PROJECT REPORT -



AI and MLOPs

**Cohort 3**

Automated Video Creation from Resumes Using LLM

Group-9

**Team Members’ Names**

* 2304306 Chamandeep Jindal
* 2303386 Ravinder Kumar Patel
* 2302901 Mohammad Abdul Khadar Zeelani
* 2303594 Pinaki Basu
* 2303311 Jyoti Radhu
* 2304044 Sukumar
* 2304442 Suvojit Kumar Chanda
* 2304007 Pranav Vetkar
* 2304008 Soumik Saha
* 2305043 Anil Sharma
* 2303292 Aman Arora
* 2304491 Andrew Dhas



## Abstract:

The rapid growth in job applications has created a need for innovative and efficient solutions to streamline the recruitment process. Traditional resume screening methods are often time-consuming and prone to overlooking critical aspects of a candidate's profile. This project introduces an AI-driven system that automates the transformation of resumes into dynamic, one-minute video summaries. By leveraging cutting-edge technologies, including Prompt Engineering, state-of-the-art large language models (LLMs) like GPT-4, text-to-speech generation, and audio-to-video synthesis, this solution provides recruiters with concise, engaging insights into candidates’ qualifications and experiences.

The system aims to revolutionize resume screening by improving efficiency, accuracy, and interactivity. It highlights key qualifications while eliminating manual effort, offering HR professionals a powerful tool to evaluate candidates effectively. This innovative approach not only enhances the recruitment experience but also demonstrates the transformative potential of generative AI in automating complex workflows.

Also, research shows that different resume parsers have around 85-90% accuracy in identifying relevant sections in standard resumes. However, no benchmark exists for automatically creating video summaries from resumes.

## Acknowledgments:

We extend our sincere gratitude to the following individuals at IISC for their invaluable support and guidance throughout the course of this project:

|  |  |
| --- | --- |
| **Person (Title and Name)** | **Title** |
| **Prof. Deepak Subramani** | Expert Faculty |
| **Prof. Sasikumaar Ganesan** | Expert Faculty |
| **Rishav Singh,Yograj** | Project Mentor/s |

**Table of Contents**

1. [Introduction 3](#_TOC_250017)
   1. [Background and context of the project 3](#_TOC_250016)
   2. [Problem statement and objectives 3](#_TOC_250015)
   3. [Motivation behind the project 3](#_TOC_250014)
2. [Methodology: 4](#_TOC_250013)
3. [Project Execution 5](#_TOC_250012)
   1. [Text Extraction from CVs 5](#_TOC_250011)
   2. LangChain & [Prompt Engineering 7](#_TOC_250010)
   3. [Text-2-speech conversion 8](#_TOC_250009)
   4. [Audio-2-Video conversion with Lip Sync 9](#_TOC_250008)
   5. [MLOps & UI 10](#_TOC_250007)
4. [Overview of Model Output on UI 13](#_TOC_250004)
5. [Challenges 14](#_TOC_250003)
6. [Conclusion 15](#_TOC_250002)
7. [Future Work 16](#_TOC_250001)
8. [References 18](#_TOC_250000)

# Introduction:

### Background and context of the project.

With an increasing number of job applicants, companies require an efficient, accurate, and engaging way to screen resumes. By automating the transformation of resume text into 1 minute video summaries, recruiters can gain a concise, dynamic overview of each candidate, reducing the chances of overlooking key qualifications.

### Problem statement and objectives.

Traditional resume screening is often time-consuming and may fail to capture critical aspects of a candidate’s skills and experience. This project aims to automate the creation of engaging video summaries of resumes using GenAI, which will streamline the recruitment process and highlight candidates' profiles effectively.

### Motivation behind the project.

We chose this project as it leverages advancements in AI to address inefficiencies in resume screening. It combines the techniques like Prompt Engineering with SOTA pretrained LLMs like GPT-4 o, Text-2-speech generation & Audio to video generation techniques to offer an innovative solution for HR, potentially transforming candidate screening into an automated, interactive experience.

# Methodology:

Our solution is essentially a 5-step process-

Step 1: Extracting CV Text

Extract text from CVs provided in PDF or DOCX format using advanced text extraction tools.

Ensure that the extracted content captures all relevant details, such as skills, experience, and education.

Step 2: Generating Textual Summaries

Use prompt engineering techniques combined with LangChain and a state-of-the-art language model (GPT-4o) to summarize the extracted CV text into a concise, readable format.

Focus on presenting the candidate's key qualifications and achievements effectively.

Step 3: Creating Audio Summaries

Leverage a Text-to-Speech (TTS) model, such as ElevenLabs, to convert the textual summaries into engaging and natural-sounding audio.

Ensure the audio retains clarity, accuracy, and a professional tone.

Step 4: Generating Video Summaries

Use an open-source GAN model like Wav2Lip to overlay the audio on an AI-generated avatar.

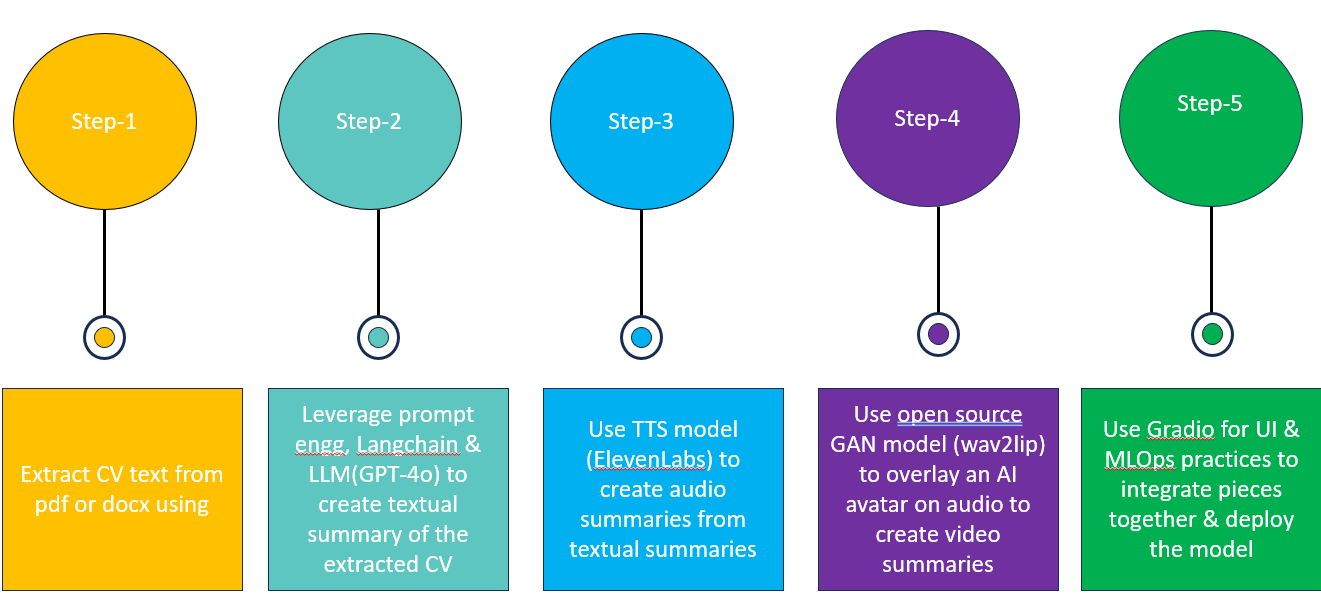
Produce dynamic, visually appealing video summaries that represent the candidate’s profile.

Step 5: Integration, Deployment & Monitoring

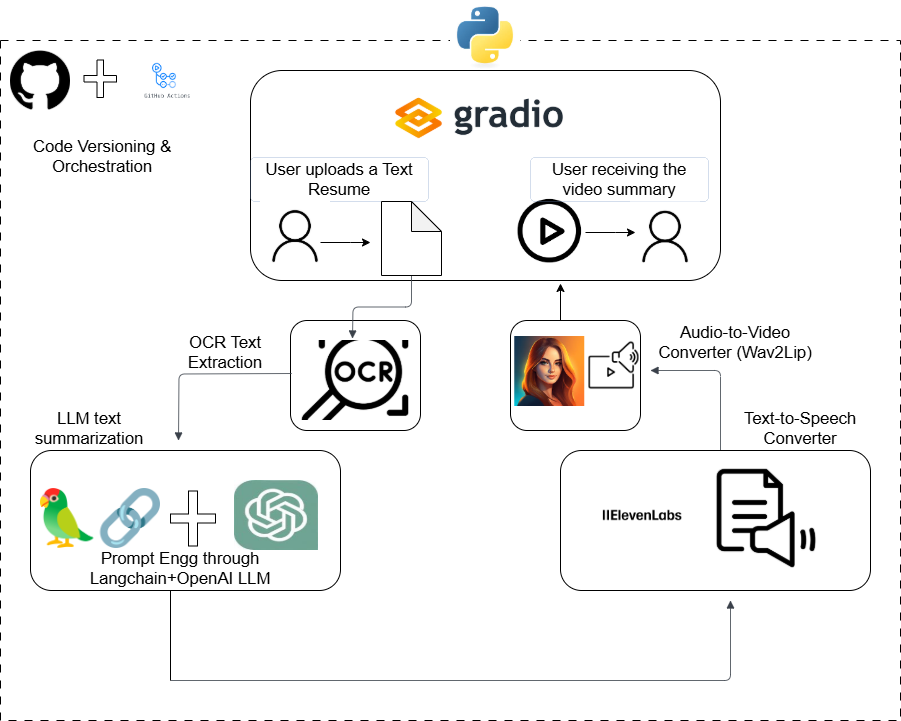
Use Gradio for building an intuitive user interface (UI) to enable seamless interaction with the system.

Apply MLOps practices to integrate the components and deploy the solution efficiently, ensuring scalability and robustness.

Integrate ClearML to monitor app KPIs



The high level solution design is shown the below-



**Fig 1: High Level Solution Architecture**

# Project Execution Details:

### Data Extraction:

### 

### 

The data extraction class extracts data from 2 types of CV formats- PDF & Docx in the string format.

**Fig 4: Data Ingestion Pipeline Technical Architecture**

### Prompt Engineering:

### 

The prompt engineering process defined for the text summarization task is essentially split into 2 steps-

1. Extracting Json output from the CV highlighting important fields required in the summary.
2. Generating a profile summary using the extracted Json output from the previous step

For both of these prompts, system & human messages are defined. The ‘system’ message basically defines the role & the exact task for the LLM. It also specifically asks the LLM to not hallucinate while generating the response.

The ‘human’ message passes the dynamic {input} to the LLM.

### LangChain & LLM:

### 

### LangChain framework is being used to interact with the LLMs. It’s a very user friendly framework & has a lot of pre-defined classes & modules to work specifically with LLMs, for example all the prompt templates are getting defined through LangChain. It also provides integration with OpenAI & so the llm object can also be defined through LangChain itself.

### Lcel, which LangChain common expression language is the latest approach to define chains-

### **chain2 = self.get\_prompt() | llm | StrOutputParser()**

### Here the prompt, the llm & the output parser are chained together to create a chain which can then be invoked for a query like this-

### **result = chain2.invoke({'input': self.resume\_data})**

### The project uses GPT-4o-mini as the LLM as GPT-4o mini is one of the most cost-efficient high performing small models in the market. GPT-4o mini scores 82% on MMLU and currently outperforms GPT-41 on chat preferences in LMSYS leaderboard.

### This is how the llm object is defined-

### **llm = ChatOpenAI(model\_name=self.llm\_name, temperature=0, api\_key=self.openai\_api\_key)**

### Since we are looking for consistent factual summaries, the temperature parameter is 0 to make it as less creative as possible.

### Text-to-Speech:



The project uses ‘ElevenLabs’ client to convert textual summary generated in previous step to audio summary. The voice settings such as voice id, stability, style etc. can be defined as arguments to the ‘text\_to\_speech.convert()’ class.

An open source library called ‘Pydub’ is also used to convert ‘mp3’ audio format into ‘wav’ format as our audio to AI avatar video model requires wav format input.

### Audio-2-video:

### 

The Audio-to-video component uses a pretrained Wav2Lip GAN checkpoint for inference. It expects two inputs-

1. Audio summary in wav format
2. Video (which is our AI-avatar in this case) in mp4 format

Using the inference.py, Wav2Lip overlays the Video with lip syncing on the audio.

The audio summary which is the input for this step comes from the TTS client ‘ElevenLabs’ in the previous step.

The video input is again created using these 2 steps-

1. Generate an AI Avatar using a Text-2-image pretrained model like MidJourney.



1. Convert the static image to a 3d video using tools like ImmersityAI



* 1. MLOps & UI

### 

### 

**Components**

* *Used Git & Github for code versioning since we were working as a team & collaborating on the project through Github by pushing our versions of the code pieces to the project repo. The above image shows*

Git & Github

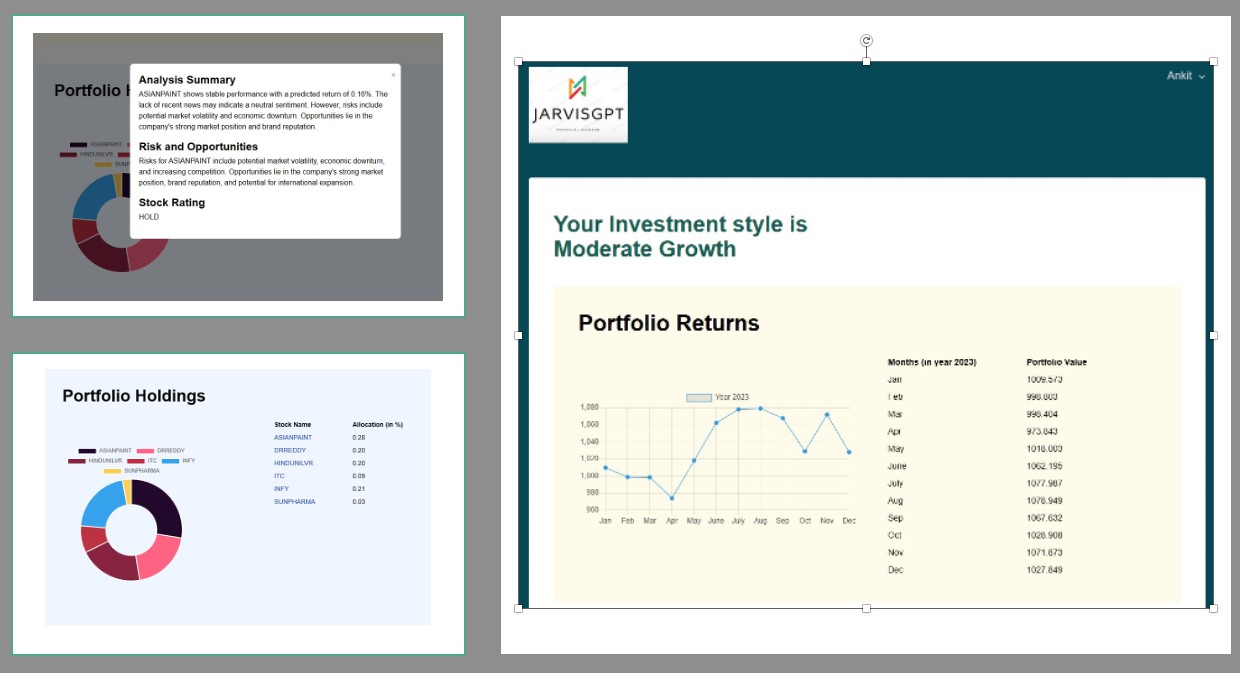
**Integration with Other Sections**

|  |  |
| --- | --- |
| Data pipeline | * Crawling news articles for a specified list of stocks (Nifty-40). * Fetching the closing price for these stocks from the National Stock Exchange (NSE). * Persisting the acquired data to Amazon S3. |
| API Binaries (versioned) | * Efficient Frontier: Utilized for constructing a user's portfolio based on a statistical formula. * LSTM: Applied for stock price forecasting. * gpt-3.5-turbo: Employed for news summarization and   generating stock analysis recommendations using the OpenAI API. |
| Continuous Integration | * A Docker image is built using QEMU for diverse architectures. * The Docker image is pushed to DockerHub (currently not pushing to ECR). * For deployment, a manual trigger is used, and self-runners are not employed. The image is pulled from DockerHub and executed as a container. |
| Continuous Deployment | * UI: Automatically deployed on Vercel post commit on master branch * Data Pipeline: Deployed on AWS. * API (container): Deployed on EC2. * Monitoring (container): Deployed on EC2. |
| Scalability | * Handling the number of users as addressed in section 3. * Managing the number of stocks through the current design, with partial handling. * For increased API calls, an ideal solution involves deploying on   EKS after partitioning the tickers list and using Dask for worker invocation. |
| Monitoring | * Data pipeline: Cloudwatch * API (container): Prometheus and Grafana |

|  |  |
| --- | --- |
|  | Number of user requests/min   1. For status code: 200 2. For status code: 5xx 3. For status code: 4xx  * API response time   1. API name   2. Response time * Number of OpenAI API:   1. Tokens count   2. (Input, Output) pair * Users   Metrics: Number of Users   * 1. Risk Bucket   2. Money Invested   3. Age |
| Resiliency | * OpenAI API calls have a retry count of 2 for API failures. * AWS calls have a retry count of 2. * The microservices-based architecture ensures independent module operation, containing failures within the specific   module. |

# Overview of Model Output on UI:

This section on the user interface provides a clear snapshot of the user's investment landscape. It highlights the following aspects:



#### Fig 8: Model Output on UI

1. **Investment Style:** The user's risk profile is prominently featured, categorized as conservative, moderate growth, growth, or aggressive growth. This categorization offers a quick glance at the user's approach to risk.
2. **Portfolio Value:** An overview of the portfolio's value over the past 12 months is depicted graphically. It illustrates the growth or contraction of the initial investment (e.g., 1000 INR) based on stock prices and units held each month.
3. **Portfolio Composition:** The user's portfolio makeup, aligned with their investment style, is visually represented through a donut chart. Each segment represents a specific stock, with its size indicating the stock's weight in the overall portfolio. This visual simplifies understanding diversification and fund allocation.
4. **News Summary Integration:** Interactivity is enhanced through the donut chart. Hovering over each stock triggers a pop-up displaying a concise news summary related to that stock. This integration with LangChain (LLM model) ensures users stay informed about developments affecting their investments.

# Challenges:

There were challenges at different stages of the development lifecycle-

1. The prompt engineering process was a pivotal challenge to begin with as multiple versions of prompts were tried and improved upon. For example, this was one of our initial prompts-

**final\_prompt = ChatPromptTemplate.from\_messages(**

**[**

**("system", "You are an HR recruitment assistant who's capable of analyzing candidate's profile by going through their CV"),**

**few\_shot\_cv\_summary\_prompt,**

**("human", '''summarize the candidate's profile mentioned here- {input}, in approximately 200 words highlighting these things in a json format-**

**1) Candidate details:**

**Name-**

**Domain- {category}**

**Years of experience-**

**2) Two latest companies that the candidate has worked in ordered by the most recent one alongwith 2 projects in each of those in this format-**

**- Company-1: Designation**

**Project-1 name: Project-1 description**

**Project-2 name: Project-2 description**

**- Company-1: Designation**

**Project-1 name: Project-1 description**

**Project-2 name: Project-2 description**

**3) Top 5 Technical Skillset**

**4) Top 5 Non Technical Skillset**

**5) Recent 2 Educational qualifications in this format-**

**- Educational Qualification-1: [College Name & GPA]**

**- Educational Qualification-2: [College Name & GPA]**

**Don't hallucinate while summarizing, if the required information isn't present, say that its not avaialble'''),**

**]**

**)**

This prompt was too verbose and was sometimes missing on the required details & generating inconsistent summaries sometimes. Also, because of the large number of tokens, this was increasing the OpenAI api costs as well. We ended up breaking the text summarization task into a 2 step process-

Step-1 extracts critical details as Json output

**prompt=ChatPromptTemplate.from\_messages([("system",'''You are an AI bot designed to act as a professional for parsing resumes. You are given with resume and your job is to extract the following information from the resume:**

**1. full name**

**2. email id**

**3. github portfolio**

**4. linkedIn id**

**5. employment details (latest 2)**

**6. technical skills (top 5)**

**7. soft skills (top 5)**

**Give the extracted information strictly in json format only. Don't hallucinate while parsing, if the required information isn't present, say that its "not available" ''' ),**

**("human",'''Here's the candidate profile mentioned- {input}''')])**

Step-2 uses this Json to generate the profile summary

**prompt\_json\_summary=ChatPromptTemplate.from\_messages([("system",'''You are an HR recruitment assistant designed to create candidate profile summaries. You are given a Json input with candidate details and your job is to create a summary out of it strictly upto 150 words.**

**Don't hallucinate while creating summaries, if the required information isn't present, say that its "not available" ''' ),**

**("human",'''Here's the candidate json profile mentioned- {json\_input}''')])**

1. The 2nd challenge was getting the right text-to-speech model. The GTTS python api generated robot like voice and was missing a few words from the text summary. This is where ElevenLabs model was helpful in generating a more human-like audio from the text summary.
2. The 3rd challenge was creating a video summaries from audio with AI avatar overlay on top & ensuring the lip syncing is appropriate. Multiple APIs were evaluated for this step-
3. Di-d
4. Akool
5. Neiro

All of these tools provide apis which are moderately expensive & don’t provide a great performance on the inference side either. This is where **Wav2Lip,** a pretrained open source GAN based model was super helpful. Even though it performed very well in creating video summaries, the challenge of reducing inference time still remains to be addressed.

1. The 4th challenge was leveraging MLOps practices and integrating the whole pipeline. There were quite a few integration errors that had to be resolved like build the end-to-end pipeline & deploy this solution as a Web App with a UI.

# Future Work:

#### Functional Areas:

1. **Enhanced Summaries -**
   * Improve profile summaries by leveraging techniques like few shot prompting, instruction finetuning or parameter finetuning techniques like PEFT, LoRA, QLoRA.

#### Candidate Job Role fitness Score -

* + Incorporating features to generate more insights for the recruiter or integrate an algorithm to provide an overall fitness score of the candidate for a specific job role.

#### Advanced Multilingual Support -

* + Explore and implement support for multiple languages both in terms of the input & output modalities to broaden the user base.

#### -

#### Non-Functional Areas:

1. **Cost optimization -**
   * Investigate and implement open source models to save on the OpenAI api costs.
   * Experiment with diverse LLMs beyond the current implementation to enhance model versatility and effectiveness.

#### Reduce Inference Time:

* + The Audio to AI avatar video creation is currently taking ~20 mins to create a 1 min video for each profile which can definitely be produced by either leveraging parallel architecture for inferencing or exploring more efficient models.

#### Scalability:

* + The deployed app should be able to handle concurrent user requests in future.

1. AI Agents:

The entire solution can be converted into a multi-Agent framework & the individual tasks like summarization, CV to JD matching etc. could be the tasks assigned to each Agent.

# References:

1. Yang, H., Liu, X.-Y., & Wang, C. D. *FinGPT: Open-Source Financial Large Language Models*. Retrieved from **ArXiv**
2. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y

ReAct: Synergizing Reasoning and Acting in Language Models. Retrieved from **ArXiv**

1. *Stock Market Analysis and Prediction Using LSTM*. Retrieved from **Kaggle**
2. LangChain. *LangChain Additional Resources and Tutorials*. Retrieved from

**LangChain**

1. Vercel. *Behind the Scenes of Vercel's Infrastructure: Achieving Optimal Scalability and Performance*. Retrieved from **Vercel**
2. AI4Finance Foundation. *FinGPT GitHub Repository README*. Retrieved from

**GitHub**

1. OpenAI . *OpenAI Platform Documentation*. Retrieved from **OpenAI**
2. Bodie, Z., Kane, A. and Marcus, A. *Investments. 8th Edition, McGraw-Hill, New York.*
3. Ranin, A***.*** *Robo-Advisor with Python: A Hands-on Guide to Building and Operating Your Own Robo-advisor.*

THE END