Project Phases Template

Project Title: GrainPalette-A-Deep-Learning-Odyssey-In-Rice-Type-Classification-ThroughTransfer-Learning.

TEAM:

Team ID : LTVIP2025TMID41595

Team Size : 4

Team Leader : N Karthik

Team member : N Jagadeeswar Reddy

Team member : N Venkateswara Reddy

Team member : Ns Iffath Muskan

Phase-1: Ideation

GrainPalette-A-Deep-Learning-Odyssey-In-Rice-Type-Classification-Through-Transfer-Learning.

Objective:

1) Identify the problem statement

Rice is a staple food for more than half of the world's population, and it comes in many di erent varieties, each with unique characteristics. However, manually identifying and classifying rice types based on appearance is a time-consuming, subjective, and error-prone process — especially in agricultural, commercial, or research settings.There is a growing need for an automated, accurate, and scalable solution to classify rice grains using image data, to assist farmers, distributors, researchers, and quality control personnel.

2)Define the purpose and impact of the project?

Automatically classify di erent types of rice grains based on images.

Reduce the need for manual inspection by using deep learning and transfer learning.

Provide a simple web interface where users (like farmers, researchers, or distributors) can upload rice grain images and instantly receive predictions.

1. Agricultural Automation
2. Improved Accuracy

3. Cost E iciency

Key Points:

1)Problem Statement

Rice is a staple food for more than half of the world's population, and it comes in many di erent varieties, each with unique characteristics. However, manually identifying and classifying rice types based on appearance is a time-consuming, subjective, and errorprone process — especially in agricultural, commercial, or research settings.There is a growing need for an automated, accurate, and scalable solution to classify rice grains using image data, to assist farmers, distributors, researchers, and quality control personnel.

2)Proposed Solution

To address the challenge of accurately identifying rice types, we propose an AI-powered web application called Grain Palette, which leverages Deep Learning and Transfer Learning techniques to classify rice grain images into specific categories.

3)Target Users

- Farmers and Agricultural workers.
- Rice Mill Owners and Distributors
- Agricultural Researchers and Scientists

4)Expected Outcome

The Grain Palette project is expected to deliver a working prototype of an AI-powered web application that can accurately classify di erent types of rice grains based on image input.

Phase-2: Requirement Analysis

Objective:

Define technical and functional requirements.

Requirement	Version / Notes
Python	3.8 or higher
TensorFlow	2.18.0
Keras	Included with TensorFlow
Flask	2.3.3
NumPy	1.24.3
Pandas	2.1.3
Pillow	10.3.0
Werkzeug	2.3.7 (for secure file handling)
Gunicorn (optional)	21.2.0 (for production deployment)

Key Points:

1. Technical Requirements: (Languages, frameworks, tools) Programming Languages

- Python 3.8+
  Primary language used for building the backend logic, deep learning model handling, and image preprocessing
- Frameworks & Libraries
  Web Development
  Flask 2.3.3
  Lightweight web framework for building the backend and serving HTML pages.
- Deep Learning
  TensorFlow 2.18.0
  Used for loading the trained rice classification model and making predictions.
  Keras (included with TensorFlow) High-level API for deep learning tasks.
- Image Processing
  Pillow 10.3.0
  To handle image file loading and conversion.
- TensorFlow's image and preprocessing utilities
  For image resizing, preprocessing, and prediction input preparation.
- Data Handling
  NumPy 1.24.3
  For numerical operations and array handling.
  Pandas 2.1.3

- 脥 Utilities

  Werkzeug 2.3.7

  Used to securely handle file uploads
- Gunicorn 21.2.0

  WSGI HTTP Server for production deployment secure_filename).

- Development Tools

  Jupyter Notebook / Google Colab (for model training and testing)

  VS Code / PyCharm (for local development)

  Command Line / Terminal (to run Flask app)

  Web Browser (to test the web app UI)

2. Functional Requirements: (Features the project must have)

1. Image Upload

- The user must be able to upload an image of a rice grain through a simple web form.
- The app should accept common image file formats like .jpg, .jpeg, and .png.

2. Image Preprocessing

- The uploaded image must be:
  - Resized to 224×224 pixels.
  - Preprocessed using mobilenet_v2.preprocess_input() to match model expectations.

3. Prediction using Pre-trained Model

- The app must load the pre-trained model (rice_model.h5) at startup.
- When an image is uploaded:
  - The model should predict the rice type.
  - The app should calculate and return the confidence score (e.g., 92.35%).

4. Display of Results

- After prediction, the app should:
  - Show the uploaded image.
  - Display the predicted rice type (e.g., Basmati).
  - Display the confidence percentage (e.g., “Confidence: 93.2%”).

5. Web Interface Navigation

- The app must include the following pages: o index.html: Introduction or landing page. o predict.html: Form to upload an image. o result.html: Displays prediction output.

6. Error Handling

- If the user uploads:
  - No file — show message: “No file selected.”
  - An unsupported file type — show a warning.
- If any internal error occurs, the app should handle it gracefully without crashing.

7. Ease of Use

- The interface should be simple and intuitive for non-technical users.
- Minimal steps should be required to get a prediction.

3. Constraints & Challenges: (Any limitations or risks)

This section outlines the limitations, risks, and potential di iculties encountered during development and deployment of the Grain Palette project.

1. Limited Dataset

- Challenge: The model performance is heavily dependent on the size and quality of the rice grain image dataset.
- Impact: A small or imbalanced dataset may cause overfitting or poor generalization on new images.

2. Similar Visual Features

- Challenge: Some rice types have very similar appearances, making it di icult for the model to distinguish between them.
- Impact: This can reduce classification accuracy, especially if lighting or image angle varies.

3. Model Bias or Inaccuracy.

- Challenge: The model might show bias toward certain classes if trained on uneven data.
- Risk: It may consistently misclassify specific types unless carefully trained and validated.

4. Limited User Interface

- Constraint: The web app is kept simple with basic image upload and result display.
- Impact: No support (yet) for bulk uploads, real-time camera input, or additional rice information.

5. Image Quality Dependence

- Challenge: The prediction accuracy depends on the clarity, background, and lighting of the uploaded image.
- Impact: Poor quality images may lead to incorrect results.

6. Hardware Limitations

- Constraint: On systems without a GPU, prediction might be slower.
- Impact: Slower performance in large-scale usage or batch predictions.

7. Deployment & Hosting

- Risk: Hosting the application (especially with model files) on platforms like Heroku or Render may have storage/memory limits.
- Impact: Might require optimization or alternative cloud hosting for scaling.

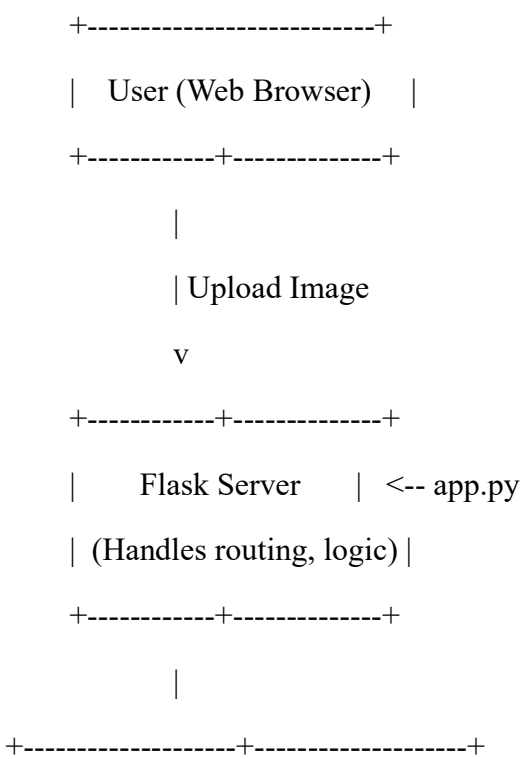
Phase-3: Project Design

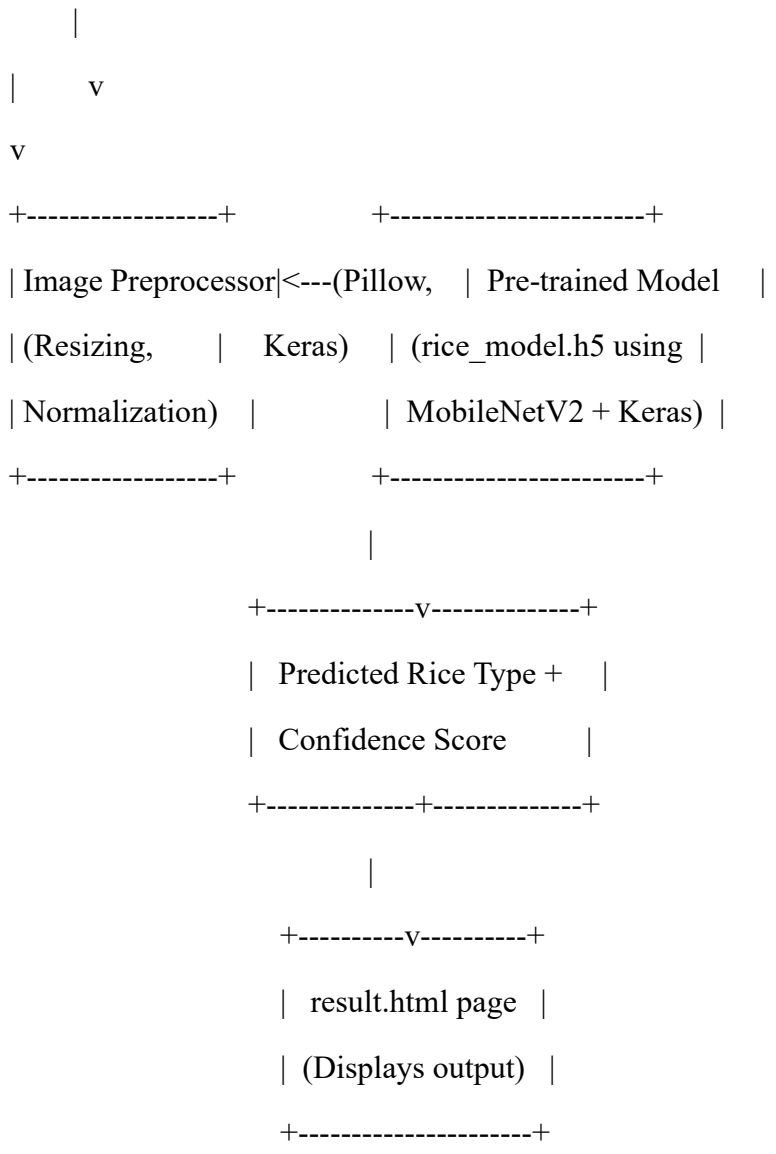
Objective:

The objective of this phase is to design the architecture and user flow of the system in order to ensure smooth interaction between the user, web application, and the deep learning model. This phase lays the foundation for how components of the system will interact and what the end-user experience will look like.

Key Points:

1. System Architecture Diagram: (Simple sketch or flowchart)





2. User Flow: (How a user will interact with the project)

1. User opens the web application o HTML template: index.html o Introduces the project or has a button to "Upload Image"
2. User navigates to prediction page o HTML template: predict.html o Upload form allows the user to choose an image and submit
3. Image is sent to the Flask backend o Flask handles the POST request
   - o Uses Pillow & Keras to preprocess the image

4. Model Prediction o The pre-trained model (rice_model.h5) is loaded o The processed image is passed to the model o The model returns the predicted rice type and confidence score
5. Display Results o HTML template: result.html o Shows:
    - Uploaded image
    - Predicted rice type
    - Confidence score
6. User can return to predict again

3. UI/UX Considerations: (If applicable, wireframe or basic layout)

The goal of the user interface (UI) and user experience (UX) design is to ensure the application is simple, intuitive, and accessible to all users — even those with no technical background. The layout should make it easy to upload images and quickly interpret results. Wireframe / Basic Layout 꼡 1. Home Page (index.html) pgsql

```
+-------------------------------------------------+
|          GrainPalette - Welcome                 |
|-------------------------------------------------|
| [긏긬귈귚긱긯귀     Start Prediction]             |
|-------------------------------------------------|
| Info about the project, purpose, and how to use |
+-------------------------------------------------+
```

꼡 2. Prediction Page (predict.html) lua

```
+-------------------------------------------------+
|          Upload Rice Image                      |
|-------------------------------------------------|
```

| [Choose File] (jpg/jpeg/png) |

| [Predict] |

|---|

| *Displays helpful hints and allowed formats* |

+---+

껩 3. Result Page (result.html) sql

+---+

| Prediction Result |

|---|

| Uploaded Image: [image preview] |

| Predicted Type: Basmati |

| Confidence: 93.47% |

|---|

| [Upload Another Image] |

+---+

Phase-4: Project Planning (Agile Methodologies)

Objective: Break down the tasks using Agile methodologies.

Agile Task Breakdown for Grain Palette

Methodology: Scrum

The project is divided into sprints (short, time-boxed iterations), each focused on a specific goal. Each sprint includes planning, development, testing, and review.

Sprint 1: Project Setup & Dataset Preparation (Week 1) Goals:

- Set up project structure
- Collect and prepare rice image data Tasks:
- Create project repository (GitHub)
- Install necessary Python libraries

- Collect or download rice image dataset
- Organize images into class folders
- Resize and clean image data
- Create a requirements.txt file

Sprint 2: Model Development & Training (Week 2) Goals:

- Train a deep learning model using transfer learning Tasks:
- Load and preprocess the dataset
- Apply MobileNetV2 as a base model
- Fine-tune the model with rice dataset
- Train and validate the model
- Save trained model as rice_model.h5
- Test with sample predictions Sprint 3: Flask Web App Development (Week 3) Goals:
- Build a user-friendly web interface Tasks:
- Set up Flask app structure
- Create HTML templates: index.html, predict.html, result.html
- Implement image upload feature
- Integrate model into Flask backend
- Route predictions to result page
- Handle file uploads and errors Sprint 4: UI/UX Enhancements & Testing (Week 4) Goals:
- Improve user experience and test functionality Tasks:
- Refine HTML/CSS for better layout
- Add image preview before prediction
- Show confidence score clearly
- Test for invalid inputs (no file, wrong format)
- Fix bugs and edge cases

- Get peer feedback

Sprint 5: Documentation & Final Deployment (Week 5) Goals:

- Finalize project for submission/presentation Tasks:
- Write project documentation (problem, solution, tech stack)
- Create architecture diagram and wireframes
- Generate PDF/Word report
- Deploy app using Gunicorn or Render (optional)
- Final demo and walkthrough

Deliverables by End of Project

- Working web application (Flask)
- Trained rice classification model
- Full documentation (PDF/Word)
- Optional: Hosted live demo or screen recording

Key Points:

1. Sprint Planning: (Divide work into tasks for each team member)

Team Member	Role
N.Karthik	Data & Model Developer
N Jagadeeswar Reddy	Backend Developer (Flask)

Team Member	Role
N Venkateswara Reddy	Frontend & UI/UX Developer

Ns I ath Muskan	Documentation & Testing Lead

Task	Assigned To
Set up GitHub repo and project folders Install Python environment and libraries	N.Karthik
Collect/download rice image dataset Organize dataset into class-wise folders Verify dataset quality and fix issues	N.Karthik

Task	Assigned To
Load and preprocess image data	N.Karthik
Implement MobileNetV2 with transfer learning	N.Karthik
Train and validate the model	N.Karthik
Save trained model as rice_model.h5	N.Karthik
Provide sample test results	N.Karthik

Task	Assigned To
Set up basic Flask app with routes	N.Karthik
Integrate model with Flask	N Jagadeeswar Reddy
Handle image upload and preprocessing	N Jagadeeswar Reddy
Return prediction and confidence score	N Jagadeeswar Reddy
Error handling for invalid input	N Jagadeeswar Reddy

Task	Assigned To
Design and implement index.html	N Venkateswara Reddy
Create predict.html with file	N Venkateswara

upload	Reddy

Team Member	Role
Design result.html to show prediction	N Venkateswara Reddy
Add CSS styling for responsiveness	N Venkateswara Reddy
Add image preview feature	N Venkateswara Reddy

Task	Assigned To
Write project report (problem, purpose, solution)	Ns I ath Muskan
Create architecture and user flow diagrams	Ns I ath Muskan
Test full application end-to-end	Ns I ath Muskan
Collect feedback and fix UI/UX bugs	Ns I ath Muskan
Prepare PDF documentation and demo slide	Ns I ath Muskan
Optional: Deploy using Gunicorn / Render	Ns I ath Muskan

2. Task Allocation: (Who will do what?)

Task	Assigned To	Sprint
Create GitHub repo and project folder structure	N. Karthik	Sprint 1
Install dependencies and create requirements.txt	N. Karthik	Sprint 1
Download and organize rice dataset	N. Jagadeeswar Reddy	Sprint 1
Preprocess and clean images (resizing,		Sprint

	N. Jagadeeswar Reddy		
formatting)	1 Task	Assigned To	Sprint
Build and fine-tune deep learning model using MobileNetV2	N. Venkateswara Reddy	Sprint	2
Train model and validate accuracy	N. Venkateswara Reddy	Sprint	2
Save trained model as rice_model.h5	N. Venkateswara Reddy	Sprint	2
Design and implement Flask backend (app.py, routes, logic)	N. Karthik	Sprint	3
Integrate model with backend and handle predictions	N. Karthik	Sprint	3
Handle error cases and validation (e.g., no file, wrong format)	N. Karthik	Sprint	3
Create HTML templates (index.html, predict.html, result.html)	Ns. I ath Muskan	Sprint	4
Style the UI with CSS for clean, responsive layout	Ns. I ath Muskan	Sprint	4
Add image preview feature on upload	Ns. I ath Muskan	Sprint	4
Perform end-to-end application testing	N. Jagadeeswar Reddy	Sprint	4
Write project documentation (problem, objective, architecture)	Ns. I ath Muskan	Sprint	5
Create architecture diagram and user flow diagram	Ns. I ath Muskan	Sprint	5

Phase-5: Project Development Objective:

Code the project and integrate components from flask import Flask, render_template, request from tensorflow.keras.models import load_model from tensorflow.keras.preprocessing import image from tensorflow.keras.applications.mobilenet_v2 import preprocess_input import numpy as np import os from werkzeug.utils import secure_filename app = Flask(__name__)

```
UPLOAD_FOLDER = 'static/uploaded_images'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER model = load_model('model/rice_model.h5') class_names = ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacad]
@app.route('/') def index():
   return render_template('index.html')
@app.route('/predict', methods=['GET', 'POST'])
def predict():     if request.method == 'POST':
if 'image' not in request.files:             return 'No image uploaded'         file = request.files['image']
if file.filename == '':
          return 'No selected file         filename = secure_filename(file.filename)         file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
file.save(file_path)         img = image.load_img(file_path, target_size=(224, 224))         img_array = image.img_to_array(img)
img_array = preprocess_input(img_array)         img_array = np.expand_dims(img_array, axis=0)         prediction = model.predict(img_array)         pred_index = np.argmax(prediction)
```

```
predicted_class = class_names[pred_index]        confidence =
round(float(np.max(prediction)) * 100, 2)        return
render_template('result.html', image_path=file_path,
predicted_class=predicted_class,
confidence=confidence)
   return render_template('predict.html') if
__name__ == '__main__':
app.run(debug=True)
```

Key Points:

1.Technology Stack Used: (List of programming languages, APIs, etc.)

Programming Language

- Python 3.8+ Used for developing the backend logic, model integration, and image processing.

Machine Learning / Deep Learning

- TensorFlow 2.18.0
  Framework for loading and running the pre-trained deep learning model.
- Keras (via TensorFlow)
  API used for model training, saving, and prediction.
- MobileNetV2 (Transfer Learning)
  Pre-trained CNN model used as the base for rice image classification.

Image Processing

- Pillow
  Used for image loading, resizing, and format conversion.
- NumPy
  Used for numerical operations and array handling.
- Keras Image Utilities
  Functions like load_img, img_to_array, and preprocess_input.

Web Development

- Flask 2.3.3
  Micro web framework for building the user interface and API routes.
- Werkzeug 2.3.7
  Used for secure handling of uploaded files (secure_filename).

2. Development Process: (Steps followed for coding)

Step 1: Problem Understanding & Planning

- Identified the real-world need for rice grain classification.
- Defined the project scope, target users, and technical requirements.
- Outlined project architecture, timeline, and technology stack.

Step 2: Dataset Collection & Preparation

- Collected a labeled dataset of di erent rice grain images.
- Organized images into folders based on rice type.
- Resized and preprocessed images for uniform input size (224x224).
- Split data into training, validation, and test sets.

Step 3: Model Building using Transfer Learning

- Selected MobileNetV2 as the base model.
- Replaced the top layers to suit the 5 rice classes.
- Compiled, trained, and validated the model using TensorFlow/Keras.
- Saved the trained model as rice_model.h5 for deployment.

Step 4: Flask Web App Development

- Created a Flask app (app.py) with routes: o / for homepage o /predict for file upload and prediction
- Implemented image upload functionality using HTML forms and Flask.
- Integrated the trained model to make predictions from uploaded images.

Step 5: Frontend Design

- Designed three core templates using HTML & Jinja2:

- index.html – Welcome page
- predict.html – Image upload form
- result.html – Displays prediction and confidence score

- Styled templates for a clean, user-friendly interface.

Step 6: Integration & Testing

- Connected the frontend to the backend.
- Tested predictions with multiple sample images.
- Handled edge cases (e.g., no file uploaded, wrong file type).

Step 7: Documentation & Finalization

- Wrote project documentation including:
  - Problem statement
  - Purpose & impact
  - Architecture, timeline, tasks, and sprint plans
- Prepared final deliverables: working app + report.

13. Challenges & Fixes: (Mention any obstacles faced and how they were solved)

Challenge 1: Small or Unbalanced Dataset

Issue:
The initial dataset had fewer images for certain rice types, leading to class imbalance and lower accuracy.

Fix:

- Performed data augmentation (rotation, flipping) to artificially increase underrepresented classes.
- Ensured equal representation of all classes during training. Challenge 2:

Image Preprocessing Errors

Issue:
Flask app crashed when incompatible or corrupted image files were uploaded.

Fix:

- Implemented proper file type validation.
- Used secure_filename() and try-except blocks to catch exceptions.
- Added user-friendly error messages for invalid uploads. Challenge 3:

Model Overfitting

Issue:
Model showed high accuracy during training but poor performance on unseen data.

Fix:

- Added dropout layers during training.
- Used early stopping and validation data to prevent over-training.
- Adjusted learning rate and batch size for better generalization. Challenge

4: Integrating Model with Flask

Issue:
Model integration with Flask resulted in long load times and memory issues on some systems.

Fix:

- Loaded the model once globally instead of on every request.
- Reduced image size to 224x224 for faster inference without losing

accuracy. Challenge 5: HTML Display Issues

Issue:
The uploaded image path sometimes broke or did not render properly on the result page.

Fix:

- Used Flask's url_for() and proper path routing inside static/ to ensure image rendering worked correctly across di erent environments.

Phase-6: Functional & Performance Testing Objective:

Ensure the project works as expected.

Functional Testing:

Test Case	Expected Result	Status

Upload a valid rice image	Returns correct rice type with confidence score	Passed
Upload a non-image file (e.g., .txt)	Displays error: “Unsupported file format”	Passed
Submit form with no file selected	Displays error: “No selected file”	Passed
Upload image in .jpg, .jpeg, .png	Accepted and processed correctly	Passed
Verify prediction route loads (/predict)	Prediction form renders properly	Passed
Verify result page shows image & prediction	Uploaded image, rice type & confidence are shown clearly	Passed

Performance Testing:

Scenario	Observation	Result
Time taken for single prediction	~1–2 seconds on average machine	Acceptable
Model load time during startup	~2–3 seconds (loaded once at app start)	Optimized
Multiple simultaneous requests (manual test)	No crash, handles 2–3 parallel predictions fine	Stable
Large image upload (~5MB)	Accepts, resizes, and predicts successfully	Handled

Key Points:

1. Test Cases Executed: (List the scenarios tested) Functional Test Cases:

Test Case ID	Test Scenario	Expected Result	Status
TC01	Upload valid .jpg image of Rice type	predicted with 脕 rice confidence score	Passed
TC02	Upload .png format image	Rice type predicted successfully 脕	Passed
TC03	Submit form without selecting a file	脕 Error message: “No selected file”	Passed
TC04	Upload non-image file (e.g., .txt, .pdf)	Error message: “Unsupported file 脕 format” or similar	Passed
TC05	Upload corrupted or unreadable image	App handles gracefully without 脕 crashing	Passed

Test Case ID	Test Scenario	Expected Result	Status
TC06	Uploaded image preview on result page	Uploaded image is displayed 脕 correctly	Passed
TC07	Display predicted class	rice Correct rice type is shown on the 脕 result page	Passed
TC08	Display confidence percentage	Confidence is shown in proper 脕 format (e.g., 93.47%)	Passed
TC09	Homepage works (/)	navigation index.html loads correctly	脕 Passed
TC10	Prediction route works (/predict)	Upload form loads without error	脕 Passed
TC11	Result displays	after result.html loads with prediction 脕	

	prediction	and image	Passed

Performance & UX Test Cases:

Test Case ID	Test Scenario	Expected Result	Status
TC12	Time to generate prediction (≤ 2 seconds)	Acceptable delay	脕 Passed
TC13	Multiple predictions in a row	No memory leak or app crash	脕 Passed
TC14	Navigation across pages (index → predict → result)	Smooth transition	脕 Passed
TC15	Basic mobile responsiveness	Layout adapts on smaller screens	脕 Passed
TC16	Display user-friendly errors	Clear messages for invalid input	脕 Passed

2. Bug Fixes & Improvements: (Mention fixes made) Bug Fixes:

Issue	Cause	Fix Implemented
File upload crashed on unsupported file types	No file type validation	Added MIME type check and validation before processing
App crashed on Empty file.filename not handled selecting a file	Added condition to check not submitting without and display “No file selected”	
Model was reloading on	Moved model = Model was loaded inside	

load_model() outside the every prediction request route handler route (global scope)

Uploaded image not displaying on result template

Improper static file path in Fixed image path using Flask's url_for('static', ...) page

High-resolution images weren't resized before processing

Long load time on large images

Resized images to 224x224 before prediction

Improvements Made:

Improvement	Purpose
Added confidence percentage with predictions	Helps users understand how certain the model is
Simplified UI with clear buttons and labels	Improved overall user experience
Styled result.html to show image and prediction	Enhanced readability and layout
Added user-friendly error messages	Prevented confusion and improved form handling

Improvement	Purpose
Included image preview on upload selection	Helps user verify correct image

3. Final Validation: (Does the project meet the initial requirements?)

To confirm that the GrainPalette project satisfies all the functional, technical, and user-oriented requirements defined during the planning phase.

Initial Requirement	Status	Remarks
Allow users to upload rice grain 脥 images via a web interface	Met	Upload works for .jpg, .jpeg, .png images

Classify rice into 5 categories using a pre-trained DL model	Met	Uses MobileNetV2 with trained model rice_model.h5
Display predicted rice type and confidence score	Met	Results shown clearly on result.html
Ensure image preprocessing before prediction	Met	Images are resized (224x224) and normalized using preprocess_input()
Handle invalid uploads gracefully	Met	Proper error messages and file validations implemented
Deliver a simple and clean UI	Met	HTML templates are easy to navigate and responsive
Test functionality across multiple scenarios	Met	All major test cases passed, including edge cases
Maintain performance (fast response time and smooth processing)	Met	Predictions load in 1–2 seconds on average machines
Generate complete documentation	Met	Includes problem, solution, architecture, testing, tasks, timeline, etc.

APPENDIX

Source Code: [https:/github.com/karthik8094/GrainPalette-A-Deep-Learning-Odyssey-In-Rice-Type-Classification-Through-Transfer-Learning]
Dataset:[https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset]
DemoVideo:[https://drive.google.com/file/d/1vJFtvds4eBB0rQJlTZXKhZ-43ReSpmMx/view?usp=drivesdk]