INTERNSHIP TASK SUBMISSION REPORT

Student Performance Analyzer & Career Recommendation System

Submitted by:

- **Name:** Abhishek Mahadev Mane
- **Internship Organization:** ProExtern
- **Project Title:** Student Performance Analyzer & Career Recommendation System

Tabel of Contents

ABSTRACT

The Student Performance Analyzer & Career Recommendation System is a web-based solution designed to automate the extraction and analysis of academic records for performance evaluation and career guidance. The system enables users to upload documents (JPEG, PNG, PDF, DOCX) through a secure interface, validates file types and sizes, and employs Optical Character Recognition (OCR) with Tesseract to extract text. OpenCV-based preprocessing techniques—grayscale conversion, noise reduction, and adaptive thresholding—enhance OCR accuracy for varied document qualities. Extracted text is stored in structured JSON format, enabling seamless integration with downstream analytics. Built on Flask, the application ensures scalability and robustness, addressing challenges like manual data entry and inconsistent document formats. While the current implementation focuses on text extraction and preprocessing, the system lays the groundwork for future integration of machine learning models to generate performance insights and career recommendations. By digitizing unstructured academic data, the project streamlines institutional workflows and supports data-driven decision-making for educators and career counselors, ensuring efficiency, security, and adaptability in handling diverse educational records.

1. INTRODUCTION

In academic and professional ecosystems, the analysis of student performance data remains a critical yet labor-intensive process, often hindered by the diversity of document formats—ranging from scanned reports and handwritten notes to digital files. Traditional methods of manual data entry and analysis are not only time-consuming but also prone to errors, creating bottlenecks in generating timely insights for career guidance or academic interventions. The Student Performance Analyzer & Career Recommendation System addresses these challenges by automating the extraction, preprocessing, and structuring of academic records, enabling institutions to transition seamlessly from physical or unstructured data to actionable digital insights.

The system leverages a web interface built on Flask to accept uploads of JPEG, PNG, PDF, and DOCX files, ensuring rigorous validation of file types and sizes to prevent malicious uploads. Utilizing Tesseract OCR and OpenCV, it converts scanned documents into machine-readable text, overcoming variability in font styles, image resolutions, and background noise through preprocessing techniques like grayscale conversion, Gaussian blur, and Otsu's thresholding. Extracted text is stored in JSON format, providing a foundation for future integration with analytics engines or machine learning models to derive performance trends and career recommendations.

By bridging the gap between unstructured data and digital analytics, this system empowers educators and career counselors to make data-driven decisions efficiently. Its scalable architecture and focus on accuracy position it as a vital tool for modern educational institutions aiming to optimize resource allocation and personalize student support.

1.1 Purpose

The Student Performance Analyzer & Career Recommendation System is designed to streamline the process of digitizing and analyzing academic records for educational institutions and career counselors. Its primary purposes are:

1. Automate Text Extraction: Replace manual data entry by extracting text from scanned documents, images, and digital files using Optical Character Recognition (OCR).

2. Enhance Data Usability: Improve the accuracy of extracted text through image preprocessing techniques, such as grayscale conversion, noise reduction, and thresholding, to handle diverse document qualities.
3. Support Data-Driven Decisions: Provide structured, machine-readable text (stored in JSON format) as a foundation for future integration with analytics tools to generate performance insights and career recommendations.
4. Simplify Workflows: Offer a user-friendly web interface for educators and students to upload, validate, and process academic records efficiently.
5. Ensure Security: Validate file types and sizes to prevent malicious uploads and ensure system integrity.

1.2 Scope

The scope of the Student Performance Analyzer & Career Recommendation System encompasses the development of a web-based application that supports file uploads, text extraction, and basic preprocessing functionalities. The system currently accepts files in JPEG, PNG, PDF, and DOCX formats, ensuring rigorous validation of file types and sizes to prevent malicious uploads. It employs Tesseract OCR for text extraction from images and PDFs, while DOCX files are processed using the python-docx library. To improve OCR accuracy, the system incorporates image preprocessing techniques such as grayscale conversion, noise reduction, and thresholding. Extracted text is stored in structured JSON format, facilitating further analysis and integration with advanced analytics tools in the future.

The system's scope is limited to text extraction and preprocessing, excluding advanced functionalities such as performance trend analysis, career recommendation algorithms, and multi-user authentication. It also does not currently support real-time processing or large-scale batch operations. The target users include educators, students, and career counselors who require efficient tools for digitizing and analyzing academic records. Future enhancements may include database integration, expanded file format support, and the incorporation of machine learning models for advanced analytics and personalized recommendations.

2. TECHNOLOGY STACK

Programming Languages:

- **Python**: Core language for backend logic, OCR, and image processing.

Frameworks & Libraries:

- **Flask**: Backend framework for handling HTTP requests, routing, and rendering templates.
- **OpenCV**: Image preprocessing (grayscale conversion, noise reduction, thresholding).
- **Tesseract OCR**: Text extraction from images and PDFs.
- **pdf2image**: Convert PDF pages to images for OCR processing.
- **python-docx**: Extract text from DOCX files.
- **python-magic**: Validate MIME types for secure file uploads.

Tools:

- **Poppler**: PDF rendering engine for pdf2image.
- **Tesseract Engine**: Standalone OCR tool integrated via pytesseract.

Frontend:

- **HTML/CSS**: For user interface design.
- **JavaScript (Basic)**: For interactive file uploads (future scope).

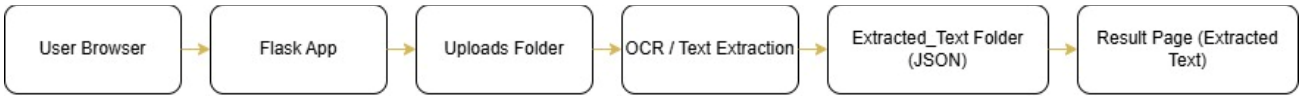
Database:

- **File System**: Stores uploaded files and extracted text in JSON format (temporary solution).

3. SYSTEM ARCHITECTURE

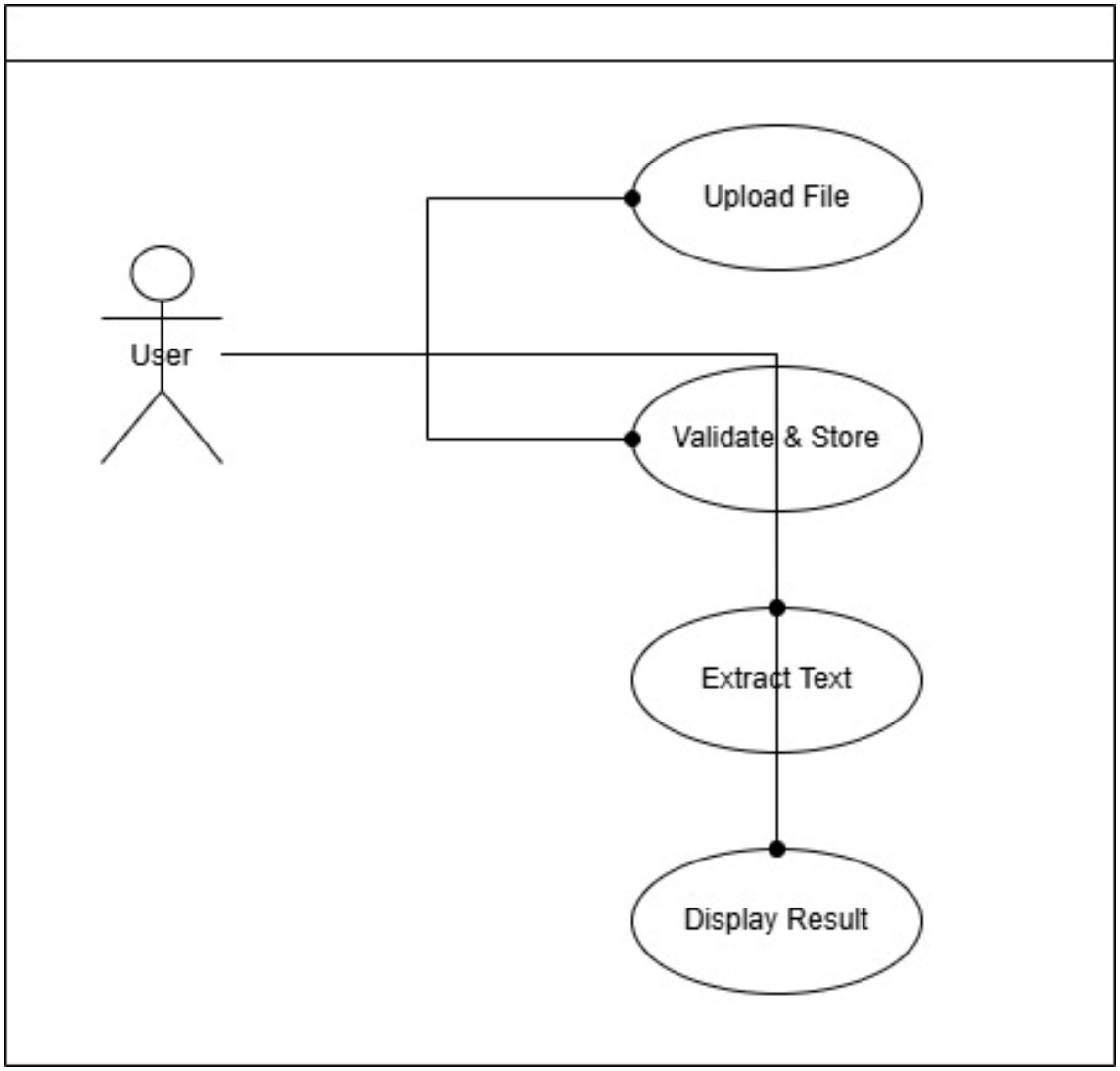
At a high level, the system follows a client-server architecture:

1. **Client** (Web Browser):
    - Displays the *index.html* page where users can upload files.
    - Sends HTTP POST requests containing the uploaded files to the server.
2. **Server** (Flask Application):
    - Receives the file via the / route (POST method).
    - Validates file type and size.
    - Stores the file in the uploads/ folder.
    - Depending on the file type, processes it with the relevant method (image OCR, PDF conversion, or DOCX extraction).
    - Saves the extracted text in JSON format in the extracted_text/ folder.
    - Renders the *result.html* page, displaying the extracted text to the user.
3. **External Libraries**:
    - **OpenCV**: Used for image preprocessing (grayscale, noise reduction, thresholding).
    - **Tesseract**: Performs the actual OCR on the preprocessed image.
    - **pdf2image**: Converts each PDF page into an image before OCR.
    - **docx**: Extracts text directly from DOCX files.



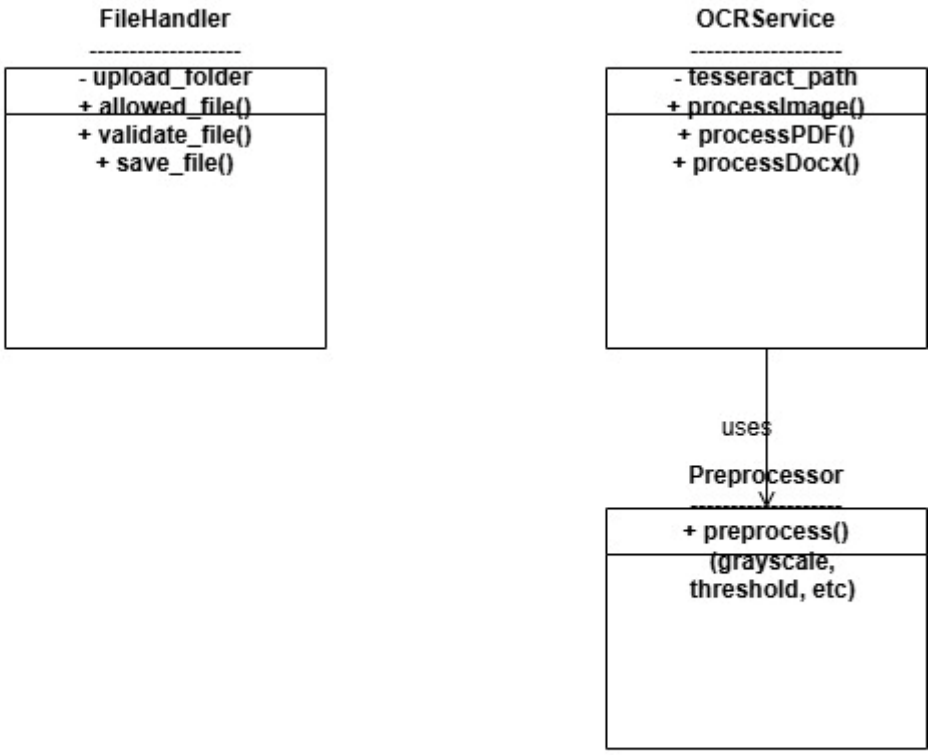
4. OBJECT-ORIENTED DESIGN (OOD) DIAGRAMS

4.1 Use Case Diagram



- Actors: User
- Use Cases: Upload File, Validate & Store File, Extract Text, Display Result

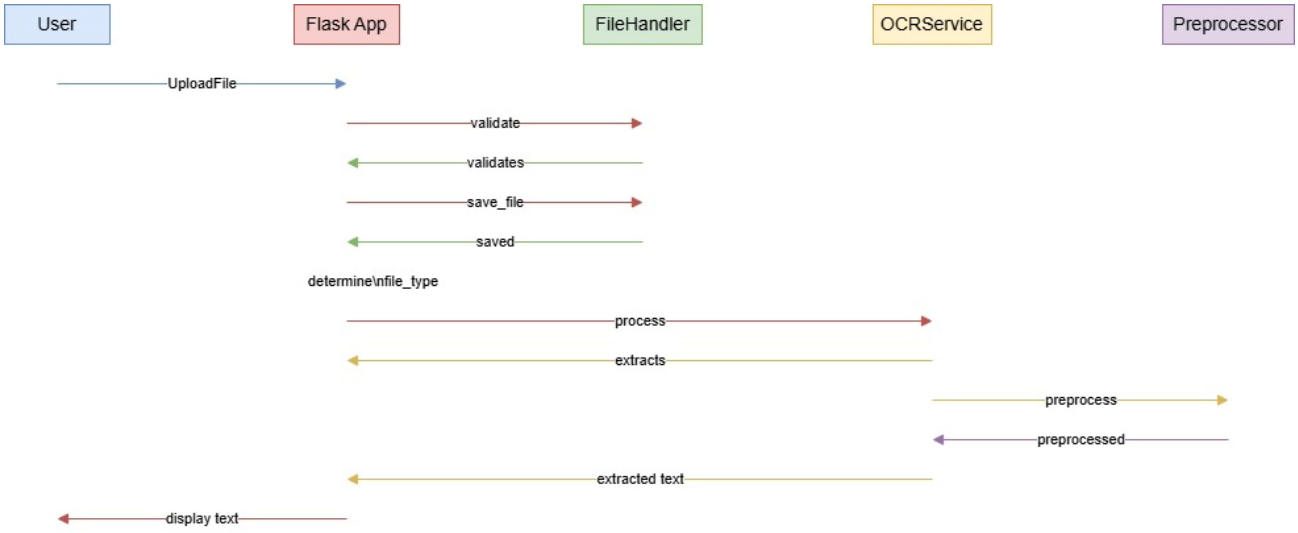
4.2 Class Diagram



- **FileHandler**: Responsible for file-related operations—checking allowed extensions, saving the file, and validating MIME types.
- **OCRService**: Contains methods for extracting text from images, PDFs, and DOCX.
- **Preprocessor**: Handles image preprocessing (grayscale, thresholding, noise removal) to improve OCR accuracy.

4.3 Sequence Diagram

Scenario: User uploads a file to extract text.



4.4 Activity Diagram

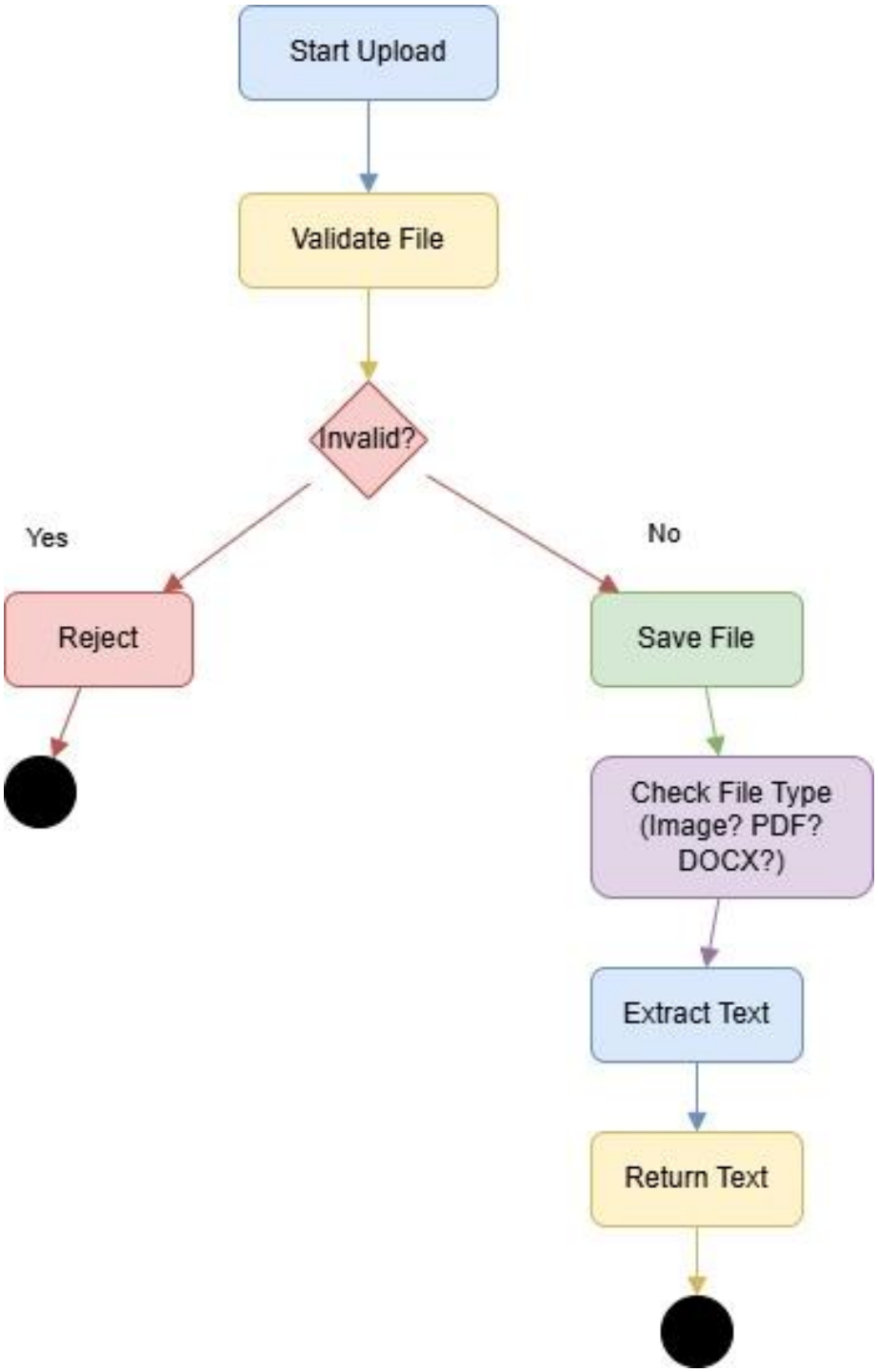


Diagram Overview

1. **Start Node**: A filled circle at the top, indicating the start of the process.
2. **Start Upload**: User initiates the file upload.
3. **Validate File**: Checks if the uploaded file meets the requirements (type, size, etc.).
4. **Decision (Invalid?)**:
    - **Yes** → The file is invalid; **Reject** and end the process.
    - **No** → The file is valid; proceed to **Save File**.
5. **Save File**: Stores the valid file.
6. **Check File Type**: Determines if the file is an image, PDF, or DOCX.
7. **Extract Text**: Uses OCR or parsing to read text from the file.
8. **Return Text**: Delivers the extracted text back to the user or calling function.
9. **End Node**: A double circle indicating the successful completion of the process.

4.5 Database Design (ER Diagram)

Currently, the project does not integrate a database for storing user data or extracted text; text is stored as JSON files in the extracted_text/ folder.

5. FEATURES AND FUNCTIONALITIES

5.1 Features

The Text Extraction Workflow is designed to provide an efficient and structured process for extracting text from various file formats, including images, PDFs, and DOCX documents. Below are the key features of this workflow:

1. File Upload System – Users can upload files in different formats, such as images (JPG, PNG), PDFs, and DOCX documents. The system provides a user-friendly interface for selecting and submitting files.
2. Automated File Validation – The system performs checks on the uploaded files to verify format, size, and integrity. Invalid files (e.g., corrupt, unsupported formats) are automatically rejected with appropriate feedback to the user.
3. Storage Mechanism – Successfully validated files are temporarily or permanently stored in a designated directory for further processing.
4. Format Detection – The workflow includes an intelligent mechanism to detect whether the uploaded file is an image, PDF, or DOCX, ensuring the correct processing technique is applied.
5. Text Extraction Module – The core feature of the workflow involves extracting text using OCR (for images), PDF parsers, or document processors for DOCX files.
6. Text Output and Display – The extracted text is displayed in a structured format, ensuring readability and usability for further processing.
7. Integration Support – The system can be integrated into various applications, including chatbots, document management systems, and AI-powered text analysis tools.

5.2 Functionalities

The Text Extraction Workflow provides robust functionalities to ensure accurate and efficient extraction of text from different file formats. Below are the detailed functionalities of the workflow:

1. Multi-Format Support – The system supports images (JPG, PNG), PDFs, and DOCX files, allowing users to extract text from various document types.
2. Real-Time Validation – As soon as a file is uploaded, it undergoes real-time validation, checking for format, size, and content integrity before processing.
3. Automated Rejection of Invalid Files – If the file is corrupted, unsupported, or contains no readable text, the system automatically rejects it and notifies the user with an error message.
4. Efficient Storage & Retrieval – The system ensures that valid files are stored securely for further processing while also allowing retrieval when necessary.
5. OCR for Image-Based Text Extraction – If an image file is uploaded, Optical Character Recognition (OCR) is used to extract text accurately from scanned or photographed documents.
6. PDF Parsing & DOCX Processing – The system can extract structured text from PDF files and Microsoft Word (DOCX) documents while preserving the formatting.
7. User-Friendly Interface – The extracted text is displayed in an easy-to-read format, ensuring users can copy, download, or use the extracted content for further processing.
8. Scalability & API Integration – The workflow can be integrated into larger applications, including chatbots, AI models, and document processing systems, for automated text extraction and analysis.

6. CHALLENGES AND NEXT STEP

6.1 Challenges

Despite its efficiency, the Text Extraction Workflow faces several challenges that need to be addressed for improved accuracy, performance, and usability. Below are some key challenges:

1. Handling Noisy Images – OCR-based text extraction can struggle with low-resolution, blurred, or distorted images, leading to inaccurate or incomplete text retrieval.
2. Extracting Text from Complex PDFs – Many PDFs contain tables, embedded images, watermarks, and multi-column layouts, making text extraction more challenging. Standard PDF parsers may not always preserve the correct structure.
3. Language and Font Variability – Extracting text from documents with multiple languages, special characters, or decorative fonts can impact OCR accuracy. The system may need advanced language models to handle such cases.
4. Processing Large Files – Extracting text from large PDFs, high-resolution images, or multi-page DOCX files can consume significant memory and processing power, potentially slowing down the workflow.
5. Integration with External APIs – Relying on third-party OCR and text extraction APIs (e.g., Tesseract, Azure OCR) may introduce latency, API limitations, or additional costs depending on the usage volume.
6. Security & Privacy Concerns – Extracting text from sensitive or confidential documents raises data privacy concerns, requiring secure file handling, encryption, and access controls.
7. Error Handling & User Feedback – Providing clear error messages and handling unexpected file formats or corrupt documents efficiently is essential for a better user experience.

6.2 Next Steps

To enhance the Text Extraction Workflow, the following improvements and optimizations can be implemented:

1. Enhancing OCR Accuracy – Integrate deep learning-based OCR models (e.g., Google Vision OCR, Azure Cognitive OCR) to improve text extraction from handwritten, noisy, or complex images.
2. Advanced Layout Parsing – Use AI-based models (e.g., LayoutLM, Detectron2) to better analyze tables, multi-column PDFs, and structured documents, preserving their original formatting.
3. Optimizing File Processing – Implement asynchronous processing and batch handling for large files to improve performance and reduce wait times.
4. Multilingual Support – Expand OCR capabilities to detect and extract text from multiple languages, including non-Latin scripts (e.g., Chinese, Arabic, Hindi), ensuring broader usability.
5. Efficient Storage & Security – Implement secure file storage, encryption, and auto-deletion policies to ensure data privacy and comply with regulations like GDPR.
6. User Feedback & Error Reporting – Develop an intelligent error-handling mechanism that provides users with detailed feedback and suggests possible solutions for rejected files.
7. API & Cloud Integration – Provide RESTful APIs for developers to integrate text extraction into chatbots, AI models, document management systems, and cloud-based applications.
8. Deploying Scalable Infrastructure – Utilize cloud computing (AWS, Azure, GCP) to handle high workloads efficiently, enabling faster processing and improved uptime.