Smart Resume Generator: Customized Resumes For Every Opportunity

Project Description:

The Resume Generator project aims to develop an AI-driven tool for automating the creation of professional resumes. The objective is to build a generative model that can craft tailored resumes based on user inputs such as personal information, job experience, and career goals. By analyzing these details, the model produces well-structured resumes that effectively highlight the candidate's skills, achievements, and qualifications. This tool streamlines the resume creation process, enabling users to generate polished and personalized resumes quickly, thereby enhancing their ability to present themselves effectively to potential employers and improve their job application success.

Scenario 1: University Career Services

A university's career services department offers a resume generator to assist students in creating polished resumes tailored to specific industries. Students input details such as their academic achievements, internships, and extracurricular activities, and the tool generates resumes highlighting the most relevant skills and experiences for fields like finance, engineering, or marketing. This service helps students stand out in competitive job markets, similar to how career advisors provide tailored guidance based on individual career goals.

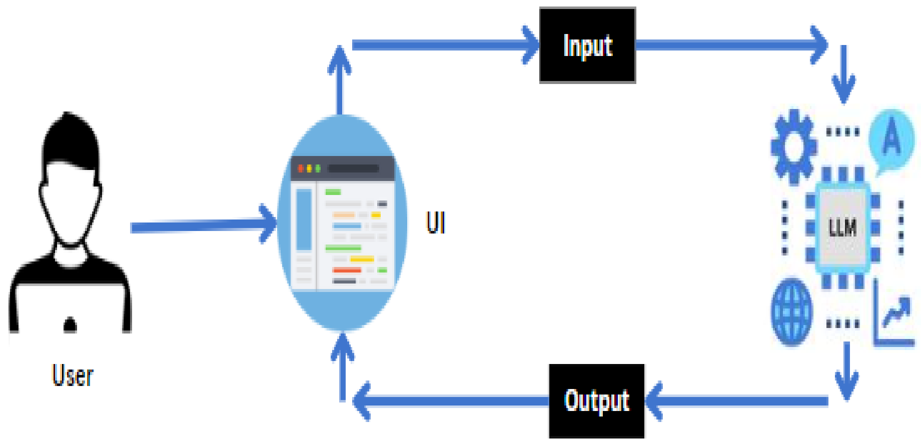
Scenario 2: Job Placement Agencies

A job placement agency uses the resume generator to streamline the job application process for clients. Candidates provide their work history, skills, and job preferences, and the tool produces multiple resume versions optimized for different job roles. This ensures that clients can quickly apply to various positions with resumes tailored to each opportunity, enhancing their chances of securing interviews. The approach is akin to personalized job coaching, where advisors help candidates, craft resumes for specific roles.

Scenario 3: Freelancers and Gig Workers

Freelancers and gig workers use the resume generator to create dynamic resumes that reflect their diverse project experience and skill sets. By inputting details of past projects, skills, and client testimonials, the tool generates resumes suited for different types of gigs, such as web development, graphic design, or writing. This allows freelancers to quickly customize their resumes when bidding for new projects, similar to how they might adjust their portfolios to match the needs of potential clients

Architecture:



Project Flow:

• ***User Input via Streamlit UI:***

Users input a prompt (e.g., topic, keywords) and specify parameters such as the desired length, tone, or style through the Streamlit interface.

• ***Backend Processing with Generative AI Model:***

The input data is sent to the backend, where it interfaces with the selected Generative AI model (e.g., GPT-4, Gemini, etc.).

The model processes the input, generating text based on the specified parameters and user input.

• ***Content Generation:***

The AI model autonomously creates content tailored to the user's specifications. This could be a blog post, poem, article, or any other form of text.

• ***Return and Display Generated Content:***

The generated content is sent back to the frontend for display on the Streamlit app.

The app presents the content to the user in an easily readable format.

• ***Customization and Finalization***:

Users can further customize the generated content through the Streamlit UI if desired. This might include editing text, adjusting length, or altering tone.

• ***Export and Usage:***

Once satisfied, users can export or copy the content for their use, such as saving it to a file or directly sharing it.

To accomplish this, we have to complete all the activities listed below,

- Initialize Gemini Pro LLM:
    - Generate Gemini Pro API
    - Initialize the pre-trained model
- Interfacing with Pre-trained Model
    - Travel itinerary Generation
- Model Deployment
    - Deploy the application using Streamlit

Prior Knowledge:

To work on a poetry generator project using Streamlit and Google Generative AI, prior knowledge in the following areas is essential:

1.***Natural Language Processing (NLP):***

Understanding the basics of NLP is crucial for processing and generating text. Familiarity with concepts such as tokenization, text generation, and language models is necessary to build a foundation for the project.

2. ***Generative AI Models:***

Knowledge of generative models, particularly those like Google's language models, is important. You should understand how these models work, how to fine-tune them, and how to manage parameters like temperature and max tokens to control the output.

3.***Python Programming:***

Proficiency in Python is required as it is the primary language used in both Streamlit and Google Generative AI. You should be comfortable writing scripts, handling APIs, and processing data.

4. ***Streamlit Framework:***

Familiarity with Streamlit is necessary to build the user interface for the poetry generator. You should know how to create interactive web applications, manage user inputs, and display generated text dynamically.

Basic knowledge of building interactive web applications using Streamlit.

Understanding of Streamlit's UI components and how to integrate them with backend logic.

https://www.datacamp.com/tutorial/streamlit

5. ***APIs and Integration:***

Understanding how to integrate different tools and services using APIs is important. This includes managing API requests, handling responses, and ensuring the seamless operation of the app. Streamlit:

Project Structure & Requirements Specification:

Create the Project folder which contains application file as shown below

Install the libraries

pip install streamlit

pip install google. generativeai

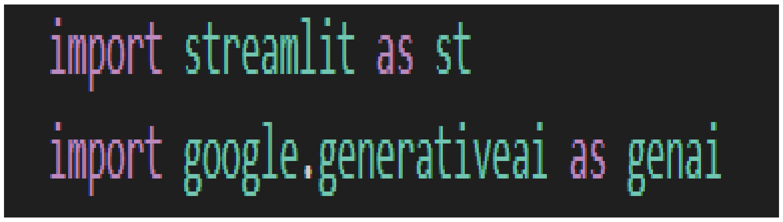
Milestone 1: Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

pip install streamlit

pip install google. Generativeai

Activity 1: Importing Libraries

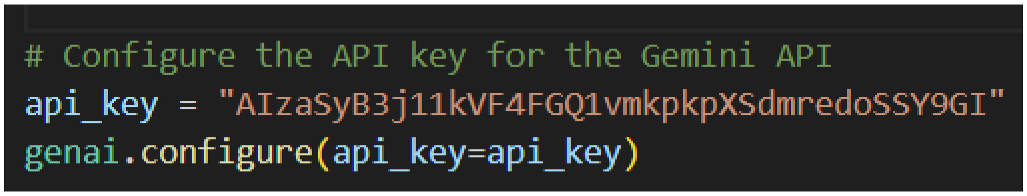


```
import streamlit as st
import google.generativeai as genai
```

1. Streamlit (streamlit as st): This is a popular open-source framework used for creating web apps with Python. The st alias allows you to easily access Streamlit's functions for building user interfaces, like adding buttons, displaying data, and more.

2. Google's Generative AI (google. generativeai as genai): This module likely provides access to Google's generative AI models. The genai alias allows you to interact with these models to generate text, images, or other outputs, depending on what Google's generative AI API offers.

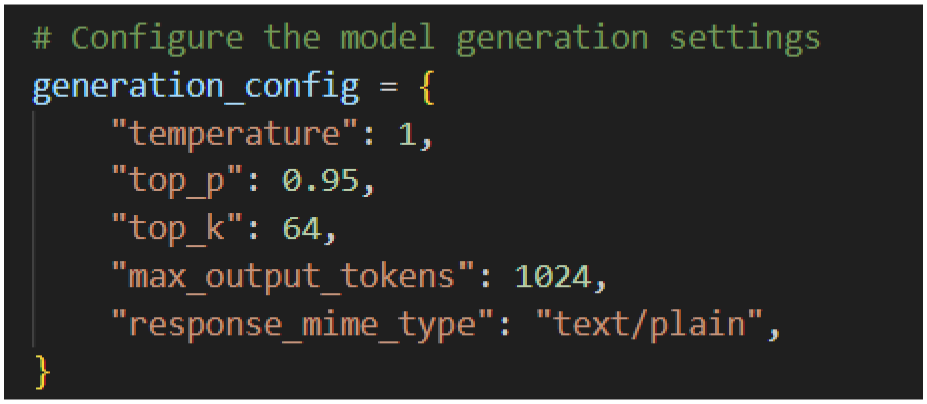
Activity 1.2: Configurating of the Gemini Pro API



```
# Configure the API key for the Gemini API
api_key = "AIzaSyB3j11kVF4FGQ1vmkpkpXSdmredoSSY9GI"
genai.configure(api_key=api_key)
```

This code sets an API key for accessing Google's Generative AI services. The api_key variable stores the key, and genai. configure(api_key=api_key) configures the genai module to use this key for authenticating requests to Google's Generative AI API. This allows secure interaction with the AI models.

Activity 1.3: Defining the model

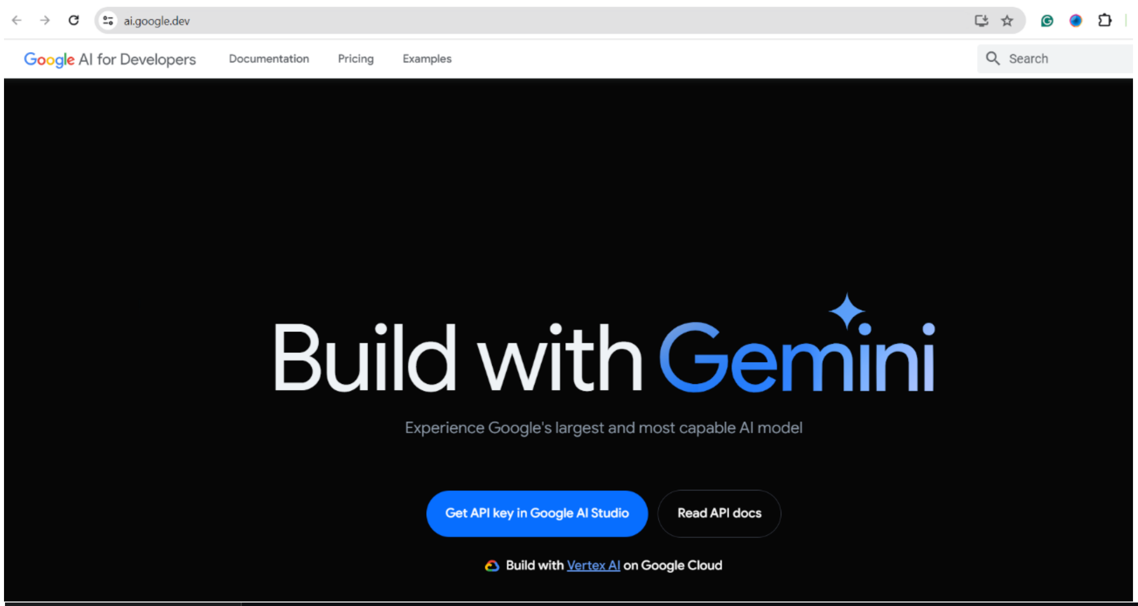


```
# Configure the model generation settings
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 1024,
    "response_mime_type": "text/plain",
}
```

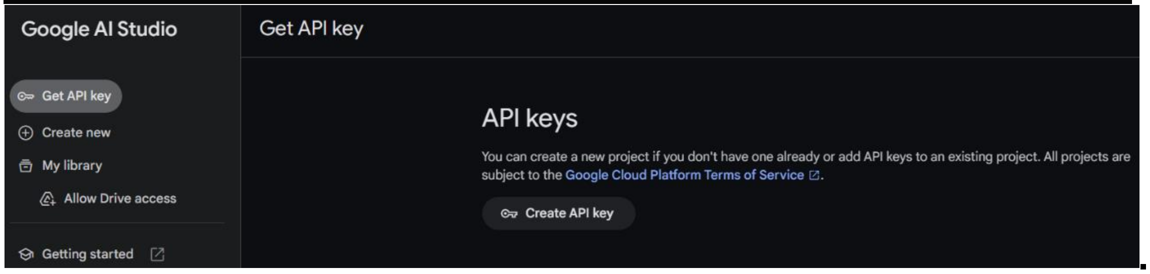
This snippet defines a dictionary called generation_config that sets various parameters for generating outputs using a generative AI model:

- temperature: Controls randomness. A value of 1 provides balanced randomness in the generated text.
- top_p: A sampling method that selects tokens from the smallest set whose probabilities sum up to 0.95, ensuring diversity.
- top_k: Limits the selection to the top 64 tokens, further controlling diversity.
- max_output_tokens: Specifies the maximum length of the generated text, capped at 1024 tokens.
- response_mime_type: Indicates the format of the response, here set to plain text.

Initializing the Models



Experience Google's largest and most capable AI model



You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the Google Cloud Platform Terms of Service.

Link: https://ai.google.dev/gemini-api

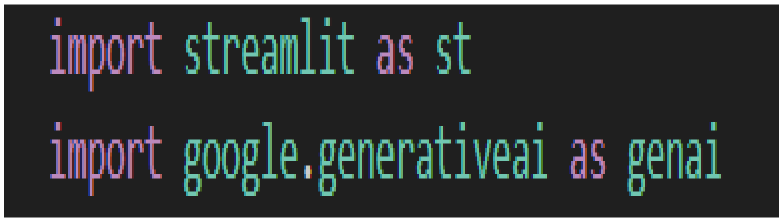
Enable the Gemini API

- Once your project is created, navigate to the API & Services Dashboard.

- Click on Enable APIs and Services at the top.
- Search for "Gemini API" in the API library.
- Select the Gemini API and click Enable.

Activity 2: Initialize the pre-trained model

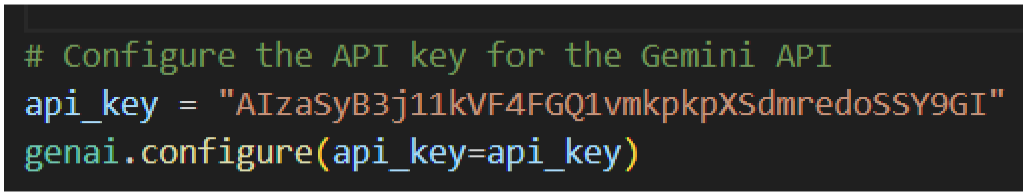
Activity 2.1: Import necessary files



```python
import streamlit as st
import google.generativeai as genai
```

- Streamlit, a popular Python library, is imported as st, enabling the creation of user interfaces directly within the Python script.
- Google Generative AI (genai): Imported to interact with the Gemini Pro model.

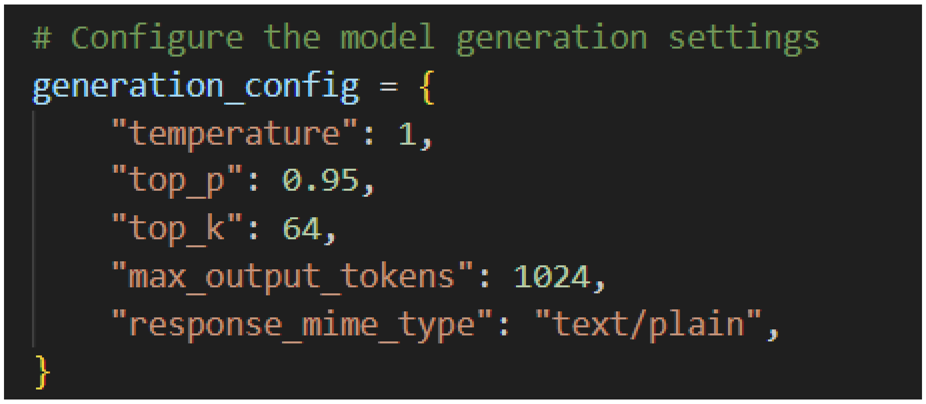
Activity 2.2: Configuration of the Gemini Pro API



```python
# Configure the API key for the Gemini API
api_key = "AIzaSyB3j11kVF4FGQ1vmkpkpXSdmredoSSY9GI"
genai.configure(api_key=api_key)
```

This code sets an API key for accessing Google's Generative AI services. The api_key variable stores the key, and genai. configure(api_key=api_key) configures the genai module to use this key for authenticating requests to Google's Generative AI API. This allows secure interaction with the AI models.

Activity 2.3: Define the model to be used



```python
# Configure the model generation settings
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 1024,
    "response_mime_type": "text/plain",
}
```

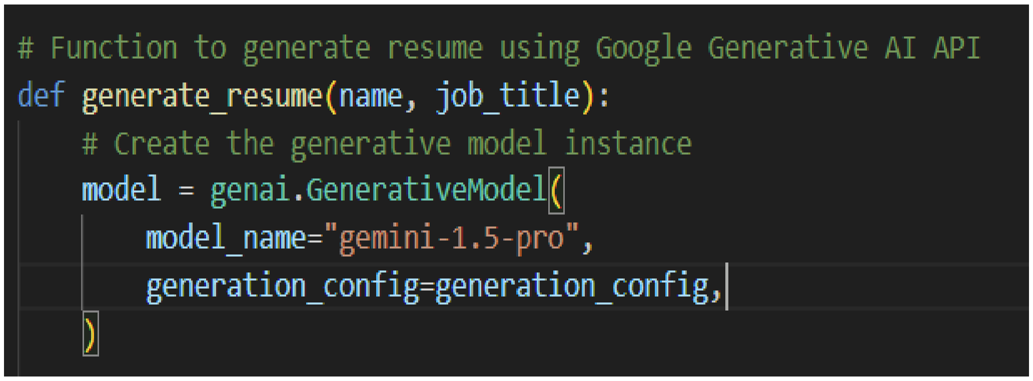
This snippet defines a dictionary called generation_config that sets various parameters for generating outputs using a generative AI model:

- ***temperature:*** Controls randomness. A value of 1 provides balanced randomness in the generated text.
- ***top_p:*** A sampling method that selects tokens from the smallest set whose probabilities sum up to 0.95, ensuring diversity.
- ***top_k:*** Limits the selection to the top 64 tokens, further controlling diversity.
- ***max_output_tokens:*** Specifies the maximum length of the generated text, capped at 1024 tokens.
- ***response_mime_type:*** Indicates the format of the response, here set to plain text.

Milestone 3: Interfacing with Pre-trained Model

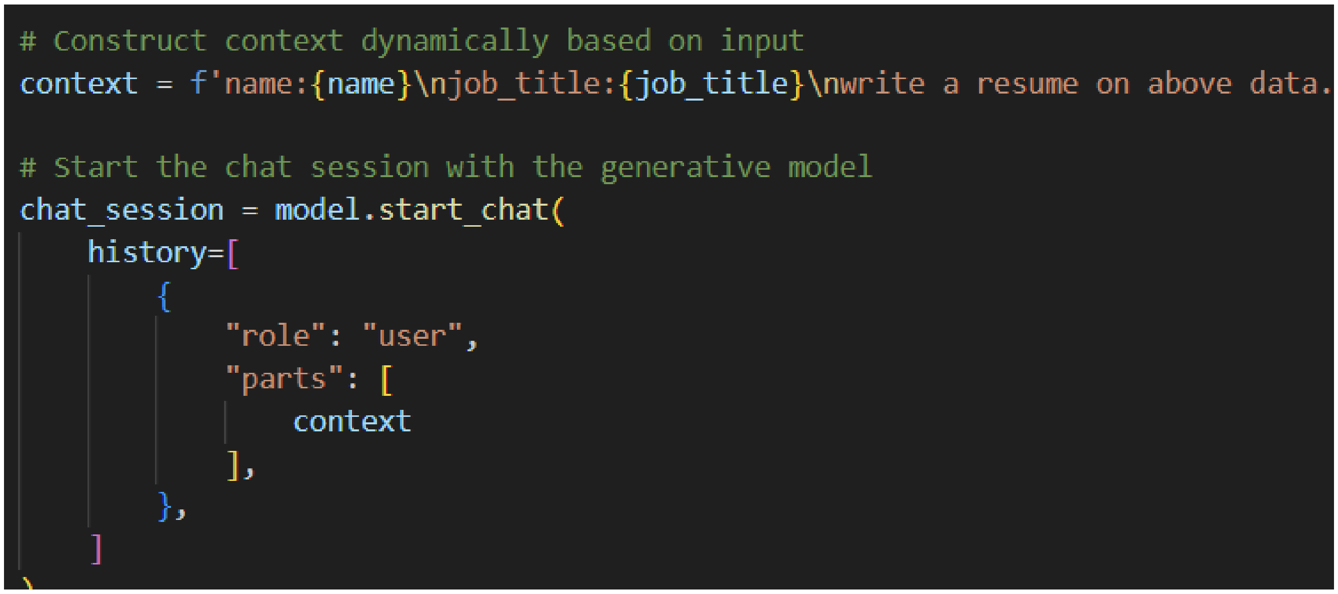
In this milestone, we will build a prompt template to generate feedback based on the project details entered by the user.

Activity 3.1: Creating Function & Defining the model



```python
# Function to generate resume using Google Generative AI API
def generate_resume(name, job_title):
    # Create the generative model instance
    model = genai.GenerativeModel(
        model_name="gemini-1.5-pro",
        generation_config=generation_config,
    )
```

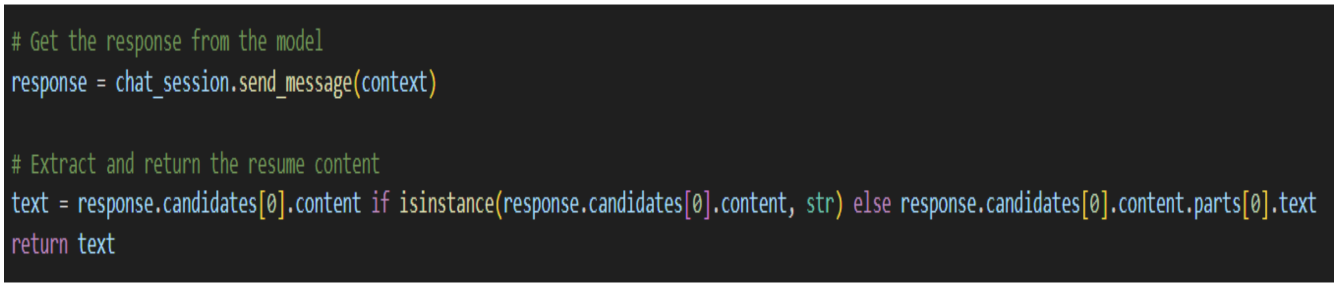
- This function initializes a generative model from the Google AI library using a specified model version and configuration settings. The generate_resume function sets up the model but doesn't yet generate or return resume content; additional code would be needed to use the model for generating a resume.



```python
# Construct context dynamically based on input
context = f'name:{name}\njob_title:{job_title}\nwrite a resume on above data.

# Start the chat session with the generative model
chat_session = model.start_chat(
    history=[
        {
            "role": "user",
            "parts": [
                context
            ],
        },
    ]
)
```

This code dynamically creates a context for the generative model based on the user's input, specifically the name and job title. The context string instructs the model to generate a resume that includes experience and knowledge, using dummy projects and experience, and formats the final output in Markdown.

Following this, a chat session with the generative model is initiated. The model. start_chat method starts a conversation with the model, providing it with an initial message (in this case, the context string) to guide the generation. The history parameter contains the initial user input, establishing the context for the model's response



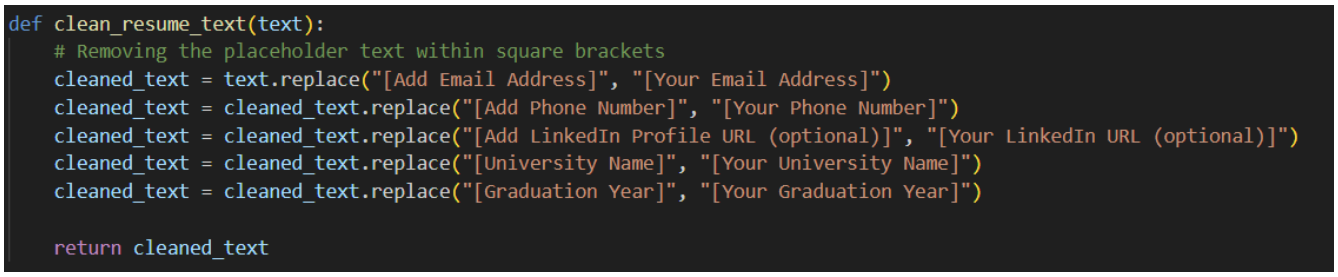
```python
# Get the response from the model
response = chat_session.send_message(context)

# Extract and return the resume content
text = response.candidates[0].content if isinstance(response.candidates[0].content, str) else response.candidates[0].content.parts[0].text
return text
```

In this code, the function retrieves and processes the model's response to generate a resume:

1. Send Message: The chat_session. Send_message(context) sends the previously constructed context to the model, requesting it to generate a resume based on that input.
2. Extract Response: The code then extracts the resume content from the model's response:

- It first checks if the content is a string directly. If so, it assigns this string to text.
- If the content is not a direct string but a more complex object, it accesses the text property of the first part of the content.

3. Return Resume: Finally, the extracted text (the generated resume) is returned from the function.



```python
def clean_resume_text(text):
    # Removing the placeholder text within square brackets
    cleaned_text = text.replace("[Add Email Address]", "[Your Email Address]")
    cleaned_text = cleaned_text.replace("[Add Phone Number]", "[Your Phone Number]")
    cleaned_text = cleaned_text.replace("[Add LinkedIn Profile URL (optional)]", "[Your LinkedIn URL (optional)]")
    cleaned_text = cleaned_text.replace("[University Name]", "[Your University Name]")
    cleaned_text = cleaned_text.replace("[Graduation Year]", "[Your Graduation Year]")

    return cleaned_text
```

This function, `clean_resume_text`, is designed to replace placeholder text in a resume with more specific, user-friendly prompts. Here's how it works:

1. Placeholder Replacement: It uses the `replace` method to substitute generic placeholders in the `text` with more personalized placeholders. For example:

` [Add Email Address] ` is replaced with ` [Your Email Address] `.

` [Add Phone Number] ` is replaced with ` [Your Phone Number] `.

` [Add LinkedIn Profile URL (optional)] ` is replaced with ` [Your LinkedIn URL (optional)] `.

` [University Name] ` is replaced with ` [Your University Name] `.

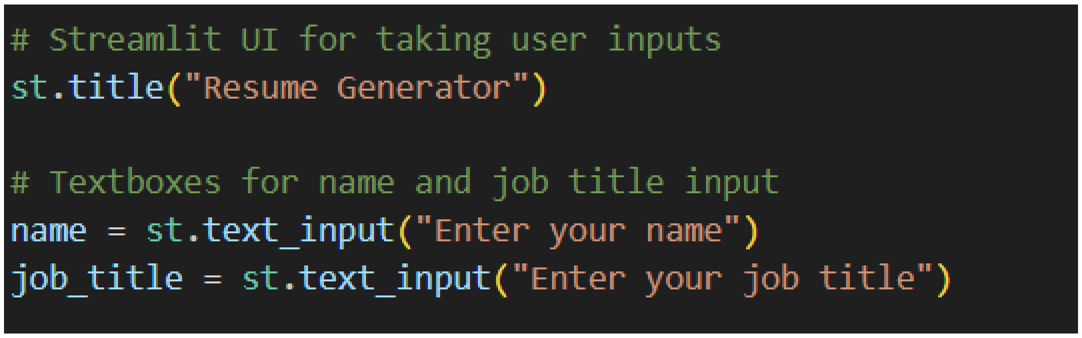
` [Graduation Year] ` is replaced with ` [Your Graduation Year] `.

2. Return Cleaned Text: The function returns the modified text with updated placeholders, making it more suitable for users to fill in with their personal information.

Milestone 4: Model Deployment

In this milestone, we deploy the created model using Streamlit. Streamlit allows us to create a user-friendly web interface, enabling users to interact with the model through their web browser

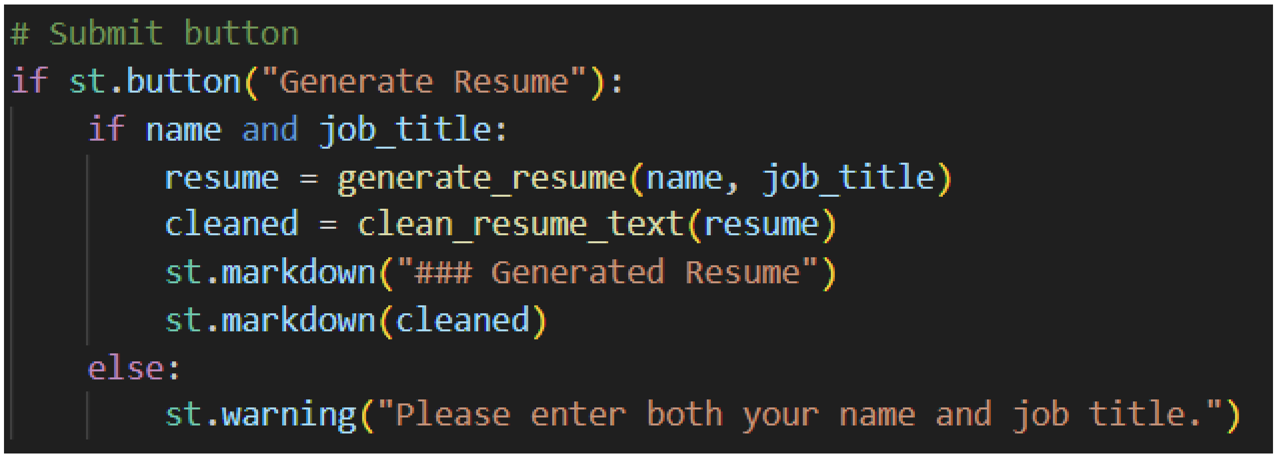
Activity 4.1: Starting Streamlit



```python
# Streamlit UI for taking user inputs
st.title("Resume Generator")

# Textboxes for name and job title input
name = st.text_input("Enter your name")
job_title = st.text_input("Enter your job title")
```

- ***St. Title("Resume Generator"):*** Displays the title "Resume Generator" at the top of the app.
- name = st.text_input ("Enter your name"): Creates a textbox for users to input their name. The entered value is stored in the name variable.
- job_title = st.text_input ("Enter your job title"): Creates a textbox for users to input their job title. The entered value is stored in the job_title variable

Activity 4.2: Displaying for user



```
# Submit button
if st.button("Generate Resume"):
    if name and job_title:
        resume = generate_resume(name, job_title)
        cleaned = clean_resume_text(resume)
        st.markdown("### Generated Resume")
        st.markdown(cleaned)
    else:
        st.warning("Please enter both your name and job title.")
```

1. ***Button Action:*** When the "Generate Resume" button is clicked, it triggers the code inside the block.

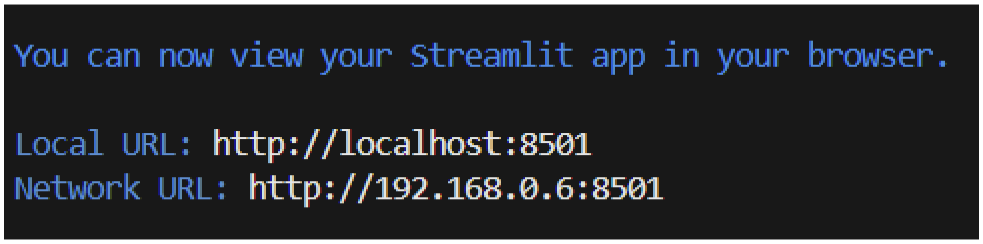
2. ***Check Inputs:*** The code checks if both `name` and `job_title` have been provided.

3. ***Generate and Clean Resume:*** If both inputs are present, it generates the resume using these inputs and then cleans up the resume text.

4. ***Display Resume:*** The cleaned resume is displayed on the app using Markdown formatting.

5. ***Error Handling***: If either `name` or `job_title` is missing, a warning message is shown asking the user to enter both fields

Activity 4.3: Running the web application

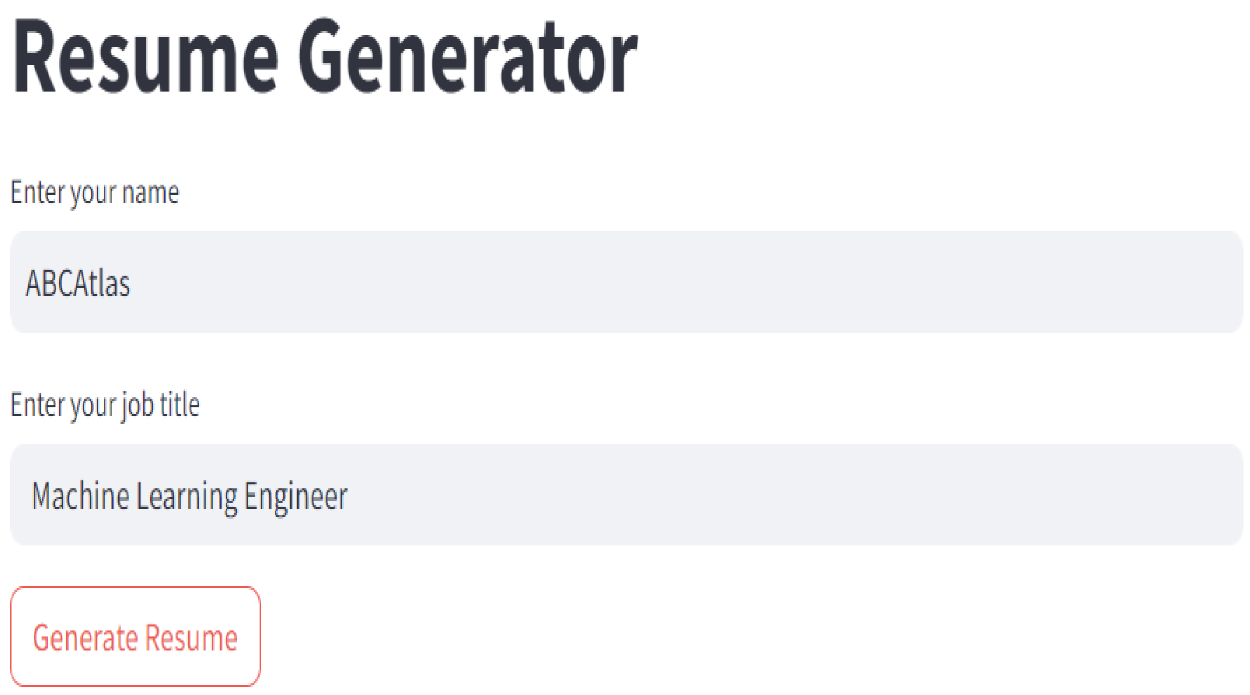


```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.6:8501
```

The application is now running and can be accessed locally through the provided URL. It is also available on the network via the network URL, allowing access from other devices on the same network

Activity 5: Output

After giving the input:


Resume Generator

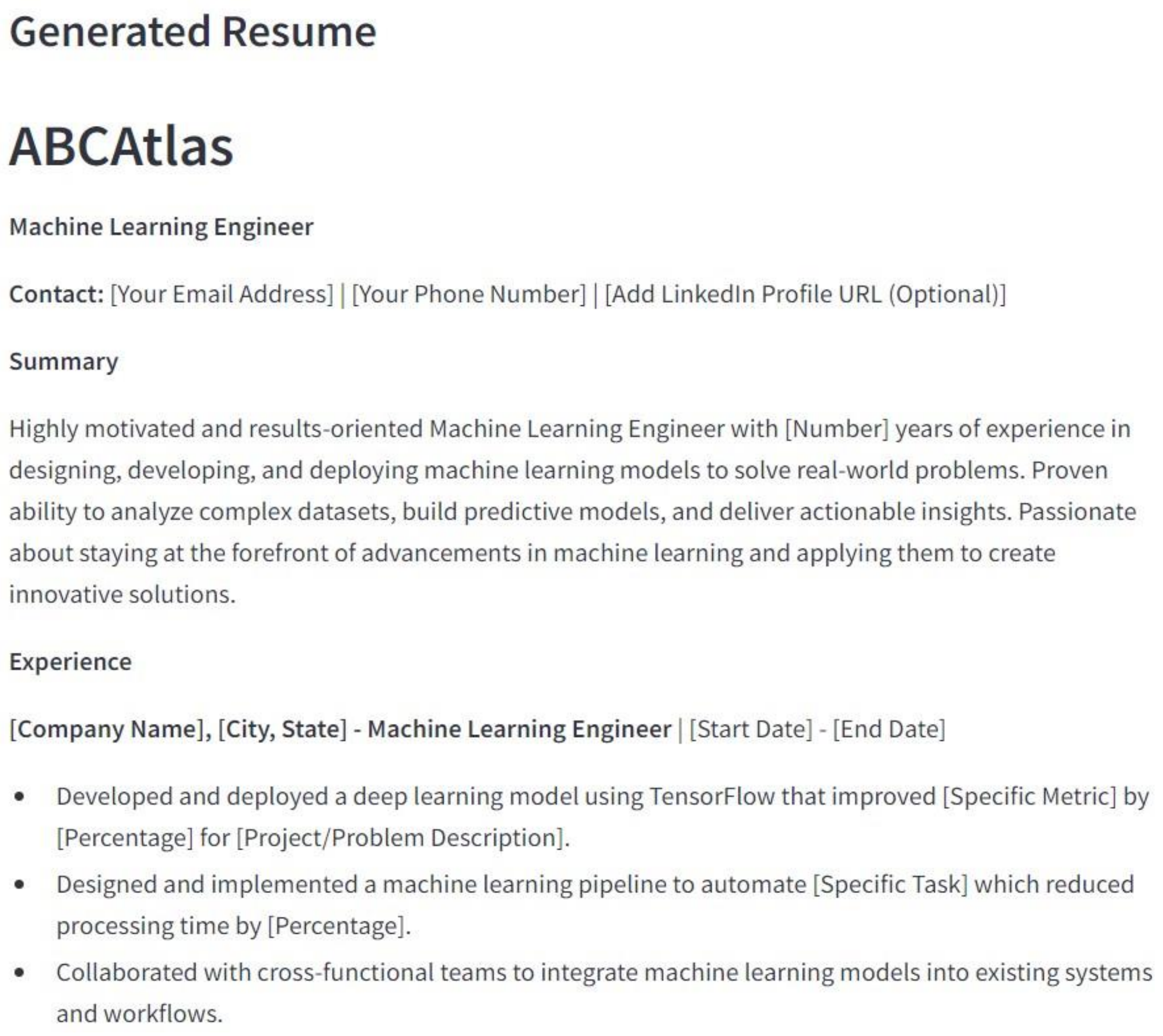
Enter your name

ABCAtlas

Enter your job title

Machine Learning Engineer

The Output generated:



Generated Resume

ABCAtlas

Machine Learning Engineer

Contact: [Your Email Address] | [Your Phone Number] | [Add LinkedIn Profile URL (Optional)]

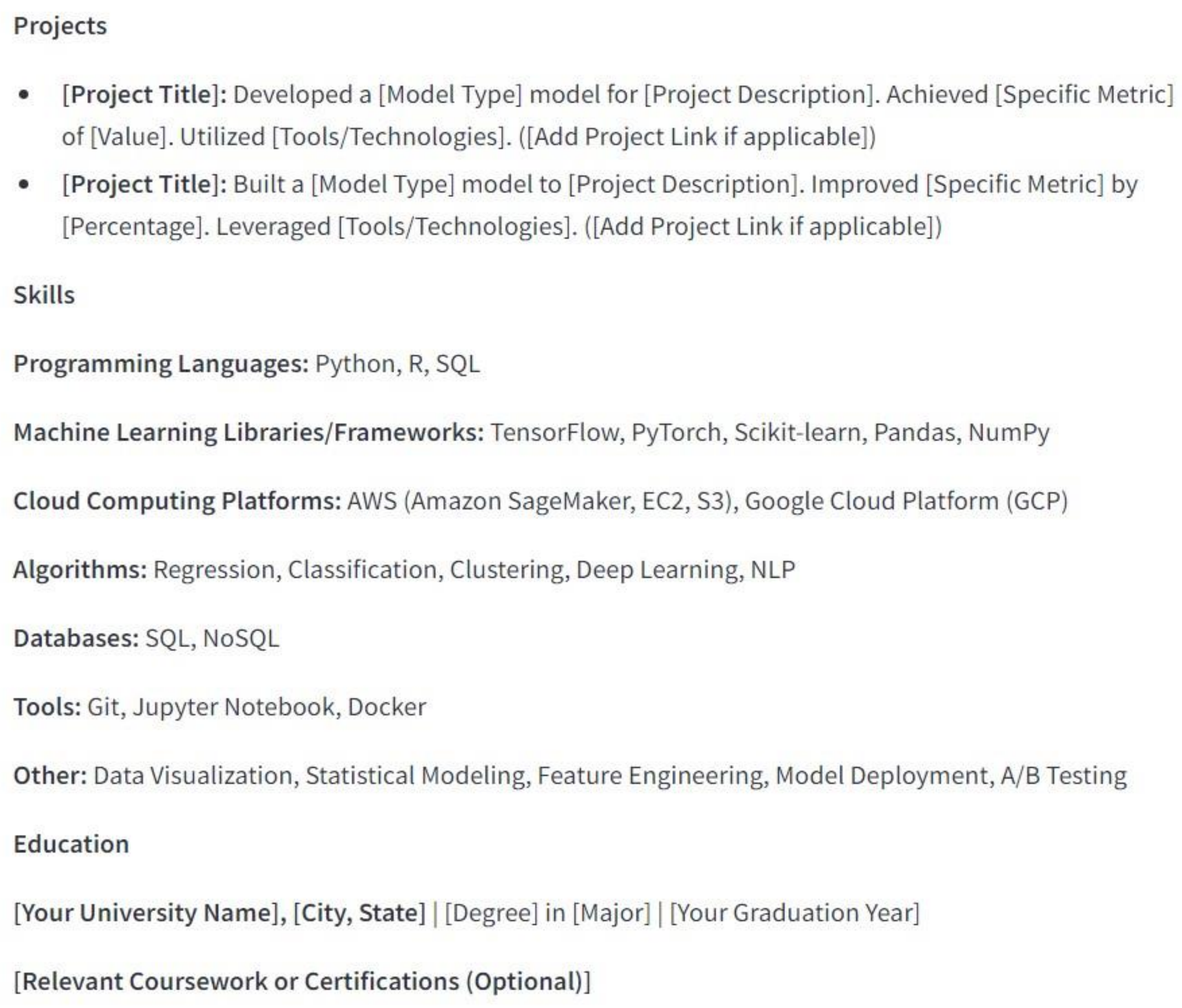
Summary

Highly motivated and results-oriented Machine Learning Engineer with [Number] years of experience in designing, developing, and deploying machine learning models to solve real-world problems. Proven ability to analyze complex datasets, build predictive models, and deliver actionable insights. Passionate about staying at the forefront of advancements in machine learning and applying them to create innovative solutions.

Experience

[Company Name], [City, State] - Machine Learning Engineer | [Start Date] - [End Date]

- Developed and deployed a deep learning model using TensorFlow that improved [Specific Metric] by [Percentage] for [Project/Problem Description].
- Designed and implemented a machine learning pipeline to automate [Specific Task] which reduced processing time by [Percentage].
- Collaborated with cross-functional teams to integrate machine learning models into existing systems and workflows.



Projects

- **[Project Title]:** Developed a [Model Type] model for [Project Description]. Achieved [Specific Metric] of [Value]. Utilized [Tools/Technologies]. ([Add Project Link if applicable])
- **[Project Title]:** Built a [Model Type] model to [Project Description]. Improved [Specific Metric] by [Percentage]. Leveraged [Tools/Technologies]. ([Add Project Link if applicable])

Skills

Programming Languages: Python, R, SQL

Machine Learning Libraries/Frameworks: TensorFlow, PyTorch, Scikit-learn, Pandas, NumPy

Cloud Computing Platforms: AWS (Amazon SageMaker, EC2, S3), Google Cloud Platform (GCP)

Algorithms: Regression, Classification, Clustering, Deep Learning, NLP

Databases: SQL, NoSQL

Tools: Git, Jupyter Notebook, Docker

Other: Data Visualization, Statistical Modeling, Feature Engineering, Model Deployment, A/B Testing

Education

[Your University Name], [City, State] | [Degree] in [Major] | [Your Graduation Year]

[Relevant Coursework or Certifications (Optional)]

Conclusion:

The Resume Generator project utilizes Google Generative AI to produce customized resumes based on user inputs. With Streamlit providing an accessible user interface and the generative model handling content creation, this tool helps users generate professional resumes tailored to their personal and career details. Users input key information such as their name, job experience, and skills, and the tool creates a polished resume that highlights their qualifications. This project underscores the efficiency of AI in automating the resume-building process, enabling users to quickly produce well-structured and impactful resumes. Overall, the Resume Generator streamlines job application efforts and illustrates the potential of AI in enhancing professional document creation.