**RESUME INSIGHTS ENGINE**

**TOPIC**: Comparative analysis on OpenAI Data Extraction in Resume document using OpenAI Keys vs Local LLM Implementation without Internet connectivity.

**INTRODUCTION:**

In the rapidly advancing landscape of technology, generative AI has assumed a central role. Possibilities such as text and image generation have become commonplace. However, the question arises: can we effectively extract specific text from a document using this technology? utilizing OpenAI Keys for remote extraction and implementing Local LLM (Large Language Model) for extraction without internet connectivity. Focusing on critical information such as names, email addresses, and phone numbers, this investigation aims to evaluate the performance, reliability, and feasibility of these methodologies in the context of a resume document. So on this specific note lets learn on the extraction methods, challenges and its solutions.

**TASK 1 & 3: OPENAI BASED DATA EXTRACTION IN A RESUME DOCUMENT**

**How It Works:** When you upload a resume, the website reads the text from the PDF. It sends this text to OpenAI, which is like a smart tool that understands and generates text. OpenAI figures out the name, email, and phone number from the resume text.

**Packages used:**

1. openai
2. pandas
3. PyPDF2
4. flask

A black screen with white text

Description automatically generated**Openai keys:** OpenAI keys are used as a means of authentication and authorization when interacting with OpenAI's API (Application Programming Interface). When you make requests to OpenAI's API for using services like GPT (Generative Pre-trained Transformer) models, you need to include your OpenAI key in the request headers but it’s important to note that this key is highly confidential and must be not shared with anyone due to security purposes. Always check your openai key usage from the official website.

**Methodology:**

1. Configure OpenAI API Key:Set up the OpenAI API key for authentication. The API key is used to interact with OpenAI's services.
2. Define Data Extraction Function:Create a function (extract\_data) that takes text input, forms a prompt using a predefined resume template, and utilizes the OpenAI ChatCompletion API to generate a response. Parse the response content and attempt to convert it into a Pandas Data Frame containing extracted measures and values.

**A computer screen shot of text

Description automatically generated**

1. Resume Template: Define a function (resume) that provides a template for the prompt. This template instructs the OpenAI model to extract specific information from a given resume.

A screen shot of a computer

Description automatically generated

1. Flask Web Application: Initialize a Flask web application. Create routes for the main page (index) and the endpoint for processing uploaded resumes (get\_insights).
2. File Upload and Text Extraction: The application handles file uploads, saves the uploaded file, extracts text from the PDF using the PyPDF module, and then calls the extract\_information function.
3. Error Handling and validation
4. Render Results: Use the extracted data to render the results on the main page (index.html).
5. Run the Application
6. Output given in a Json Format

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Challenges:**

1. Resource Limitations: The dependency on OpenAI's API and the consumption of API credits can be a challenge. If API credits are depleted or unavailable, it may hinder the application's functionality.
2. Authentication Dependency: The reliance on OpenAI keys for authentication poses a challenge, especially when users may not have immediate access to their own API keys.

**Limitations:**

1. Single Data Source: The application currently focuses on extracting information exclusively from PDF files. Expanding to other file formats or web-based sources might enhance its versatility.
2. Dependence on PDF Structure: The application relies on the assumption that resumes are in a readable format within PDF files. Variations in PDF structure might lead to inaccuracies in text extraction.

**Future Works:**

1. Multi-Format Support: Enhance the application to support multiple document formats (e.g., Word, plain text) to broaden its compatibility and usability.
2. Keyword Matching and Analysis: Implement advanced keyword matching algorithms to identify relevant terms or phrases within resumes. This could improve the application's ability to extract contextually relevant information.

**TASK 2 & 4: OPENAI BASED DATA EXTRACTION IN A RESUME DOCUMENT**

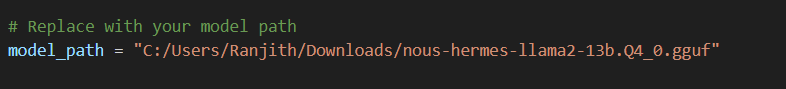
**How It Works:**  when you upload a resume pdf in the flask API it will be sent to our Local LLM to process and once it runs the extracted information will be displayed on the new webpage. This Process usually takes time to extract the information.

**Packages used:**

1. Langchain
2. Llama-cpp-python
3. PyPDF
4. Flask
5. Openai
6. GPT4ALL

**Methodology:**

1. Installation of GPT4ALL model from model explorer and mentioning the path of the downloaded file in the code. I installed a model called “mistral-7b-openorca.Q4\_0.gguf”



A screenshot of a computer

Description automatically generated

1. LlamaCpp Language Model Configuration: The script configures the LlamaCpp language model, specifying the model path, the number of GPU layers, batch size, and setting up callback handlers for streaming standard output

A screenshot of a computer

Description automatically generated

1. Prompt Template Setup: A prompt template is defined using the Prompt Template class, incorporating placeholders for text and variables like name, phone, and email.

A blue and black background

Description automatically generated with medium confidence

1. Information Extraction Function: The extract\_information function is implemented to process text in chunks, leveraging the LlamaCpp language model to extract information such as name, phone number, and email using regular expressions.
2. Flask Web Application Initialization: A Flask web application is initialized, and routes are defined for the main page (index) and the endpoint for processing uploaded resumes (get\_insights).
3. File Upload and Text Extraction: The application handles file uploads, saves the uploaded file, extracts text from the PDF using the PyPDF module, and then calls the extract\_information function.
4. A screenshot of a computer

   Description automatically generatedA close-up of a business card

   Description automatically generatedRun the application

**Challenges:**

1. Large Model Downloads: Downloading large language models from platforms like GPT4ALL can be time-consuming, especially due to the high file sizes. Slow or unreliable internet connections may further exacerbate this challenge.
2. Local Execution Time: Running the local LlamaCpp language model on a machine, especially with large models, may require substantial computational resources and time. The performance may be hindered by the local hardware's processing capabilities.
3. Unnecessary Outputs: The local execution might generate verbose outputs in the terminal, potentially causing information overload or making it challenging to discern relevant information from the log.
4. Exceeded Tokens Handling: When dealing with large text inputs, the model's maximum token limit could be exceeded, necessitating additional code to split the text into manageable chunks. Managing these chunks and ensuring a coherent response becomes a challenge.

**Limitations:**

1. Local Resource Requirements: The local execution of large language models(Some) may require substantial resources, including GPU capabilities and memory. This limits the applicability of the solution to machines with sufficient hardware resources.
2. Processing Time: The time taken to process large volumes of text locally can be a limiting factor. Users may experience delays in obtaining results, impacting the real-time usability of the application.

**Future Work:**

1. Optimized Model Distribution: Explore strategies for optimizing the distribution and download of large language models. This could involve implementing efficient download protocols, differential updates, or utilizing model compression techniques.
2. Local Execution Optimization: Investigate methods to enhance the local execution speed, such as parallel processing, model optimizations, or leveraging hardware accelerators to improve overall performance.
3. User-Friendly Interfaces: Develop user-friendly interfaces that abstract away unnecessary outputs and provide a cleaner, more comprehensible view of the extraction results. This could include graphical interfaces or logging improvements.
4. Handling Exceeded Tokens Automatically: Enhance the application to automatically handle cases where the input text exceeds the model's maximum token limit. Implement mechanisms to intelligently split and process large texts seamlessly.

**Code is available in the GitHub : https://github.com/Ranjith-Kumar007/Resume-Insights-Engine**

**References:**

1. <https://cookbook.openai.com/>
2. <https://python.langchain.com/docs/use_cases/extraction>
3. <https://python.langchain.com/docs/integrations/llms/gpt4all>
4. <https://huggingface.co/docs/transformers/main/model_doc/llama2>