Build Resume Generation Using LLaMA3 Locally

1. Objectives

- Develop an AI-powered agent that generates professional resumes based on user input.
- Utilize LLaMA 3 locally to ensure data privacy and offline capabilities.
- Implement the solution entirely within a Jupyter Notebook for ease of experimentation and learning.

2. Problem Statement

- Job seekers often struggle to create tailored resumes that highlight their strengths and align with specific job roles. This exercise aims to build an AI agent that:
  - Collects user information interactively.
  - Generates a well-structured, professional resume.

3. Inputs / Shared Artifacts

- Installed and configured LLaMA 3 model locally using llama_cpp or LangChain's useLlama2Chat.
- Basic understanding prompting techniques.

4. Expected Outcomes

- A complete, professional resume generated based on user input.
- Enhanced understanding of integrating LLaMA 3 with Python for practical AI applications.
- Experience in prompt engineering and handling model outputs within a local environment.

5. Concepts Covered

- **AI-Powered Resume Generation:** Leveraging large language models like LLaMA 3 to generate structured, human-like resumes from raw user inputs.
- **Local Model Deployment:** Running LLaMA 3 locally to ensure data privacy, offline capability, and control over the AI model.
- **Prompt Engineering:** Crafting effective prompts to guide the model's output towards clear, relevant, and professional resume formats.

6. Example: Step-by-Step Instructions

```
# prompt: generate python code to use LLaMA cpp and a huggingface model, code must handle the exception and organize as OOP

!pip install llama-cpp-python

from llama_cpp import Llama
import os

class LlamaModel:
    def __init__(self, model_path, n_ctx=512):
        """
        Initializes the Llama model.

        Args:
            model_path (str): The path to the LLaMA model file.
            n_ctx (int): The context size.
        """
        if not os.path.exists(model_path):
            raise FileNotFoundError(f"Model file not found at: {model_path}")
        try:
            self.llm = Llama(model_path=model_path, n_ctx=n_ctx)
            print(f"LLaMA model loaded successfully from {model_path}")
        except Exception as e:
```

```
            raise RuntimeError(f"Failed to load LLaMA model: {e}")

    def generate_text(self, prompt, max_tokens=100, temperature=0.7):
        """
        Generates text using the loaded Llama model.

        Args:
            prompt (str): The input prompt for text generation.
            max_tokens (int): The maximum number of tokens to generate.
            temperature (float): The sampling temperature.

        Returns:
            str: The generated text.
        """
        try:
            output = self.llm(prompt, max_tokens=max_tokens,
temperature=temperature)
            return output['choices'][0]['text']
        except Exception as e:
            raise RuntimeError(f"Failed to generate text: {e}")

# Example usage:
# Replace 'path/to/your/llama.gguf' with the actual path to your LLaMA model
file.
# You would typically download this file from a source like Hugging Face.

# You might need to download a model first.
# For example, using a shell command to download from Hugging Face:
# Make sure to choose a compatible model file (e.g., .gguf format)
!wget https://huggingface.co/TheBloke/Llama-2-7B-Chat-
GGUF/resolve/main/llama-2-7b-chat.Q4_K_M.gguf -O llama-2-7b-chat.Q4_K_M.gguf

model_file_path = 'llama-2-7b-chat.Q4_K_M.gguf' # Replace with your
downloaded model path

try:
```

```
    # Initialize the model
    llama_model = LlamaModel(model_file_path)

    # Generate text
    prompt_text = "Once upon a time,"
    generated_text = llama_model.generate_text(prompt_text, max_tokens=50)
    print("\nGenerated Text:")
    print(generated_text)

except FileNotFoundError as e:
    print(f"Error: {e}")
    print("Please ensure the model file exists at the specified path.")
except RuntimeError as e:
    print(f"Error during model operation: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

7. Final Submission Checklist

- Submit your Python script or notebook containing the full code.
- Use a dummy input list (auto input). Do not use manual input (input() built-in).
- Include sample inputs and outputs. (3 samples)