AI PLANET – ASSESSMENT

High-Level Design (HLD)

1. Project Overview

The "PDF Analyzer using AI" project is designed to provide automated insights from PDF documents. This tool leverages Artificial Intelligence (AI) models to perform text analysis, extract key entities, and generate summaries. It simplifies document analysis by transforming complex or lengthy PDF content into structured, concise, and meaningful outputs, thereby improving productivity and decision-making.

2. System Architecture

The system is structured around multiple core components, each performing specific tasks to ensure a seamless flow of data and functionality. The high-level architecture is depicted below:

Components:

- **User Interface (Frontend):** Handles user interactions and facilitates the uploading of PDF files.
- **Backend Application:** Manages incoming requests, processes PDF content, and interacts with AI models.
- **AI Modules:** Includes models for natural language processing (NLP) tasks like text summarization and named entity recognition.
- **PDF Parser Module:** Extracts raw text and structured data from uploaded PDF files.
- **Database/Storage:** Stores processed data, user inputs, and intermediate results, if necessary.
- **External APIs or Libraries:** Uses APIs or libraries to enhance specific functionalities, such as optical character recognition (OCR) or advanced NLP features.

System Flow Diagram: The overall workflow is as follows:

```
[User] ---> [Frontend/UI] ---> [Backend Application] ---> [PDF Parser Module] ---> [AI Models] ---> [Results/Insights]
```

3. Key Components

Frontend

The application interface allows users to interact with the system, upload PDF documents, and view the results. This can be implemented as a web application or a command-line interface (CLI).

Backend

The backend serves as the system's core, where it processes user requests, orchestrates workflows, and returns processed results. It connects the frontend to internal modules and ensures efficient task execution.

AI Modules

AI models are integral to this system:

- **Summarization Model:** Reduces lengthy document content into shorter, meaningful summaries.

- **Entity Recognition Model:** Identifies key entities like names, dates, and locations from the extracted text.
- **OCR (if required):** Converts scanned PDFs into machine-readable text for analysis.

PDF Parsing

This module utilizes libraries such as PyPDF2 or PDFMiner to extract textual data from PDF files. For scanned documents, OCR tools like Tesseract are employed to convert images into text.

Database/Storage

The system may use storage solutions like SQLite or cloud storage to save extracted data and user-provided inputs for further processing or historical analysis.

4. Functional Flow

The system processes data through the following steps:

1. **Input:** Users upload a PDF document using the application interface.
2. **Processing:**
    - The backend forwards the file to the PDF Parser module, which extracts raw text from the document.
    - The extracted text is cleaned and processed through the AI models for summarization, entity extraction, or other specified tasks.
3. **Output:**
    - The backend compiles the processed insights and sends them to the frontend for user presentation.
    - Results are displayed in a structured and user-friendly format.

Low-Level Design (LLD)

1. Detailed Module Design

The low-level design breaks down the major components of the system into specific modules and provides an in-depth explanation of their implementation and interactions.

1.1 User Interface (UI)

- **Functionality:** Allows users to upload PDF files and view the processed results.
- **Implementation Details:**
    - **Framework:** Flask for backend integration or HTML/CSS/JavaScript for a simple frontend.
    - **Endpoints:**
        - `/upload`: Accepts PDF files from the user.
        - `/results`: Displays processed data.
    - **Validation:** Checks for valid file formats (e.g., `.pdf`) before proceeding.

1.2 PDF Parser Module

- **Functionality:** Extracts raw text and metadata from PDF files.
- **Implementation Details:**
    - **Libraries:** PyPDF2 or PDFMiner for text extraction.

- **Scanned PDF Handling:** Utilizes Tesseract OCR for image-to-text conversion if the PDF contains scanned images.
- **Output:** Provides raw text and metadata (e.g., number of pages, author information).
- **Example Code:**

```
from PyPDF2 import PdfReader

def extract_text(file_path):
    reader = PdfReader(file_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text
```

1.3 AI Models

- **Summarization Model:**
  - **Algorithm:** Uses a pre-trained transformer model (e.g., Hugging Face's T5 or BERT) to summarize text.
  - **Input:** Cleaned raw text from the parser.
  - **Output:** A concise summary of the document.
  - **Example:**

```
from transformers import pipeline

summarizer = pipeline("summarization")
def generate_summary(text):
    return summarizer(text, max_length=150, min_length=30, do_sample=False)
```

- **Named Entity Recognition (NER):**
  - **Algorithm:** Pre-trained SpaCy model for entity extraction.
  - **Entities Extracted:** Person, Organization, Date, etc.
  - **Example:**

```
import spacy

nlp = spacy.load("en_core_web_sm")
def extract_entities(text):
    doc = nlp(text)
    return [(ent.text, ent.label_) for ent in doc.ents]
```

- **OCR Module (if required):**
  - **Library:** Tesseract.
  - **Usage:** Converts scanned PDF images to text.

1.4 Backend Application

- **Functionality:** Orchestrates workflows between modules, manages requests, and provides responses.
- **Implementation Details:**
  - **Framework:** Flask or Django.
  - **Endpoints:**
    - `/upload`: Accepts file uploads.
    - `/process`: Coordinates parsing, AI analysis, and results generation.
    - `/results`: Returns processed insights to the user.
  - **Middleware:** Validates user inputs and handles exceptions.

1.5 Database (Optional)

- **Functionality:** Stores processed insights, user information, or historical data for reuse.
- **Implementation Details:**
  - **Database:** SQLite or PostgreSQL.
  - **Schema:**
    - **User:** Stores user IDs, uploaded file details.
    - **Results:** Links uploaded files to extracted data and insights.

2. Workflow Implementation

The detailed workflow ensures seamless functionality of all modules.

2.1 File Upload

- **Input:** PDF file.
- **Validation:** Ensures the file is a valid `.pdf` format.
- **Output:** Accepted file is saved temporarily on the server.

2.2 Text Extraction

- If the PDF contains selectable text:
  - Use PyPDF2 to extract raw text.
- If the PDF contains scanned images:
  - Convert images to text using Tesseract OCR.

2.3 AI Processing

- Summarization: Applies the summarization model to extracted text.
- Entity Recognition: Processes the text to identify key entities using NER models.
- Results Compilation: Merges insights into a user-readable format.

2.4 Result Presentation

- **Output Format:**
  - Summary: A concise overview of the document.
  - Entities: List of extracted entities with their types.
- **Frontend:** Displays the processed data in a structured format.

3. Code Snippets

Below are a few implementation examples:

Summarization Endpoint

```
from flask import Flask, request, jsonify
from transformers import pipeline

app = Flask(__name__)
summarizer = pipeline("summarization")
```

```
@app.route('/summarize', methods=['POST'])
def summarize():
    data = request.get_json()
    text = data.get("text", "")
    summary = summarizer(text, max_length=150, min_length=30, do_sample=False)
    return jsonify(summary)
```

Entity Recognition Endpoint

```
from flask import Flask, request, jsonify
import spacy

app = Flask(__name__)
nlp = load("en_core_web_sm")

@app.route('/entities', methods=['POST'])
def entities():
    data = request.get_json()
    text = spacy.data.get("text", "")
    doc = nlp(text)
    entities = [{"text": ent.text, "label": ent.label_} for ent in doc.ents]
    return jsonify(entities)
```

4. Error Handling

The application is designed to gracefully handle errors:

- **Invalid File:** Returns a clear message if the uploaded file is not a PDF.
- **AI Model Errors:** Logs errors if models fail to process text or generate outputs.
- **Timeouts:** Implements a timeout for long-running tasks to prevent server crashes.