**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.