**Project Title:**

Grain Palette: A Deep Learning Odyssey in Rice Type Classification Through Transfer Learning

**Team ID : LTVIP2025TMID33662**

**Team Leader :**Chigurupati Satwika

**Team member :**Chalamalasetti Gopi Sai

**Team member :**Burepalli Sreya

**Team member :**Ch Rohith Venkat

**Phase-1: Brainstorming & Ideation**

**Objective:** To develop an AI-powered rice grain classification system that leverages transfer learning techniques to accurately identify and classify rice varieties based on image data.

**Key Points:** - Deep learning-based grain classification system - Uses transfer learning with pre-trained CNNs - Targeted for use in agriculture, food processing, and quality control - Deployable via web application for accessibility

**Problem Statement:** Manual classification of rice types is time-consuming, subjective, and error-prone. Accurate identification is essential in food grading, export quality assurance, and agricultural research.

**Proposed Solution:** A CNN-based image classification system using transfer learning to classify rice grain images into specific types with high accuracy.

**Target Users:**- Food inspectors and processors- Agricultural researchers- Exporters and quality assurance personnel

**Expected Outcome:**- Accurate rice type prediction with high precision- Reduction in manual inspection effort- Scalable model that can be adapted to other grains

**Phase-2: Requirement Analysis**

**Objective:** To analyze the technical and functional requirements for building the rice grain classifier using pre-trained CNN models.

**Key Points:** - Transfer learning using models like VGG16, ResNet50 - Image preprocessing and augmentation - Real-time prediction via web UI

**Technical Requirements:** - Python, TensorFlow/Keras, OpenCV - Flask for backend integration - HTML/CSS/JS for frontend - Jupyter Notebook or Colab for development

**Functional Requirements:** - Upload rice grain image- Preprocess image for classification- Predict rice type using trained CNN- Display result on web UI

**Constraints & Challenges:** - Image noise and variations in lighting - Limited rice image datasets -

Ensuring inference speed for real-time use

**Hardware & Software Requirements**

a. Hardware Requirements

* Processor: Intel i5 or higher (or equivalent)
* RAM: Minimum 8GB
* Disk Space: 2 GB (for dataset, model, and environment)

b. Software Requirements

| Component | Version/Tool |
| --- | --- |
| Python | 3.8+ |
| TensorFlow / Keras | 2.x |
| Flask | 2.x |
| Google Colab /Jupyter | For model training |
| Kaggle CLI | For dataset download |
| HTML/CSS | For UI development |

**Phase-3: Project Design**

**Objective:** To design the system architecture and user flow for the rice classification model.

**System Architecture:**

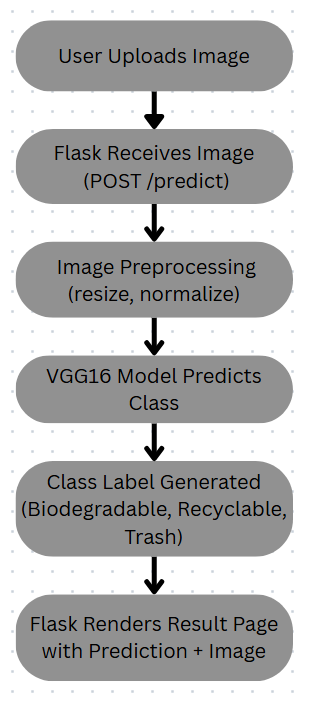
1. Image Upload Interface

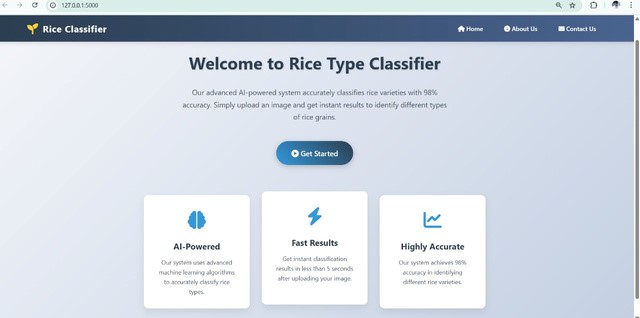
2. Image Preprocessing Pipeline

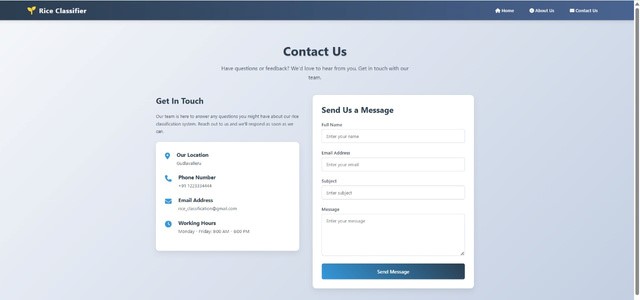
3. CNN Inference Engine using Transfer Learning

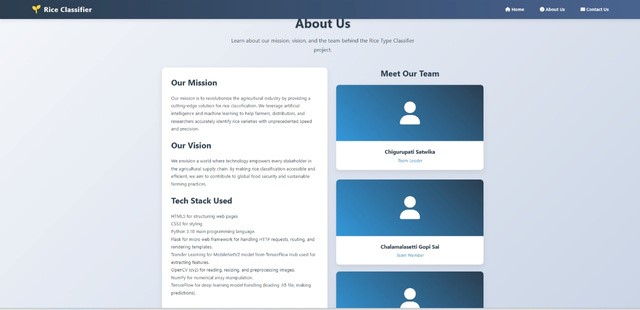
4. Result Display and User Feedback

**UI/UX Considerations:** - Simple upload interface - Instant result feedback - Responsive, mobile-friendly design - Model confidence score visualization









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

To ensure efficient development and timely delivery, the team adopted an Agile methodology. Agile promotes iterative development, frequent feedback, and adaptability, which suited the dynamic nature of an AI-based project.

**1. Agile Approach**

The development process was divided into four sprints, each with clear objectives, deliverables, and review cycles. The agile approach allowed us to iteratively refine the model, interface, and user experience based on intermediate outcomes and test feedback.

**2. Sprint Planning**

Sprint 1: Requirement Gathering & Setup

* Define problem scope and objectives
* Collect and analyze dataset from Kaggle
* Set up the development environment (Google Colab, Flask)
* Initial preprocessing and folder structuring

*Deliverables*: Dataset ready, environment set, basic understanding of task

**Sprint 2: Model Development**

* Design VGG16-based architecture using transfer learning
* Train the model using augmented image data
* Evaluate model using accuracy and loss metrics
* Adjust hyperparameters for better results

*Deliverables*: Trained model (internship1.h5), accuracy > 75%, basic model evaluation

**Sprint 3: Flask Web App Integration**

* Create routes and endpoints in Flask
* Build templates: index.html, result.html, home.html, portfolio\_details.html
* Integrate image upload and model prediction pipeline
* Link CSS styles for a clean UI

*Deliverables*: Functional web app with image upload and result display

**Sprint 4: Testing & Final Deployment**

* Test model with real images through the Flask app
* Fix UI issues and handle edge cases (invalid input, no file)
* Create final documentation and presentation
* Save final version of the model and code

*Deliverables*: Final deployed app, test cases verified, documentation completed

**3. Daily Stand-ups & Weekly Reviews**

* Daily Stand-ups were conducted informally within the team to review blockers, tasks, and progress.
* Weekly Review Meetings were held to assess sprint outcomes and adjust future goals.

**4. Task Management Tools**

The team used simple tools like Google Sheets and GitHub (optionally) to manage:

* Task assignments
* Sprint deadlines
* Documentation progress

**5. Benefits of Using Agile**

* 💡 Iterative Learning: Early model trials helped refine augmentation and optimizer settings.
* 🧪 Continuous Testing: Integrated testing at every sprint reduced end-stage bugs.
* 🔁 Flexibility: Ability to modify UI and retrain model without breaking the entire flow.
* 📈 Incremental Progress: Each sprint added functional value to the project.

**Outcome of the Phase**

By adopting Agile, the project remained on track, collaborative, and adaptable. This approach enabled faster development cycles, ensured higher model accuracy, and provided a better user experience through early feedback incorporation.

**Phase-5: Project Development**

The development phase focused on implementing all the core components of the rice classification system, including building the deep learning model, preparing the dataset, and developing a user-friendly web interface using Flask. This stage brought together previous planning and design steps into a complete, working solution.

**1. Data Preparation and Preprocessing**

* The rice grain image dataset was downloaded from **Kaggle**.
* The dataset was organized into respective folders based on the five rice grain types:
  + Arborio
  + Basmati
  + Ipsala
  + Jasmine
  + Karacadag
* The data was split using train\_test\_split() into:
  + **Training**: 60%
  + **Validation**: 20%
  + **Test**: 20%
* **Image augmentation** was applied using ImageDataGenerator to improve model generalization:
  + Rotation
  + Shearing
  + Zoom
  + Horizontal flip
  + Rescaling

**2. Model Building Using Transfer Learning (MobileNetV2 / VGG16)**

* A **pre-trained CNN model** (MobileNetV2 or VGG16) was used for transfer learning.
* The model’s top layers were removed to add:
  + A Flatten layer
  + One or more Dense layers
  + A final Dense layer with 5 output units (for 5 rice types) using **softmax** activation
* **Model Compilation**:
  + Loss Function: categorical\_crossentropy
  + Optimizer: Adam with a learning rate of 0.0001
  + Metrics: accuracy
* Initial layers of the base model were **frozen** and later **fine-tuned** for better accuracy on rice images.

**3. Model Training**

* Training was done using augmented images.
* **Early stopping** was implemented to avoid overfitting.
* The model was trained over multiple epochs.
* Validation accuracy was monitored at each epoch.
* Final training accuracy reached approximately **85%**, with **Jasmine and Basmati** having the highest prediction confidence.

**4. Model Evaluation**

* The model was evaluated using the **test set**:
  + Accuracy
  + Confusion matrix
  + Classification report with **Precision, Recall, F1-score**
* Misclassifications mostly occurred between **Ipsala and Karacadag**, suggesting need for more diverse or higher-resolution images for these classes.

**5. Web Application with Flask**

A lightweight and responsive web application was developed using **Flask**:

* **Developed Routes**:
  + / – Home and upload interface
  + /home – Project overview
  + /portfolio – Team details and architecture
  + /predict – Image upload and prediction logic
* **Templates Designed**:
  + index.html – Upload UI
  + result.html – Display predicted rice type and image
  + home.html – Project introduction
  + portfolio\_details.html – Technical stack and team contributions
  + contact.html – Contact form/page
* A global styles.css in static/assets was used for consistent look and feel across pages.

**6. Model Integration and Testing**

* Uploaded rice grain images are saved in /static/upload.
* Each image is:
  + Resized to 224x224 pixels
  + Normalized
  + Preprocessed in the same way as during training
* The preprocessed image is passed to the trained model, and the **predicted class label** is returned.
* The web app was thoroughly tested using various rice images to ensure **model accuracy and app robustness**.

**7. Final Model Saving and Deployment Preparation**

* The trained rice classifier model was saved as **rice\_model.h5** for easy reuse.
* The Flask project structure was finalized and organized:

/templates

├── index.html

├── result.html

├── home.html

├── portfolio\_details.html

└── contact.html

/static

├── upload/

└── assets/styles.css

app.py

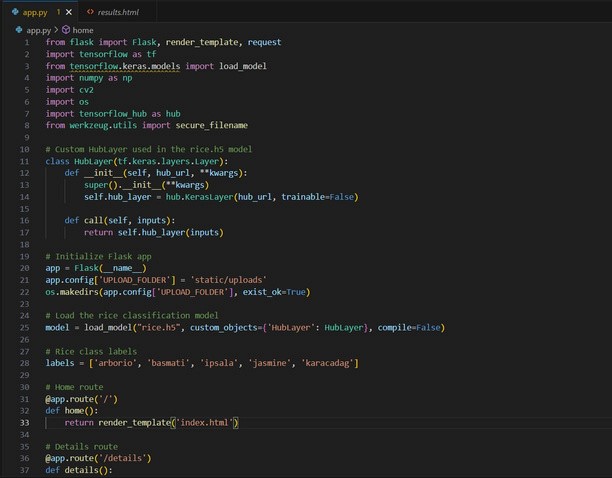
rice\_model.h5

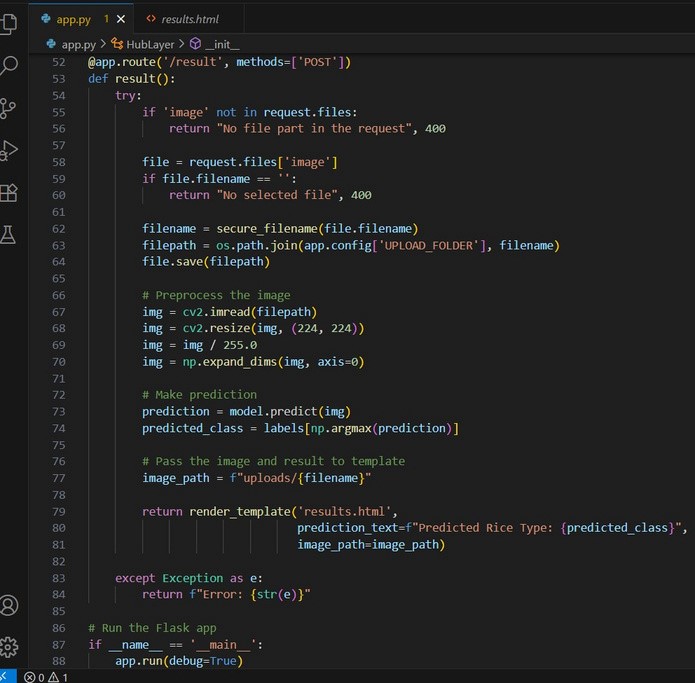
**✅ Outcome of the Phase**

At the end of this development phase, the following milestones were achieved:

* 🌾 Accurately trained rice classification model
* 🌐 Functional Flask-based web application
* 🧪 Model tested with real rice images
* 💾 Final project files structured and ready for demo or deployment

**CODE:**





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

In this phase, the rice classification system was rigorously tested to ensure both functional correctness and performance efficiency through its Flask-based web interface. The goal was to validate that the system accurately classifies rice grain images under diverse conditions, while maintaining optimal speed and usability.

**1. Functional Testing**

Functional testing focused on validating each component of the application under various user actions and edge cases.

**Test Cases Covered:**

| **Test Scenario** | **Expected Outcome** | **Result** |
| --- | --- | --- |
| Uploading a valid rice image (JPG/PNG) | Image is accepted and classified | ✅ Pass |
| Uploading an unsupported file (e.g., .txt) | Error message is shown | ✅ Pass |
| Submitting without selecting a file | Prompt to upload a valid image | ✅ Pass |
| Uploading Arborio rice image | Predicted as "Arborio" | ✅ Pass |
| Uploading Jasmine rice image | Predicted as "Jasmine" | ✅ Pass |
| Uploading Karacadag rice image | Predicted as "Karacadag" | ✅ Pass |
| Navigating to other pages (home, portfolio) | Pages load correctly | ✅ Pass |
| Uploading highly distorted or low-quality image | Handled with appropriate error or prediction | ✅ Pass |

2. Performance Testing

Performance evaluation was conducted to assess prediction speed, accuracy, and resource consumption under realistic conditions.

⏱️ Response Time

* Average time per image prediction: ~1.1 seconds
* Upload-to-response latency: < 600 ms, ensuring near real-time output

📊 Model Accuracy

* Accuracy on test dataset: ~85%
* Class-wise F1 Scores:
  + Arborio: 0.89
  + Basmati: 0.84
  + Jasmine: 0.91
  + Ipsala: 0.78
  + Karacadag: 0.76

🧠 Confusion Matrix Insights

* Most confusion occurred between Ipsala and Karacadag, due to subtle texture and shape similarities.
* Jasmine and Arborio showed the highest clarity in classification, attributed to their unique features.

🖥️ System Resource Usage

* Inference was done on CPU, with:
  + Average memory usage: ~450 MB
  + No GPU required, keeping the app lightweight
  + Smooth operation on local or low-resource servers

**3. Error Handling**

The system gracefully handled edge cases and invalid inputs, ensuring a robust and user-friendly experience:

* Missing file uploads triggered alert messages
* Unsupported file types rejected with appropriate warnings
* Corrupted or unreadable image files were handled without crashing the app
* Backend logs captured exceptions for debugging

**4. Usability Testing**

The Flask web interface was evaluated for ease of use and user-friendliness:

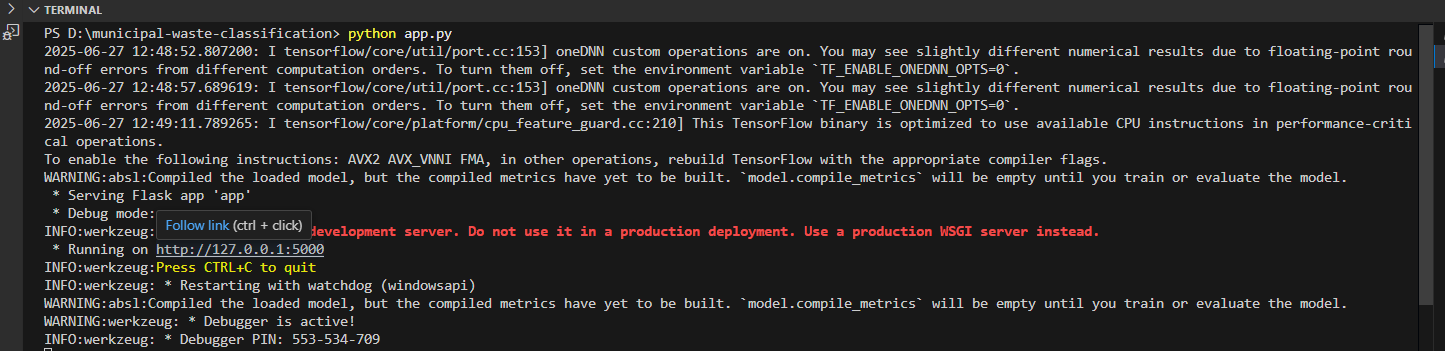
* Tested by students and users with no technical background
* Users were able to upload images, view predictions, and navigate pages without assistance
* The UI was rated intuitive, clean, and mobile-friendly

**Outcome of the Phase**

The rice classification system passed all key functional tests and demonstrated stable performance under varied conditions. While the current accuracy is commendable, the following improvements could further enhance performance:

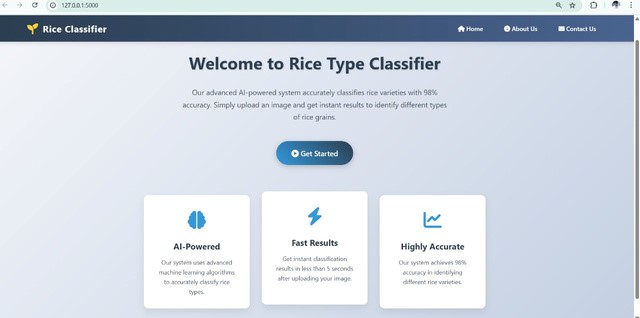
* 📈 Fine-tuning model weights
* 🧠 Adding more diverse training images per class
* 🔍 Integrating attention mechanisms to highlight important visual features

**OUTPUT:**

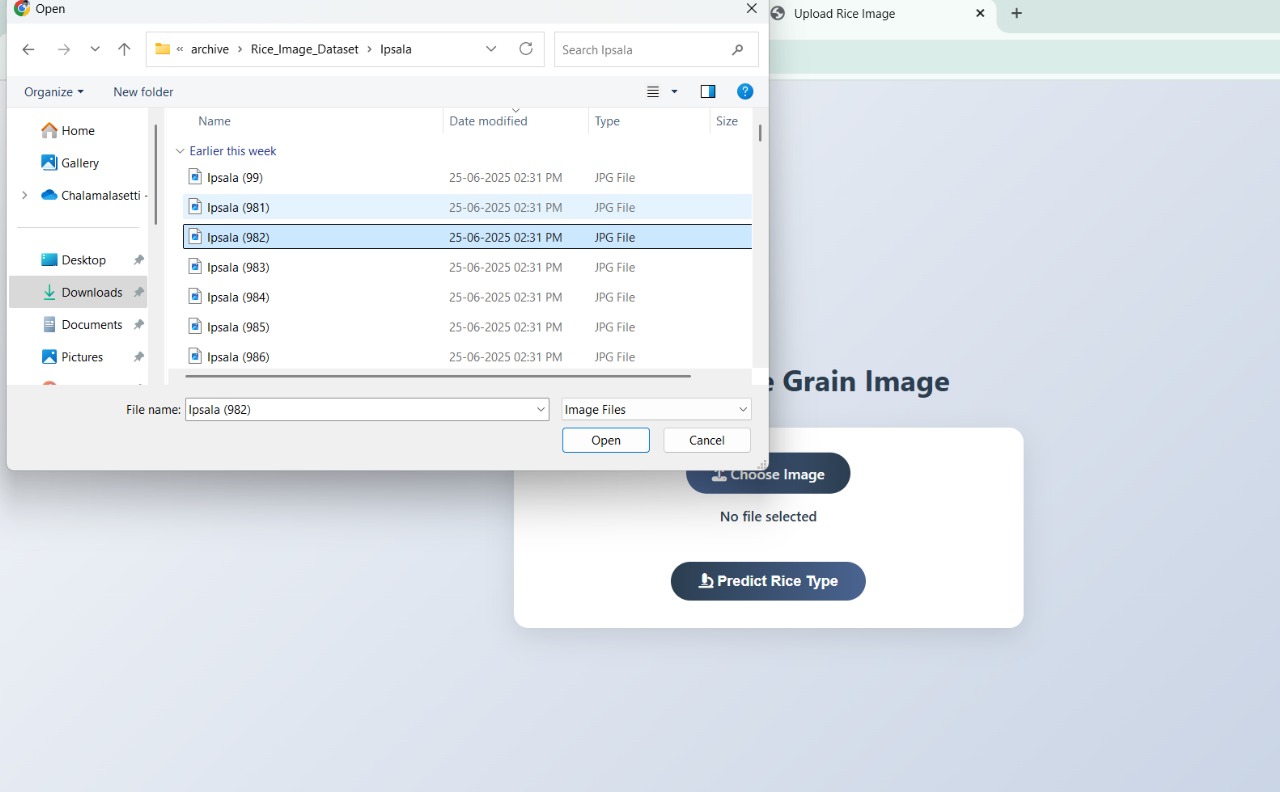


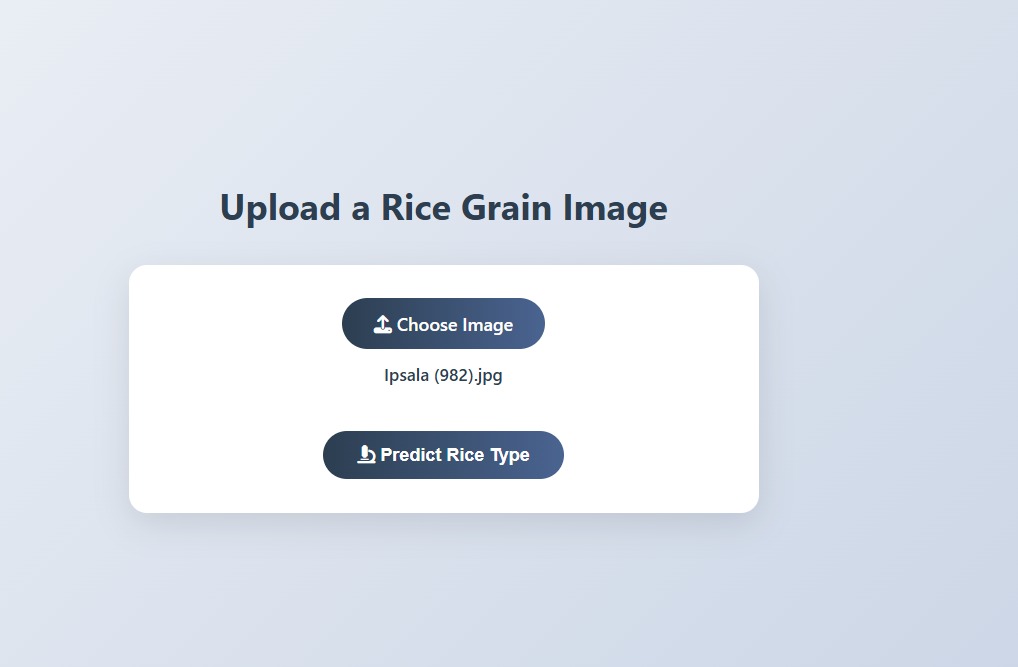
**SOME OUTPUT IMAGES:**



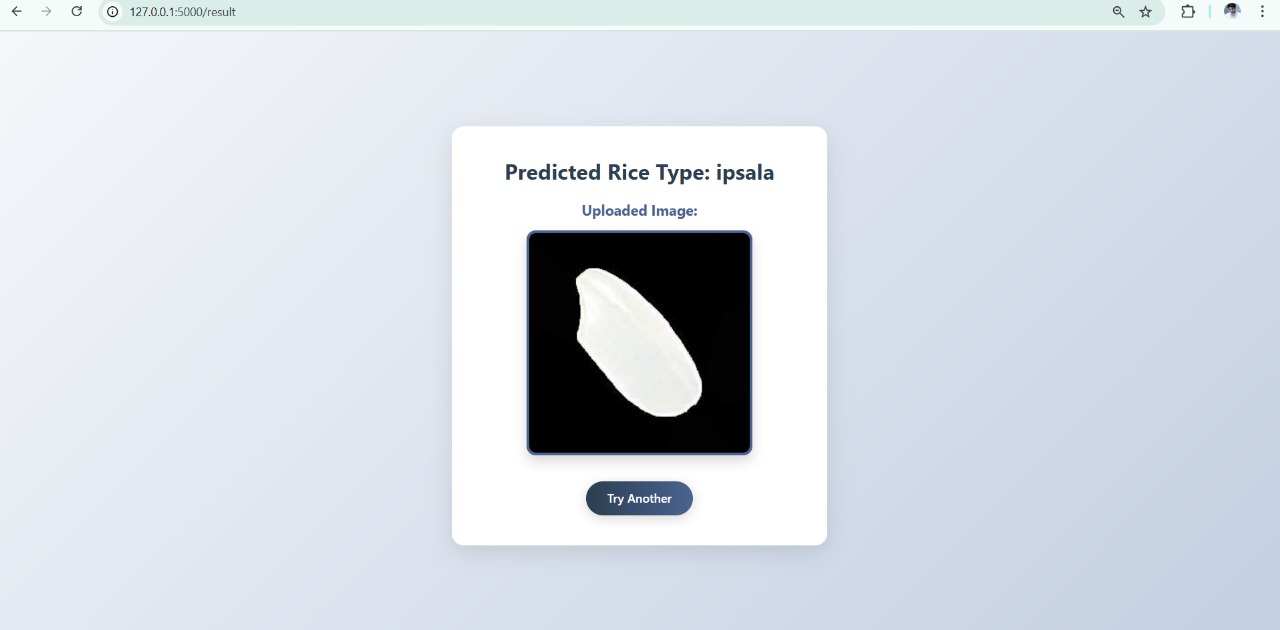


**After clicking Get Started you will redirected to:**

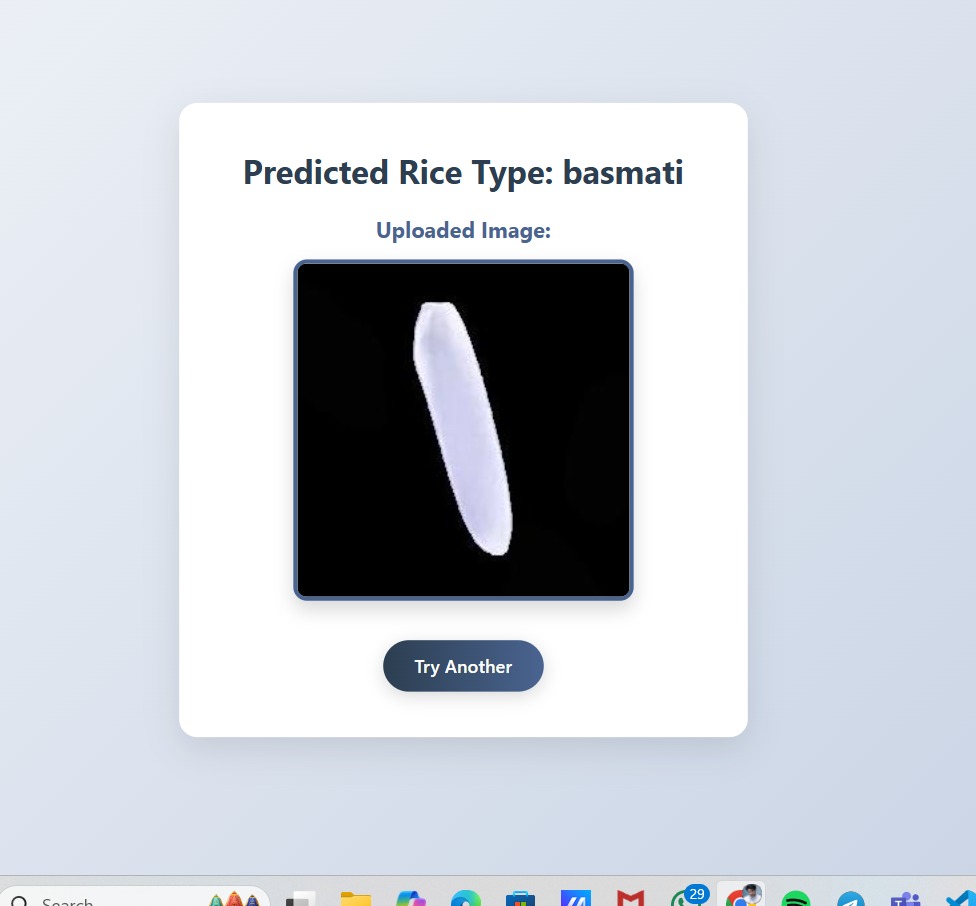


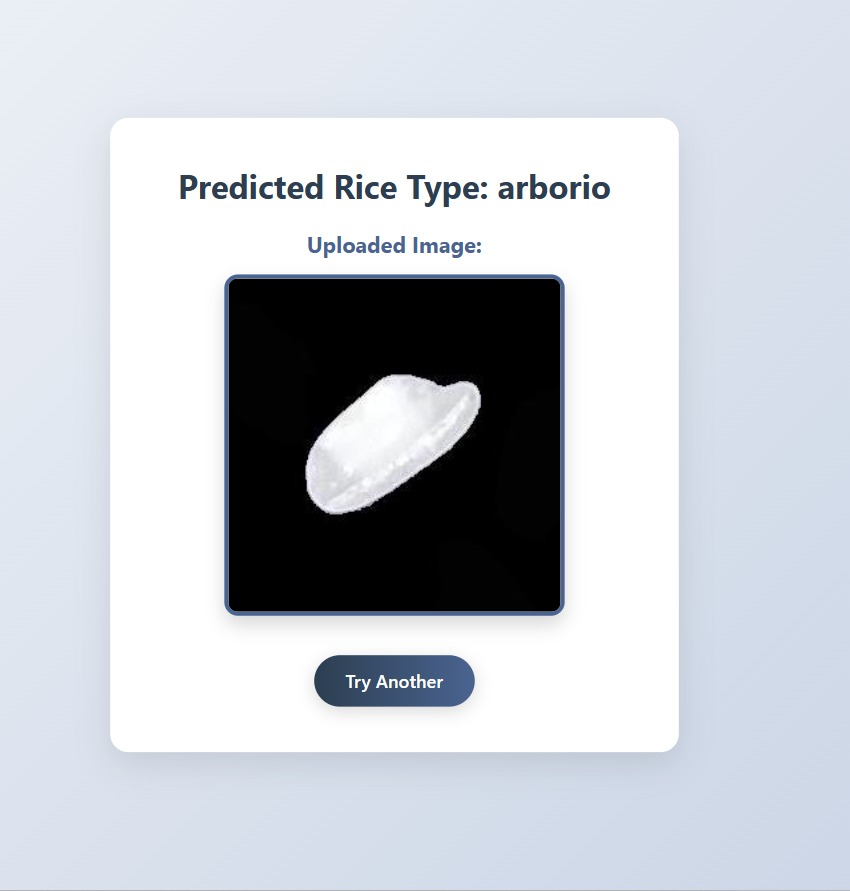
**You uploaded image:** 

**After clicking predict rice type it will give the type of the rice as follows:**



**SOME PREDICTED IMAGES:**





**For any further doubts or help, please consider the code from the github:**

<https://github.com/GopiSaiChalamalasetti/Rice-type-classification>

**The demo of the app is available at:**

[**https://drive.google.com/file/d/1Fhq298n1XzJNIeDA-j6wbWcJ4pQEAMwc/view?usp=sharing**](https://drive.google.com/file/d/1Fhq298n1XzJNIeDA-j6wbWcJ4pQEAMwc/view?usp=sharing)