Cheat Sheet: Build GenAI Application With LangChain

Estimated time needed: 5 minutes

Package/Method	Description	Code Example

mkdir and cd	Create and navigate into a new project directory.	
		```
		mkdir genai_flask_app
		cd genai_flask_app
		```

Virtual environment	Set up a Python virtual environment for package management.	
		```
		python3.11 -m venv venv
		source venv/bin/activate
		```

pip install ibm-watsonx-ai	Install the IBM watsonx AI library for LLM interactions.	`pip install ibm-watsonx-ai`

Credentials	Authenticate with IBM watsonx AI using credentials.	
		```
		from ibm_watsonx_ai import Credentials

		credentials = Credentials(
		    url = “https://us-south.ml.cloud.ibm.com",
		    # api_key = “<YOUR_API_KEY>“
		)
		```

Model parameters	Define parameters for model inference.	
		```
		from ibm_watsonx_ai.metanames import GenTextParamsMetaNames

		params = {
		    GenTextParamsMetaNames.DECODING_METHOD: “greedy”,
		    GenTextParamsMetaNames.MAX_NEW_TOKENS: 100
		}
		```

Model inference	Initialize an AI model for text generation.	
		```
		from ibm_watsonx_ai.foundation_models import ModelInference

		model = ModelInference(
		    model_id=”ibm/granite-3-3-8b-instruct”,
		    params=params,
		    credentials=credentials,
		    project_id=”skills-network”
		)
		```

Generating AI response	Use an AI model to generate text based on a prompt.	`text = """`
		`Only reply with the answer. What is the capital of Canada?`
		`"""`
		
		`print(model.generate(text)[‘results’][0][‘generated_text’])`

LangChain prompt templates	Define reusable prompt templates for different models.	`from langchain.prompts import PromptTemplate`
		
		`llama3_template = PromptTemplate(`
		`    template=’’’<|begin_of_text|><|start_header_id|>system<|end_header_id|>`
		`{system_prompt}<|eot_id|><|start_header_id|>user<|end_header_id|>`
		`{user_prompt}<|eot_id|><|start_header_id|>assistant<|end_header_id|>`
		`‘’’,`
		`    input_variables=[“system_prompt”, “user_prompt”]`
		`)`

LangChain chaining	Pipe a prompt template into an AI model to generate structured output.	`def get_ai_response(model, template, system_prompt, user_prompt):`
		`    chain = template | model`
		`    return chain.invoke({'system_prompt': system_prompt, 'user_prompt': user_prompt})`

Tokenization and prompt formatting	Specialized token formatting for different AI models.	`# Llama 3 formatted prompt`
		`text = """`
		`<|begin_of_text|><|start_header_id|>system<|end_header_id|>`
		`You are an expert assistant who provides concise and accurate answers.<|eot_id|>`
		
		`<|start_header_id|>user<|end_header_id|>`
		`What is the capital of Canada?<|eot_id|>`
		
		`<|start_header_id|>assistant<|end_header_id|>`
		`“””`

JSON output parser	Parse and structure AI-generated responses using LangChain.	`from langchain_core.output_parsers import JsonOutputParser`
		`from pydantic import BaseModel, Field`
		
		`class AIResponse(BaseModel):`
		`    summary: str = Field(description=”Summary of the user’s message”)`
		`    sentiment: int = Field(description=”Sentiment score from 0 to 100”)`
		`    response: str = Field(description=”Generated AI response”)`
		
		`json_parser = JsonOutputParser(pydantic_object=AIResponse)`

Enhancing AI outputs	Modify LangChain chaining to ensure structured JSON output.	`def get_ai_response(model, template, system_prompt, user_prompt):`
		`    chain = template | model | json_parser`
		`    return chain.invoke({`
		`        'system_prompt': system_prompt,`
		`        'user_prompt': user_prompt,`
		`        'format_prompt': json_parser.get_format_instructions()`
		`    })`

Flask API integration	Create an API endpoint for AI model interactions.	`from flask import Flask, request, jsonify`
		`from model import get_model_response`
		
		`app = Flask(name)`
		
		`@app.route(‘/generate’, methods=[‘POST’])`
		`def generate():`
		`    data = request.json`
		`    model_name = data.get(‘model’)`
		`    user_message = data.get(‘message’)`
		
		`if not user_message or not model_name:`
		
		`    return jsonify({"error": "Missing message or model selection"}), 400`
		
		`system_prompt = "You are an AI assistant helping with customer inquiries. Provide a concise response."`
		
		`try:`
		
		`    response = get_model_response(model_name, system_prompt, user_message)`
		
		`    return jsonify(response)`
		
		`except Exception as e:`
		
		`    return jsonify({"error": str(e)}), 500`
		
		`if name == ‘main‘:`
		`    app.run(debug=True)`

Author

Hailey Quach

Skills Network