# 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```<br>```cd genai_flask_app``` |
| **Virtual environment** | Set up a Python virtual environment for package management. | ```python3.11 -m venv venv```<br>```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```<br><br>```credentials = Credentials(```<br>```    url = "https://us-south.ml.cloud.ibm.com",```<br>```    # api_key = "<YOUR_API_KEY>"```<br>```)``` |
| **Model parameters** | Define parameters for model inference. | ```from ibm_watsonx_ai.metanames import GenTextParamsMetaNames```<br><br>```params = {```<br>```    GenTextParamsMetaNames.DECODING_METHOD: "greedy",```<br>```    GenTextParamsMetaNames.MAX_NEW_TOKENS: 100```<br>```}``` |
| **Model inference** | Initialize an AI model for text generation. | ```from ibm_watsonx_ai.foundation_models import ModelInference```<br><br>```model = ModelInference(```<br>```    model_id="ibm/granite-3-3-8b-instruct",```<br>```    params=params,```<br>```    credentials=credentials,```<br>```    project_id="skills-network"```<br>```)``` |

| | | |
|---|---|---|
| **Generating AI response** | Use an AI model to generate text based on a prompt. | ```<br>text = """<br>Only reply with the answer. What is the capital of Canada?<br>"""<br><br>print(model.generate(text)['results'][0]['generated_text'])<br>``` |
| **LangChain prompt templates** | Define reusable prompt templates for different models. | ```<br>from langchain.prompts import PromptTemplate<br><br>llama3_template = PromptTemplate(<br>    template='''<|begin_of_text|><|start_header_id|>system<|end_header_id|><br>{system_prompt}<|eot_id|><|start_header_id|>user<|end_header_id|><br>{user_prompt}<|eot_id|><|start_header_id|>assistant<|end_header_id|><br>''',<br>    input_variables=["system_prompt", "user_prompt"]<br>)<br>``` |
| **LangChain chaining** | Pipe a prompt template into an AI model to generate structured output. | ```<br>def get_ai_response(model, template, system_prompt, user_prompt):<br>    chain = template | model<br>    return chain.invoke({'system_prompt': system_prompt, 'user_prompt': user_prompt})<br>``` |
| **Tokenization and prompt formatting** | Specialized token formatting for different AI models. | ```<br># Llama 3 formatted prompt<br>text = """<br><|begin_of_text|><|start_header_id|>system<|end_header_id|><br>You are an expert assistant who provides concise and accurate answers.<|eot_id|><br><br><|start_header_id|>user<|end_header_id|><br>What is the capital of Canada?<|eot_id|><br><br><|start_header_id|>assistant<|end_header_id|><br>"""<br>``` |
| **JSON output parser** | Parse and structure AI-generated responses using LangChain. | ```<br>from langchain_core.output_parsers import JsonOutputParser<br>from pydantic import BaseModel, Field<br><br>class AIResponse(BaseModel):<br>    summary: str = Field(description="Summary of the user's message")<br>    sentiment: int = Field(description="Sentiment score from 0 to 100")<br>    response: str = Field(description="Generated AI response")<br><br>json_parser = JsonOutputParser(pydantic_object=AIResponse)<br>``` |
| **Enhancing AI outputs** | Modify LangChain chaining to ensure structured | ```<br>def get_ai_response(model, template, system_prompt, user_prompt):<br>    chain = template | model | json_parser<br>    return chain.invoke({<br>        'system_prompt': system_prompt,<br>        'user_prompt': user_prompt,<br>        'format_prompt': json_parser.get_format_instructions()<br>``` |

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

## Author

Hailey Quach