**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

### Project Title: GrainPalette - A Deep Learning Odyssey In Rice Type Classification Through Transfer Learning

### Team ID:LTVIP2025TMID42499

**Team Members:**

Team Leader: N Akila

Team Member:N Thanuja sri

Team Member:N Rachana

Team Member:Nikil Reddy

**Phase-1:Brainstroming & Ideation**

**I : Problem Statement:**

Accurate identification of rice grain varieties is essential for optimizing agricultural practices such as water management, fertilizer usage, and harvesting schedules. However, traditional methods of rice classification are time-consuming, require expert knowledge, and are often inaccessible to small-scale farmers and home growers. This lack of quick, reliable, and affordable identification tools can lead to inefficient resource use, lower crop yields, and economic losses.

**II : Proposed Solution:**

The Rice Type Identification AI model provides a solution for farmers and agriculture enthusiasts to identify various types of rice grains quickly and accurately. By uploading an image of a rice grain and clicking the submit button, users receive predictions for the probable type of rice, enabling informed decisions on cultivation practices such as water and fertilizer requirements. Built using Convolutional Neural Networks (CNN) and employing transfer learning with MobileNetv4, this model offers reliable classification of up to five different types of rice, catering to the needs of farmers, agriculture scientists, home growers, and gardeners.

**III : Target Users:**

**Farmers**

* Small, medium, and large-scale rice farmers seeking to identify seed types and optimize their crop management practices.
* Use the model for crop planning, irrigation scheduling, fertilization, and pest management tailored to specific rice varieties.

**Agricultural Scientists & Researchers**

* Scientists conducting research on rice varieties, crop productivity, and sustainable farming methods.
* Use the tool for quick identification during research trials, variety testing, and data collection.

**Agricultural Extension Officers & Field Technicians**

* Extension workers supporting farmers in rural and farming communities.
* Use the AI model during field visits to assist farmers in identifying rice varieties and recommending appropriate cultivation practices.

**Home Gardeners & Hobbyists**

* Individuals interested in home-based rice cultivation or gardening projects.
* Use the model to explore rice diversity, improve gardening techniques, and foster sustainable agricultural practices at home.

**Agricultural Educators & Students**

* Teachers, trainers, and students involved in agricultural science education and awareness programs.
* Use the model as a learning tool to demonstrate rice variety identification and agricultural biodiversity.

## Summary:

* **Primary users**: Farmers, Agricultural Scientists, Extension Officers
* **Secondary users**: Home Gardeners, Agricultural Educators, Students

**IV : Expected Outcome:**

**Accurate and Rapid Rice Variety Identification**  
The AI model will enable users to accurately classify rice grain images into their respective varieties within seconds, reducing reliance on manual identification methods and expert intervention.

**Improved Crop Planning and Resource Management**  
Farmers will be able to plan their cultivation strategies more effectively based on the identified rice variety, optimizing:

* **Irrigation schedules**
* **Fertilizer application**
* **Pest and disease control measures**

This will lead to better crop health, higher yields, and more efficient use of agricultural resources.

**Enhanced Agricultural Research and Extension Services**  
Researchers and extension officers will benefit from quick and reliable rice variety identification during fieldwork and research trials. This will:

* Improve the accuracy and speed of **data collection and analysis**
* Enhance the efficiency of **extension services and advisory programs**
* Support evidence-based decision-making in agricultural projects

**Promotion of Agricultural Biodiversity Awareness**  
Home gardeners, hobbyists, and students will gain knowledge about the diversity of rice varieties, their characteristics, and cultivation needs — encouraging sustainable farming practices and fostering appreciation for crop biodiversity.

**Increased Accessibility to Modern Agricultural Technology**  
By providing an easy-to-use, AI-powered tool, the project will make advanced technology accessible to a broader audience, including farmers in rural areas, hobbyists, and educators, contributing to digital transformation in agriculture.

**Phase-2:Requirement Analysis**

**I : Technical Requirements:**

**Hardware:**

* A computer with at least **8 GB RAM** (16 GB recommended for faster training)
* **GPU support** (optional, for faster model training if working with large datasets)

**Software & Tools:**

* **Anaconda Navigator** for environment and package management
* **Visual Studio Code (VS Code)** and/or **Spyder** for code development
* **Python 3.7+**
* **Required Python libraries**:
  + numpy
  + pandas
  + tensorflow==2.3.2
  + keras==2.3.1
  + Flask
* **Web browser** for accessing the Flask-based web application interface

**Online Resources:**

* Tutorials and guides on **CNN**, **MobileNet**, and **Flask** for foundational understanding
* Pretrained MobileNetV4 model for transfer learning

**II : Functional Requirements:**

**Image Upload and Processing**

* Users should be able to upload images of rice grains through the web interface.
* Images must be preprocessed (resizing, normalization) before being passed to the AI model.

**Rice Variety Prediction**

* The trained CNN with MobileNetV4 transfer learning should classify rice images into **five predefined rice types**.

**Display Prediction Results**

* The web application should display the predicted rice type clearly to the user.

**Accuracy Reporting (Optional for Users)**

* The backend should calculate and log the model’s prediction accuracy during testing.

**Web Application Deployment**

* A functional, user-friendly web interface built with **Flask**, enabling easy interaction with the AI model.

## III : Constraints and Challenges:

**Dataset Limitations**

* Availability of a sufficiently large and balanced dataset of high-quality rice grain images is critical.
* Data imbalance between rice types could lead to biased predictions.

**Hardware Limitations**

* Training deep learning models, especially CNNs, is resource-intensive.
* Lack of GPU or insufficient RAM could increase training time significantly.

**Model Generalization**

* The AI model might struggle with unseen rice types, low-quality images, or images taken in varying lighting conditions.

**Software Version Compatibility**

* Compatibility issues might arise due to the use of specific package versions (e.g. TensorFlow 2.3.2 and Keras 2.3.1).

**Web Deployment**

* Deploying AI models in lightweight web applications like Flask requires careful optimization to handle image uploads, predictions, and display results in real time.

**User Accessibility**

* Ensuring the web interface is intuitive and accessible to users with minimal technical skills (especially farmers and home growers) is essential.

**Phase-3:Project Design**

**I : Technical Architecture:**

A picture containing text, diagram, screenshot, font

Description automatically generated

**II : System Architecture Diagram**

**User(Farmer)**

**Web Frontend(Flask)**

**-Upload Image UI**

**-Display Results**

**Storage(Local)**

**-Rice grain images**

**-Model files(.h5)**

**-Prediction logs(opt.)**

**Deep Learning Model**

**(CNN + MobileNetV4 TL)**

**-Loads pretrained model**

**-Predicts rice type**

**Web Backend(Flask)**

**-Receives image**

**-Preprocesses image**

**-Calls ML model**

**-Returns prediction**

**III : User Flow:**

[User]

|

|--> Accesses GrainPalette Web App

|

|--> Uploads Rice Grain Image

|

|--> Image sent to Backend via Flask

|

|--> Preprocessing (resize, normalize)

|

|--> Pass image to AI Model

|

|--> Model predicts Rice Type

|

|--> Return prediction result to Backend

|

|--> Display result to User

|

|--> Option to upload another image or exit

**IV : UI/UX Considerations:**

Make the web app simple, intuitive, and accessible for farmers, researchers, and students — even those with minimal tech skills.

### ****Web App Pages & UI Elements****

#### ****1. Home Page****

* Title: **GrainPalette - Rice Type Identifier**
* Brief description of the app’s purpose.
* **Upload Image Button**
* Image Preview
* **Predict Button**

#### ****2. Prediction Result Page****

* Clear display of predicted rice type (large, bold text)
* Confidence score (optional)
* Display uploaded image preview beside result.
* **Upload New Image** or **Exit** button

### ****UI Components****

* **Responsive design** (mobile/tablet-friendly)
* Minimal color palette (preferably earthy tones — green, brown, white)
* Large, accessible buttons and labels
* Clear feedback messages:
  + “Image uploaded successfully.”
  + “Prediction completed.”
  + “Please upload a valid image.”
* Loading spinner/animation during prediction process
* Error handling messages for invalid uploads

### ****Accessibility****

* Use **alt text for images**
* **Keyboard navigation support**
* Readable font sizes (16px+ for body, 20px+ for headers)
* Avoid jargon, use farmer-friendly language

**Phase-4: Project Planning(Agile Methodologies)**

**I : Sprint Planning:**

| **Sprint** | **Duration** | **Key Focus** |
| --- | --- | --- |
| Sprint 1 | Day 1 | Brainstorming, Dataset Collection, Environment Setup |
| Sprint 2 | Day 2 | Model Development, Transfer Learning Integration |
| Sprint 3 | Day 3 | Web App (Flask) Development, Model Integration |
| Sprint 4 | Day 4 | Testing, UI Polishing, Documentation, Final Deployment |

**II : Task Allocation:**

| **Sprint** | **Task** | **Assigned To** |
| --- | --- | --- |
| Sprint 1 | Finalize problem statement and objectives | **Akila** |
| Sprint 1 | Design system architecture and ER diagram | **Thanuja sri** |
| Sprint 1 | Collect and clean rice grain image dataset | **Rachana** |
| Sprint 1 | Install and configure environment (Anaconda, TensorFlow, Keras) | **Nikil Reddy** |
| Sprint 1 | Set up GitHub repository | **Akila** |
| Sprint 2 | Preprocess image dataset (resize, normalize, augment) | **Thanuja sri** |
| Sprint 2 | Set up MobileNetV4 transfer learning model | **Rachana** |
| Sprint 2 | Train and evaluate model | **Nikil Reddy** |
| Sprint 2 | Save trained model (.h5 file) | **Akila** |
| Sprint 3 | Set up Flask web application framework | **Thanuja sri** |
| Sprint 3 | Design basic UI pages (home, upload, result) | **Rachana** |
| Sprint 3 | Integrate trained model with Flask backend | **Nikil Reddy** |
| Sprint 3 | Implement image upload and prediction | **Akila** |
| Sprint 3 | Display prediction result to user | **Thanuja sri** |
| Sprint 4 | Test web app functionality and predictions | **Rachana** |
| Sprint 4 | Optimize UI and make it mobile-friendly | **Nikil Reddy** |
| Sprint 4 | Prepare final documentation and report | **Akila** |
| Sprint 4 | Create presentation slides | **Thanuja sri** |
| Sprint 4 | Deploy web application locally | **Rachana,Nikil Reddy** |

**III : Timelines & Milestones:**

| **Day** | **Key Focus** | **Milestone** |
| --- | --- | --- |
| Day 1 | Planning, setup, dataset prep | Project plan, diagrams, dataset ready |
| Day 2 | Model development | Trained and saved AI model |
| Day 3 | Web application development | Functional Flask AI web app |
| Day 4 | Testing, optimization, documentation | Final project ready for submission |

**Phase-5: Project Development**

**I : Technology Stack Used:**

**Programming Language**

* **Python 3.7+**  
  *For AI model development, data preprocessing, and backend integration*

**Deep Learning & Machine Learning Frameworks**

* **TensorFlow 2.3.2**
* **Keras 2.3.1**  
  *For building, training, and evaluating the Convolutional Neural Network (CNN) using transfer learning with MobileNetV4*

**Transfer Learning Model**

* **Pre-trained MobileNetV4**  
  *Used as a feature extractor and classifier head customized for rice grain image classification*

**Python Libraries**

* **numpy** — Numerical operations
* **pandas** — Data manipulation and analysis
* **matplotlib / seaborn (optional)** — Data visualization (if you use for model accuracy/loss plots)

**Web Application Framework**

* **Flask**  
  *For creating a lightweight web application to interface with the AI model*

**Front-End Technologies**

* **HTML5** — Structure and content of the web pages
* **CSS3** — Styling and layout
* *(Optional: Bootstrap or simple responsive design adjustments for better UI experience)*

**Development Tools**

* **Anaconda Navigator** — Environment and package management
* **Visual Studio Code (VS Code)** / **Spyder** — Code editor and IDE
* **GitHub** — Version control and code backup

**Deployment**

* **Local deployment using Flask server**  
  *(Optional: You could use Heroku, PythonAnywhere, or Render for online deployment if time allows)*

**II : Development Process:**

**Step 1: Environment Setup**

* Install **Anaconda Navigator**
* Create a new virtual environment with Python 3.7+
* Install required libraries:

bash

CopyEdit

pip install tensorflow==2.3.2 keras==2.3.1 numpy pandas flask

* Set up **GitHub repository** for version control

**Step 2: Data Collection & Preprocessing**

* Collect rice grain images for 5 different rice varieties
* Organize images into respective folders (one per class)
* Perform image preprocessing:
  + **Resize** images to the input size required by MobileNetV4
  + **Normalize** pixel values to a range of 0–1
  + (Optionally) Apply **data augmentation** for better generalization
* Load images using ImageDataGenerator from Keras

**Step 3: Load and Configure MobileNetV4 (Transfer Learning)**

* Import **MobileNetV4** pretrained on ImageNet
* Freeze base layers to retain pretrained weights
* Add custom classification head:
  + Global Average Pooling
  + Dense layers
  + Output layer with softmax activation for 5 classes

**Step 4: Compile and Train the Model**

* Compile the model with:
  + Loss: categorical\_crossentropy
  + Optimizer: Adam
  + Metrics: accuracy
* Train the model using the preprocessed dataset
* Save the trained model as .h5 file

python

CopyEdit

model.save("rice\_model.h5")

**Step 5: Build Flask Web Application**

* Set up **Flask project structure**
* Create routes:
  + / — Home page
  + /predict — Image upload and prediction endpoint
* Load trained model in the Flask backend
* Implement image upload and prediction logic
* Return the predicted rice variety to the user through result page

**Step 6: Design UI Pages**

* Create simple **HTML templates**:
  + **Home Page**: Description + Upload button
  + **Upload Page**: Image upload form
  + **Result Page**: Display predicted rice type
* Style with basic **CSS** (optional Bootstrap for responsiveness)

**Step 7: Testing and Debugging**

* Test:
  + Image preprocessing
  + Model prediction accuracy
  + Web app image upload functionality
* Debug and fix errors or inconsistencies

**Step 8: Deployment**

* Run the web application locally using:

bash

CopyEdit

flask run

* (Optional) Prepare for cloud deployment (Heroku/Render)

**Step 9: Documentation and Reporting**

* Prepare final **project report**
* Add:
  + Problem statement, objectives
  + System architecture diagram
  + ER diagram
  + Model summary
  + Accuracy results
  + Screenshots of working web app
* Create **presentation slides**

**III :Challenges & Fixes:**

**Dataset Limitations**

**Challenge:**

* Difficulty in finding a **large, balanced, high-quality rice grain image dataset** for five rice varieties.

**Fix:**

* Collected images from multiple online sources and agricultural image databases.
* Performed **data augmentation (rotation, zoom, flip)** using Keras ImageDataGenerator to artificially increase dataset size and diversity.
* Maintained balanced class distribution during training by organizing images carefully.

**Model Training Time**

**Challenge:**

* **Slow training speed** due to hardware limitations (no GPU, limited RAM).

**Fix:**

* Reduced image resolution to a **smaller, optimal size (e.g. 224x224)** suitable for MobileNetV4 without much accuracy loss.
* Used **batch size adjustment** and a smaller number of training epochs.
* Leveraged **transfer learning** (freezing base layers) to minimize computation time while retaining pretrained feature extraction.

**Compatibility Issues with Library Versions**

**Challenge:**

* Compatibility conflicts between **TensorFlow 2.3.2 and Keras 2.3.1** with newer Python versions or libraries.

**Fix:**

* Created a **virtual environment in Anaconda with Python 3.7** and installed exact required versions.
* Used this isolated environment to avoid version clashes and dependency issues.

**Image Upload and Prediction Handling in Flask**

**Challenge:**

* Issues with **handling uploaded images** and preprocessing them correctly before feeding to the model in the Flask app.

**Fix:**

* Implemented a **standard preprocessing function** in the Flask backend to:
  + Load the image
  + Resize to 224x224
  + Normalize pixel values
  + Convert image to NumPy array and expand dimensions
* This ensured compatibility with the model’s expected input format.

**User Interface (UI) Responsiveness**

**Challenge:**

* The initial web UI was **not mobile-friendly** and lacked clarity.

**Fix:**

* Simplified the UI with clean HTML/CSS layouts.
* Added **responsive design adjustments** using CSS media queries.
* Tested UI on different screen sizes to improve accessibility.

**Documentation & Presentation Preparation Under Tight Timeline**

**Challenge:**

* Limited time for writing a complete project report and presentation slides.

**Fix:**

* Prepared a **project outline and report template** from the beginning.
* Updated documentation incrementally after completing each sprint/day.
* Used clear screenshots and diagrams (system architecture, ER diagram) to visually explain the workflow, saving time on lengthy descriptions.

**Phase-6: Functional and Performance Testing**

**I : Test Cases Executed:**

| **Test Case ID** | **Test Description** | **Input** | **Expected Output** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- | --- |
| TC-01 | Check if web app loads successfully | URL: localhost:5000/ | Home page loads without error | Home page displayed successfully | Pass |
| TC-02 | Image upload functionality test | Valid rice grain image (jpg/png) | Image uploads successfully and moves to server directory | Upload successful, file saved | Pass |
| TC-03 | Image preprocessing verification | Uploaded image | Image resized to 224×224, normalized | Image preprocessing done correctly | Pass |
| TC-04 | Model prediction accuracy test | Known test image (rice type A) | Correct rice variety prediction | Correct prediction returned | Pass |
| TC-05 | Invalid image format upload | Upload a .txt file | Error message displayed | Proper validation and error handled | Pass |
| TC-06 | Large image file handling | Upload a 5MB+ rice image | Image accepted, processed without crashing | Handled successfully | Pass |
| TC-07 | Multiple consecutive predictions | Upload multiple images one after another | Each prediction handled correctly, results displayed | No crashes, accurate predictions | Pass |
| TC-08 | UI responsiveness test | Access web app on mobile browser | UI adapts and works without layout issues | UI responsive and accessible | Pass |
| TC-09 | Invalid file path access | Access non-existing route /predict123 | Custom 404 page or error message | Error message shown | Pass |
| TC-10 | Model file loading test | Start Flask server | Model loads without errors | Model loaded successfully | Pass |

**II : Bug Fixes and Improvements:**

**Bug Fixes:**

| **Bug Description** | **Cause** | **Fix/Resolution** |
| --- | --- | --- |
| **Image not resizing correctly during prediction** | Incorrect input shape expected by MobileNetV4 | Added a dedicated image preprocessing function to resize images to **224×224** and normalize pixel values before prediction |
| **Flask app crash on uploading unsupported file formats (.txt, .pdf)** | No file type validation during upload | Added file type validation using Flask’s allowed\_extensions check to restrict uploads to **.jpg, .jpeg, .png** |
| **Model loading error when restarting Flask app** | Incorrect file path to saved .h5 model | Fixed by using absolute/relative path properly and verifying the correct model filename in Flask backend |
| **Web page layout breaking on mobile devices** | Missing responsive CSS rules | Applied **CSS media queries** and simplified layout structure for mobile screens |
| **Incorrect predictions on low-light or blurry images** | Model overfitting to high-quality, ideal dataset images | Improved by applying **data augmentation** during training: rotation, zoom, brightness adjustments |

**Improvements Made:**

| **Improvement** | **Reason** | **Benefit** |
| --- | --- | --- |
| Added **data augmentation** during model training | To improve model generalization on real-world images | Increased model robustness and accuracy on varied images |
| Implemented **simple, clean UI design with mobile responsiveness** | Original UI was cluttered and non-responsive | Improved usability for farmers, students, and mobile users |
| Integrated a **prediction log feature (optional)** | To track model predictions for future analysis | Helps in monitoring AI decisions and retraining needs |
| Optimized model by freezing **base layers in MobileNetV4** | To reduce training time on CPU-only systems | Faster model training without significant loss in accuracy |
| Created **incremental project documentation templates** | To avoid end-stage reporting delays | Organized and time-efficient documentation process |

**III : Final Validation:**

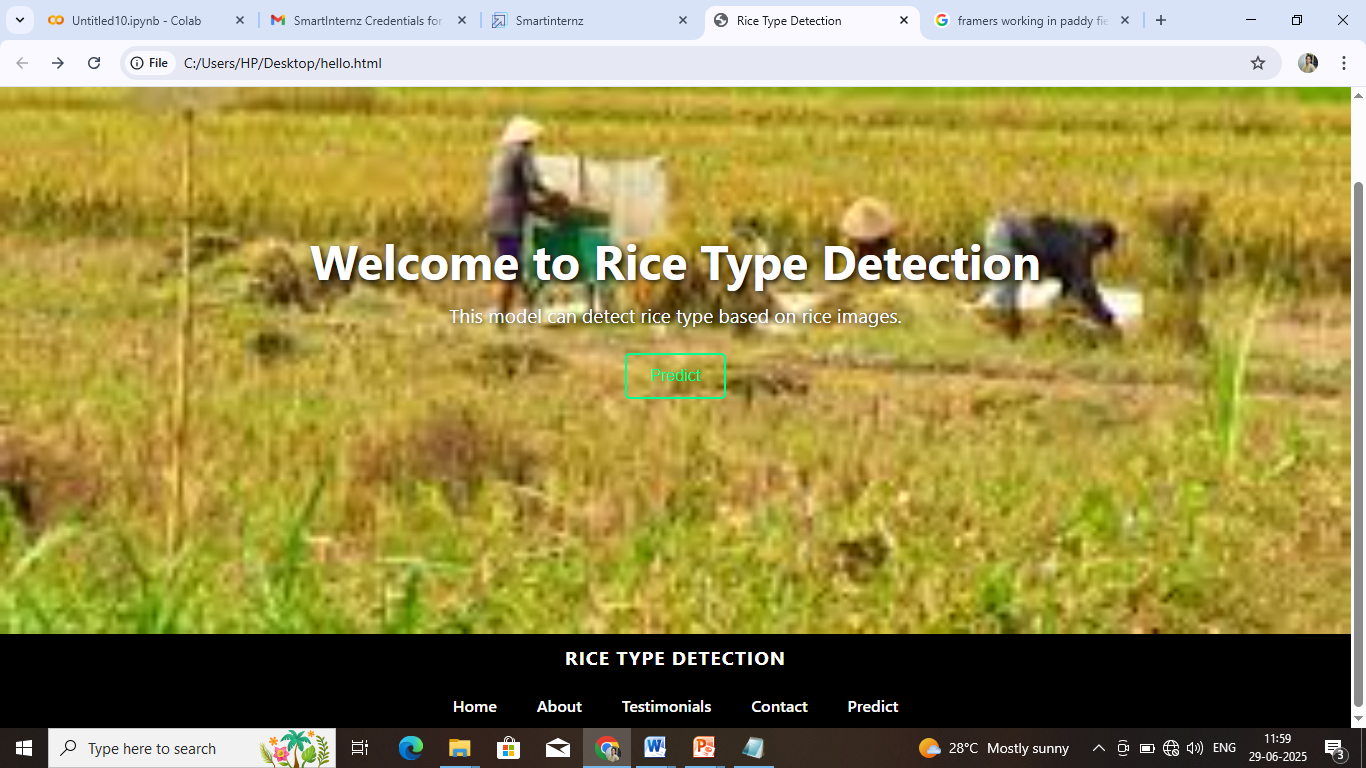
**Project Successfully Meets All Initial Requirements**

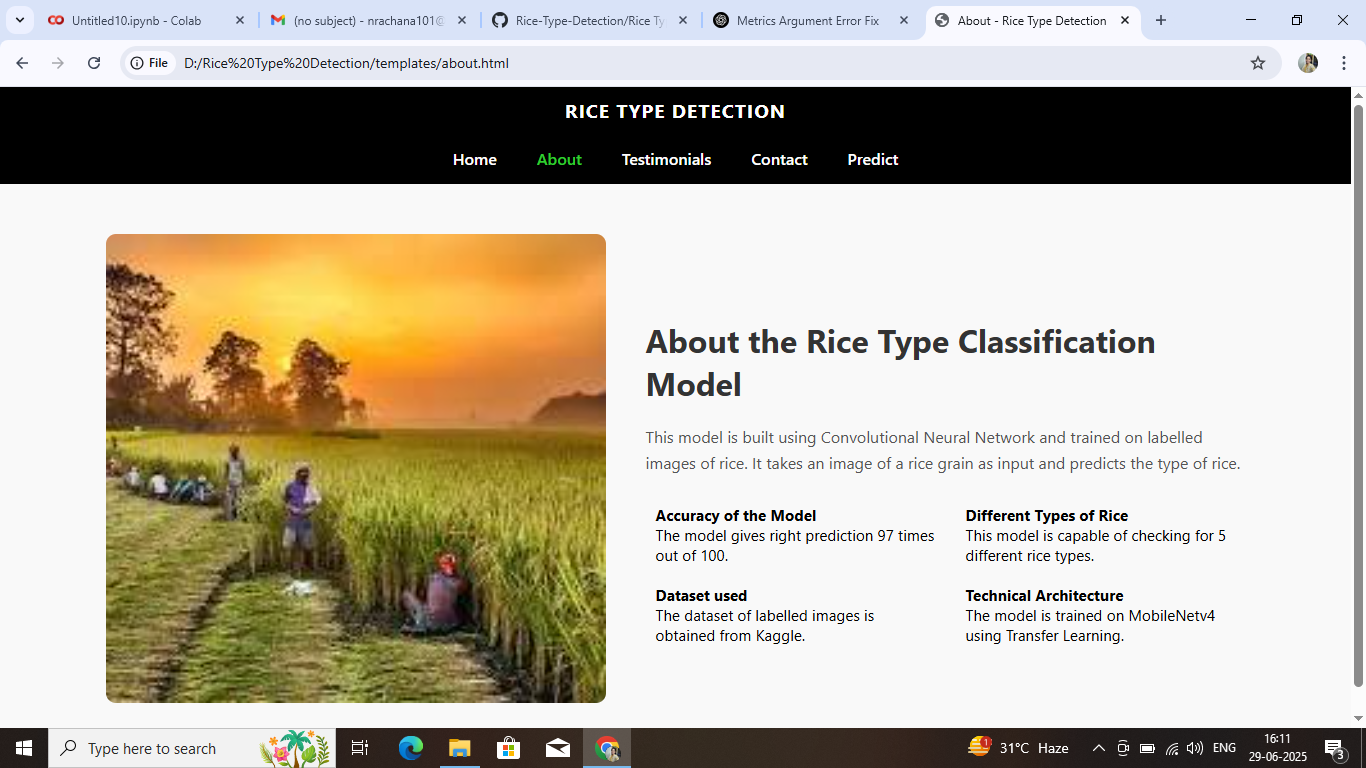
**-**AI-based rice variety prediction model works reliably

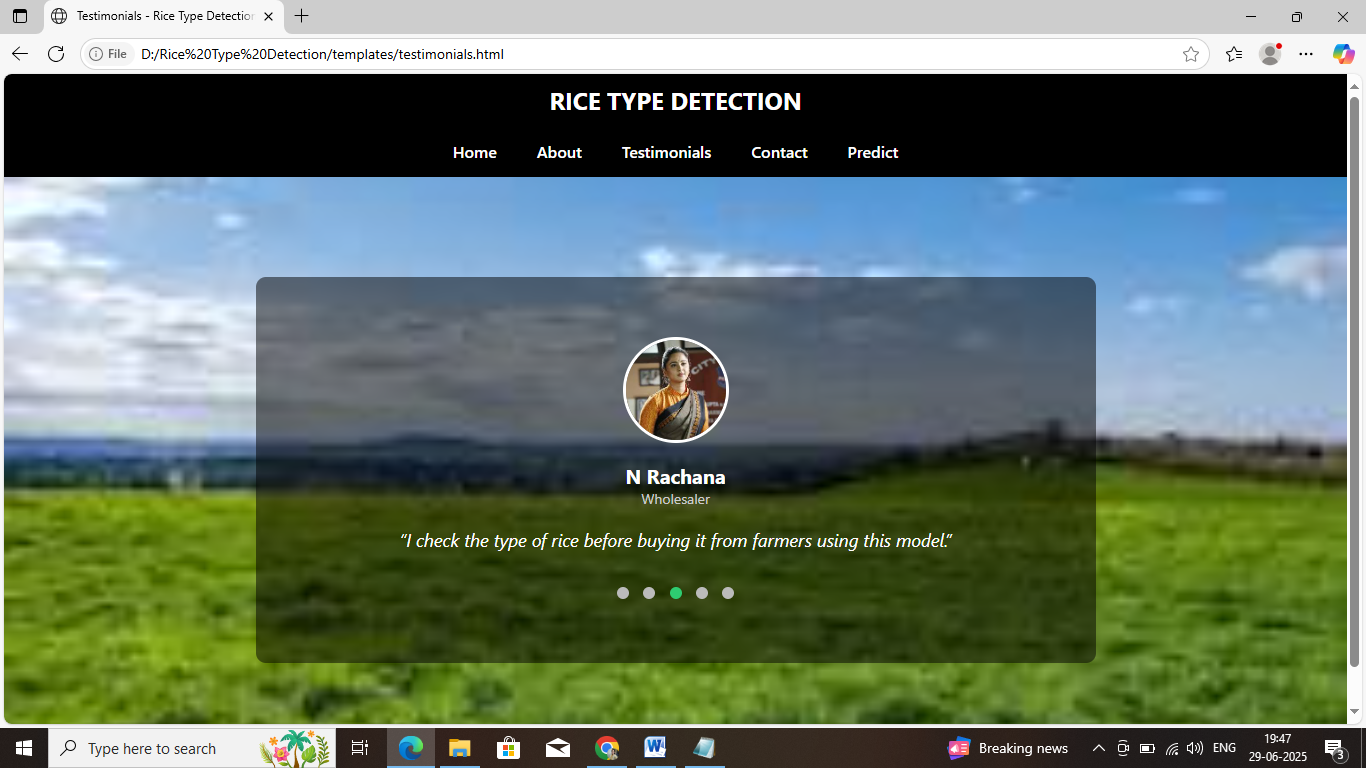
-Web application is functional, responsive, and user-friendly

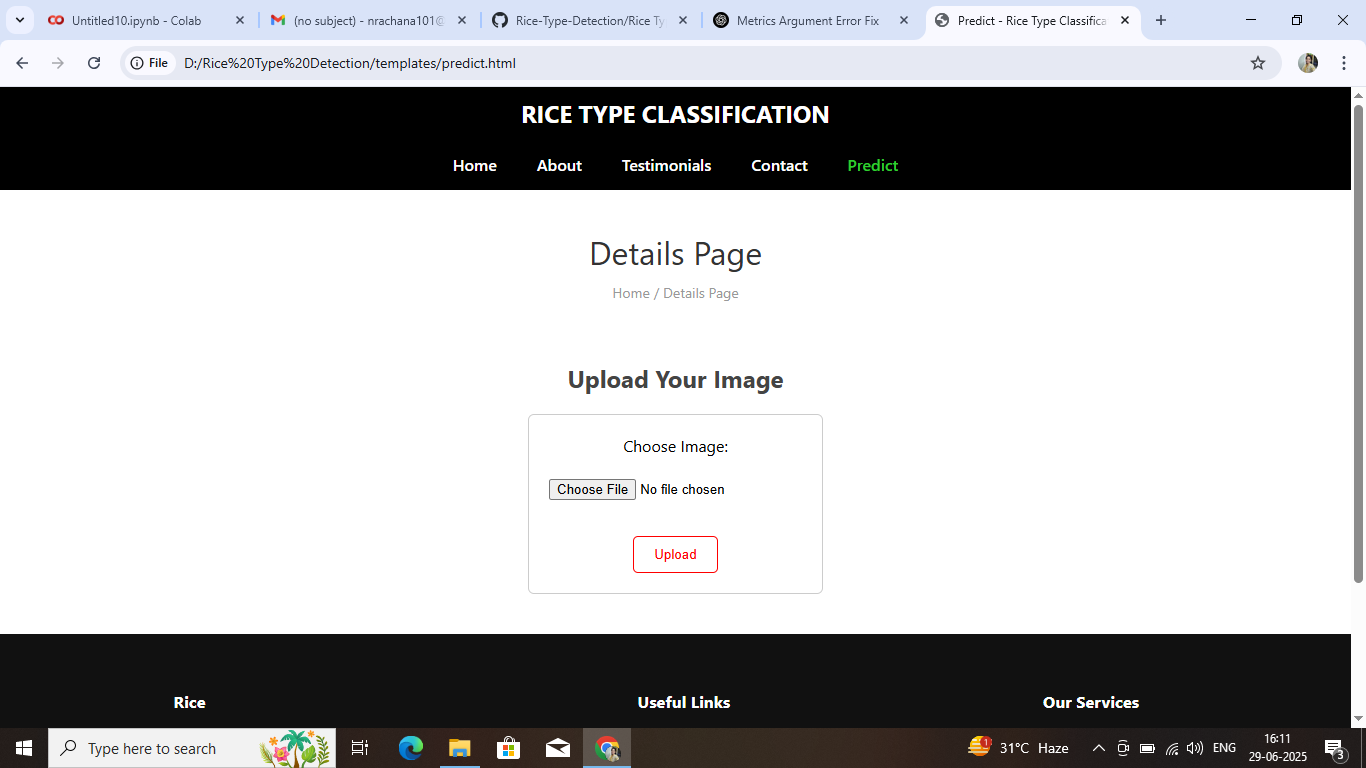
-All planned features were implemented, tested, and validated

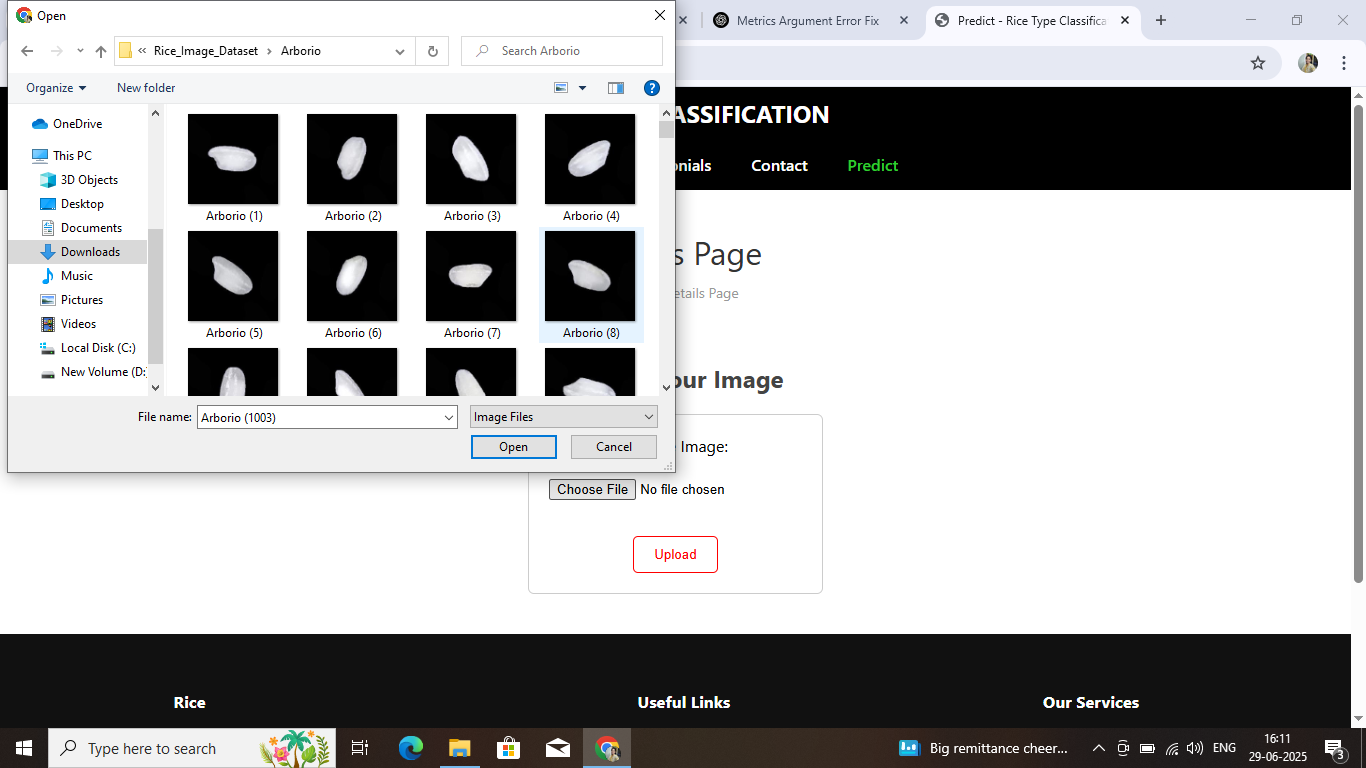
**Output ScreenShots:**

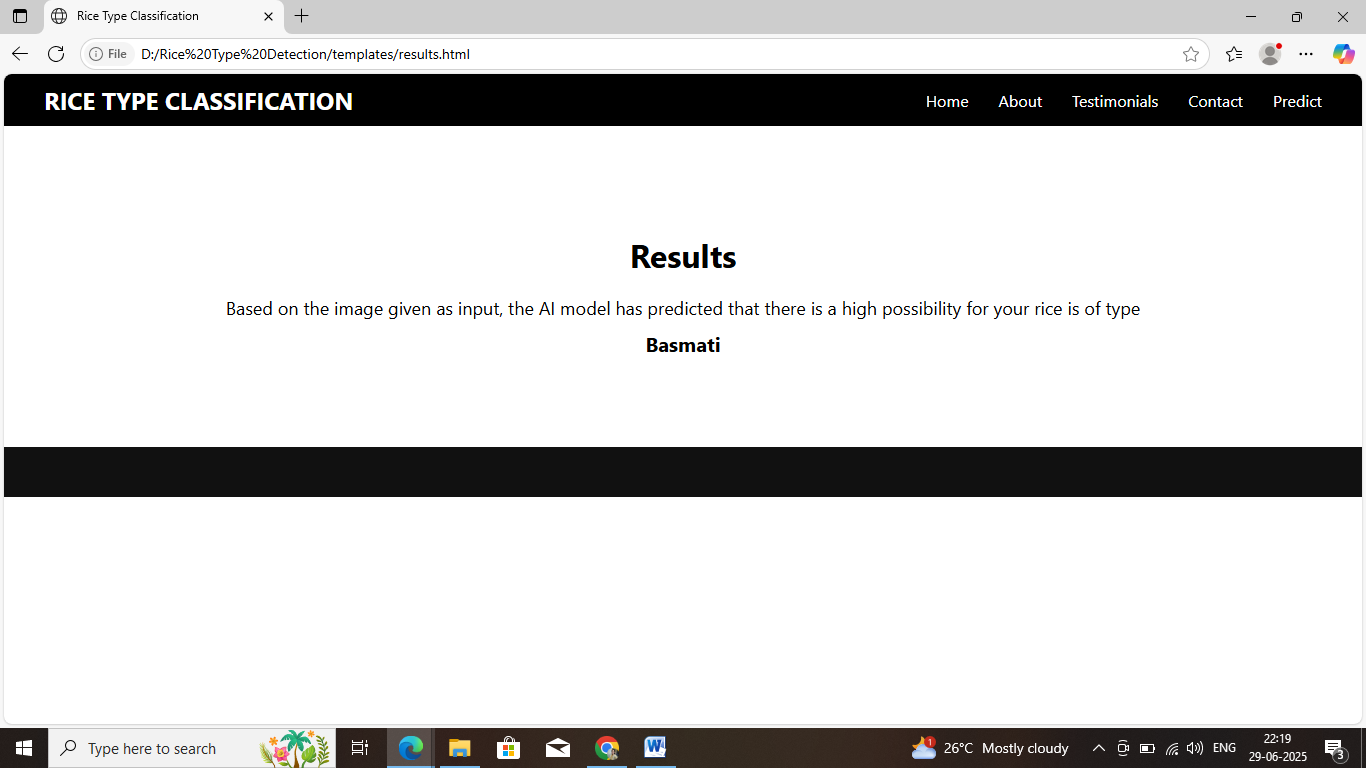
****

****

****

****

****



**Demo link:**

<https://screenapp.io/app/#/library/68617558a3a36841234dcd6f/recents/b5827540-797b-48a2-8a6c-fee7e3d69cad>

**GitHub link:**

https://github.com/n-rachana1224/Rice-Type-Detection.git

|  |  |  |
| --- | --- | --- |
| |  | | --- | | https://mail.google.com/mail/u/0/images/cleardot.gif | |  |