# 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` Copied!Wrap Toggled! |
| **Virtual environment** | Set up a Python virtual environment for package management. | `python3.11 -m venv venv` `source venv/bin/activate` Copied!Wrap Toggled! |
| **pip install ibm-watsonx-ai** | Install the IBM watsonx AI library for LLM interactions. | `pip install ibm-watsonx-ai` Copied!Wrap Toggled! |
| **Credentials** | Authenticate with | `from ibm_watsonx_ai import Credentials` |

| | | |
|---|---|---|
| | IBM watsonx AI using credentials. | `credentials = Credentials(`<br>`  url = “https://us-south.ml.cloud.ibm.com",`<br>`  # api_key = “<YOUR_API_KEY>“`<br>`)`<br><br>Copied!Wrap Toggled! |
| **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>`}`<br><br>Copied!Wrap Toggled! |
| **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>`)`<br><br>Copied!Wrap Toggled! |
| **Generating AI response** | Use an AI model to generate | `text = """` |

| | | |
|---|---|---|
| | text based on a prompt. | Only reply with the answer. What is the capital of Canada?<br>"""<br><br>print(model.generate(text)[‘results’][0][‘generated_text’])<br>Copied!Wrap Toggled! |
| **LangChain prompt templates** | Define reusable prompt templates for different models. | 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>Copied!Wrap Toggled! |
| **LangChain chaining** | Pipe a prompt template into an AI model to generate structured output. | 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>Copied!Wrap Toggled! |

| | | |
|---|---|---|
| **Tokenization and prompt formatting** | Specialized token formatting for different AI models. | # 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><\|start_header_id\|>assistant<\|end_header_id\|><br>“””<br>Copied!Wrap Toggled! |
| **JSON output parser** | Parse and structure AI-generated responses using LangChain. | 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>json_parser = JsonOutputParser(pydantic_object=AIResponse)<br>Copied!Wrap Toggled! |
| **Enhancing AI outputs** | Modify LangChain | def get_ai_response(model, template, system_prompt, user_prompt): |

| | | |
|---|---|---|
| | chaining to ensure structured JSON output. | chain = template \| model \| json_parser<br><br>return chain.invoke({<br><br>'system_prompt': system_prompt,<br><br>'user_prompt': user_prompt,<br><br>'format_prompt': json_parser.get_format_instructions()<br><br>})<br><br>Copied!Wrap Toggled! |
| **Flask API integration** | Create an API endpoint for AI model interactions. | from flask import Flask, request, jsonify<br><br>from model import get_model_response<br><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><br>system_prompt = "You are an AI assistant helping with customer inquiries. Provide a concise response."<br><br><br>try:<br><br>response = get_model_response(model_name, system_prompt, user_message) |

| | | |
|---|---|---|
| | | return jsonify(response)<br><br>except Exception as e:<br><br>return jsonify({"error": str(e)}), 500<br><br><br>if **name** == ‘**main**‘:<br>app.run(debug=True)<br><br>Copied!Wrap Toggled! |

**Author**

Hailey Quach