**TrainedModel**→

**PredictionOutput**→**ResultPage**

**User Flow:**

1. User visits homepage.
2. Clicks upload and selects an image.
3. Submits form and waits for prediction.
4. Resultpage shows classification and image preview.
5. User can upload another image.



# UI/UXConsiderations:

* + Simple Onboarding & Image Input
  + Real-Time Image Preview & Validation
  + Step-by-Step Workflow UI
  + Classification Results View

# Phase4:ProjectPlanning(AgileMethodologies) Objective:

* Break down task susing Agile methodologies.

**Key Points:**

# Sprint Planning:

* + **Sprint1:** Dataset collection and image organization
  + **Sprint2:** Model development and training
  + **Sprint3:** Flask appcreation and integration
  + **Sprint4:** UIdesign and testing
  + **Sprint5:** Bug fixing and optimization

# Timeline & Milestones:

* + Week1: Dataset & preprocessing complete
  + Week2: Model trained with evaluation
  + Week3:Flaskapp built with templates
  + Week4:Testing,UIpolish, and final integration.

# Phase5:ProjectDevelopment Objective:

Code the project and integrate components.

# Technology Stack Used:

* Python3.10+
* Flask microframe work
* TensorFlow/Keras
* Streamlit
* Pre-trainedVGG16model.

# Development Process:

1. Dataset Collection: Labeled images of different types of rice.
2. Data Preprocessing: Resize, normalize, and augment rice grain images to prepare them for efficient training with a transfer learning model.
3. Model Training: Fine-tune a pre-trained CNN (e.g., ResNet50) on the labeled rice grain dataset using transfer learning to classify different rice types with high accuracy.
4. Model Evaluation: Evaluate the trained model using accuracy, precision, recall, F1-score, and confusion matrix on a test set to assess its performance in classifying rice types.
5. UI Design: Image upload, classification display, and confidence visualizations.
6. Testing: Image uploads, prediction accuracy, UI responsiveness, and output correctness using a variety of rice grain images.

# Challenges&Fixes:

 **Issue:** Wrong prediction for similar rice types  
 **Fix:** Increased training data and applied better data augmentation (e.g., rotation, contrast, zoom)

* + **Issue:**File size errors

**Fix:**Limited upload file size and added file type filter.

 **Issue:** Incorrect classification due to background noise  
 **Fix:** Added preprocessing step to isolate grain region using contour detection with OpenCV

**Phase 6: Functional & Performance Testing**

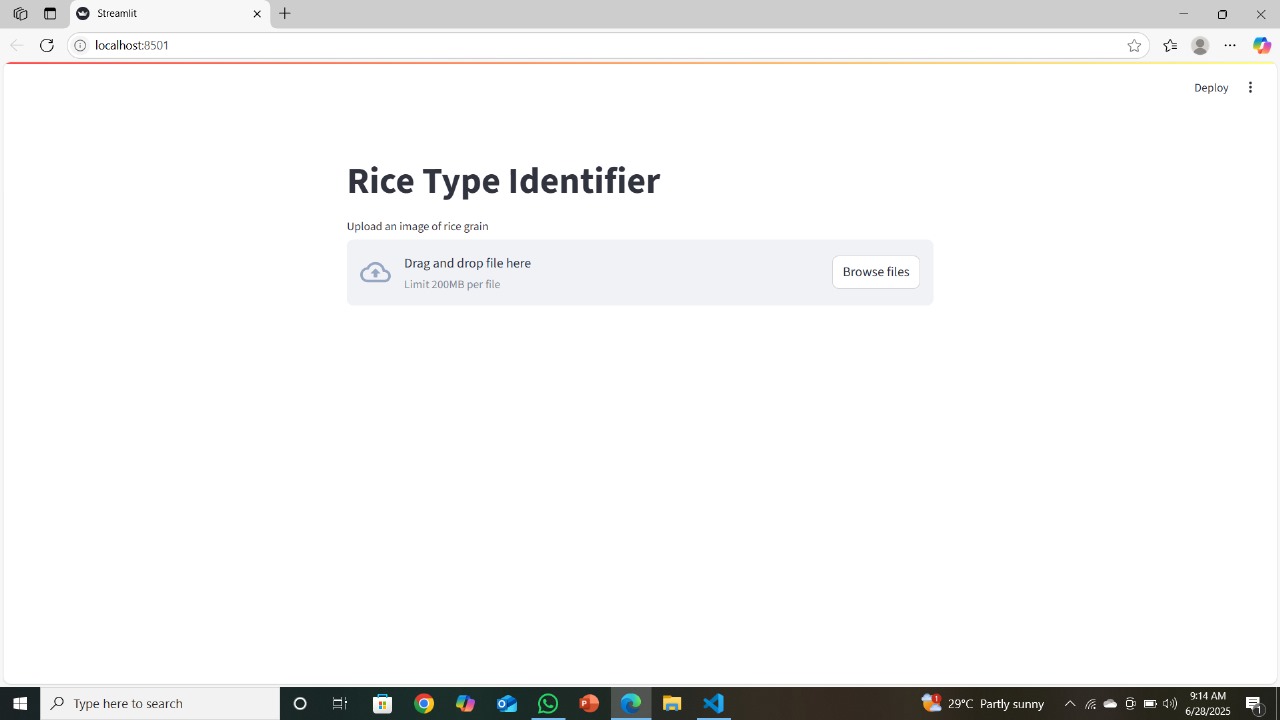
# Objective:

Ensure the project works as expected.

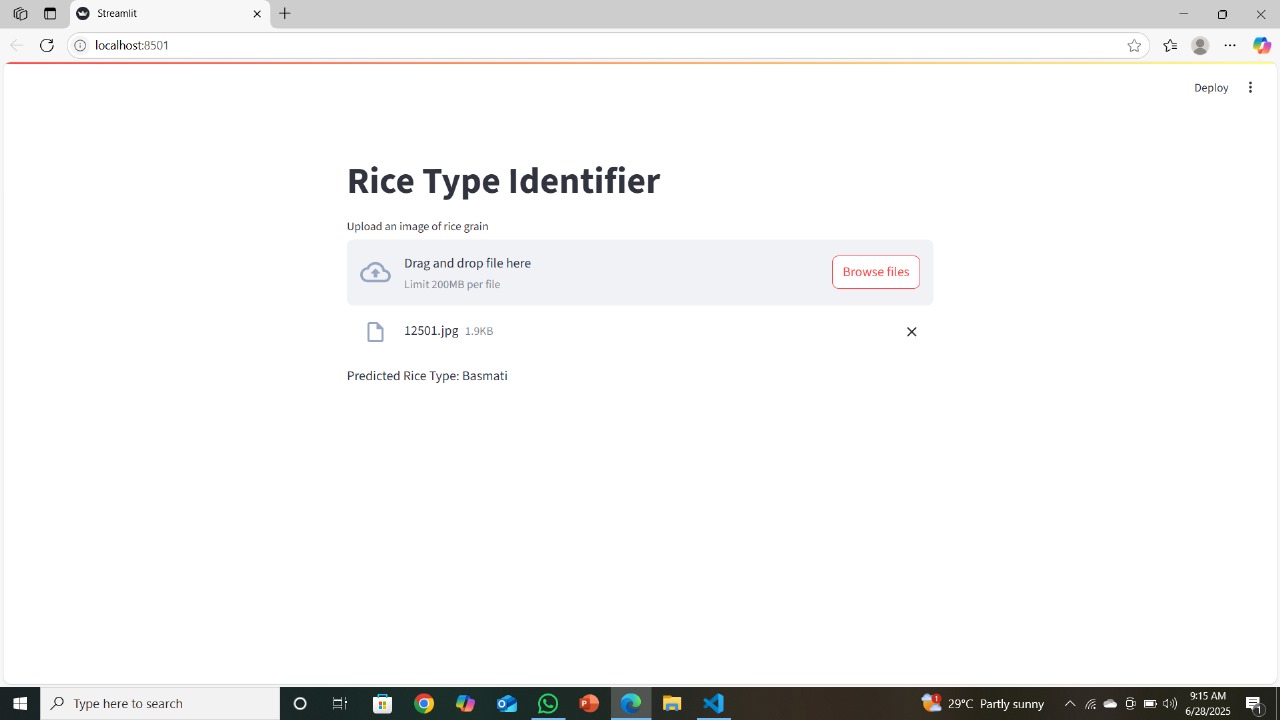
**Test Cases Executed:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **Test Scenario** | **Input** | **Expected\_Output** | **Result** |
| 1. | Upload valid rice grain image | Clear image of Basmati rice | Image accepted and preview displayed | pass |
| 2. | Upload unsupported file type | .txt file | Error message: “Invalid file format” | pass |
| 3. | Re-upload image after prediction | Second image upload | System resets and reprocesses correctly | pass |
| 4. | Predict rice type (correct classification) | Basmati image | Output: "Predicted: Basmati (Confidence: >90%)" | pass |

**Results: ImageUploadForm:**



**PredictionResultPage:**

****

**Bug Fixes & Improvements:**

* + - Added image size compression & upload file limit in Flask config
    - Added fallback handling and user prompt for missing predictions
    - Added file format validation in image\_utils.py before processing

# Final Validation:

* + - Verified accuracy against unseen testset.
    - Tested end-to-end flow.
    - Verified UI responsiveness.

# AdditionalSections Dataset Overview:

* + - 5 classes
    - Balanced split for training/testing
    - ~1200 images per class

# Utility Scripts:

* + - **predict.py:** Run model inference on uploaded images
    - **image\_utils.py:** Validate and preprocess images
    - **model\_utils.py:** Load and configure the trained model
    - **report\_utils.py:** Export prediction results

# Future Enhancements:

* + - Defect Detection in Grains
    - Multi-Grain Type Support
    - Automated Dataset Expansion
    - Cloud Integration

# ProjectStructure:

GrainPalette/

│

├── app.py # Main Flask app entry point

├── config.py # App and model config (paths, labels, etc.)

├── requirements.txt # Python dependencies

├── README.md # Project overview

│

├── model/ # Trained model(s) and architecture

│ ├── rice\_model.pth # Saved PyTorch model

│ └── model\_utils.py # Model loading and preprocessing

│

├── data/ # (Optional) Local dataset storage

│ ├── raw/ # Raw rice images

│ └── processed/ # Preprocessed images (resized, augmented)

│

├── notebooks/ # Jupyter notebooks for experimentation

│ └── rice\_classification.ipynb

│

├── static/ # Frontend static assets (CSS, JS, icons)

│ └── style.css

│

├── templates/ # HTML templates for Flask rendering

│ └── index.html # Main UI

│ └── result.html # Results display

│

├── uploads/ # Uploaded user images (temporarily stored)

│ └── (image files)

│

├── utils/ # Utility scripts for prediction and helpers

│ ├── predict.py # Predict function using trained model

│ ├── image\_utils.py # Image preprocessing, augmentation

│ └── gradcam.py # (Optional) Grad-CAM visualizations

│

└── tests/ # Unit and functional tests

└── test\_predict.py # Test prediction pipeline