

# Cheat Sheet: Build GenAI Application With LangChain

Estimated time needed: 5 minutes

Package/Method	Description	Code Example
<b>mkdir and cd</b>	Create and navigate into a new project directory.	<pre>mkdir genai_flask_app cd genai_flask_app</pre>
<b>Virtual environment</b>	Set up a Python virtual environment for package management.	<pre>python3.11 -m venv venv source venv/bin/activate</pre>
<b>pip install ibm-watsonx-ai</b>	Install the IBM watsonx AI library for LLM interactions.	<pre>pip install ibm-watsonx-ai</pre>
<b>Credentials</b>	Authenticate with IBM watsonx AI using credentials.	<pre>from ibm_watsonx_ai import Credentials  credentials = Credentials(     url = "<a href="https://us-south.ml.cloud.ibm.com">https://us-south.ml.cloud.ibm.com</a>",     # api_key = "&lt;YOUR_API_KEY&gt;" )</pre>
<b>Model parameters</b>	Define parameters for model inference.	<pre>from ibm_watsonx_ai.metanames import GenTextParamsMetaNames  params = {     GenTextParamsMetaNames.DECODING_METHOD: "greedy",     GenTextParamsMetaNames.MAX_NEW_TOKENS: 100 }</pre>

<b>Model inference</b>	Initialize an AI model for text generation.	<pre> 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" ) </pre>
<b>Generating AI response</b>	Use an AI model to generate text based on a prompt.	<pre> text = """ Only reply with the answer. What is the capital of Canada? """  print(model.generate(text)['results'][0]['generated_text']) </pre>
<b>LangChain prompt templates</b>	Define reusable prompt templates for different models.	<pre> from langchain.prompts import PromptTemplate  llama3_template = PromptTemplate(     template='''&lt; begin_of_text &gt;&lt; start_header_id &gt;system&lt; end_header_id &gt; {system_prompt}&lt; eot_id &gt;&lt; start_header_id &gt;user&lt; end_header_id &gt; {user_prompt}&lt; eot_id &gt;&lt; start_header_id &gt;assistant&lt; end_header_id &gt; '''     input_variables=["system_prompt", "user_prompt"] ) </pre>
<b>LangChain chaining</b>	Pipe a prompt template into an AI model to generate structured output.	<pre> def get_ai_response(model, template, system_prompt, user_prompt):     chain = template   model     return chain.invoke({'system_prompt': system_prompt, 'user_prompt': user_prompt}) </pre>
<b>Tokenization and prompt formatting</b>	Specialized token formatting for different AI models.	<pre> # Llama 3 formatted prompt text = """ </pre>

		<pre>&lt; begin_of_text &gt;&lt; start_header_id &gt;system&lt; end_header_id &gt; You are an expert assistant who provides concise and accurate answers.&lt; eot_id &gt;  &lt; start_header_id &gt;user&lt; end_header_id &gt; What is the capital of Canada?&lt; eot_id &gt;  &lt; start_header_id &gt;assistant&lt; end_header_id &gt; """</pre>
JSON output parser	Parse and structure AI-generated responses using LangChain.	<pre>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)</pre>
Enhancing AI outputs	Modify LangChain chaining to ensure structured JSON output.	<pre>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()     })</pre>
Flask API integration	Create an API endpoint for AI model interactions.	<pre>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)</pre>

```
except Exception as e:
    return jsonify({"error": str(e)}), 500

if name == 'main':
    app.run(debug=True)
```

## Author

Hailey Quach



# Skills Network