Enhancing Resume Shortlisting and QA using LangChain, LLM and Retrieval-Augmented Generation (RAG)

Naveen Raju S G

Email - naveenraju100@gmail.com

LinkedIn - https://www.linkedin.com/in/naveen-raju-s-g-bb1486124/

Contact number - +1 312 912 2878

Project Motivation:

By addressing these current problems and implementing innovative solutions, we can revolutionize the resume shortlisting process, providing a more accurate and fair representation of candidates' qualifications and experiences.

Current Problems with Resume Shortlisting:

1. Keyword-Centric Focus:

ATS systems predominantly rely on keywords, leading to a lack of holistic understanding of resume content.

2. Limited Context Understanding:

The inability to comprehend project titles, descriptions, and nuanced skill similarities hinders accurate shortlisting.

3. Uninformed Skill Assessment:

HR may lack understanding of how mentioned skills are practically utilized in specific contexts and how are they useful for job for which resume is being shortlisted thus affecting decision-making.

4. Oversights in Work Experience Evaluation:

Strict adherence to exact work experience matches may overlook relevant experiences that align with job descriptions.

5. Page Limit Constraints:

Students often struggle to include comprehensive project details within the confined space of a one-page resume. As a result, they may end up omitting some projects, fearing that HR professionals won't have time to review a resume longer than one page. This concern may lead students to exclude projects that could be crucial for HR in the resume shortlisting process.

6. Under utilization of Portfolios:

Even though students include portfolio links containing detailed project descriptions and a wealth of projects, these links often go unopened or are not thoroughly understood during the shortlisting process.

Overcoming Challenges with Innovative Solutions using LLM, RAG and LangChain:

1. Utilizing Large Language Models (LLM):

Leverage advanced large language models like GPT-3.5, to comprehensively understand resumes and portfolios, enabling HR to efficiently and accurately shortlist deserving candidates.

2. Enhancing Contextual Analysis with RAG:

Incorporate Retrieval-Augmented Generation (RAG) for a more comprehensive understanding of project titles, descriptions, and skill relevancy beyond exact keyword matches.

3. LangChain for Skill-Context Chains:

Utilize LangChain to build sophisticated chains for meticulous skill assessment, offering detailed analysis and feedback on the relevance of mentioned skills to the given job description. This includes acknowledging skills that, while not explicitly stated in the job description, contribute to the overall suitability for the role.

4. Strategic QA for Resume and Portfolio Understanding:

Develop QA strategies using LangChain and LLM to extract and evaluate relevant details from portfolio links and resume, ensuring a more thorough assessment.

5. Addressing Page Limit Challenges by analyzing portfolio website content:

Utilizing LangChain, LLM, and RAG, review the HTML content of portfolios to shortlist candidates based on provided job descriptions. If there's no match for the given job description, subsequently assess if the portfolio aligns with any other job descriptions within the company

6. Educating ATS Systems with Advanced Technologies:

Advocate for the integration of advanced technologies, such as LLM, RAG, and LangChain, within ATS systems to enhance efficiency and accuracy in the shortlisting process.

7. Promoting Fair Opportunities:

Raise awareness about the limitations of existing ATS systems and champion the adoption of more advanced, inclusive technologies to ensure deserving candidates do not miss interview opportunities.

Project Details:

Load docs, create chroma db using embedding, use it as a retriever, create Retrieval QA

All the below code is in new_1.ipynb

- Load documents from a folder consisting of multiple resume PDF file using DirectoryLoader

```
import os
import openai
import pinecone
from langchain.document_loaders import DirectoryLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import Pinecone
from langchain.llms import OpenAI
from langchain.chains.question_answering import load_qa_chain
```

```
C:\Users\navee\anaconda3\envs\llm_project\lib\site-packages\pinecone\index.py:4: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets.
See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from tqdm.autonotebook import tqdm
```

```
directory = 'E:/LangChain course/Lang Chain for LLM application development/cv_read/'

def load_docs(directory):
  loader = DirectoryLoader(directory,show_progress=True) #unstructuredLoader by default has used this auto identify file type and load it, mode="single",
  documents = loader.load()
  return documents

documents = load_docs(directory)
len(documents)
```

Default config of DirectoryLoader (mode=”single” which treats a document as single entities instaed of “elements” where it break downs. strategy='fast' where it splits based on [“\n”,”\n\n”,” “,””] alternative option is “hi_res” which uses Yolo algorithm.

- Split the documents using RecursiveCharacterTextSplitter with chunk size = 1000, chunk overlap=100
- Set the environment with OPENAI_API_KEY
- Create OpenAIEmbeddings instance

```
def split_docs(documents, chunk_size=1000, chunk_overlap=100):
  text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
  docs = text_splitter.split_documents(documents)
  return docs

docs = split_docs(documents)
print(len(docs))
```

```
47
```

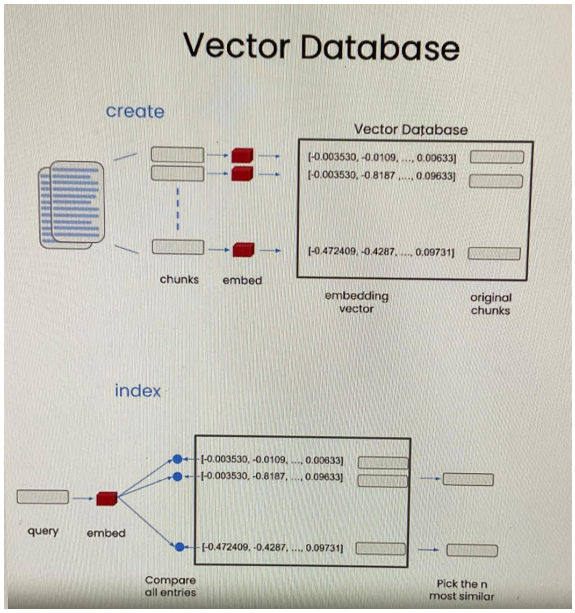
```
import os

os.environ["OPENAI_API_KEY"] = "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"
```

```
from langchain.embeddings import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(openai_api_key="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")
```

- Let’s use Chroma DB from langchain vector stores to embed and store the vector database on local hard disk
- Now load the persisted database from disk
- Initiate the Chroma vector database retriever with k=2, where default search type is similarity_score_threshold
- The above retriever helps us to retrieve document section from PDF’s which is suitable for give role




```
from langchain.vectorstores import Chroma
# Embed and store the texts
# Supplying a persist_directory will store the embeddings on disk
persist_directory = 'db'

## here we are using OpenAI embeddings but in future we will swap out to local embeddings
embedding = OpenAIEmbeddings()

vectordb = Chroma.from_documents(documents=docs,
                                 embedding=embedding,
                                 persist_directory=persist_directory)
```

```
# persiste the db to disk
vectordb.persist()
vectordb = None
```

```
# Now we can load the persisted database from disk, and use it as normal.
vectordb = Chroma(persist_directory=persist_directory,
                  embedding_function=embedding)
```

```
retriever = vectordb.as_retriever(search_kwargs={"k": 2}) # by default search_type="similarity_score_threshold"
```

- One of the resumes loaded before had skills and other information related to Machine learning role, so it extracted contents of that resume

```
docs = retriever.invoke("which candidate is good fit for Machine learning engineer roles.")
```

```
print(docs[0].page_content)
```

```
TECHNICAL SKILLS Cloud : AWS (Amazon Web Services) Deep learning framework : Keras, TensorFlow Other libraries : Nump
y, Pandas, Matplotlib, scikit-learn Distributed programming : Apache PySpark Generative AI : Large language models (LL
M) Other relevant skills : MLOPS, Machine learning, Deep Learning/ComputerVision, NLP

Programming languages : Python, R, C++, SQL Image processing libraries : Open CV, scikit-image Data Pipelining : Apach
e Airflow Version Control : Git and GitHub

EDUCATION Illinois Institute of Technology Master of Science Artificial Intelligence - GPA: 3.833/4 May 2024 Courses:
Machine Learning, Data Mining, Applied Statistics,Big Data Technologies, Data Preparation and Analysis, Computer Visio
n, Natural Language Processing, Deep Learning

Visvesvaraya Technological University, India Bachelor of Engineering, Information Science and Engineering

July 2018

WORK EXPERIENCE Graduate Teaching Assistance - Data Mining course (Computer Science Department)
```

- One of the resumes loaded before had skills and other information related to Data Analyst role, so it extracted contents of that resume

```
# docs = retriever.invoke("which candidate is good fit for Data Analyst roles.")
docs = retriever.invoke("Give name of candidate who is good fit for Data Analyst roles.")
```

```
print(docs[0].page_content)
```

```
BYJUS THINK AND LEARN, Bangalore, India Data Analyst - Operations (Power BI | Data Analysis| Data Visualization) Augus
t 2019 - August 2021  Achieved a 30% reduction in processing time by optimizing logistics and supply chain processes
using

Python and Tableau.

Delivered a remarkable 25% improvement in on-time delivery rates and reduced asset loss through data driven solutions.

Streamlined operations with data insights, resulting in 8% improvement in customer satisfaction through enhanced order
fulfillment and reduced delivery time.

EDUCATION

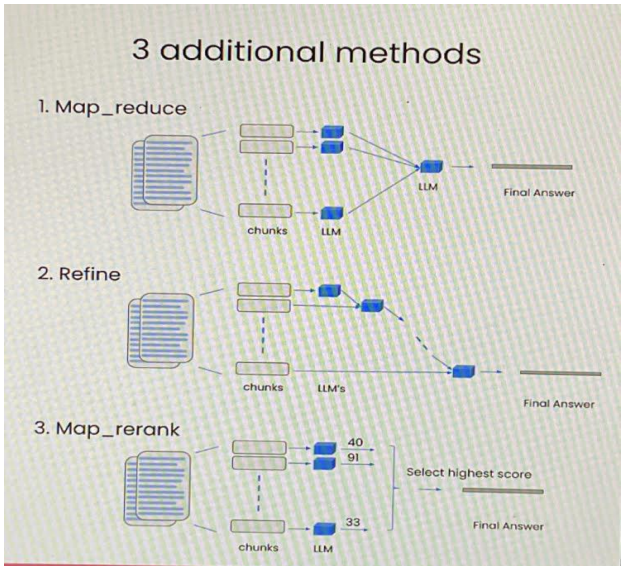
ILLINOIS INSTITUTE OF TECHNOLOGY, Chicago, IL Master of Science in Data Science May 2023 Courses - Machine Learning, N
atural Language Processing, Statistics, Time Series.

PRESIDENCY UNIVERSITY, India

Bachelor of Technology in Computer Science

PROJECTS
```

- Create a ChatOpenAI instance with temperature=0 with model as 'gpt-3.5-turbo'
- Initiate a RetrievalQA chain with ChatOpenAI instance created in last step and with following config : chain_type="stuff"(means we just stuff entire data into context to pass to LLM. But other methods include Map_reduce,Refine and Map_rerank), retriever (created in previous steps)



- Write a function to post process LLM response, Usually in the response of LLM it has meta data I.e source of a document from it answered. So we should capture that detail and print it.

```
from langchain.chat_models import ChatOpenAI
turbo_llm = ChatOpenAI(
    temperature=0,
    model_name='gpt-3.5-turbo'
)
```

```
from langchain.chains import RetrievalQA
# create the chain to answer questions
qa_chain = RetrievalQA.from_chain_type(llm=turbo_llm,
                                       chain_type="stuff",
                                       retriever=retriever,
                                       return_source_documents=True)
```

```
## Cite sources
def process_llm_response(llm_response):
    print(llm_response['result'])
    print('\n\nSources:')
    for source in llm_response["source_documents"]:
        print(source.metadata['source'])
```

- Now lets create a prompt which contains a warning so LLM would not hallucinate, job description of some job from linkedIn and message to short list resumes which is good fit based on skills,education and work experience mentioned in it
- Finally run function returned from LLM to retrieve pdf name(I.e resume file name)

```
# full example
warning = "If you don't know the answer, just say that you don't know, don't try to make up an answer"
job_description = "MS or PhD in computer science or a related technical field,5+ years of industry work experience. Good sense of product with a focus o
Excellent coding, analysis, and problem-solving skills. Proven knowledge of data structure and algorithms. \
Familiarity in relevant machine learning frameworks and packages such as Tensorflow, PyTorch and HuggingFace\
Experience working with Product Management and decomposing feature requirements into technical work items to ship products\
Experience with generative AI, knowledge of ML Ops and ML services is a plus. This includes Pinecone, LangChain, Weights and Biases etc. \
Familiarity with deployment technologies such as Docker, Kubernetes and Triton are a plus\
Strong communication and collaboration skills"
question = warning+job_description + " Based on the given job description"
query = question + "short list resumes which is good fit based on skills,education and work experience mwntioned in it? also provide the candidate name
# query = "short list resumes which is good fit for Data analysis roles based on skills,education and work experience mwntioned in it?"

llm_response = qa_chain(query)
process_llm_response(llm_response)
```

```
Based on the given job description, the following candidate seems to be a good fit:

Candidate Name: Not provided in the given context

Skills:
- Expertise in Python and Python based ML/DL and Data Science frameworks
- Excellent coding, analysis, and problem-solving skills
- Proven knowledge of data structure and algorithms
- Familiarity with machine learning frameworks such as TensorFlow, PyTorch, and HuggingFace
- Experience with generative AI
- Knowledge of ML Ops and ML services (e.g., Pinecone, LangChain, Weights and Biases)
- Familiarity with deployment technologies such as Docker, Kubernetes, and Triton
- Strong communication and collaboration skills

Education:
- Master of Science in Artificial Intelligence from Illinois Institute of Technology (GPA: 3.833/4)
- Bachelor of Engineering in Information Science and Engineering from Visvesvaraya Technological University, India

Work Experience:
- Graduate Teaching Assistance - Data Mining course (Computer Science Department)

Please note that the candidate's name is not provided in the given context.


Sources:
E:\LangChain course\Lang Chain for LLM application development\cv_read\CV_ML_NAVEEN RAJU SREERAMA RAJU GOVINDA RAJU.pdf
E:\LangChain course\Lang Chain for LLM application development\cv_read\CV_ML_NAVEEN RAJU SREERAMA RAJU GOVINDA RAJU.pdf
```

All the below code is in new_3.ipynb

- Created steps same as previously mentioned but one change made was did not split the document after embedding.

Here it was able to identify Name in resume as whole document was given in context. Here in below text LLM is trying to extract Name, and few other information based on prompt give such that it matches with job description given. Followed by extracting source document using custom post processing function written.
This is the output of Retrieval QA chain.

```
# full example
warning = "If you don't know the answer, just say that you don't know, don't try to make up an answer"
job_description = "MS or PhD in computer science or a related technical field,5+ years of industry work experience. Good sense of product with a focus o
Excellent coding, analysis, and problem-solving skills. Proven knowledge of data structure and algorithms. \
Familiarity in relevant machine learning frameworks and packages such as Tensorflow, PyTorch and HuggingFace\
Experience working with Product Management and decomposing feature requirements into technical work items to ship products\
Experience with generative AI, knowledge of ML Ops and ML services is a plus. This includes Pinecone, LangChain, Weights and Biases etc. \
Familiarity with deployment technologies such as Docker, Kubernetes and Triton are a plus\
Strong communication and collaboration skills"
question = warning+job_description + " Based on the given job description"
query = question + "short list resumes which is good fit based on skills,education and work experience mwntioned in it? also provide the candidate name
# query = "short list resumes which is good fit for Data analysis roles based on skills,education and work experience mwntioned in it?"

llm_response = qa_chain(query)
process_llm_response(llm_response)
```

```
Based on the given job description, the following candidates seem to be a good fit based on their skills, education, and work experience:

1. Naveen Raju S G - The candidate has a Master of Science in Artificial Intelligence, experience in Machine Learning, Deep Learning/Computer Vision, an
d Data Science. They have worked on projects involving Deep Learning algorithms and image processing pipelines. They also have experience with Python, T
ensorFlow, and other relevant libraries. They have experience working with Product Management and have knowledge of ML Ops and ML services. They have go
od communication and collaboration skills.

Please note that I cannot provide the candidate's name without a subheading as it is not mentioned in the provided context.


Sources:
E:\LangChain course\Lang Chain for LLM application development\cv_read\CV_ML_NAVEEN RAJU SREERAMA RAJU GOVINDA RAJU.pdf
```

- Retrieving resume content which is suitable based on given job description

```
warning = "If you don't know the answer, just say that you don't know, don't try to make up an answer"
job_description = "MS or PhD in computer science or a related technical field,5+ years of industry work experience. Good sense of product with a focus o
Excellent coding, analysis, and problem-solving skills. Proven knowledge of data structure and algorithms. \
Familiarity in relevant machine learning frameworks and packages such as Tensorflow, PyTorch and HuggingFace\
Experience working with Product Management and decomposing feature requirements into technical work items to ship products\
Experience with generative AI, knowledge of ML Ops and ML services is a plus. This includes Pinecone, LangChain, Weights and Biases etc. \
Familiarity with deployment technologies such as Docker, Kubernetes and Triton are a plus\
Strong communication and collaboration skills"
question = warning+job_description + " Based on the given job description"
query = question + "retrive the full document information of a resume which is good fit based on skills,education and work experience mwntioned in it? "
```

```
resume_doc = retriever.invoke(query)
```

```
print(resume_doc)
```

```
resume_doc = resume_doc[0].page_content
```

- Demonstrating the use of prompt template:

Define a prompt template to extract skills, education, projects, publications, work experience as comma separated python list.

```
review_template = """\
For the following text, extract the following information:

Skills: what are the technical and non technical skills? \
Answer output them as a comma separated Python list.

Education: What is the highest education of the candidate and what is the GPA as mentioned in the text?\
Answer Output should be the university/college name and GPA if given in text, output them as a comma separated Python list.

Projects: Extract all project titles mentioned in a text\
and output them as a comma separated Python list.

Publications: Extract all publication titles mentioned in a text\
and output them as a comma separated Python list.

Work experience: Extract all organisation name where he/she has worked along with number of years or months worked there and also extract designation\
and output them as a comma separated Python list.

Format the output as JSON with the following keys:
Skills
Education
Projects
Publications
Work experience

text: {text}
"""
```

```
from langchain.prompts import ChatPromptTemplate

prompt_template = ChatPromptTemplate.from_template(review_template)
print(prompt_template)
```

```
input_variables=['text'] messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['text'], template='For the following text, extract
the following information:\n\nSkills: what are the technical and non technical skills? Answer output them as a comma separated Python list.\n\nEducatio
n: What is the highest education of the candidate and what is the GPA as mentioned in the text?Answer Output should be the university/college name and
GPA if given in text, output them as a comma separated Python list.\n\nProjects: Extract all project titles mentioned in a textand output them as a com
ma separated Python list.\n\nPublications: Extract all publication titles mentioned in a textand output them as a comma separated Python list.\n\nWork
experience: Extract all organisation name where he/she has worked along with number of years or months worked there and also extract designationand out
put them as a comma separated Python list.\n\nFormat the output as JSON with the following keys:\nSkills\nEducation\nProjects\nPublications\nWork exper
ience\n\ntext: {text}\n'))]
```

Create a conversation chain with memory

- Memory type used here is ConversationBufferWindowMemory

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferWindowMemory
memory = ConversationBufferWindowMemory(k=1)
memory.save_context({"input": "Hi"},
                    {"output": "What's up"})
memory.save_context({"input": "Not much, just hanging"},
                    {"output": "Cool"})
memory.load_memory_variables({})

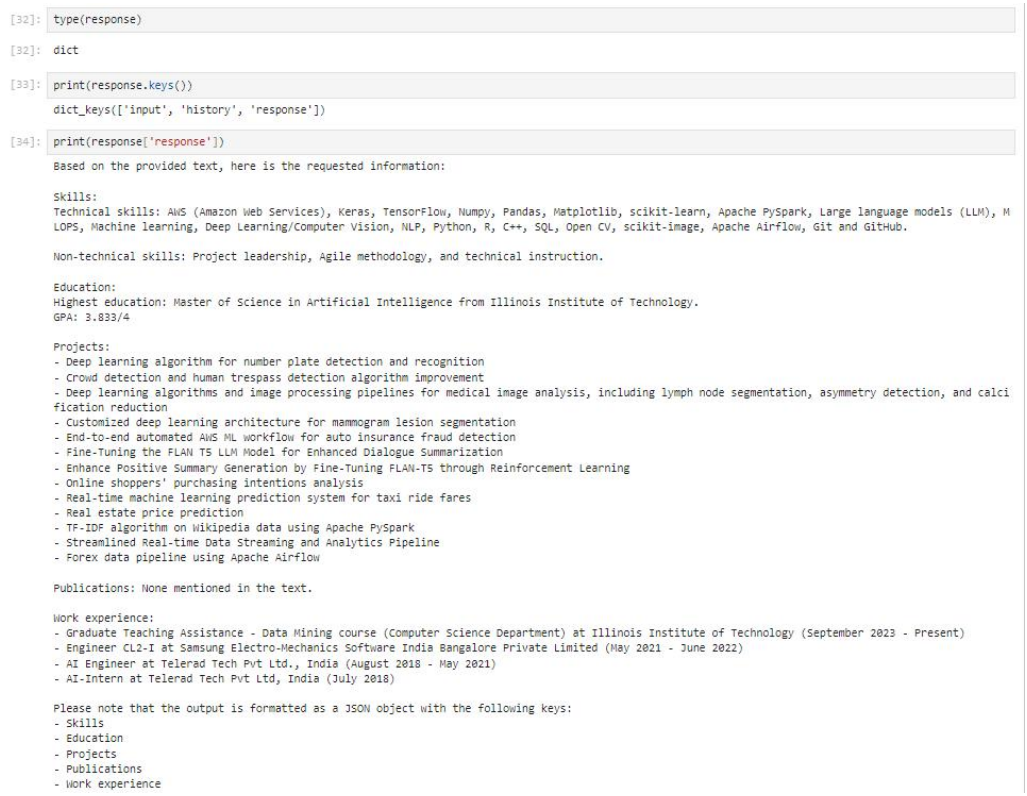
turbo_llm_memory = ChatOpenAI(
    temperature=0,
    model_name='gpt-3.5-turbo'
)


memory_llm_conversation = ConversationChain(
    llm=turbo_llm_memory,
    memory = memory,
    verbose=True
)
```

```
messages = prompt_template.format_messages(text=resume_doc)
# chat = ChatOpenAI(temperature=0.0, model=turbo_llm_memory)
response = memory_llm_conversation(messages)
```

Here, resume_doc assigned to text is a document retrived in previous steps based on the job description.

Output:



```
[32]: type(response)

[32]: dict

[33]: print(response.keys())

dict_keys(['input', 'history', 'response'])

[34]: print(response['response'])

Based on the provided text, here is the requested information:

Skills:
Technical skills: AWS (Amazon Web Services), Keras, TensorFlow, Numpy, Pandas, Matplotlib, scikit-learn, Apache PySpark, Large language models (LLM), M
LOPS, Machine learning, Deep Learning/Computer Vision, NLP, Python, R, C++, SQL, Open CV, scikit-image, Apache Airflow, Git and GitHub.

Non-technical skills: Project leadership, Agile methodology, and technical instruction.

Education:
Highest education: Master of Science in Artificial Intelligence from Illinois Institute of Technology.
GPA: 3.833/4

Projects:
- Deep learning algorithm for number plate detection and recognition
- Crowd detection and human trespass detection algorithm improvement
- Deep learning algorithms and image processing pipelines for medical image analysis, including lymph node segmentation, asymmetry detection, and calci
fication reduction
- Customized deep learning architecture for mammogram lesion segmentation
- End-to-end automated AWS ML workflow for auto insurance fraud detection
- Fine-Tuning the FLAN T5 LLM Model for Enhanced Dialogue Summarization
- Enhance Positive Summary Generation by Fine-Tuning FLAN-T5 through Reinforcement Learning
- Online shoppers' purchasing intentions analysis
- Real-time machine learning prediction system for taxi ride fares
- Real estate price prediction
- TF-IDF algorithm on Wikipedia data using Apache PySpark
- Streamlined Real-time Data Streaming and Analytics Pipeline
- Forex data pipeline using Apache Airflow

Publications: None mentioned in the text.

Work experience:
- Graduate Teaching Assistance - Data Mining course (Computer Science Department) at Illinois Institute of Technology (September 2023 - Present)
- Engineer CL2-I at Samsung Electro-Mechanics Software India Bangalore Private Limited (May 2021 - June 2022)
- AI Engineer at Telerad Tech Pvt Ltd., India (August 2018 - May 2021)
- AI-Intern at Telerad Tech Pvt Ltd, India (July 2018)

Please note that the output is formatted as a JSON object with the following keys:
- Skills
- Education
- Projects
- Publications
- Work experience
```

- Parsing the LLM output using output parser

Define response schemas for skills, projects and work experience that need to be extracted, using which create structured output parser.
Create a prompt template to instruct to extract skills, projects and work experience, which also include placeholder for context(I.e resume in our case) and format instructions(derived from StructuredOutputParser).
Now send the formatted prompt template to LLM whose output will be dictionary thus we can easily retrieve required answers using specific keys.

```
from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser
```

```
skills_schema = ResponseSchema(name="Skills",
                               description="what are the technical and non technical skills? \
Answer output them as a comma separated Python list.")

# Education_schema = ResponseSchema(name="Education",
#                                   description="What is the highest education of the candidate and what is the GPA as mentioned in the text?\
# Answer Output should be the university/college name and GPA if given in text, output them as a comma separated Python list.")


Projects_schema = ResponseSchema(name="Projects",
                                 description="Extract all project titles mentioned in a text\
and output them as a comma separated Python list.")

# Publications_schema = ResponseSchema(name="Publications",
#                                   description="Extract all publication titles mentioned in a text\
# and output them as a comma separated Python list.")

Work_experience_schema = ResponseSchema(name="Work experience",
                                 description="Extract all organisation name where he/she has worked along with number of years or months worked there
and output them as a comma separated Python list.")

response_schemas = [skills_schema,
                    # Education_schema,
                    Projects_schema,
                    # Publications_schema,
                    Work_experience_schema]
```

```
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

```
format_instructions = output_parser.get_format_instructions()
```

```
print(format_instructions)
```

```
The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```json
{
        "Skills": string  // what are the technical and non technical skills? Answer output them as a comma separated Python list.
        "Projects": string  // Extract all project titles mentioned in a textand output them as a comma separated Python list.
        "Work experience": string  // Extract all organisation name where he/she has worked along with number of years or months worked there and also
extract designationand output them as a comma separated Python list.
}
```
```

```
review_template_2 = """\
For the following text, extract the following information:

Skills: what are the technical and non technical skills? \
Answer output them as a comma separated Python list.

Projects: Extract all project titles mentioned in a text\
and output them as a comma separated Python list.

Work experience: Extract all organisation name where he/she has worked along with number of years or months worked there and also extract designation\
and output them as a comma separated Python list.

Format the output as JSON with the following keys:
Skills
Projects
Work experience

text: {text}

{format_instructions}
"""


prompt = ChatPromptTemplate.from_template(template=review_template_2)

messages = prompt.format_messages(text=resume_doc,
                                  format_instructions=format_instructions)
```

```
response2 = turbo_llm_memory(messages)
```

```
print(response2.content)
```

```
```json
{
        "Skills": "AWS (Amazon Web Services), Keras, TensorFlow, Numpy, Pandas, Matplotlib, scikit-learn, Apache PySpark, Large language models (LLM), MLOPS, Machine learning, Deep Learning/ComputerVision, NLP, Python, R, C++, SQL, Open CV, scikit-image, Apache Airflow, Git and GitHub",
        "Projects": "End-to-end automated AWS ML workflow for auto insurance fraud detection, Fine-Tuning the FLAN T5 LLM Model for Enhanced Dialogue Summarization, Enhance Positive Summary Generation by Fine-Tuning FLAN-T5 through Reinforcement Learning, Online shoppers' purchasing intentions, Real-time machine learning prediction system for taxi ride fares, Real estate price prediction, TF-IDF algorithm on Wikipedia data using Apache PySpark, Streamlined Real-time Data Streaming and Analytics Pipeline, Forex data pipeline using Apache Airflow",
        "Work experience": "Data Mining course (Computer Science Department), Samsung Electro-Mechanics Software India Bangalore Private Limited, Telerad Tech Pvt Ltd., India, Telerad Tech Pvt Ltd, India"
}
```
```

```
type(response2.content)
```

```
str
```

```
output_dict = output_parser.parse(response2.content)
```

```
output_dict
```

```
{'Skills': 'AWS (Amazon Web Services), Keras, TensorFlow, Numpy, Pandas, Matplotlib, scikit-learn, Apache PySpark, Large language models (LLM), MLOPS, Machine learning, Deep Learning/ComputerVision, NLP, Python, R, C++, SQL, Open CV, scikit-image, Apache Airflow, Git and GitHub',
 'Projects': "End-to-end automated AWS ML workflow for auto insurance fraud detection, Fine-Tuning the FLAN T5 LLM Model for Enhanced Dialogue Summarization, Enhance Positive Summary Generation by Fine-Tuning FLAN-T5 through Reinforcement Learning, Online shoppers' purchasing intentions, Real-time machine learning prediction system for taxi ride fares, Real estate price prediction, TF-IDF algorithm on Wikipedia data using Apache PySpark, Streamlined Real-time Data Streaming and Analytics Pipeline, Forex data pipeline using Apache Airflow",
 'Work experience': 'Data Mining course (Computer Science Department), Samsung Electro-Mechanics Software India Bangalore Private Limited, Telerad Tech Pvt Ltd., India, Telerad Tech Pvt Ltd, India'}
```

```
type(output_dict)
```

```
dict
```

```
output_dict.get('Skills')
```

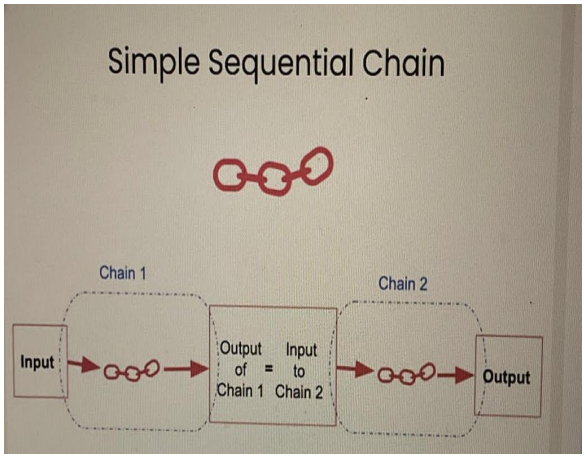
```
'AWS (Amazon Web Services), Keras, TensorFlow, Numpy, Pandas, Matplotlib, scikit-learn, Apache PySpark, Large language models (LLM), MLOPS, Machine learning, Deep Learning/ComputerVision, NLP, Python, R, C++, SQL, Open CV, scikit-image, Apache Airflow, Git and GitHub'
```

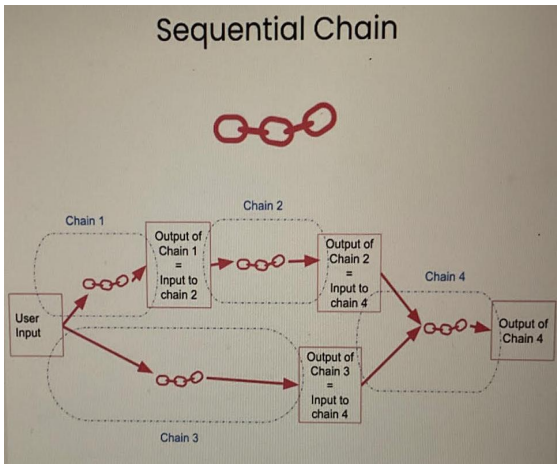
```
output_dict.get('Projects')
```

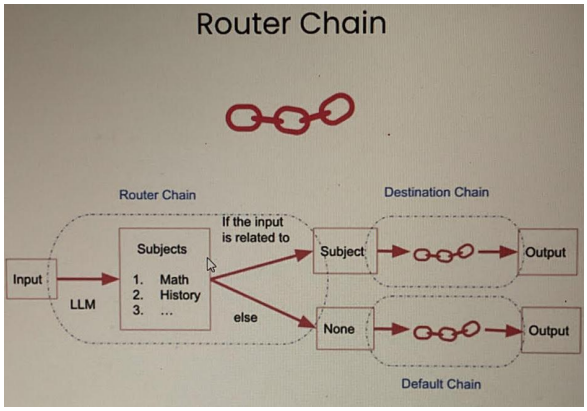
```
"End-to-end automated AWS ML workflow for auto insurance fraud detection, Fine-Tuning the FLAN T5 LLM Model for Enhanced Dialogue Summarization, Enhance Positive Summary Generation by Fine-Tuning FLAN-T5 through Reinforcement Learning, Online shoppers' purchasing intentions, Real-time machine learning prediction system for taxi ride fares, Real estate price prediction, TF-IDF algorithm on Wikipedia data using Apache PySpark, Streamlined Real-time Data Streaming and Analytics Pipeline, Forex data pipeline using Apache Airflow"
```

Types of Chains:

Simple sequential Chains, Sequential chains and Router Chains







Demonstrating Sequential Chains

First chain is to extract technical and non technical skills.
Second chain is for what are the job roles among Data Scientist, Machine learning Engineer, Software Engineer, Data Engineer, Devops Engineer, Cloud Architect. Are suited based on the given skill sets.
Third chain is for explaining each skill as for what kind of projects are these useful.

```
from langchain.chains import SequentialChain
from langchain.chat_models import ChatOpenAI
```

```
import os

os.environ["OPENAI_API_KEY"] = "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"
```

```
llm = ChatOpenAI(temperature=0.9, model="gpt-3.5-turbo")
```

```
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain

first_prompt = ChatPromptTemplate.from_template(
    "Skills: what are the technical and non technical skills? \
Answer output them as a comma separated Python list."
    "\n\n{resume_doc}"
)

chain_one = LLMChain(llm=llm, prompt=first_prompt,
                     output_key="skills"
                    )
```

```
second_prompt = ChatPromptTemplate.from_template(
    "Can you name what the job roles among Data Scientist, Machine learning Engineer, Software Engineer, Data Engineer, Devops Engineer, Cloud Architect
    "\n\n{skills}"
)

chain_two = LLMChain(llm=llm, prompt=second_prompt,
                     output_key="job_titles"
                    )
```

```
third_prompt = ChatPromptTemplate.from_template(
    "Explain each skill as for what kind of projects are these usefull:\n\n{skills}"
)
# chain 3: input= Review and output= Language
chain_three = LLMChain(llm=llm, prompt=third_prompt,
                       output_key="skills_explanation"
                      )
```

```
overall_chain = SequentialChain(
    chains=[chain_one, chain_two, chain_three],
    input_variables=["resume_doc"],
    output_variables=["skills", "job_titles","skills_explanation"],
    verbose=True
)
```

```
seqchain_output = overall_chain(resume_doc)
```

```
> Entering new SequentialChain chain...

> Finished chain.
```

```
type(seqchain_output)
```

```
dict
```

```
seqchain_output.keys()
```

```
dict_keys(['resume_doc', 'skills', 'job_titles', 'skills_explanation'])
```

```
print(seqchain_output['skills'])
```

```
['AWS (Amazon Web Services)', 'Keras', 'TensorFlow', 'Numpy', 'Pandas', 'Matplotlib', 'scikit-learn', 'Apache PySpark', 'Large language models (LLM)',
'MLOPS', 'Machine learning', 'Deep Learning/ComputerVision', 'NLP', 'Python', 'R', 'C++', 'SQL', 'Open CV', 'scikit-image', 'Apache Airflow', 'Git and
GitHub']
```

```
print(seqchain_output['job_titles'])
```

```
Based on the given skill sets, the job roles that are suited include:

- Data Scientist: Machine learning, Deep Learning/ComputerVision, NLP, Python, R, SQL, Apache PySpark, Apache Airflow, Git and GitHub.
- Machine learning Engineer: Machine learning, Deep Learning/ComputerVision, NLP, Python, R, C++, SQL, Apache PySpark, Large language models (LLM), MLO
PS, Git and GitHub.
- Software Engineer: Python, R, C++, SQL, Git and GitHub.
- Data Engineer: Python, SQL, Apache PySpark, Git and GitHub.
- Devops Engineer: Python, Git and GitHub, AWS (Amazon Web Services).
- Cloud Architect: Python, Git and GitHub, AWS (Amazon Web Services).

Note that there may be some overlap in required skills between these roles, and the specific skill sets required can vary depending on the organization
and the nature of the job.
```

```
print(seqchain_output['skills_explanation'])
```

```
- AWS (Amazon Web Services): Useful for projects involving cloud computing, storage, and data analysis. It provides a wide range of services such as EC
2, S3, and Redshift, which are valuable for scalable and secure applications.

- Keras: Particularly helpful for building neural network models, especially deep learning models. It is a high-level neural networks API written in Py
thon, capable of running on top of TensorFlow, Theano, or CNTK.

- TensorFlow: Beneficial for machine learning projects, especially those involving deep learning. TensorFlow is an open-source library that offers a fl
exible ecosystem for building and deploying machine learning models.

- Numpy: Ideal for projects involving numerical computations and data manipulation. Numpy is a fundamental library in Python that provides support for
large, multi-dimensional arrays and matrices along with a vast collection of mathematical functions.

- Pandas: Valuable for data manipulation and analysis tasks. Pandas is a powerful Python library that offers data structures and functions to efficient
ly work with structured data, enabling tasks like data cleaning, filtering, and transformation.

- Matplotlib: Suitable for projects requiring data visualization. Matplotlib is a plotting library in Python that allows the creation of various types
of plots, charts, and graphs, making it easier to interpret and present data.

- scikit-learn: Useful for machine learning projects, specifically those involving data preprocessing, model selection, and evaluation. scikit-learn pr
ovides a comprehensive set of tools and algorithms for common machine learning tasks.

- Apache PySpark: Helpful for big data processing and analysis projects. PySpark is the Python API for Apache Spark, a powerful open-source analytics e
ngine providing distributed computing capabilities for large datasets.

- Large language models (LLM): Valuable for projects involving natural language processing (NLP), text generation, or understanding human language. Lar
ge language models like GPT-3 are designed to perform complex language-related tasks.

- MLOPS: Useful for projects involving the deployment and management of machine learning models in production. MLOPS focuses on automating the machine
learning lifecycle, including model training, testing, deployment, and monitoring.

- Machine learning: Applicable to various projects that aim to create predictive or analytical models using algorithms to learn patterns and make predi
ctions or decisions based on data.

- Deep Learning/Computer Vision: Particularly suitable for projects involving image or video analysis, object detection, and recognition. Deep learning
and computer vision techniques enable the extraction of meaningful information from visual data.

- NLP: Beneficial for projects involving natural language processing tasks such as text classification, sentiment analysis, language translation, or ch
atbots. NLP focuses on understanding and processing human language.

- Python: Versatile and widely used in various projects, especially in data analysis, machine learning, and web development. Python has rich libraries,
frameworks, and a clean syntax that makes it easy to work with.

- R: Particularly helpful for statistical analysis and data visualization projects. R is a programming language specialized for data analysis, providin
g extensive libraries and packages for statistical modeling.

- C++: Suitable for projects requiring high-performance computing, system-level programming, or optimization. C++ is a low-level language known for its
speed and efficiency.



- SQL: Essential for projects involving relational databases, data querying, and manipulation. SQL is a programming language used to manage and retriev
e data from databases.

- Open CV: Valuable for computer vision projects, image processing, and video analysis. OpenCV is an open-source library that provides tools, algorithm
s, and functions for image and video processing.

- scikit-image: Useful for image processing and analysis projects. scikit-image is a Python library that offers a collection of algorithms and function
s for image manipulation, enhancement, and segmentation.

- Apache Airflow: Beneficial for projects involving workflow management, scheduling, and automation. Apache Airflow is a platform for programmatically
creating, scheduling, and monitoring workflows.

- Git and GitHub: Essential for version control and collaborative software development projects. Git is a distributed version control system, and GitHu
b is a web-based hosting service, allowing seamless collaboration and code management.
```

Demonstrating langchain Tools and Agents

Agents : The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hard coded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

Tools: are interfaces that an agent can use to interact with the world.

Here we define one custom tool to return job descriptions, other inbuilt tools which I used was llm-math and Wikipedia.

query: what are the technical and non technical skills?

```
from langchain_experimental.agents.agent_toolkits import create_python_agent
from langchain.agents import load_tools, initialize_agent
from langchain.agents import AgentType
from langchain.python import PythonREPL
from langchain.chat_models import ChatOpenAI
from langchain.agents import tool
```

```
@tool
def job_desription(text: str)-> str:
 """Returns job disriptions mentioned below, use this for any \
 questions related to knowing the job disription. \
 The input should always be an empty string, \
 and this function will always return a string containing job disriptions.\ """

 return "Job discriptions:\
 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algorithms to solve complex busi
 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan structure extraction and model development. Collaborates with product
```

```
tools = load_tools(["llm-math","wikipedia"], llm=turbo_llm)
```

```
agent= initialize_agent(
    tools+ [job_desription],
    turbo_llm,  #turbo_llm, qa_chain,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    handle_parsing_errors=True,
    verbose = True)
```

```
# print(result)
```

```
agent_template = """\
The following is the resume and query:

resume: {resume}

query: {query}
"""


prompt = ChatPromptTemplate.from_template(template=agent_template)
query_human = 'Skills: what are the technical and non technical skills? \Answer output them as a comma separated Python list.'
messages = prompt.format_messages(resume=resume_doc,
                                  query=query_human)
```

```
> Entering new AgentExecutor chain...
Thought: I need to extract the technical and non-technical skills from the resume.
Action:
```
{
  "action": "job_desription",
  "action_input": ""
}
```

Observation: Job discriptions: 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algor
ithms to solve complex business problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Py
thon, TensorFlow, and PyTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a co
llaborative approach and a proven ability to drive projects to completion. 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan
structure extraction and model development. Collaborates with product and research teams to enhance current products and enable new ones. Experienced w
ith massive datasets, 2D Deep Learning, and Computer Vision using PyTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML mode
ls transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:I have found the job descriptions for Machine Learning Engineer and Computer Vision Engineer. Now I can extract the technical and non-technical
skills from these descriptions.
Action:
```
{
  "action": "job_desription",
  "action_input": "Machine Learning Engineer"
}
```

Observation: Job discriptions: 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algor
ithms to solve complex business problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Py
thon, TensorFlow, and PyTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a co
llaborative approach and a proven ability to drive projects to completion. 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan
structure extraction and model development. Collaborates with product and research teams to enhance current products and enable new ones. Experienced w
ith massive datasets, 2D Deep Learning, and Computer Vision using PyTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML mode
ls transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:I have found the job description for Machine Learning Engineer. Now I can extract the technical and non-technical skills from this description.
Action:
```
{
  "action": "job_desription",
  "action_input": "Computer Vision Engineer"
}
```
```

```
Observation: Job discriptions: 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algor
ithms to solve complex business problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Py
thon, TensorFlow, and PyTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a co
llaborative approach and a proven ability to drive projects to completion. 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan
structure extraction and model development. Collaborates with product and research teams to enhance current products and enable new ones. Experienced w
ith massive datasets, 2D Deep Learning, and Computer Vision using PyTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML mode
ls transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:I have found the job description for Computer Vision Engineer. Now I can extract the technical and non-technical skills from this description.
Action:
```
{
  "action": "job_desription",
  "action_input": "Computer Vision Engineer"
}
```
Observation: Job discriptions: 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algor
ithms to solve complex business problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Py
thon, TensorFlow, and PyTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a co
llaborative approach and a proven ability to drive projects to completion. 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan
structure extraction and model development. Collaborates with product and research teams to enhance current products and enable new ones. Experienced w
ith massive datasets, 2D Deep Learning, and Computer Vision using PyTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML mode
ls transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:I have extracted the technical and non-technical skills from the job descriptions of Machine Learning Engineer and Computer Vision Engineer. No
w I can provide the answer to the query.

Final Answer: The technical and non-technical skills mentioned in the resume are:
Technical Skills: AWS (Amazon Web Services), Keras, TensorFlow, Numpy, Pandas, Matplotlib, scikit-learn, Apache PySpark, Open CV, scikit-image, Apache
Airflow, Git and GitHub.
Non-Technical Skills: MLOPS, Machine learning, Deep Learning/Computer Vision, NLP.

> Finished chain.
```

query: Give me the available job descriptions?

```
> Finished chain.
```

```
agent_template = """\
The following is the resume and query:

resume: {resume}

query: {query}
"""


prompt = ChatPromptTemplate.from_template(template=agent_template)
query_human = 'Give me the available job discriptions?'
messages = prompt.format_messages(resume=resume_doc,
                                  query=query_human)
```

```
result = agent(messages)
```

```
> Entering new AgentExecutor chain...
Could not parse LLM output: Thought: The user wants to know the available job descriptions.
Action: I will use the `job_desription` tool to get the job descriptions.

Observation: Invalid or incomplete response
Thought:I need to use the `job_desription` tool to get the job descriptions mentioned in the resume.
Action:
```
{
  "action": "job_desription",
  "action_input": ""
}
```
Observation: Job discriptions: 1)Machine learning Engineer:Machine Learning Engineer with expertise in designing and developing robust models and algor
ithms to solve complex business problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Py
thon, TensorFlow, and PyTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a co
llaborative approach and a proven ability to drive projects to completion. 2) Computer Vision Engineer:Computer Vision Engineer specializing in 3D scan
structure extraction and model development. Collaborates with product and research teams to enhance current products and enable new ones. Experienced w
ith massive datasets, 2D Deep Learning, and Computer Vision using PyTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML mode
ls transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:The available job descriptions mentioned in the resume are:

1) Machine Learning Engineer: Machine Learning Engineer with expertise in designing and developing robust models and algorithms to solve complex busine
ss problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Python, TensorFlow, and PyTorc
h. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a collaborative approach and a p
roven ability to drive projects to completion.

2) Computer Vision Engineer: Computer Vision Engineer specializing in 3D scan structure extraction and model development. Collaborates with product and
research teams to enhance current products and enable new ones. Experienced with massive datasets, 2D Deep Learning, and Computer Vision using PyTorch
and/or TensorFlow. Balances generalist and researcher roles, ensuring ML models transition into meaningful production. Works closely with product owner
s to deliver value efficiently to customers.

Final Answer: The available job descriptions mentioned in the resume are Machine Learning Engineer and Computer Vision Engineer.

> Finished chain.
```

Demonstrating Multi Prompt chain

Demonstrating use the RouterChain paradigm to create a chain that dynamically selects the prompt to use for a given input. Specifically use the MultiPromptChain to create a question-answering chain that selects the prompt which is most relevant for a given question, and then answers the question using that prompt.

Define 3 prompt template:

Job description:

```
“””You are good at matching available job description with resume.\
Steps:\
1.Retrieve job descriptions from given tool attached with agent \
2.Compare if resume can be selected based on any job description, if yes then return that specific job description
3.If no job description matches the return None
Here is a resume:
{input}"""
```

Portfolio :

```
"""
You are good at finding portfolio link from the given resume and return that link to the user.If link not found return None.

Here is a question:
{input}"""
```

Summary:

```
"""
You are good at summerising the given resume. You will include skills, professional experience, education in the summary.

Here is a question:
{input}"""
```

```
job_description_template = """
You are good at matching available job description with resume.\
Steps:\
1.Retreive job discriptions from givel tool attached with agent \
2.Compare if resume can be selected based on any job discription, if yes then retuen that specific job discription
3.If no job discription matches the return None

Here is a resume:
{input}"""


portfolio_finder_template = """

You are good at finding portfolio link from the given resume and return that link to the user.If link not found return None.

Here is a question:
{input}"""

summary_template = """
You are good at summerising the given resume. You will include skills, professional experience, education in the summary.

Here is a question:
{input}"""
```

```
prompt_infos = [
    {
        "name": "job_description",
        "description": "Good for providing job discription that is matched",
        "prompt_template": job_description_template
    },
    {
        "name": "portfolio",
        "description": "Good for returning portfolio link from resume",
        "prompt_template": portfolio_finder_template
    },
    {
        "name": "summary",
        "description": "Good for providing summary of resume",
        "prompt_template": summary_template
    }
]
```

Create destination chains:

```
from langchain.chains.router import MultiPromptChain
from langchain.chains.router.llm_router import LLMRouterChain,RouterOutputParser
from langchain.prompts import PromptTemplate
```

```
llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")
```

```
destination_chains = {}
for p_info in prompt_infos:
    name = p_info["name"]
    prompt_template = p_info["prompt_template"]
    prompt = ChatPromptTemplate.from_template(template=prompt_template)
    if name == "job_description":
        chain = agent
    elif name == "portfolio" :
        chain = LLMChain(llm=llm, prompt=prompt)
    else:
        chain = LLMChain(llm=llm, prompt=prompt)

    destination_chains[name] = chain

destinations = [f"{p['name']}: {p['description']}" for p in prompt_infos]
destinations_str = "\n".join(destinations)
```

```
destinations
```

```
['job_description: Good for providing job discription that is matched',
 'portfolio: Good for returning portfolio link from resume',
 'summary: Good for providing summary of resume']
```

```
destinations_str
```

```
'job_description: Good for providing job discription that is matched\nportfolio: Good for returning portfolio link from resume\nsummary: Good for provi
ding summary of resume'
```

Define multi prompt router prompt and multi prompt chain:

```
default_prompt = ChatPromptTemplate.from_template("{input}")
default_chain = LLMChain(llm=llm, prompt=default_prompt)
```

```
MULTI_PROMPT_ROUTER_TEMPLATE = """Given a raw text input to a \
language model select the model prompt best suited for the input. \
You will be given the names of the available prompts and a \
description of what the prompt is best suited for. \
You may also revise the original input if you think that revising\
it will ultimately lead to a better response from the language model.

<< FORMATTING >>
Return a markdown code snippet with a JSON object formatted to look like:
```json
{{{{
    "destination": string \ name of the prompt to use or "DEFAULT"
    "next_inputs": string \ a potentially modified version of the original input
}}}}
```

REMEMBER: "destination" MUST be one of the candidate prompt \
names specified below OR it can be "DEFAULT" if the input is not\
well suited for any of the candidate prompts.
REMEMBER: "next_inputs" can just be the original input \
if you don't think any modifications are needed.

<< CANDIDATE PROMPTS >>
{destinations}

<< INPUT >>
{{input}}

<< OUTPUT (remember to include the ```json)>>"""
```

```
router_template = MULTI_PROMPT_ROUTER_TEMPLATE.format(
    destinations=destinations_str
)
router_prompt = PromptTemplate(
    template=router_template,
    input_variables=["input"],
    output_parser=RouterOutputParser(),
)

router_chain = LLMRouterChain.from_llm(llm, router_prompt)
```

```
chain = MultiPromptChain(router_chain=router_chain,
                         destination_chains=destination_chains,
                         default_chain=default_chain, verbose=True
                        )
```

Let’s define a input prompt to for extracting portfolio link from resume:

```
review_template1 = """\
For the following text, extract the following information:

Portfolio link: Extract portfolio link from given document.

text: {text}
"""
```

```
from langchain.prompts import ChatPromptTemplate

prompt_template1 = ChatPromptTemplate.from_template(review_template1)
messages1 = prompt_template1.format_messages(text=resume_doc[:]) #resume_doc[300:] to test portfolio link :None
print(prompt_template)
```

```
You are good at summerising the given resume. You will include skills, professional experience, education in the summary.

Here is a question:
{input}
```

```
res=chain.run(messages1)
```

```
type(res)
```

```
str
```

```
print(res)
```

```
Portfolio link: https://naveenrajusg.github.io/Portfolio/
```

Let’s define a input prompt to give matching job_description, if nothing matches give None:
Output:

```
ensuring ML models transition into meaningful production. Works closely with product owners to deliver value efficiently to customers.
Thought:I have extracted the job descriptions from the given text. The job descriptions are:

1) Machine Learning Engineer: Machine Learning Engineer with expertise in designing and developing robust models and algorithms to solve complex bus
iness problems. Experienced in end-to-end machine learning pipelines, from data preprocessing to deployment. Proficient in Python, TensorFlow, and P
yTorch. Skilled in data preprocessing, feature engineering, and cloud platforms (AWS, Azure, GCP). Strong communicator with a collaborative approach
and a proven ability to drive projects to completion.

2) Computer Vision Engineer: Computer Vision Engineer specializing in 3D scan structure extraction and model development. Collaborates with product
and research teams to enhance current products and enable new ones. Experienced with massive datasets, 2D Deep Learning, and Computer Vision using P
yTorch and/or TensorFlow. Balances generalist and researcher roles, ensuring ML models transition into meaningful production. Works closely with pro
duct owners to deliver value efficiently to customers.

Final Answer: The matching job descriptions from the given text are Machine Learning Engineer and Computer Vision Engineer.

> Finished chain.
```

Not let’s create a response schema to extract portfolio link:

```
from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser
```

```
portfolio_link_schema = ResponseSchema(name="portfolio_link",
                                       description="Give portfolio link from the given resume and return that link to the user\
Answer output them as a comma separated Python list.")
```

```
response_schemas = [portfolio_link_schema]
```

```
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

```
format_instructions = output_parser.get_format_instructions()
```

```
print(format_instructions)
```

```
The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```json
{
        "portfolio_link": string  // Give portfolio link from the given resume and return that link to the userAnswer output them as a comma separated
Python list.
}
```
```

```
review_template1 = """\
For the following text, extract the following information:

Portfolio link: Extract portfolio link from given document.

text: {text}

{format_instructions}
"""
```

```
from langchain.prompts import ChatPromptTemplate

prompt_template1 = ChatPromptTemplate.from_template(review_template1)
messages1 = prompt_template1.format_messages(text=res,format_instructions=format_instructions)
```

```
print(messages1[0].content)
```

```
For the following text, extract the following information:

Portfolio link: Extract portfolio link from given document.

text: Portfolio link: https://naveenrajusg.github.io/Portfolio/

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```json
{
	"portfolio_link": string  // Give portfolio link from the given resume and return that link to the userAnswer output them as a comma separated Python list.
}
```
```

```
messages1
```

```
[HumanMessage(content='For the following text, extract the following information:\n\nPortfolio link: Extract portfolio link from given document.\n\ntext: Portfolio link: https://naveenrajusg.github.io/Portfolio/\n\nThe output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":\n\n```json\n{\n\t"portfolio_link": string  // Give portfolio link from the given resume and return that link to the userAnswer output them as a comma separated Python list.\n}\n```\n')]
```

```
result = turbo_llm_memory(messages1)
```

```
print(result.content)
```

```
```json
{
	"portfolio_link": "https://naveenrajusg.github.io/Portfolio/"
}
```
```

```
output_dict = output_parser.parse(result.content)
```

```
output_dict
```

```
{'portfolio_link': 'https://naveenrajusg.github.io/Portfolio/'}
```

```
link = output_dict.get('portfolio_link')
```

```
link
```

```
'https://naveenrajusg.github.io/Portfolio/'
```

Scrape the information from the https link (portfolio link) for performing QA:

```
from langchain.document_loaders import AsyncHtmlLoader
```

```
loader = AsyncHtmlLoader([link])
```

```
html = loader.load()
```

```
Fetching pages: 100%|#############################################################| 1/1 [00:00<00:00,  2.56it/s]
```

```
html
```

```
[Document(page_content='<!DOCTYPE html>\r\n<html lang="en">\r\n    <head><!-- Global site tag (gtag.js) - Google Analytics -->\r\n        <script async src="https://www.googletagmanager.com/gtag/js?id=G-JZWJ68MYX4"></script>\r\n        <script>\r\n          window.dataLayer = window.dataLayer || [];\r\n          function gtag(){dataLayer.push(arguments);}\r\n          gtag(\'js\', new Date());\r\n        \r\n          gtag(\'config\', \'G-JZWJ68MYX4\');\r\n        </script>\r\n        <!-- Google Tag Manager -->\r\n<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({\'gtm.start\':\r\n new Date().getTime(),event:\'gtm.js\'});var f=d.getElementsByTagName(s)[0],\r\n    j=d.createElement(s),dl=l!=\'dataLayer\'?\'&l=\'+l:\'\';j.async=true;j.src=\r\n    \'https://www.googletagmanager.com/gtm.js?id=\'+i+dl;f.parentNode.insertBefore(j,f);\r\n    })(window,document,\'script\',\'dataLayer\',\'GTM-P9ZS443\');</script>\r\n    <!-- End Google Tag Manager -->\r\n        <meta charset="utf-8" />\r\n        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />\r\n        <meta name="description" content="" />\r\n        <meta name="author" content="" />\r\n        <title>Portfolio - Naveen Raju S G</title>\r\n        <link rel="icon" type="image/x-icon" href="assets/img/favicon.ico" />\r\n        <!-- Font Awesome icons (free version)-->\r\n        <script src="https://use.fontawesome.com/releases/v5.13.0/js/all.js" crossorigin="anonymous"></script>\r\n        <!-- Google fonts-->\r\n        <link href="https://fonts.googleapis.com/css?family=Saira+Extra+Condensed:500,700" rel="stylesheet" type="text/css" />\r\n        <link href="https://fonts.googleapis.com/css?family=Muli:400,400i,800,800i" rel="stylesheet" type="text/css" />\r\n        <!-- Core theme CSS (includes Bootstrap)-->\r\n        <link href="css/styles.css" rel="stylesheet" />\r\n        <!-- Global site tag (gtag.js) - Google Analytics -->\r\n<script async src="https://www.googletagmanager.com/gtag/js?id=UA-189123490-1">\r\n</script>\r\n<script>\r\n  window.dataLayer = window.dataLayer || [];\r\n  function gtag(){dataLayer.push(arguments);}\r\n  gtag(\'js\', new Date());\r\n\r\n  gtag(\'config\', \'UA-189123490-1\');\r\n</script>\r\n    </head>\r\n    <body id="page-top" style="text-size-adjust: auto;">\r\n        <!-- Google Tag Manager (noscript) -->\r\n<noscript><iframe src="https://www.googletagmanager.com/ns.html?id=GTM-P9ZS443"\r\n    height="0" width="0" style="display:none;visibility:hidden">
```

```
from langchain_core.runnables import RunnablePassthrough
from langchain.schema import StrOutputParser
from langchain import hub

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=300)
splits = text_splitter.split_documents(html)

vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())
retriever = vectorstore.as_retriever()

prompt = hub.pull("rlm/rag-prompt")

def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

```
rag_chain.invoke("Extract project names related to  (LLM)large language models? from projects section")
```

```
'Fine-Tuning the FLAN T5 LLM Model for Enhanced Dialogue Summarization.'
```

```
rag_chain.invoke("Education?")
```

```
'The individual has a Master of Science degree in Artificial Intelligence from the Illinois Institute of Technology with a GPA of 3.9/4. They have expe
rtise in machine learning, generative AI, deep learning, convolutional neural networks, recurrent neural networks, and image processing. They also have
knowledge in data mining, computer vision, deep learning, natural language processing, and introduction to AI.'
```

Conclusion:

The aim of this project was to demonstrate how LLM can be made use of to achieve various functionalities using langchain and RAG (Retrieval-Augmented Generation).

Future work:

Develop an efficient and intricate architecture of LangChains and RAG, incorporating advanced logic's that encompass various evaluation criteria for candidates. This includes the extraction of information from diverse sources, such as LinkedIn, by analyzing recommendation sections. The goal is to create a sophisticated product with comprehensive capabilities.