

# **AI BASED SOCIAL SUMMARIZER**

A major project report submitted in partial fulfillment of the requirement  
for the award of degree of

**Bachelor of Technology**

in

**Computer Science & Engineering / Information Technology**

*Submitted by*

**Sumeet Birendra Singh (201214)**

*Under the guidance & supervision of*

**Dr. Deepak Gupta**



**Department of Computer Science & Engineering and  
Information Technology**

**Jaypee University of Information Technology, Wagnaghat,  
Solan - 173234 (India)**

# CERTIFICATE

This is to certify that the work which is being presented in the project report titled “AI Based Social Summarizer” in complete fulfilment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Wagnaghat is an authentic record of work carried out by “Sumeet Birendra Singh (201214)” during the period from August 2023 to May 2024 under the supervision of Dr. Deepak Gupta, Assistant Professor (SG) Department of Computer Science and Engineering, Jaypee University of Information Technology, Wagnaghat.

Sumeet Birendra Singh  
(201214)

The above statement made is correct to the best of my knowledge.

Dr. Deepak Gupta  
Assistant Professor (SG)  
Computer Science & Engineering and Information Technology  
Jaypee University of Information Technology, Wagnaghat

# CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled '**AI Based Social Summarizer**' in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of our own work carried out over a period from August 2023 to May 2023 under the supervision of **Dr. Deepak Gupta** (Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)

Sumeet Birendra Singh

201214

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)

Supervisor Name: Dr. Deepak Gupta

Designation: Assistant Professor (SG)

Department: Computer Science & Engineering and Information Technology

Dated:

# ACKNOWLEDGEMENT

Firstly, i express our heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I would like to express our sincere gratitude to our supervisor, **Dr. Deepak Gupta**, for his valuable guidance and support throughout the development of our project, titled "**AI Based Social Summarizer.**"

His expertise has played a great role in our progress so far. I am grateful for his willingness to review our work and provide feedback, we am confident that his guidance will lead us to the successful completion of our project. We am truly honored and thus i extend our heartfelt gratitude for his mentoring and contributions to my project.

I would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, i might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

No expression of appreciation is complete without recognition of the prayers, good wishes, advice and moral support of my affectionate parents, which helped me immensely to achieve my goal.

# TABLE OF CONTENTS

<b>LIST OF ABBREVIATIONS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>x</b>
<b>ABSTRACT</b>	<b>xi</b>
<b>1 INTRODUCTION</b> .....	<b>1</b>
1.1 Introduction .....	1
1.2 Problem Statement .....	2
1.3 Objectives .....	3
1.4 Significance and motivation of the project report .....	5
1.5 Organization of project report .....	6
<b>2 LITERATURE SURVEY</b> .....	<b>7</b>
2.1 Overview of relevant literature .....	7
2.2 Key gaps in the literature .....	16
<b>3 System Development</b> .....	<b>18</b>
3.1 Requirements and Analysis .....	18
3.2 Project Design and Architecture .....	20
3.3 Data Preparation .....	25
3.4 Implementation .....	26
3.5 Key Challenges .....	38
<b>4 Testing</b> .....	<b>40</b>
4.1 Testing Strategy .....	40

4.2 Test Cases and Outcomes .....	41
<b>5 Results and Evaluation .....</b>	<b>50</b>
5.1 Results .....	50
<b>6 Conclusions and Future Scope .....</b>	<b>54</b>
6.1 Conclusion .....	54
6.2 Future Scope .....	55
<b>REFERENCES .....</b>	<b>56</b>

# LIST OF ABBREVIATIONS

<b>S. No.</b>	<b>Short Form</b>	<b>Full Form</b>
1	AI	Artificial Intelligence
2	LLM	Large Language Model
3	GPT	Generative Pre-Training
4	RNN	Recurrent Neural Network
5	CNN	Convolutional Neural Network
6	UI	User Interface
7	BERT	Bidirectional Encoder Representations from transformers
8	NLU	Natural Language Understanding
9	LAMA	Language Model Analysis
10	CoT	Chain of Thoughts
11	HTML	Hyper Text Markup Language
12	CSS	Cascading Style Sheet
13	JSON	JavaScript Object Notation

# LIST OF FIGURES

<b>S. No.</b>	<b>Title</b>	<b>Page No.</b>
3.1	Representation of the project entities	19
3.2	Idea of LangChain	22
3.3	Flow chart	24
3.4	Representation of linkedin_lookup_agent.py file	27
3.5	Representation of custom_chains.py file	28
3.6	Representation of script in the style.css file	29
3.7	Continued representation of script in the style.css file	30
3.8	Representation of script in index.html file	31
3.9	Representation of linkedin.py file	32
3.10	Representation of tools.py file	33
3.11	Representation of app.py file	34
3.12	Representation of ice_breaker.py file	35
3.13	Continued representation of ice_breaker.py file	36
3.14	Representation of output_parsers.py file	37
4.1	Representation of frontend application	41
4.2	Representation of terminal while testing the ProxyCurl API	42
4.3	Representation of chains file	42



<b>S. No.</b>	<b>Title</b>	<b>Page No.</b>
4.4	LinkedIn profile from the URL produced by API	43
4.5	Representation of terminal while testing the ProxyCurl API for test case 2	43
4.6	LinkedIn profile from the URL produced by API for test case 2	44
4.7	Representation of test case 1 for OpenAI API testing	45
4.8	Representation of console for test case 1	45
4.9	Representation of frontend output for test case 1	46
4.10	Representation of test case 2 for OpenAI API testing	47
4.11	Representation of console for test case 2	48
4.12	Representation of frontend output for test case 2	49
5.1	Passing the LinkedIn profile user name	50
5.2	Representation of console	51
5.3	Representation of frontend output	52

# LIST OF TABLES

1.1	Literature review of research papers.....	15
-----	---	----

# ABSTRACT

It can be hard and tedious to keep up with important social media details from websites like LinkedIn in present times as there is a lot of information overload and profiles flooding all over the platform some of which seem to be irrelevant. Thus, the development of an effective social summarizer web application based on AI that integrates LinkedIn data compresses it into concise and meaningful abstracts, and communicates the data is needed to address this issue. This would provide customers with a handy means of getting updates on all essential information about some users globally in the internet world. The rise of generative AIs such as Generative Pre-Training 3 (GPT3) can be attributed to the development of Large Language Model (LLM) based applications. These models have been quite efficient in natural language processing and production applications though several applications of these LLMs have not yet been discovered or even thought of, be simple LLMs have a wider scope in varying domains which when applied can be of great use.

LLMs have been used for text summarizing and content extraction. Previous studies on social summarization have concentrated on the processing of data from LinkedIn networks respectively. Agents and scraping approaches have been used to collect data from several sources which was preprocessed later. Nevertheless, adding advanced AI features such as LLMs which most current summaries lack could greatly improve the quality of output summaries. There are several other applications of LLM in the AI domain that have been proven to revolutionize. Though the use of LLMs has been limited to certain fields it can be employed in several cases which are yet to be discovered. Also, LangChain is a powerful Python-based framework not known to everyone that provides the complete infrastructure, tools, and overall environment in general to develop certain projects. LLM coupled with the use of LangChain or related framework can do wonders in the Artificial Intelligence domain.

# CHAPTER 1: INTRODUCTION

## 1.1 INTRODUCTION

There are various social networking platforms such as LinkedIn, which is very important in professional activities. It is used for posting one's professional successes and networking with other individuals who share similar interests while also seeking employment opportunities. It is not simple as a lot of data is present there which nobody can tell is true or not. Therefore, new applications should be developed to use the strength of artificial intelligence to help tackle this issue of information overload.

The purpose of this project is to address the information overload challenges that exist in the social media environment using generative AI technology. Our web-based application will go through numerous critical phases, i will utilize the specially designed agents and APIs to gather data from LinkedIn. Data preprocessing will remove noise and useless information. Our application will majorly deal with the integration of LLM driven by LangChain. This will allow for precise, logical, and contextually relevant summaries of the accumulated data. Summaries will reflect the very specific key ideas of the original content.

I will develop an appropriate user interface, in which the users can state their specific requirements as well as view the summaries presented. I will use Flask to build our application and it will comprise of friendly interface for users and an integrated AI backend. For this to work, communication between components must be seamless.

I propose a solution that uses generative AI methods with effective data gathering and processing by creating this AI-based Social Summarizer. Our project would meet the growing demand for clear information. The application would aim to provide users with a tool that makes reading social media profiles an effective experience through the combination of technology and user-centric design. This web application would be able to provide the medium where the user could look for a brief description of some person primarily the summary generated would be from

his/her LinkedIn profile. The core of the AI-based Social Summarizer relies on the extraction of relevant information from LinkedIn profiles. To accomplish this, the project leverages the ProxyCurl API, a robust tool designed for secure and ethical data scraping from LinkedIn. This ensures compliance with LinkedIn's terms of service while allowing the project to access profile information seamlessly.

The project makes use of the ChatGPT3.5 Turbo, which is a strong language model that comprehends and produces natural language. Applying this model makes it possible for social summarizers powered with AI technology to extract informative write-ups of LinkedIn profiles. To ensure that the generated summaries capture the nuances and salient features of each profile, the LangChain framework is used for enhancement.

This AI-powered social summarizer will be seen as one of the uncommon approaches in the area of AI-driven social network analysis. The project can change the way people access and navigate through LinkedIn profiles by using LLM's advanced language models, ethically scraped data, and web applications connected to external APIs.

## **1.2 PROBLEM STATEMENT**

As the number of users on LinkedIn continues to grow exponentially, leading to uncountable profiles, getting to a certain significant profile becomes an increasingly challenging task. The vast volume of information load makes it difficult for the user to fetch the relevant details about a person they are looking for in the desired way. This problem is unaddressed due to the absence of intelligent tools that can distill and summarize necessary information from LinkedIn profiles. This major project deals with the lengthiness of LinkedIn profiles as it is one of the critical issues while going through the platform. Understanding many shades of every profile has become a tedious task for the majority of its users as the platform increasingly finds its use among professionals for networking and communication purposes. However, it is difficult to analyze traditional profiles using old means designed for such purposes as there exists a lot of information that can affect an individual's personality. Thus, this calls for a smart approach that

simplifies the generation of profile summaries allowing customers to comprehend required information within a small space.

The underlying idea of the AI-based social Summarizer is that it can redefine user engagement with LinkedIn profiles. Manual profile analysis is not feasible, and users are also unaware of key details hidden in long contents. Advanced language models and data scraping help tackle the limitations by giving users a quick but useful way of comprehending profiles.

The adoption of AI-based social summarizers is likely to produce positive outcomes including increased user efficiency, better understanding of profile information, and more precise starting points for communicating with others depending on the created summaries. The success of the project will be determined by users' input, the quality of summaries, and its success in everyday use.

## **1.3 OBJECTIVES**

### **1.3.1 CONTENT SUMMARIZATION:**

The main goal of the development of an AI-based Social Summarizer is building a strong system for generating content summaries taking into account the power of LLM. Textual information is vast on social media profiles, especially the sites like LinkedIn. Making an intelligent natural language system necessary. To understand the basis of each LinkedIn profile, OpenAI's ChatGPT3.5 Turbo has been used. This can get major data including career events, skills, schooling, and others to create brief but meaningful outlines.

Content summarization is made up of extracting significant parts of information, such that the system correctly presents every detail associated with each profile. LLM facilitates the contextual approach that understands that the profession is broad and varies in scope or field among different individuals. The system seeks to give relevant summaries of important issues as most people struggle to get meaning out of too much information available on these sites.

### **1.3.2 LANGCHAIN INTEGRATION:**

The project uses LangChain technology to provide better summary outputs. LangChain is essential for improving the validity, uniformity, and general soundness of language development. Whereas LLM like ChatGPT3.5 Turbo is good at producing text, LangChain serves as an additional level that provides a meaningful storyline in the generated summaries.

The major achievement contributed by LangChain is in generating summaries not only with accuracy but also with well-structured logic based on relationships among different parts of the texts. To tackle such an issue of separated summaries, a process for integrating these results into natural language has been provided. The duo of LLM and LangChain improves the understandability and fluency of the created content.

### **1.3.3 USER-FRIENDLY INTERFACE:**

The AI-based social summarizer works under advanced language processing and its interface is a web application developed using Flask. It should have a friendly interface that provides easy access and use, leading to the success of this endeavor and to make the summary tool feasible.

Thus, it acts as a link between content summary, integration with LangChain, data extraction, and users' needs. This makes it easy for users to enter the LinkedIn username and access the process of summary generation. It has a simple design that renders easy to follow the constructed summaries. The aim is to provide an effective UI coupled with several other features that give higher productivity at any time.

The summaries produced by this process are made accessible to users through a user-friendly, simple and easy to access interface where professionals, employers, and individuals can find relevant information about various LinkedIn profiles without much efforts. A user-friendly design improves on the practical aspects of the AI Based Social Summarizer that can be understood by different users having different skill levels.

## **1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK**

The project is significant due to its potential to revolutionize the way people access and extract information from LinkedIn profiles. Also, this addresses the issue by bringing in use the advanced technologies, including LLM and LangChain, to cater to the summarization process.

The system's ability to suggest conversation topics based on the summarization output improves its value even more. This feature would not only help initiating conversations but also enhance the overall user experience. Hence, the AI-Based Social Summarizer can be considered as a transformative tool, offering users a time-efficient and insightful means of getting through the struggles of social media profiles. The AI-Based Social Summarizer might prove significant for professionals, recruiters, and individuals seeking to optimize their interactions on LinkedIn. By employing OpenAI's ChatGPT3.5 Turbo, the project ensures that the generated summaries are not only accurate but also contextually relevant and desired.

The motivation behind the "AI-Based Social Summarizer" project is from the recognition of existing challenges in the realm of profile analysis on professional networking platforms. Traditional methods of manually sifting through extensive profiles not only consume valuable time but also risk missing out on crucial details. The motivation to develop an automated summarization tool is rooted in the desire to empower users with a more efficient and effective means of profile comprehension.

The scraping of data from LinkedIn via the ProxyCurl API adds another layer of motivation by addressing ethical considerations. The project is designed to adhere to LinkedIn's policies, ensuring that data extraction is conducted ethically and securely. This commitment to ethical practices aligns with the broader motivation to create a tool that not only meets technical excellence but also upholds ethical standards in data usage. Also, the motivation to develop and learn the advanced technologies in Artificial Intelligence like LLM based applications, learning about the LangChain framework and it's working along with third party APIs had driven the working on the project.



## 1.5 ORGANIZATION OF PROJECT REPORT

The report is organized as follows:

- Chapter 1 is all about the idea of the need of the proposed system and how it can help. In present times we deal with a lot of information overload on social networking platforms. Therefore, i describe a problem statement and how to deal with it with the help of LLM and I have also defined a set of objectives for this project.
- Chapter 2 outlines the existing related work in the field of LLMs and LangChain powered web applications in Artificial Intelligence. It further presents the outputs which i eventually compare and discuss in this report.
- Chapter 3 puts forward the system that is formulated to cater the summarization of LinkedIn profiles precisely and is designed to work to fetch the information about a user. This is where we cover the requirements, project design and implementation along with challenges faced.
- Chapter 4 is all about testing the system for the accuracy and precision of the generated summary of the desired profile discussing the test strategy, test cases and outcomes thus produced.
- Chapter 5 puts forward the analysis of the results in depth and also with content to existing work in the field.
- Finally, Chapter 6 presents the conclusion of the study. It also contains the application contribution with future scope of the project.

# CHAPTER 2: LITERATURE SURVEY

## 2.1 OVERVIEW OF RELEVANT LITERATURE

**M Konda et al. [1]**, demonstrated how to combine OpenAI and LangChain to construct a basic client-server application based on LLM dubbed "Answer Bot." A client-side API was made accessible for connecting to and invoking a Flask-based Python server. A basic user interface was constructed after the endpoint was tested with Postman. The tech stack featured the Open AI GPT 3.5 model (gpt-3.5-turbo), the LangChain framework, Python, Flask for server-side web server support, and the Streamlit framework for the client's user interface. created a server providing the API for talking with the server, designed in Streamlit, and successfully deployed LangChain. The client then makes a call to this API to obtain the results.

**Dash ICT et al. [2]**, posted another article that provides a detailed investigation of using LangChain and some of the trendiest subjects. The author also detailed how he established his chatbot, which he utilizes in conjunction with proprietary APIs to assist users and give insights from their web data. An AI based personal shop advisor was developed to aid consumers in reviewing their online store data and making advice on how to increase sales and where to commit more cash to improve average revenue per visitor. The Chatbot verified an endpoint depending on the user's response to the query using the app's API swagger file in order to obtain vital data from the app's backend for corrective feedback. It goes over LLM again, as well as its text production, translation, summarizing, answering queries, finishing papers, and language comprehension capabilities. It also offers free and paid models such as GPT-3 (Generative Pre-trained Transformer 3), Falcon LLM, and LLaMA. Using the Planner agent as a tool to improve planning and analysis of user inputs, it updated the prompt template while keeping the app's API in mind. The planner agent analyzes the yaml file, converts all endpoints into tools that the agent can use, generates a plan with all the APIs that must be called in order to provide the best response to a human question, calls these APIs in order to analyze the data, and returns the best response to the user.

**Keita et al. [3]**, illustrates how to design a system that can communicate with any PDF and image file. The entire chat system's workflow was documented from beginning to end. The user must first submit the document to be processed, which could be in image or PDF format. The second module was built up to recognize the file format and then start the appropriate content extraction process. The Data Splitter module was used to separate the document's information into smaller bits. Finally, the chunk converter translated the chunks into embeddings, which were then saved in the vector storage. In the last phase, each chunk held an answer to the user's query, and the results were provided to the user in the form of a JSON response. The approach employed was decided by whether the input document was an image or a PDF. To extract data from a specific type of file, utilized a LangChain library with a lot of separate components. A larger document, such as a research paper, may, on the other hand, comprise numerous sections. Chunk embeddings are created. A vector store was designed to identify the response to a given query concerning a defined collection of chunks that are most comparable to that query. It constructed a dictionary with two keys, the confidence level associated with the query's answer. When communicating with the PDF document, we had to specify the file's path as well as the question to which we wanted our model to answer. The model responded like a human in a matter of seconds. Depending on the length of the document, putting a human through the same process could take minutes or even hours.

**C. Greyling et al. [4]**, focused on utilizing LangChain to the HuggingFace inference API for a Q&A chatbot. It was then followed by a few real-world examples of how to use LangChain and HuggingFace to insert context into a debate utilizing a few-shot learning technique. offered a simple example of how to govern conversational context in an LLM-based chatbot. An LLM was employed in a generating process, and the LLM got the initial input. This initial query includes a description of the chatbot, the first human input, and the LLM response. To continue the conversation, a fresh additional block was delivered. The LLM's response is well-informed and based on the context of the blocks provided collectively. Simply, the conversational flow buffered each discussion turn. Furthermore, it suggested two approaches for resolving lengthy conversations, deleting the first part of the conversation history at specific moments by truncating the conversational history. This method is comparable to using rolling logs to limit the size of log files. The second way is to utilize LLMs to record the history of the debates as

they occur. Discusses on the ChainBufferMemory that is a form of memory that adds all previously sent text both input and output to the context that is provided with each user message. LangChain introduced three ways to context management: combination, summary, and buffering. The application code that uses the HuggingFace inference API was successfully executed.

**S. Talebi et al. [5]**, published research work on the use of LLMs. In this situation, three phases of dealing with LLMs were presented, as well as an introduction to them. It is described a language model that is smaller and more general than a huge language model. Not all squares are rectangles, just as not all rectangles are squares. All LLMs are language models, it is a relatively recent innovation in artificial intelligence and machine learning. ChatGPT was characterized as a chat interface that ran on the GPT-3 LLM (which has since been improved to either GPT-3.5 or GPT-4 as of this writing). It evaluated both qualitative and quantitative sorts of LLM depending on the number and type of metrics utilized or displayed.

**A. Biswas et al. [6]**, offered a chatbot that overcomes the shortcomings of typical chatbots. The most recent ChatGPT API version was used. OpenAI's GPT-3.5-Turbo big language model recognizes and produces natural language or code. It is one of the most capable and competitively priced devices in Open AI's current GPT3.5 series. The ConversationChain, which has a simple memory type that preserves all previous inputs and outputs and appends them to the passed context; and the memory made out of a buffer that can take n user interactions as a context summary, which can provide an overview of prior talks, were employed. Both can occasionally be present in the same memory at the same time. Front-end programming was utilized to create the chatbot utilizing an online DataButton platform that comprised an integrated code editor (IDE), a package plus configuration maintenance environment, and a real-time development viewing area (localhost). Because DataButton makes use of the Streamlit framework, the code was written in basic Streamlit syntax. This resulted in the successful development of a Memory Bot that can be further customized and expanded with one's own datasets. It can have genuine human-like discussions while still being aware of the talks and context.

**O. Topsakal et al. [7]**, stresses at the usage of LLMs in the rapid development of applications. This article focuses on the open-source software library LangChain. It begins with talking about revolution in AI highlighting ImageNet Challenge [8], [9] and implementation of reinforcement learning [10]. LLMs have gained popularity since they can handle a number of duties including as writing code, explaining things, writing essays, and troubleshooting. LLMs have been utilized by millions of individuals courtesy to OpenAI's ChatGPT. The core emphasis of the research is LangChain, which is aimed to speed up the development of specialized AI applications employing LLMs. LangChain is well-known in the field of artificial intelligence for its smooth integration with a varied variety of apps and data sources. The paper analyzes LangChain's core parts, such as its chains and components, which serve as adaptive, use-case-specific pipelines and modular abstractions, respectively. Using a variety of real-world examples, the study clarifies this framework's ability to accelerate the development of LLM-based applications. A "prompt" is an LLM's input. They are commonly generated dynamically in an LLM application and contain the user's input (question). LangChain provides a set of classes for building prompts by employing numerous customizable Prompt Templates. A prompt template is a repeatable way for constructing a prompt. The chain is typically made up of an LLM and a prompt. When an application requires a flexible chain of calls to LLMs and other tools that rely on user input, agents can be deployed. An agent selects the correct tool from a selection of tools to utilize for user input. The development of LLMs, such as OpenAI's ChatGPT, signifies a paradigm change in AI research with multiple applications. Because of its versatility to interface with a large number of data sources and applications, the open-source library LangChain is a helpful resource in the AI field. LangChain's modular architecture, which provides pipelines that can be customized for various use cases, speeds up the creation of LLM applications. This work contributes to the topic on LLM application development in an intent to motivate additional research into LangChain and comparable technologies.

**Wei et al. [11]**, looked at language models' ability to construct a logical chain of thought, or a sequence of brief words that resembles the type of reasoning a human may use in response to an inquiry. Experiments reveal that when pressed to induce a chain of thought, sufficiently big language models perform better on reasoning problems with flat scaling curves. This research explored the use of chain of thought prompting to improve the reasoning task performance of

language models. Before arriving at a solution, language models could construct a logical path of thought, comparable to how people do when faced with a multi-step reasoning difficulty. When ordinary few-shot urging is insufficient for a certain reasoning activity, chain of thought prompting is easy to utilize and increases overall performance. Importantly, the trial results hint to the likelihood that successful chain of thought prompting is an emergent attribute of model scale, which means that its benefits become apparent only at a sufficiently large model scale (about 100B parameters). The datasets investigated were the SingleOp [12], SingleEq [13], AddSub [14], ASDiv [15], MultiArith [16], and GSM8K [17]. Several complex left-to-right transformer language models with just decoders were utilized. The models were pre-trained using a combination of web publications, dialog data, and Wikipedia. Chain of thought prompting is a basic and often utilized method that was studied in this work to improve reasoning in language models. As shown by tests on symbolic, arithmetic, and commonsense reasoning, chain of thought processing is an emergent property of model size that permits sufficiently big language models to execute reasoning tasks that would otherwise have flat scaling curves. It is expected that increasing the spectrum of reasoning problems that language models can handle will inspire additional study into language-based reasoning methodologies. CoT prompts foster more thorough and logical reasoning. The model's performance variations are dictated by prompt quality.

**Xiao et al. [18]**, introduced P-Tuning, a revolutionary technique that blends discrete prompts with trainable continuous prompt embeddings. Natural language understanding (NLU) can be accomplished by using natural language patterns to prompt a pre-trained language model. However, preliminary research indicates that manual discrete prompts frequently result in unstable performance; for example, changing a single word in the prompt can result in a substantial reduction in performance. P-Tuning not only minimizes the time gap between discrete prompts to stabilize training, but it also enhances performance on a range of NLU tasks such as SuperGLUE and LAMA. It works effectively for adjusted or frozen language models in both fully-supervised and few-shot conditions. The datasets and tools used were LAMA-TREx (LAMA-34k and LAMA-29k) and AutoPrompt. P-tuning improves best knowledge probing outcomes. Furthermore, P-tuning outperforms prior discrete prompt searching algorithms as AutoPrompt [19] and LPAQA [20] on the same-size models. As this indicates,

distinct prompts may not be the ideal method. This work presented the P-Tuning technique, which combines discrete and continuous prompts. In both the few-shot and fully-supervised scenarios, P-tuning is useful with both tuned and frozen language models. It increases performance and stabilizes training for the adaptability of pre-trained language models. GPT-3 goes beyond pattern memorization to indicate linguistic understanding. Excellent performance on a number of language-related tasks.

**Liu et al. [21]**, presented a new work coupled with a discussion of Bidirectional Encoder Representations from Transformers (BERT). This paper illustrates that considerable performance benefits are attainable and models (Transformers) function best when matched with this method. It also indicated higher transferable representations and language comprehension. This enables for enhanced generalization and insights into contextual representations. It was utilized as a pre-trained language model to improve a range of natural language processing jobs. This study created a thorough framework for both extractive and abstractive models and proved how BERT may be productively applied to text summarization. Many intersections Transformer layers are built on top of the encoder to build the extractive model. A new fine-tuning schedule for abstractive summarization was presented in order to alleviate the mismatch between the pre-trained encoder and the non-pre-trained decoder. This is accomplished by deploying distinct optimizers for each. A two-staged fine-tuning technique was also presented, which could help to boost the quality of the summary supplied. Experiments were done utilizing three datasets, and the findings demonstrated that the model consistently provides cutting-edge outcomes in both extractive and abstractive settings [22]. This work shows how pretrained BERT may be employed in text summarization. A unique encoder at the document level was also introduced, as well as a basic framework for abstractive and extractive summarization. The focus of this research was on document encoding for summary. As a result, in terms of generating coherent summaries, the system proposed in this research beats earlier summary systems.

**A Radford et al. [23]**, revealed great improvements on a range of tasks, including textual entailment, question answering, semantic similarity assessment, and document classification. can be performed by discriminatively fine-tuning a language model on each particular task following generative pre-training on a variety of unlabeled text. Although huge unlabeled text corpora are routinely available, labeled data for these specific tasks is limited, limiting the performance of discriminatively trained models. In contrast to other methods, due to the application of task-aware input transformations during fine-tuning, successful transfer was achieved with minimal model architectural changes. The usefulness of this method has been proved over a wide range of natural language understanding criteria. Through pre-training on a vast corpus with extended portions of continuous text, model got considerable expertise and the capacity to handle long-term dependencies. These abilities were then successfully used to discriminative tasks such as text classification, entailment determination, question answering, and semantic similarity evaluation, enhancing the state of the art on 9 of the 12 datasets investigated.

**A. Vaswani et al. [24]**, presented the Transformer, a revolutionary neural network design that altered tasks involving natural language processing (NLP), was the first attempt that came near to bringing about a revolution. Vaswani et al. published it in 2017. Convolutional neural networks (CNNs) were extensively used prior to transformers and was utilized in NLP applications, but they had several drawbacks. The authors suggested a fully attention-based model that outperformed earlier methods while being easier to train and more parallelizable. The key idea behind the Transformer is the self-attention mechanism, which allows the model to determine the relative relevance of multiple phrases or tokens in a sequence while processing the sequence. Because it does not employ sequential processing, the Transformer beats RNNs and CNNs in capturing word dependencies. The Transformer architecture is made up of an encoder and a decoder. The encoder and decoder are built of multiple layers, each of which use position-wise feed-forward neural networks and a self-attention mechanism. Using the self-attention method, the model can pay attention to different words in the input sequence and recognize their associations. In order for the self-attention mechanism to work, the query, key, and value linear transformations of the input must be computed. Following these input projections into various subspaces, the model computes attention ratings for each query and key



pair. The output of the self-attention layer is obtained by adding the corresponding values, which are weighted depending on these scores. In addition to the self-attention technique, the Transformer features positional encoding to incorporate the word order in the sequence. For this, many datasets were applied. Hyperparameters were chosen following testing on the development set. When possible, the maximum output length during inference to input length + 50 was terminated as soon as practicable [25]. The Transformer may be learned significantly faster than systems based on recurrent or convolutional layers. This Transformer design surpassed the competition in terms of competitive performance, faster training times, and excellent efficacy in parallelization and capturing long-range links.

Table 1.1: Literature review of research papers

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
1.	Oguzhan Topsakal et.al., “Creating LLM Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast” [7]	All Sciences Proceedings (2023)	LLM based API for making an API call.	Insights into LangChain’s usage, fostering rapid application development using LangChain for LLM applications.	Security concerns in LLM application development.
2.	Jason Wei et.al., “Chain of Thought Prompting Elicits Reasoning in LLMs” [11]	Advances in Neural Information Processing Systems 35, NeurIPS (2022)	SingleOp, SingleEq, AddSub, ASDiv, MultiArith, GSM8K	CoT prompts lead to more in-depth and coherent reasoning. Model performance varies based on prompt quality.	Performance can still vary depending on prompt design. Limited exploration of prompt variations.
3	Xiao Liu et al., “GPT Understands, Too” [18]	AI Open (2021)	LAMA-TREx dataset (LAMA-34k and LAMA-29k) AutoPrompt	GPT-3 exhibits linguistic comprehension beyond pattern memorization. Strong performance in various linguistic tasks.	There were instances where GPT-3 generates incorrect or irrelevant responses
4.	Yang Liu et.al., “Text Summarization with Pre-Trained Encoders” [21]	Association for Computational Linguistics (2019)	Summarization datasets: CNN/DailyMail news highlights dataset,	Improved performance in terms of generating coherent summaries, surpassing previous summarization methods.	Challenges in achieving fluency and accuracy.
5.	Alec Radford et.al., “Improving Language Understanding by Generative Pretraining” [23]	OpenAI (2018)	Natural language inference: SNLI, MultiNLI, etc. Question Answering: RACE, Story Cloze Sentence similarity	Improved language understanding, transferable representations. Enables better generalization, insights into contextual representations.	Challenges in handling out-of-distribution inputs.
6.	Ashish Vaswani et.al., “Attention Is All You Need” [25]	Neural Information Processing Systems, NeurIPS (2017)	Various language datasets	Competitive performance, faster training times. Effective in capturing long-range dependencies, parallelization	Less effective for tasks requiring structured outputs.

## 2.2 KEY GAPS IN THE LITERATURE

While these papers/articles revolutionized the Artificial Intelligence domain there are some key gaps:

In A. Vaswani et al.'s work, the model used here faced difficulties when dealing with very long sentences or extensive variations. Training very big models can prove costly when it comes to computing resources and they still can prove to be unreasonable for any research or industrial application.

In A. Radford et al.'s work, it is significant to note that different evaluation metrics may be required for different tasks, and improvement of one parameter does not mean better performance. The scalability of the approach is not discussed. The study considers benchmark datasets used in research, its application in real life situation has not been fully addressed.

In Liu et al.'s work the LLM used BERT can be very demanding when it comes to computations. The proposed framework was not enough to address large datasets and for different domains. However, the paper did not discuss on how readable the generated summaries were.

In Liu, Xiao et al.'s paper there was no information provided about how prompt embeddings are created but have been discussed many times in the paper. The paper addresses briefly the quality of P-Tuning in few-shot situations in general but does not study its behavior in different contexts. The paper overlooks the possible influence of P-tuning on the transformed models.

In Jason Wei et al. the author talked about chain-of-thought prompts which may not offer a clear way to measure improvements numerically in terms of scores. Pre-training method and process can influence the output accuracy of a given model greatly. The description is not explicit for each dataset.

O. Topsakal et al.'s paper made no mention about LangChain's limitations and barriers in its usage but kept on talking about its positive aspects. Developer experience was not discussed while working on user information and adopting to challenges. The paper lacks a detailed look at other available frameworks or libraries offering similar services.

In A. Biswas et al.'s work there was no discussion on how the proposed architecture would deal with extended conversations. No discussion of its performance aspect with respect to the how it handles several users at the same time and keeping up with the conversation. Data security and privacy concerns where there as conversational agents were used.

In S. Talebi et al.'s work it was not clear how LLMs distinguish themselves from other language models. The details about LLMs were not specified by the article that how using ChatGPT will make an impact when one updates these models. No practical examples where LLMs such as GPT 3.5 was used.

# CHAPTER 3: SYSTEM DEVELOPMENT

## 3.1 REQUIREMENTS AND ANALYSIS

### 3.1.1 SOFTWARE REQUIREMENTS:

- PyCharm IDE: To cater the need for user friendly interface and environment for development and testing.
- ProxyCurl API: The system must be capable of securely and ethically scraping data from LinkedIn profiles using the ProxyCurl API.
- OpenAI's ChatGPT3.5 Turbo: Integration with OpenAI's ChatGPT3.5 Turbo is a critical component for generating accurate and contextually relevant summaries of LinkedIn profiles.
- LangChain: The LangChain technology must be integrated to enhance the consistency of the generated summaries. This involves ensuring a logical and structured flow of information in the summaries for improved user comprehension.
- Flask Web App: The development of a user-friendly web application using Flask is essential.

### 3.1.2 HARDWARE REQUIREMENTS:

There are no special requirements concerning hardware other than those needed to run all of the necessary software and the entire project itself.

### 3.1.3 TECHNICAL ANALYSIS:

The project brings together various modern AI technologies such as LLM using OpenAI's ChatGPT3.5 Turbo model and LangChain. These particular technologies have been chosen due to their ability to produce good quality and rich summaries with focus on natural language processing and understanding. Flask is perfectly suitable as a web application framework

according to the ease of use and the range of customizations. The flask program helps to integrate the backend processes easily and has user friendly interface.

### 3.1.4 LANGUAGE AND TOOL ANALYSIS:

Python is used as a main programming language due to its flexibility and vast libraries of web development, data scraping, natural language processing, etc. The frontend development is done using HTML and CSS creating an appealing and responsive user interface. Use of APIs such as ProxyCurl API, SerpAPI, and GPT 3.5 API is an extension made in this regard. These are used for data extraction and language processing in order to produce the final output. For our AI-Based Social Summarizer to be strong, flexible and efficient these technologies, programming languages, as well as other tools are chosen.

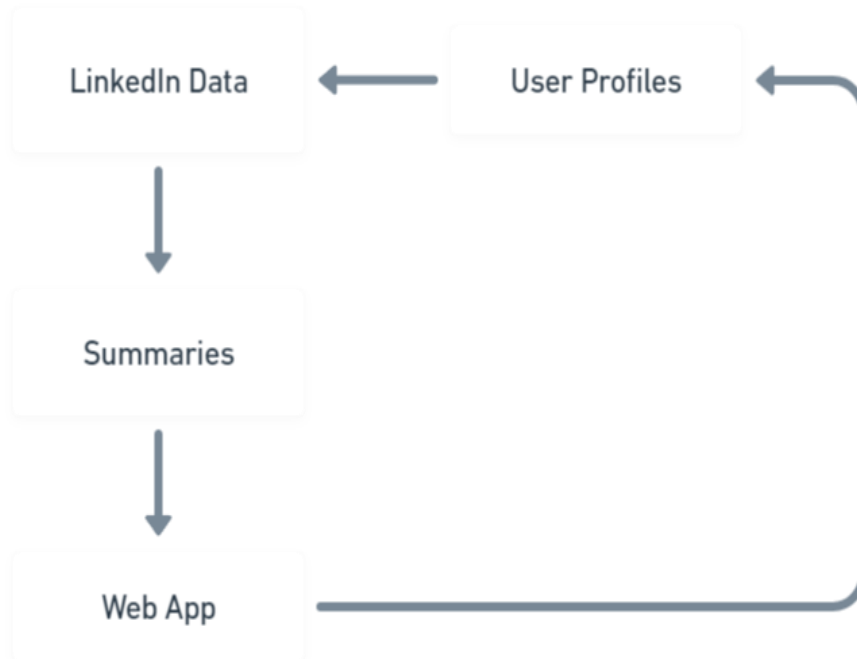


Figure 3.1: Representation of the project entities

## **3.2 PROJECT DESIGN AND ARCHITECTURE**

This section covers the design and architecture of the project, the development of “AI-Based Social Summarizer” is based on a holistic approach that ensures that different elements can work with each other to provide an excellent system. The following components play an important role in shaping the architecture.

### **3.2.1 DATA SCRAPING MODULE:**

The extraction of the information is done through the ethical way by using ProxyCurl API. Through ProxyCurl, LinkedIn’s security is assured and hence adheres to platform policies. This part provides the vital information including experiences, skills, and educational achievements on personal profiles for further processing.

### **3.2.2 LANGUAGE PROCESSING MODULE:**

The system employs ChatGPT3.5 Turbo and LangChain. ChatGPT3.5 Turbo performs content summarization by presenting main points of the scraped data in a precise form referring to the context. Integrating LangChain leads to better logicity of the summaries, making them more relevant for consumption.

### **3.2.3 WEB APPLICATION MODULE (FLASK):**

As for the user interface, it is a Flask-based web application that enables smooth interaction between the users and the system. The user interface is easy to use and requires users to simply enter in a LinkedIn profile username resulting in the data scraping and language processing algorithms. After that, the application showcases the created summaries and conversation starter (ice-breaking) points in an easy-to-read fashion.

### **3.2.4 SERP API:**

SerpAPI is an API to access Google search results on real time basis. It handles proxies, solves captchas, and parse all rich structured data for further use. In order to improve the produced summarization, the approach uses this for searching the most relevant data about the LinkedIn

pages. Furthermore, it helps to build the context of all the summaries providing the user with a broader view on one's web-presence.

### **3.2.5 INTEGRATION WITH LLM:**

OpenAI API provides variety of models with different capabilities, ChatGPT3.5 Turbo is one of them. It understands and generates natural language contexts. In regard to the project, the extracted data is sent to the API by the system, resulting in syntactically and semantically correct summaries. The output goes through the LangChain technology in order to make summaries more logical.

### **3.2.6 LANGCHAIN INTEGRATION:**

LangChain is a framework for developing applications using language models. It enables context aware applications and used to connect language models. This framework consists of several libraries and provides chains and agents the important entity in AI. The language processing pipeline incorporates this technology. Making sure that it considers the consecutive nature of the summary story and maintaining proper sequence to facilitate consistency by processing the GPT3.5 Turbo output. The presentation of this information is made structural as LangChain has been able to explain it within its context and this makes the summaries to be of good quality.



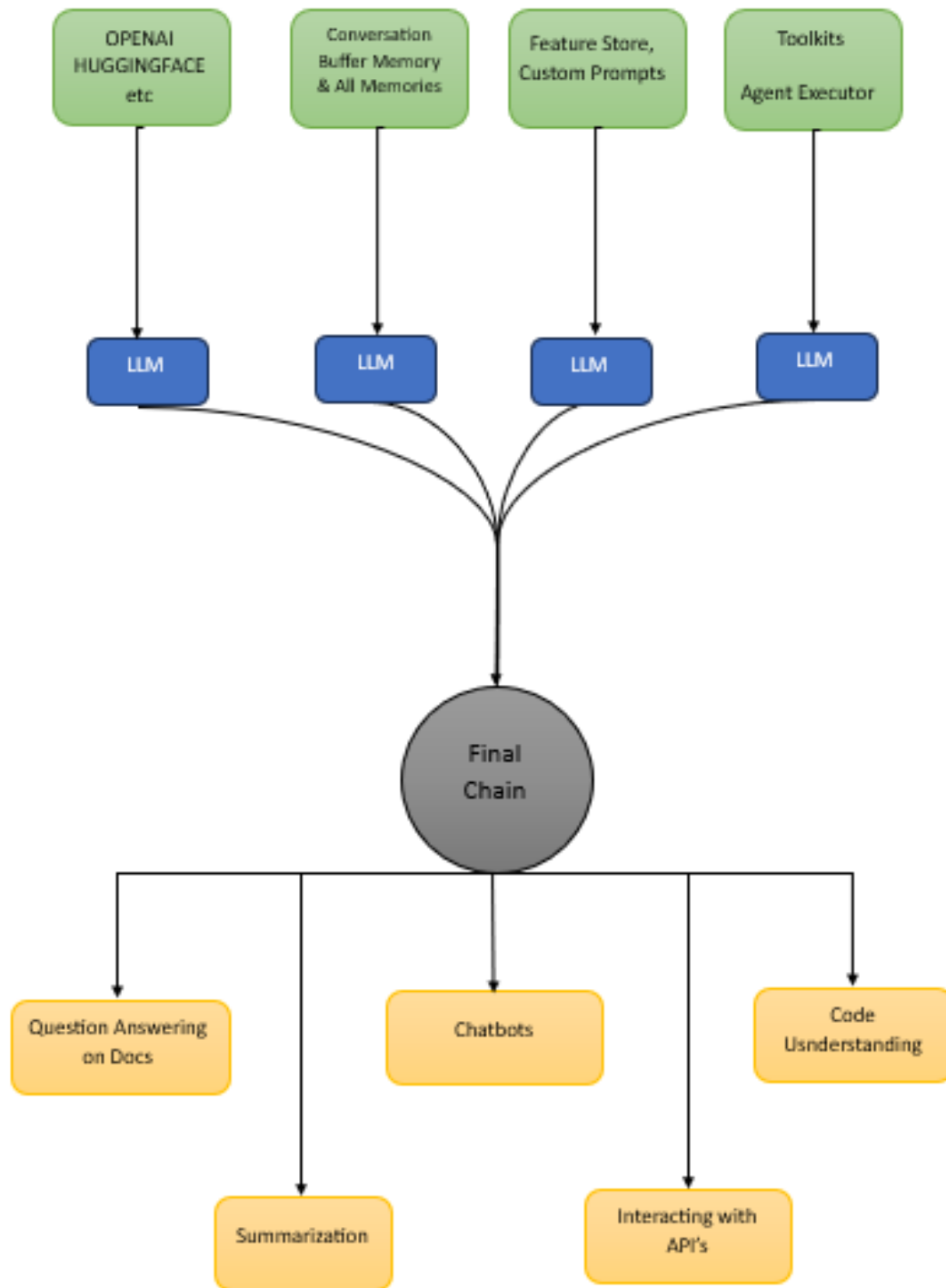


Figure 3.2: Idea of LangChain

### **3.2.7 FLASK WEB APPLICATION:**

Flask provides an interface for users to interact with the system. Once users enter the LinkedIn profile username, web scraping data extraction module gets triggered. Then the raw data will be processed thorough language processing module and finally presented to a customer through the web-interface. The application also gives users suggestions of what to chat about, following up on the summarized information.

### **3.2.8 DATA FLOW:**

The data flow within the system follows a sequential process:

- Via a Flask web interface, the user places an input of a LinkedIn profile URL.
- Data scraping module is triggered by the web application by use of ProxyCurl API to pull the desired data.
- Scraped data is related to a language processing component relying on OpenAI's ChatGPT3.5 Turbo for content recapitulation.
- Once the output is processed, it will guarantee cohesion and consistency for ease of comprehension and fluency in the flow.
- At the same time, SerpAPI obtains extra data from relevant Google queries about the LinkedIn page.
- A user is presented with an extensive summary of his/her profile that was supplemented with third party data as well as suggested discussion topics.

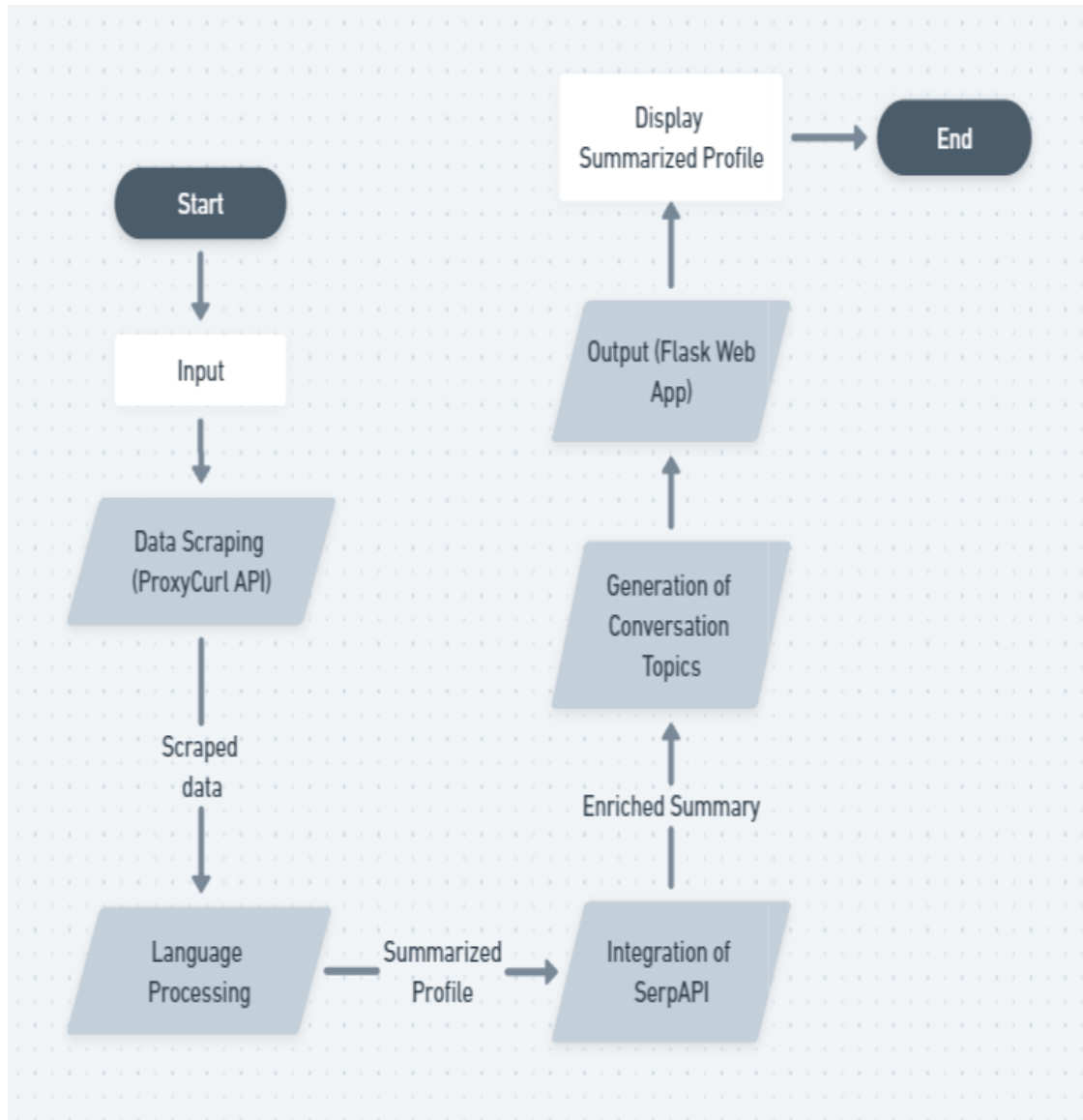


Figure 3.3: Flow chart

This interaction flow ensures an uninterrupted user experience, utilizing the power of each file to deliver accurate and relevant summaries of LinkedIn profiles within the Flask web application. The architecture is carefully designed to balance efficiency, security, and user-friendly approach, making the "AI-Based Social Summarizer" a powerful and accessible tool for professionals and users on LinkedIn.

### **3.3 DATA PREPARATION**

This section is all about the process of collecting and refining data from LinkedIn using ProxyCurl API and employing SerpAPI for integrating external information from Google searches.

#### **3.3.1 LINKEDIN DATA SCRAPING:**

For scraping data ethically from LinkedIn profiles, ProxyCurl API is used keeping in mind the policies of the platform. This involves extracting a range of information from the profile like full name, professional experiences, skills, educational background, and other relevant details. This extracts the information in JSON format which can further be implemented in some sort of application or can be accessed by simply printing the value. One API call costs one credit. This API provides the data ethically ensuring system's adherence to valid scraping practices and the data hence collected can be put to further use for developing applications such as the 'AI Based Social Summarizer'.

#### **3.3.2 SERPAPI:**

This API is basically used to scrape Google searches, here this is utilized to gather additional context from Google searches about the LinkedIn profiles. This helps in understanding an individual's existence online by providing supplementary information beyond the LinkedIn. This helps in improving the prepared summaries by scraping data from search results which eventually offers users a more knowledge about the individual's online activity and professional life. This also extracts the information in JSON format which can further be implemented in some sort of application or can be accessed by simply printing the value. And same as ProxyCurl one API call costs one credit.

The prepared data from these API is properly formatted for optimal utilization by LLM - ChatGPT 3.5 Turbo and LangChain framework for generating desired summaries. LangChain enhances the logical structure of the generated summaries whereas the ChatGPT 3.5 LLM is used to distill the required information.

## 3.4 IMPLEMENTATION

This section showcases the implementation done so far in regards to the project undertaken. The AI Based Social Summarizer is a Flask web application designed to analyze the LinkedIn profiles and generate concise summaries about an individual along with topics (Ice Breakers) on which the conversation can be started. The project is based on advanced AI technologies including LLMs and LangChain. This summarization and ice-breaking functionalities are in existence by OpenAI's GPT-3.5 Turbo API. And the data extraction is facilitated by ProxyCurl API while SerpAPI is used for scraping Google searches.

### 3.4.1 PROJECT CODE:

Set up a working directory in PyCharm IDE and imported and created desired files. The project is organized into several modules to ensure maintainability.

- Agents:

This consists of two files `init.py` that is an initialization file and `linkedin_lookup_agent.py` which has import statements importing functions and classes from various modules and also functions related to agents. The lookup function initializes a ChatGPT 3.5 Turbo model for language processing. PromptTemplate is defined specifying that the input variable would be the name of the person. A tool is defined specifying the `get_profile_url` function for scraping LinkedIn profile URLs. The agent is initialized and the agent type is specified as `ZERO_SHOT_LEARNING`. `agent_run` method is called to generate a response and finally LinkedIn username is returned by the lookup function.

```
linkedin_lookup_agent.py X
agents > linkedin_lookup_agent.py > ...
1 from tools.tools import get_profile_url
2
3 from langchain.prompts import PromptTemplate
4 from langchain.chat_models import ChatOpenAI
5
6 from langchain.agents import initialize_agent, Tool
7 from langchain.agents import AgentType
8
9
10 def lookup(name: str) -> str:
11     llm = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo", openai_api_key='sk-N94QIjBxS2KqW105JUHT3B1bkFJUczSrC3EsK5boKxt1FFs')
12
13     template = """given the full name {name_of_person} I want you to get it me a link to their LinkedIn profile page.
14                 Your answer should contain only a URL"""
15     tools_for_agent1 = [
16         Tool(
17             name="Crawl Google 4 linkedin profile page",
18             func=get_profile_url,
19             description="useful for when you need get the LinkedIn Page URL",
20         ),
21     ]
22
23     agent = initialize_agent(
24         tools_for_agent1, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True
25     )
26     prompt_template = PromptTemplate(
27         input_variables=["name_of_person"], template=template
28     )
29     linkedin_username = agent.run(prompt_template.format_prompt(name_of_person=name))
30
31     return linkedin_username
32
```

Figure 3.4: Representation of linkedin\_lookup\_agent.py file

- Chains:

This again consists of two files `init.py` that is an initialization file and `custom_chains.py` which defines the LangChain chains used for generating summaries and conversation topics. Several classes and functions are imported from the LangChain library. Two instances of `ChatOpenAI` are created and configured with different temperature settings; these are `llm` used for creating summaries and `llm_creative` for generating ice-breakers. The code defines three functions each returning an instance of LLM Chain. `get_summary_chain` is designed to generate a short summary and two interesting facts about a person based on LinkedIn information. `get_interests_chain` is designed to generate three topics of interest based on LinkedIn information.

```

custom_chains.py X
chains > custom_chains.py > get_ice_breaker_chain
1 from langchain import PromptTemplate
2 from langchain.chat_models import ChatOpenAI
3 from langchain.chains import LLMChain
4
5 from output_parsers import summary_parser, ice_breaker_parser, topics_of_interest_parser
6
7 llm = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo", openai_api_key='sk-N94QIjBXbS2KqW105JUht3B1bkFJUczSrC3EsK5boKxt1FFs')
8 llm_creative = ChatOpenAI(temperature=1, model_name="gpt-3.5-turbo", openai_api_key='sk-N94QIjBXbS2KqW105JUht3B1bkFJUczSrC3EsK5boKxt1FFs')
9
10
11 def get_summary_chain() -> LLMChain:
12     summary_template = """
13     | given the information about a person from linkedin {information} I want you to create:
14     | 1. a short summary
15     | 2. two interesting facts about them
16     | \n{format_instructions}
17     | """
18
19     summary_prompt_template = PromptTemplate(
20     | input_variables=["information"],
21     | template=summary_template,
22     | partial_variables={
23     |     "format_instructions": summary_parser.get_format_instructions()
24     | },
25     | )
26
27     return LLMChain(llm=llm, prompt=summary_prompt_template)
28
29
30 def get_interests_chain() -> LLMChain:
31     interesting_facts_template = """
32     | given the information about a person from linkedin {information}, I want you to create:
33     | 3 topics that might interest them
34     | \n{format_instructions}
35     | """
36
37     interesting_facts_prompt_template = PromptTemplate(
38     | input_variables=["information"],
39     | template=interesting_facts_template,
40     | partial_variables={
41     |     "format_instructions": topics_of_interest_parser.get_format_instructions()
42     | },
43     | )
44
45     return LLMChain(llm=llm, prompt=interesting_facts_prompt_template)
46
47
48 def get_ice_breaker_chain() -> LLMChain:
49     ice_breaker_template = """
50     | given the information about a person from linkedin {information}, I want you to create:
51     | 2 creative Ice breakers with them that are derived from their activity on LinkedIn, preferably on latest tweets
52     | \n{format_instructions}
53     | """
54
55     ice_breaker_prompt_template = PromptTemplate(
56     | input_variables=["information"],
57     | template=ice_breaker_template,
58     | partial_variables={
59     |     "format_instructions": ice_breaker_parser.get_format_instructions()
60     | },
61     | )
62
63     return LLMChain(llm=llm_creative, prompt=ice_breaker_prompt_template)
64

```

Figure 3.5: Representation of custom\_chains.py file

- Static:

It consists of CSS for styling the web application and an image used in the application.

```
static > css > style.css > ...
1  /* static/css/style.css */
2
3  * {
4      box-sizing: border-box;
5      margin: 0;
6      padding: 0;
7  }
8
9  body {
10     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
11     background-color: #f5f5f5;
12     color: #333;
13     line-height: 1.6;
14 }
15
16 .container {
17     width: 80%;
18     margin: 0 auto;
19     padding: 30px;
20     background-color: #ffffff;
21     box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
22     border-radius: 5px;
23     display: flex;
24     flex-direction: column;
25     align-items: center;
26 }
27
28 h1 {
29     font-size: 32px;
30     margin-bottom: 20px;
31 }
32 ul {
33     list-style-type: none;
34 }
35
36 input[type="text"] {
37     width: 100%;
38     padding: 12px 20px;
39     margin: 8px 0;
40     box-sizing: border-box;
41     border: 2px solid #ccc;
42     border-radius: 4px;
43     background-color: #f8f8f8;
44     font-size: 14px;
45 }
46
47 button {
48     background-color: #4caf50;
49     border: none;
50     color: white;
51     padding: 15px 32px;
52     text-align: center;
53     text-decoration: none;
54     display: inline-block;
55     font-size: 16px;
56     margin: 4px 2px;
57     cursor: pointer;
58     border-radius: 4px;
59     transition: 0.3s;
60 }
61
62 button:hover {
63     background-color: #45a049;
64 }
65
66 #result {
67     margin-top: 30px;
68     width: 100%;
69     text-align: center;
70 }
```

Figure 3.6: Representation of script in the style.css file



```

73 #loading {
74   display: none;
75   position: fixed;
76   top: 0;
77   right: 0;
78   bottom: 0;
79   left: 0;
80   z-index: 999;
81   background-color: rgba(255, 255, 255, 0.8);
82 }
83
84 .loader {
85   position: absolute;
86   top: 50%;
87   left: 50%;
88   transform: translate(-50%, -50%);
89   border: 8px solid #f3f3f3;
90   border-top: 8px solid #3498db;
91   border-radius: 50%;
92   width: 50px;
93   height: 50px;
94   animation: spin 2s linear infinite;
95 }
96
97 p {
98   font-size: 18px;
99   margin-bottom: 10px;
100 }
101
102 h2 {
103   font-size: 24px;
104   margin-bottom: 10px;
105   margin-top: 20px;
106 }
107
108 #profile-pic
109 {
110   width: 300px;
111 }
112 /* style.css */
113
114 body {
115   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
116   background-color: #f5f5f5;
117   color: #333;
118   line-height: 1.6;
119   background-image: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
120 }
121
122 .spinner-container {
123   position: fixed;
124   display: flex;
125   align-items: center;
126   justify-content: center;
127   top: 0;
128   right: 0;
129   bottom: 0;
130   left: 0;
131   z-index: 999;
132 }
133
134 #loading-spinner {
135   font-size: 48px;
136 }
137
138 button {
139   background-color: #007AFF;
140   font-weight: bold;
141   border: #000000 3px solid;
142   color: white;
143   padding: 15px 32px;
144   text-align: center;
145   text-decoration: none;
146   display: inline-block;
147   font-size: 16px;
148   margin: 4px 2px;
149   cursor: pointer;
150   border-radius: 35px;
151   transition: 0.3s;
152 }
153

```

Figure 3.7: Continued representation of script in the style.css file

- **Template:**

Contains index.html, the HTML template for rendering the web application interface and also includes the JavaScript for the code.

```
templates > index.html > ...
35 <script>
36 $(document).ready(function () {
37     $('#name-form').on('submit', function (e) {
38         e.preventDefault();
39         $('#spinner-container').show();
40         $.ajax({
41             url: '/process',
42             data: $('#name-form').serialize(),
43             type: 'POST',
44             success: function (response) {
45                 $('#profile-pic').attr('src', response.picture_url); // Update the profile picture src attribute
46                 $('#profile-pic').show(); // Show the profile picture
47                 $('#summary-and-facts').text(response.summary_and_facts.summary);
48                 $('#interests').html('<ul>' + response.summary_and_facts.facts.map(fact => '<li>' + fact + '</li>').join('') + '</ul>');
49                 $('#ice-breakers').html('<ul>' + response.ice_breakers.ice_breakers.map(ice_breaker => '<li>' + ice_breaker + '</li>').join('') + '</ul>');
50                 $('#topics-of-interest').html('<ul>' + response.interests.ice_breakers.map(topic => '<li>' + topic + '</li>').join('') + '</ul>');
51             },
52             error: function (error) {
53                 console.log(error);
54             },
55             complete: function () {
56                 $('#spinner-container').hide();
57             }
58         });
59     });
60 });
61 </script>
62 </body>
63 </html>
64
```

Figure 3.8: Representation of script in index.html file

- **Third parties:**

It comprises init.py, the initialization file and also linkedin.py which implements functionalities related to LinkedIn, including the interaction with the ProxyCurl API. scrape\_linked\_in function is responsible for scraping information from a LinkedIn profile. Function uses requests.get method to get requests to the specified api\_endpoint with parameters, including LinkedIn profile URL and authorization headers. The response is converted to JSON using response.json() and the processed data is returned.

```

third_parties > linkedin.py > ...
1  import os
2  import requests
3
4
5  def scrape_linkedin_profile(linkedin_profile_url: str):
6      """scrape information from LinkedIn profiles,
7      Manually scrape the information from the LinkedIn profile"""
8      api_endpoint = "https://nubela.co/proxycurl/api/v2/linkedin"
9      header_dic = {"Authorization": f'Bearer {os.environ.get("PROXYCURL_API_KEY")}'}
10
11     response = requests.get(
12         | api_endpoint, params={"url": linkedin_profile_url}, headers=header_dic
13         | )
14
15     data = response.json()
16     data = {
17         | k: v
18         | for k, v in data.items()
19         | if v not in ([], "", "", None)
20         | and k not in ["people_also_viewed", "certifications"]
21         | }
22     if data.get("groups"):
23         for group_dict in data.get("groups"):
24             | group_dict.pop("profile_pic_url")
25
26     return data
27

```

Figure 3.9: Representation of linkedin.py file

- Tools:

It also comprises init.py, the initialization file and also tools.py which contains utility functions used throughout the project. CustomSerpAPIWrapper class is initialized with the super call to the parent class SerpAPIWrapper. get\_profile\_url function searches for LinkedIn profile pages based on a given name. This handles interaction with SerpAPI for obtaining information related to LinkedIn profiles and relevant information.

```
tools.py ×
tools > tools.py > CustomSerpAPIWrapper > _process_response
1 from langchain.utilities import SerpAPIWrapper
2
3
4 class CustomSerpAPIWrapper(SerpAPIWrapper):
5     def __init__(self):
6         super(CustomSerpAPIWrapper, self).__init__()
7
8     @staticmethod
9     def _process_response(res: dict) -> str:
10        """Process response from SerpAPI."""
11        if "error" in res.keys():
12            raise ValueError(f"Got error from SerpAPI: {res['error']}")
13        if "answer_box" in res.keys() and "answer" in res["answer_box"].keys():
14            toret = res["answer_box"]["answer"]
15        elif "answer_box" in res.keys() and "snippet" in res["answer_box"].keys():
16            toret = res["answer_box"]["snippet"]
17        elif (
18            "answer_box" in res.keys()
19            and "snippet_highlighted_words" in res["answer_box"].keys()
20        ):
21            toret = res["answer_box"]["snippet_highlighted_words"][0]
22        elif (
23            "sports_results" in res.keys()
24            and "game_spotlight" in res["sports_results"].keys()
25        ):
26            toret = res["sports_results"]["game_spotlight"]
27        elif (
28            "knowledge_graph" in res.keys()
29            and "description" in res["knowledge_graph"].keys()
30        ):
31            toret = res["knowledge_graph"]["description"]
32        elif "snippet" in res["organic_results"][0].keys():
33            toret = res["organic_results"][0]["link"]
34
35        else:
36            toret = "No good search result found"
37        return toret
38
39
40 def get_profile_url(name: str):
41     """Searches for LinkedIn or twitter Profile Page."""
42     search = CustomSerpAPIWrapper()
43     res = search.run(f"{name}")
44     return res
45
```

Figure 3.10: Representation of tools.py file

- Venv- Library Roots:

This part of the code includes header files and python standard library. It also contains scripts for activating virtual environments.

- .env:

It is the file that contains configuration files like environment variables and other sensitive information like API keys.

- app.py:

This is the main application file responsible for handling web requests and integrating various components of the project. It starts by importing necessary modules from Flask and a custom ice\_breaker module. The process route handles POST requests made from the front end, calls the ice\_breaker\_with function and results in a JSON response. Also, this is the HTML file for the application and all the other files are integrated here.

```
app.py > ...
1  from flask import Flask, render_template, request, jsonify
2  from ice_breaker import ice_break_with
3
4  app = Flask(__name__)
5
6
7  @app.route("/")
8  def index():
9      return render_template("index.html")
10
11
12 @app.route("/process", methods=["POST"])
13 def process():
14     print('hello: '+request.form["name"])
15     name = request.form["name"]
16     summary_and_facts, interests, ice_breakers, profile_pic_url = ice_break_with(
17         name=name
18     )
19     return jsonify(
20         {
21             "summary_and_facts": summary_and_facts.to_dict(),
22             "interests": interests.to_dict(),
23             "ice_breakers": ice_breakers.to_dict(),
24             "picture_url": profile_pic_url,
25         }
26     )
27
28
29 if __name__ == "__main__":
30     app.run(debug=True, host='0.0.0.0', port=5000)
31
```

Figure 3.11: Representation of app.py file

- ice\_breaker.py:

This implements the ice-breaking functionality using GPT3.5 Turbo LLM from OpenAI API to generate conversation starter topics. Starts by importing required modules and functions like lookup from linkedin\_lookup\_agent and many more. ice\_break\_with function takes LinkedIn user's name as input, looks up their username and scrapes the data using scrape\_linkedin\_profile and returns a tuple containing the summary, icebreaker, topics of interest.

```
ice_breaker.py > ice_break_with
1  from typing import Tuple
2  from agents.linkedin_lookup_agent import lookup as linkedin_lookup_agent
3
4  from chains.custom_chains import (
5      get_summary_chain,
6      get_interests_chain,
7      get_ice_breaker_chain,
8  )
9  from third_parties.linkedin import scrape_linkedin_profile
10
11 from output_parsers import (
12     summary_parser,
13     topics_of_interest_parser,
14     ice_breaker_parser,
15     Summary,
16     IceBreaker,
17     TopicOfInterest,
18 )
19
20
21 def ice_break_with(name: str) -> Tuple[Summary, IceBreaker, TopicOfInterest, str]:
22     linkedin_username = linkedin_lookup_agent(name=name)
23
24     linkedin_data = scrape_linkedin_profile(linkedin_profile_url=linkedin_username)
25
26     summary_chain = get_summary_chain()
27
28     summary_and_facts = summary_chain.run(
29         information=linkedin_data
30     )
31     print(summary_and_facts)
32     summary_and_facts = summary_parser.parse(summary_and_facts)
33     print(summary_and_facts)
```

Figure 3.12: Representation of icebreaker.py file

```

36 | interests_chain = get_interests_chain()
37 | interests = interests_chain.run(information=linkedin_data)
38 | interests = topics_of_interest_parser.parse(interests)
39 |
40 | ice_breaker_chain = get_ice_breaker_chain()
41 | ice_breakers = ice_breaker_chain.run(
42 | | information=linkedin_data
43 | )
44 | ice_breakers = ice_breaker_parser.parse(ice_breakers)
45 |
46 | return (
47 | | summary_and_facts,
48 | | interests,
49 | | ice_breakers,
50 | | linkedin_data.get("profile_pic_url"),
51 | )
52 |
53 |
54 | if __name__ == "__main__":
55 | | pass

```

Figure 3.13: Continued representation of ice\_breaker.py file

- output\_parsers.py:

This part of the code formats the output generated by the summarizer for user-friendly presentation along with conversation starters. The code defines three Pydantic models (Summary, IceBreaker, TopicOfInterest) and output parsers for these models. to\_dict method converts model instances into a dictionary. Instances are created for each of the models that are summary which represents the summary and interesting facts, IceBreaker represents a list of icebreakers that could be used, TopicOfInterest represents the topics that might interest an individual.

```

output_parsers.py > TopicOfInterest
1  from typing import List
2
3  from langchain.output_parsers import PydanticOutputParser
4  from pydantic import BaseModel, Field
5
6
7  class Summary(BaseModel):
8      summary: str = Field(description="summary")
9      facts: List[str] = Field(description="interesting facts about them")
10
11     def to_dict(self):
12         return {"summary": self.summary, "facts": self.facts}
13
14
15     class IceBreaker(BaseModel):
16         ice_breakers: List[str] = Field(description="ice breaker list")
17
18         def to_dict(self):
19             return {"ice_breakers": self.ice_breakers}
20
21
22     class TopicOfInterest(BaseModel):
23         topics_of_interest: List[str] = Field(
24             description="topic that might interest the person"
25         )
26
27         def to_dict(self):
28             return {"topics_of_interest": self.topics_of_interest}
29
30
31     summary_parser = PydanticOutputParser(pydantic_object=Summary)
32     ice_breaker_parser = PydanticOutputParser(pydantic_object=IceBreaker)
33     topics_of_interest_parser = PydanticOutputParser(pydantic_object=IceBreaker)
34

```

Figure 3.14: Representation of output\_parsers.py file

The AI Based Social Summarizer project uses a technological stack consisting of web development using Flask, OpenAI's GPT3.5 Turbo model for AI capabilities, profile scraping of LinkedIn via ProxyCurl API and Google searches scraping by SerpAPI. The use of LangChain is crucial for generation of concise and rich valid summaries and ice-breaking topics.



### **3.4.2 ALGORITHM PROPOSED:**

- User accesses the web application through Flask based interface and inputs the LinkedIn username which needs to be summarized.
- The application triggers the LinkedIn data scraping module using ProxyCurl API data ethically.
- Extracting the relevant information such as experiences, skills and education etc.
- Google searches integration is performed by SerpAPI to gather additional information related to LinkedIn profiles.
- Content summarization is done by integrating ChatGPT-3.5 Turbo model of OpenAI API which enables the generation of coherent summaries of the LinkedIn profile.
- LangChain framework is integrated with the project code to enhance the logicity of the generated summaries.
- Output parsers are employed to parse the generated summaries and additional information which is further represented to the user with a clear and organized view of summarized LinkedIn profile and suggested conversation topics.

### **3.5 KEY CHALLENGES**

Below mentioned are the key challenges faced and their solution during the implementation of the project:

- Working in the field which is new and less discovered is always challenging, so was the case with the implementation of this project as there are not many resources available to go through as and when needed. For this, as the project started the next steps in the project got clear automatically.
- The first challenge in our implementation was ethical data scraping from LinkedIn, as while scraping data one needs to obey the guidelines specified by the platform, keeping in mind the privacy and related security concerns. To address this issue, ProxyCurl API was used for ethical data scraping without violating the policies of the platform.

- OpenAI recently updated their API accessing procedure billing that had to be done prior to using the API and it is no longer free, which was again challenging. To address this issue, billing of 5\$ was done to get access to the OpenAI API.
- Handling diverse search results produced by SerpAPI and its integration to the summarizing process of the project. To address this issue, A parsing mechanism was employed to extract relevant information from different search results providing more information about an individual.
- Learning about the LangChain framework was a tough task as it is vast and has plenty of functionalities and its integration into the project flooded the code with errors. To address this issue, resolving the errors caused due to LangChain integration was the only way, it was a long process; few of them still occur at times.
- Having no prior experience with Flask development was challenging the development of the user interface that needed to be intuitive and should be capable of presenting complex information in appealing format. This was managed anyhow and also the library root provided with the dedicated templates for HTML, hence the web application was carefully designed to present the summaries and other information in simple and organized manner.
- Managing all the external dependencies and third-party libraries was a tedious task leading to a lot of code errors and compatibility issues. To address this issue, online resources helped tackle many of the issues along with LangChain documentation.

# CHAPTER 4: TESTING

## 4.1 TESTING STRATEGY

- For testing purposes in each module, unit testing was performed by using the print statements in between lines of the code while debugging whenever encountered the errors. The above strategy made it clear which part of the code was responding and which was not.
- Except the above generic approach of testing to ensure seamless integration amongst different modules connection between them, the importing modules were being checked if the code was responding.
- Errors are the best indicators of a fault in the code. Thus, rectifying the errors was the main method adopted.
- Also had to fetch new API keys and integrate them with the rest of the code since the API keys, especially the ProxyCurl API, had a limited fixed number of credits which were mostly exhausted while running these tests.
- The above method and checking if the API key being used was valid and if the API is responding correctly served as a major component of testing, this is discussed with a test case in the following section.
- The user interface developed using Flask, HTML and CSS, underwent testing to ensure that it is responsive and accessible. Manual testing of the web application was performed.

Coupled with other uncommon developer-specific testing methods it was ensured that the project works fine enough and offers a smooth user experience.

## 4.2 TEST CASES AND OUTCOMES

### 4.2.1 TESTING THE PROXYCURL API:

On entering the username of an individual, we want to know about on the web application and upon clicking the 'Do Your Magic' button we test if the project is able to make a request to the ProxyCurl API.

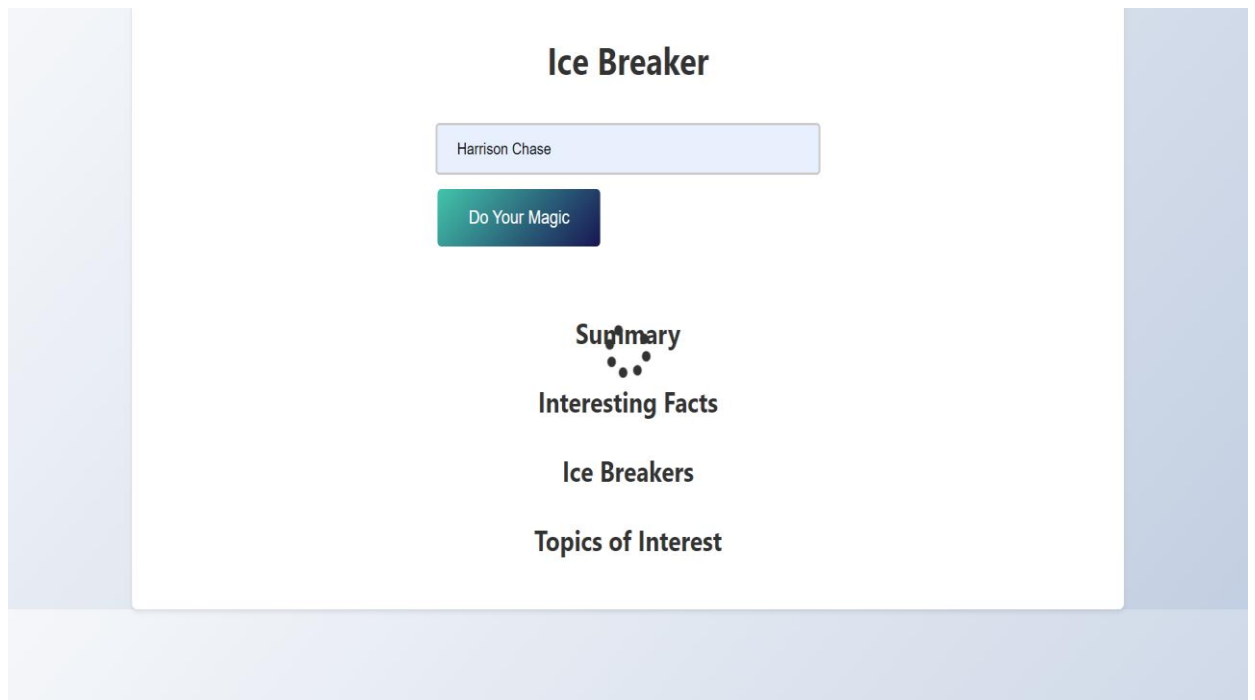


Figure 4.1: Representation of frontend application

For this we go back to the terminal and check if the call to the API was made or not.

```
hello: Harrison Chase

> Entering new AgentExecutor chain...
I need to search for the LinkedIn profile page of Harrison Chase.
Action: Crawl Google 4 linkedin profile page
Action Input: "Harrison Chase LinkedIn"
Observation: https://www.linkedin.com/in/harrison-chase-961287118
Thought: I have found the LinkedIn profile page for Harrison Chase.
Final Answer: https://www.linkedin.com/in/harrison-chase-961287118

> Finished chain.
{"properties": {"summary": {"title": "Summary", "description": "summary", "type": "str
```

Figure 4.2: Representation of terminal while testing the ProxyCurl API

As we can see from the figure that the API call was successfully made and the LinkedIn URL to the profile was provided. Upon clicking the link, we check if we found the actual profile we were looking for.

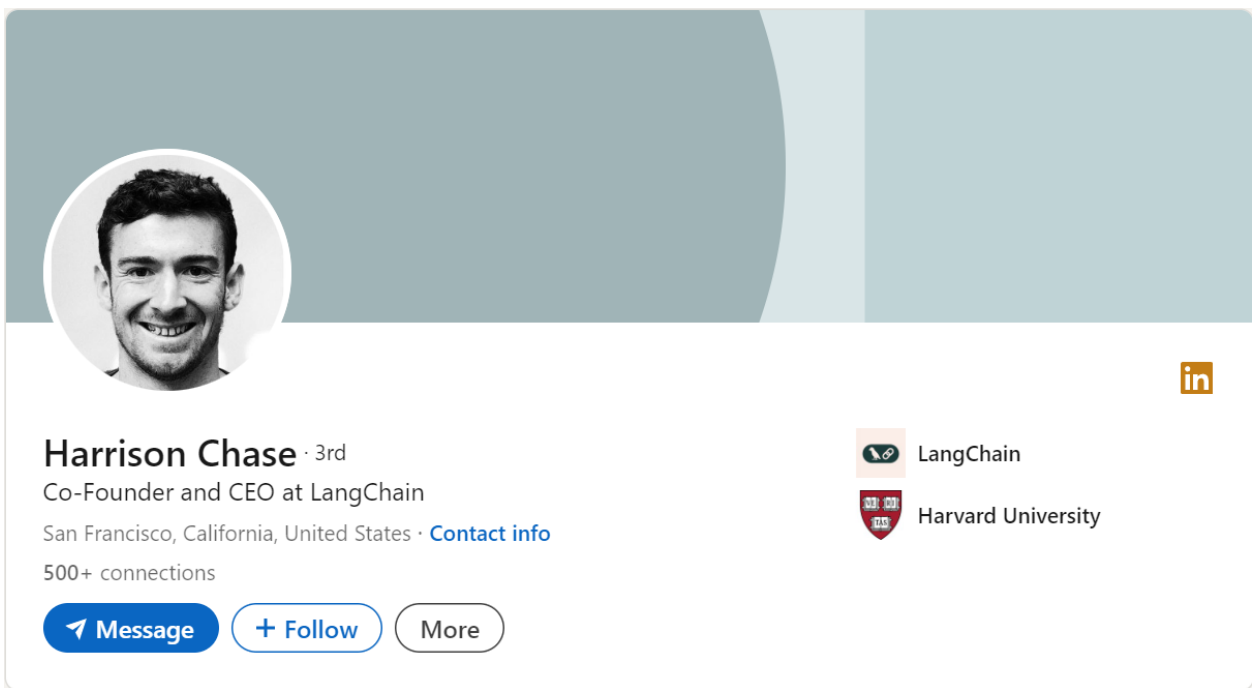


Figure 4.3: Representation of LinkedIn profile from the URL produced by API

As we can see that we got the profile we were looking for and hence it can now be summarized, and summary hence generated can be seen on the web application.

Let's run one more test for the same:

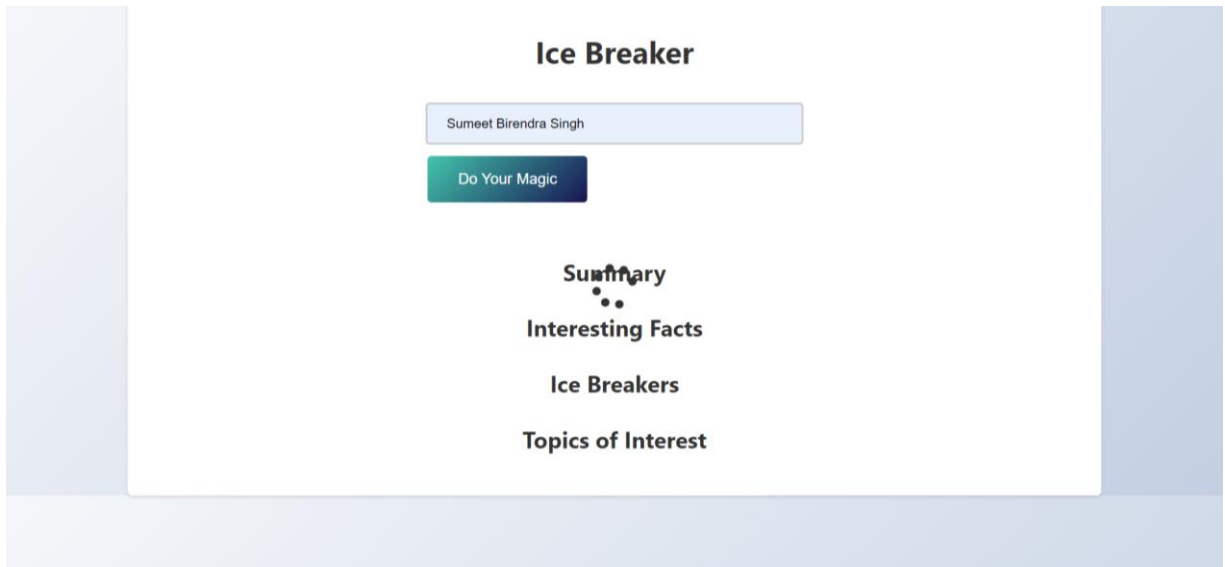


Figure 4.4: Representation of test case 2

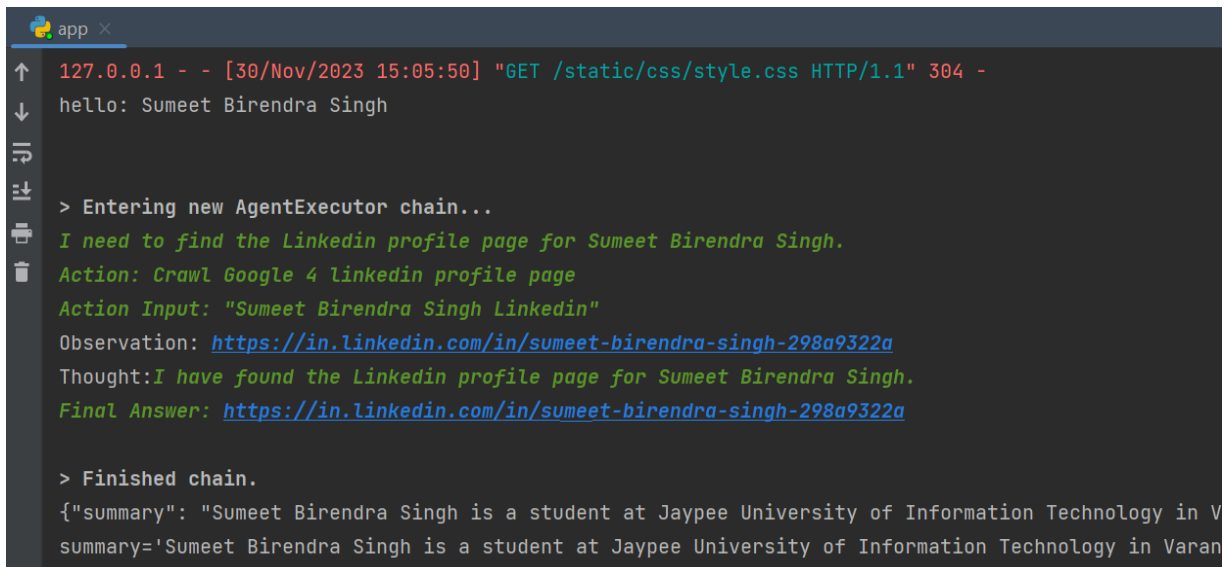


Figure 4.5: Representation of terminal while testing the ProxyCurl API for test case 2

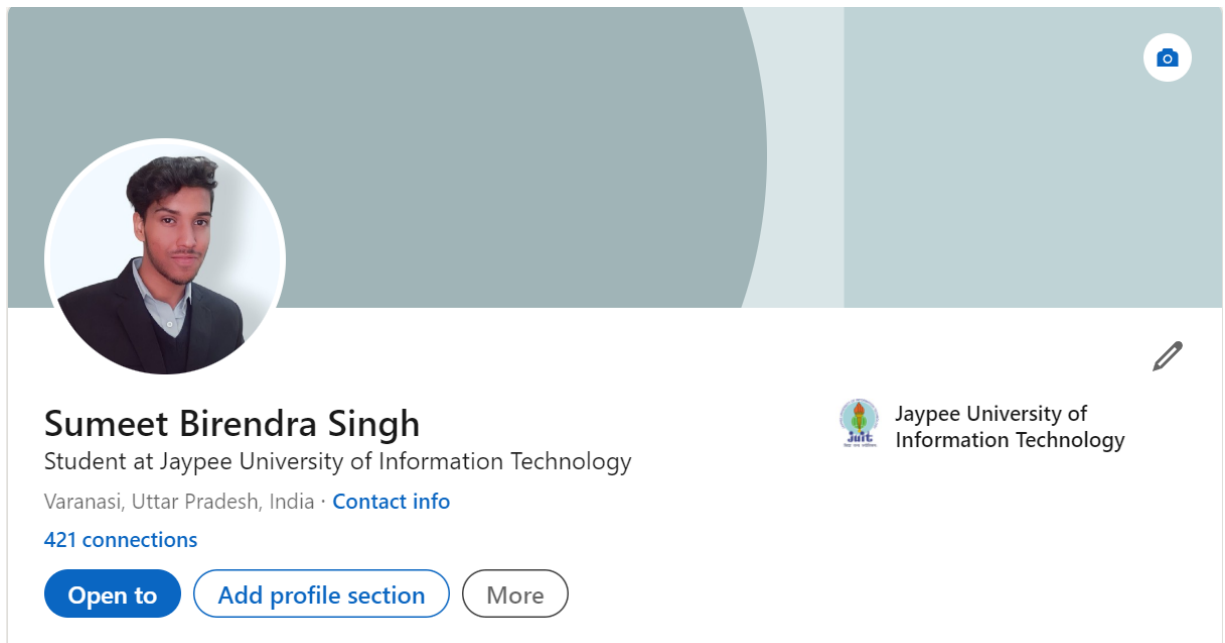


Figure 4.6: LinkedIn profile from the URL produced by API for test case 2

Thus, we can say that our project passes the test case of ProxyCurl API call. Similarly, the call for SerpAPI and OpenAI API showcased success too.

#### **4.2.2 TESTING THE OPENAI API:**

The ChatGPT3.5 Turbo LLM by OpenAI API could be tested by the same process of passing the input to the Flask front end. By providing the name of the LinkedIn user whose profile's summary we are interested in generating in the input section of the flask frontend, we saw in the previous section that the code implemented was able to fetch the LinkedIn profile of that particular user, thus in this section we tested whether the heart and soul of the whole project summaries were being generated in a desired way or not by the help of ChatGPT3.5 Turbo LLM.

## AI Based Social Summarizer

Summary  
Interesting Facts  
Ice Breakers  
Topics of Interest

Figure 4.7: Representation of test case 1 for OpenAI API testing

After entering the user name of the LinkedIn user we hit the 'Go' button lets see what actions does this trigger in the console:

```
Thought:I need to verify if this is the correct LinkedIn profile for Harrison Chase
Action: Crawl Google 4 linkedin profile page
Action Input: Harrison Chase LinkedIn profile
Observation: https://www.linkedin.com/in/harrison-chase-961287118
Thought:I have found the correct LinkedIn profile for Harrison Chase
Final Answer: https://www.linkedin.com/in/harrison-chase-961287118

> Finished chain.
{'Authorization': 'Bearer rwT9YG5M-0f0R9dktovPWg'}
{
  "summary": "Harrison Chase is the Co-Founder and CEO of LangChain, with a background in machine learning and software engineering.",
  "facts": [
    "Harrison has a keen ability to identify bottlenecks and prioritize next steps, leading to efficient project management.",
    "He is known for his strong Python programming skills, careful code reviewing, and knack for generating significant value."
  ]
}
summary='Harrison Chase is the Co-Founder and CEO of LangChain, with a background in machine learning and software engineering.'
127.0.0.1 - - [14/May/2024 01:02:50] "POST /process HTTP/1.1" 200 -
```


Figure 4.8: Representation of console for test case 1



Hence we can see the output in the console, what happens here is that on hitting the go button the processes in the background get triggered which in return by the help of the ProxyCurl API and Serp API fetch the required LinkedIn data related to that particular LinkedIn user along with ChatGPT 3.5 Turbo model which provides summarized and concise information about the same. Alone showing the result in the console is not enough we need the output on our Flask based frontend:

## AI Based Social Summarizer

GO !



### Summary

Harrison Chase is the Co-Founder and CEO of LangChain, with a background in machine learning, software engineering, and statistics.

### Interesting Facts

Harrison has a strong ability to prioritize next steps and isolate bottlenecks, making him a valuable asset in project management. He is known for his uncanny ability to generate significant model improvements by analyzing errors and thinking of features that may explain those differences.

### Ice Breakers

I saw you're a Co-Founder and CEO at LangChain, what inspired you to start this venture?  
I noticed you have a background in machine learning and software engineering, what advice do you have for someone looking to enter this field?

### Topics of Interest

- Sports analytics
- Machine learning applications in sports
- Software engineering best practices

Figure 4.9: Representation of frontend output for test case 1

Hence we can say that the frontend part is also working completely fine since the summary generated was presented on the frontend cleanly and concisely, easy to read with genuine information, and simple, easy-to-understand language providing valuable insights about the user's profile.

For better understanding let's test the code again for another LinkedIn user, the process would remain the same, on hitting the 'Go' button upon entering the user name we could see the output in the console for some time and frontend too. Let us look at the console part first:

## AI Based Social Summarizer

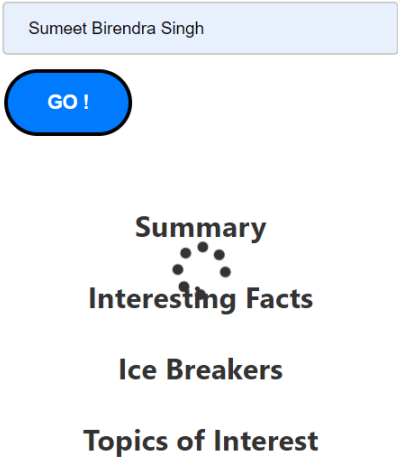


Figure 4.10: Representation of test case 2 for OpenAI API testing

```
I need to search for the LinkedIn profile of Sumeet Birendra Singh
Action: Crawl Google 4 linkedin profile page
Action Input: Sumeet Birendra Singh LinkedIn
Observation: https://in.linkedin.com/in/sumeet-birendra-singh-298a9322a
Thought:I have found the LinkedIn profile for Sumeet Birendra Singh
Final Answer: https://in.linkedin.com/in/sumeet-birendra-singh-298a9322a

> Finished chain.
{'Authorization': 'Bearer rwT9YG5M-0f0R9dktovPWg'}
{
  "summary": "Sumeet Birendra Singh is a student at Jaypee University of Information Technology in Varanasi, Uttar Pradesh, India,
  "facts": [
    "Sumeet Birendra Singh has 418 followers on LinkedIn.",
    "Sumeet Birendra Singh started studying at Jaypee University of Information Technology in January 2020 and is expected to gra
  ]
}
summary='Sumeet Birendra Singh is a student at Jaypee University of Information Technology in Varanasi, Uttar Pradesh, India, studyin
127.0.0.1 - - [14/May/2024 01:07:40] "POST /process HTTP/1.1" 200 -
```

Figure 4.11: Representation of console for test case 2

So now we know what information will be shown on our front end, the summarized information generated by applying the ability of ChatGPT 3.5 Turbo model which would present the required data in a very readable and user-friendly manner. The project code has been tested on numerous such test cases and hence can be considered to perform well enough as expected by generating a crisp and clear profile summary.

Let us take a look at our frontend part for this test case:

## AI Based Social Summarizer

Sumeet Birendra Singh

GO !



### Summary

Sumeet Birendra Singh is a student at Jaypee University of Information Technology in India and has a follower count of 418 on LinkedIn

### Interesting Facts

Sumeet is pursuing a Bachelor of Technology in Computer Science.  
Sumeet is from Varanasi, Uttar Pradesh.

### Ice Breakers

Ask Sumeet about their experience studying Computer Science at Jaypee University of Information Technology  
Inquire about some of Sumeet's favorite activities and societies during their time as a student at Jaypee University

### Topics of Interest

The future of Artificial Intelligence and Machine Learning  
Digital transformation and innovation in the education sector  
Latest trends in the world of technology and computer science

Figure 4.12: Representation of frontend output for test case 2

# CHAPTER 5: RESULTS AND EVALUATION

## 5.1 RESULTS

The result of this project is that I was successfully able to build a simple user-friendly Flask web application that takes the LinkedIn username of an individual as input and generates the summary of his/her or some other LinkedIn user's profile.

Let us finally test the project code for a LinkedIn user to evaluate the final result. This starts with entering the profile name of that LinkedIn user in the input field of the Flask-based front end:

### AI Based Social Summarizer

Summary  
Interesting Facts  
Ice Breakers  
Topics of Interest

Figure 5.1: Passing the LinkedIn profile user name

Now let's check the console to see if the username we provided is being processed or not this can only be visible and accessible to the developer side as once this gets deployed or in for production services, the background processes will get abstracted, and only the frontend would be visible. But still, to evaluate the result let's look into the console:

```
I should use the tool to crawl Google for the LinkedIn profile page of Raj Vikramaditya.
Action: Crawl Google 4 linkedin profile page
Action Input: Raj Vikramaditya LinkedIn profile
Observation: https://in.linkedin.com/in/rajstriver
Thought: I now know the final answer
Final Answer: https://in.linkedin.com/in/rajstriver

> Finished chain.
{'Authorization': 'Bearer rwT9YG5M-0f0R9dktovPWg'}
{
  "summary": "Raj Vikramaditya is a Software Engineer III at Google based in Warsaw, Poland. He is also a YouTuber with over 400K subscribers.",
  "facts": [
    "Raj has won the 'Geek of the Year' award from GeeksforGeeks and stood second in the IEST Shibpur Tech Fest Coding event.",
    "Raj has achieved impressive rankings in coding competitions such as securing 67th rank in ACM ICPC Kanpur regionals and 127th rank in ACM ICPC Noida regionals."
  ]
}
summary='Raj Vikramaditya is a Software Engineer III at Google based in Warsaw, Poland. He is also a YouTuber with over 400K subscribers.'
127.0.0.1 - - [14/May/2024 01:10:58] "POST /process HTTP/1.1" 200 -
```

Figure 5.2: Representation of console

It is evident that by incorporating the ChatGPT 3.5 Turbo model and Langchain, a summary, as well as an icebreaker topic, is being generated. To have a clear picture of the summary generated let us get back to the Flask-based frontend where the final output would be displayed.

## AI Based Social Summarizer

GO !



### Summary

Raj Vikramaditya is a Software Engineer III at Google based in Warsaw, Poland. He is also a YouTuber with over 400K subscribers and has a background in software development and education.

### Interesting Facts

Raj has won the 'Geek of the Year' award from GeeksforGeeks and stood second in the IEST Shibpur Tech Fest Coding event. Raj has achieved impressive rankings in coding competitions such as securing 67th rank in ACM ICPC Kanpur regionals and an AIR of 16 in Codevita.

### Ice Breakers

Hey Raj, I see you're a Software Engineer at Google and a YouTuber with over 400K subscribers. What inspired you to start your YouTube channel?  
Hi Raj, I noticed you were awarded 'Geek of the Year' from GeeksforGeeks. That's incredible! How did you feel when you received that honor?

### Topics of Interest

Latest trends in software engineering  
YouTube content creation strategies  
Coding competition tips and tricks

Figure 5.3: Representation of frontend output

The findings are as follows, outlining the key outcomes and their interpretation:

LinkedIn profile summaries generated were accurate, and the core functionality on which the project is based on content summarization using OpenAI's ChatGPT3.5 Turbo and LangChain proved to be favorable. The generated summaries filtered the relevant details from LinkedIn

profiles including interesting facts about an individual. LLM combined with LangChain ensures a clear and concise overview of certain profiles.

Integration of SerpAPI to scrape data from Google searches enriched the summaries by providing additional information related to LinkedIn profiles. The project adheres to the policies of LinkedIn and respects the privacy of LinkedIn users as the API applied to scrape data is purely ethical.



# CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE

## 6.1 CONCLUSION

The AI Based Social Summarizer proved to be successful and is a significant approach in AI driven social media analysis. The main findings of the project are about its ability to generate accurate profile summaries using OpenAI's ChatGPT3.5 Turbo and LangChain. Integration of these technologies resulted in coherent and relevant summaries as well as provided topics on which conversation can be started with an individual on the basis of his/her profile.

Despite the pros there are few limitations to our project. The user interface can be more appealing and attractive, also other features can be provided to the web application which the project lacks. Also, the quality of summary generated depends highly on the data availability on LinkedIn. Project is highly dependent on third party APIs which are expensive and one cannot always rely on third party resources. Moreover, scraping data from every networking platform is not possible as each platform has its own policies regarding data security and user's privacy.

The information displayed as an output on the web application was well organized and accurate, similar to the natural way of writing as a result, users can quickly understand the profile details. Moreover, the Flask web application is quite simple and user-friendly thus easy to access.

The system hence designed shows readiness for cloud deployment as it is adaptable to future growth and demand. The positive outcomes prove the effectiveness of chosen technologies and algorithms. To conclude, the 'AI Based Social Summarizer' project enhanced our learning about AI, LLM, particularly and working with them added up to our knowledge and experience. And this can prove as a promising tool for professionals, recruiters and any individual seeking for brief analysis on social networking platforms.

## 6.2 FUTURE SCOPE

The ‘AI Based Social Summarizer’ project lays a foundation for further development in the AI driven social media analysis domain. The project has abundant future scope.

- It can be aligned with other emerging technologies and can be improved continuously by training as it employs LLM, keeping it up to date with evolving patterns in the field.
- The system can be introduced with multiple languages support and translation of the information.
- The web application can be improved a lot by designing the front end more appealing and providing features like switching modes (dark and light) or by providing user specific customized summaries
- The system can be integrated with other social media platforms like LinkedIn, scraping the data from them and producing rich summaries.
- Introducing the real time data updates which can be reflected while a profile is being analyzed.

There is a large scope of improvement in this proposed system which can enhance the overall project eventually.

## REFERENCES

- [1] Konda, "Developing LLM-based Applications: Client-Server Answer Bot," Medium, <https://mkonda007.medium.com/developing-llm-based-applications-client-server-answer-bot-aab54469879d>, 2023.
  
- [2] Dash ICT, "Build Chatbot with LLMs and LangChain," Medium, <https://medium.com/@dash.ps/build-chatbot-with-llms-and-langchain-9cf610a156ff>, 2023.
  
- [3] Keita, "How to Chat With Any PDFs and Image Files Using Large Language Models," Towards Data Science, 2023.
  
- [4] Greyling, "Using LangChain To Create Large Language Model (LLM) Applications Via HuggingFace," Medium, <https://cobusgreyling.medium.com/langchain-creating-large-language-model-llm-applications-via-huggingface-192423883a74>, 2023.
  
- [5] Talebi, "A Practical Introduction to LLMs," Towards Data Science, <https://towardsdatascience.com/a-practical-introduction-to-llms-65194dda1148>, 2023.
  
- [6] Biswas, "How to build a Chatbot with ChatGPT API and a Conversational Memory in Python," Medium, 2023.
  
- [7] Topsakal et al., "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast," All Sciences Proceedings, 2023.
  
- [8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng, Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 2015.
  
- [9] Krizhevsky, Alex & Sutskever, et al. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

- [10] Sean D. Holcomb, William K. Porter, et al. 2018. Overview on DeepMind and Its AlphaGo Zero AI. In Proceedings of the 2018 International Conference on Big Data and Education (ICBDE '18). Association for Computing Machinery, New York, NY, USA, 67–71. <https://doi.org/10.1145/3206157.3206174>
- [11] Wei et al., "Chain of Thought Prompting Elicits Reasoning in Large Language Models," International Conference on Neural Information Processing Systems (NeurIPS), 2022.
- [12] Roy, S., Vieira, T., and Roth, D. Reasoning about Quantities in Natural Language. Transactions of the Association for Computational Linguistics, 2015. doi: 10.1162/tacl\_a\_00118.
- [13] Koncel-Kedziorski, R., Hajishirzi, H., Sabharwal, A., Etzioni, O., and Ang, S. D. Parsing Algebraic Word Problems into Equations. Transactions of the Association for Computational Linguistics, 2015. doi: 10.1162/tacl\_a\_00160.
- [14] Hosseini, M. J., Hajishirzi, H., Etzioni, O., and Kushman, N. Learning to solve arithmetic word problems with verb categorization. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. doi: 10.3115/v1/D14-1058.
- [15] Miao, S. Y., Liang, C. C., and Su, K. Y. A diverse corpus for evaluating and developing English math word problem solvers. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.92.
- [16] Roy, S. and Roth, D. Solving general arithmetic word problems. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015. doi: 10.18653/v1/D15-1202.
- [17] Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.
- [18] Liu, Xiao, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. "GPT understands, too." AI Open (2023).

- [19] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. “Autoprompt: Eliciting knowledge from language models with automatically generated prompts,” 2020, arXiv preprint arXiv:2010.15980.
- [20] Zhengbao Jiang, Frank F Xu, et al. 2020b. “How can we know what language models know,” Transactions of the Association for Computational Linguistics, 2020, 8:423–438.
- [21] Liu et al., "Text Summarization with Pretrained Encoders," Association for Computational Linguistics, 2019.
- [22] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450.
- [23] Alec Radford, K. Narasimhan, T. Salimans, and I. Sutskever, " Improving Language Understanding by Generative Pretraining," arXiv:1801.06146 [cs.CL], 2018.
- [24] Vaswani Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).
- [25] Yonghui Wu, Mike Schuster, Zhifeng Chen, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

## LLM

### ORIGINALITY REPORT

<b>3%</b> SIMILARITY INDEX	<b>3%</b> INTERNET SOURCES	<b>1%</b> PUBLICATIONS	<b>1%</b> STUDENT PAPERS
-------------------------------	-------------------------------	---------------------------	-----------------------------

### PRIMARY SOURCES

<b>1</b>	<b>www.gabormelli.com</b> Internet Source	<b>1%</b>
<b>2</b>	<b>ir.juit.ac.in:8080</b> Internet Source	<b>1%</b>
<b>3</b>	<b>Submitted to Somaiya Vidyavihar</b> Student Paper	<b>&lt;1%</b>
<b>4</b>	<b>export.arxiv.org</b> Internet Source	<b>&lt;1%</b>
<b>5</b>	<b>www.mathaware.org</b> Internet Source	<b>&lt;1%</b>
<b>6</b>	<b>researchonline.ljmu.ac.uk</b> Internet Source	<b>&lt;1%</b>
<b>7</b>	<b>Submitted to University of West London</b> Student Paper	<b>&lt;1%</b>
<b>8</b>	<b>Submitted to Melbourne Institute of Technology</b> Student Paper	<b>&lt;1%</b>
<b>9</b>	<b>Submitted to University College London</b> Student Paper	<b>&lt;1%</b>

10	Submitted to University of College Cork Student Paper	<1 %
11	Chao Chang, Junming Zhou, Xiangwei Zeng, Yong Tang. "Chapter 21 SUMOPE: Enhanced Hierarchical Summarization Model for Long Texts", Springer Science and Business Media LLC, 2023 Publication	<1 %
12	arxiv.org Internet Source	<1 %
13	pureadmin.unileoben.ac.at Internet Source	<1 %
14	zdocs.ro Internet Source	<1 %
15	Jie Mei, Xiang Jiang, Aminul Islam, Abidalrahman Moh'd, Evangelos Milios. "Integrating Global Attention for Pairwise Text Comparison", Proceedings of the ACM Symposium on Document Engineering 2018 - DocEng '18, 2018 Publication	<1 %
16	dokumen.pub Internet Source	<1 %
17	"Advanced Computing Technologies and Applications", Springer Science and Business Media LLC, 2020 Publication	<1 %

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT**  
**PLAGIARISM VERIFICATION REPORT**

Date: .....

Type of Document (Tick):  PhD Thesis  M.Tech Dissertation/ Report  B.Tech Project Report  Paper

Name: \_\_\_\_\_ Department: \_\_\_\_\_ Enrolment No \_\_\_\_\_

Contact No. \_\_\_\_\_ E-mail. \_\_\_\_\_

Name of the Supervisor: \_\_\_\_\_

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): \_\_\_\_\_

**UNDERTAKING**

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

**FOR DEPARTMENT USE**

We have checked the thesis/report as per norms and found **Similarity Index** at ..... (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

**FOR LRC USE**

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> <li>• All Preliminary Pages</li> <li>• Bibliography/Images/Quotes</li> <li>• 14 Words String</li> </ul>		Word Counts	
<b>Report Generated on</b>			Character Counts	
		<b>Submission ID</b>	Total Pages Scanned	
			File Size	

Checked by  
Name & Signature

Librarian

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at [plagcheck.juit@gmail.com](mailto:plagcheck.juit@gmail.com)**