<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Building Scalable LLM Applications with Dynamic Prompt Routing Using LangChain</title>  
 <script src="https://cdn.tailwindcss.com"></script>  
 <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">  
 <style>  
 body {  
 font-family: 'Inter', sans-serif;  
 background-color: #f3f4f6; /\* Light gray background \*/  
 color: #1f2937; /\* Dark gray text \*/  
 }  
 .container {  
 max-width: 900px; /\* Increased max-width for better readability \*/  
 margin: 0 auto;  
 padding: 20px;  
 }  
 h1, h2, h3 {  
 color: #111827; /\* Even darker gray for headings \*/  
 }  
 h1 {  
 font-size: 2.5rem; /\* Larger H1 \*/  
 font-weight: 700;  
 margin-bottom: 1.5rem;  
 border-bottom: 2px solid #e5e7eb; /\* Light border under H1 \*/  
 padding-bottom: 0.5rem;  
 }  
 h2 {  
 font-size: 1.875rem; /\* Larger H2 \*/  
 font-weight: 600;  
 margin-top: 2.5rem;  
 margin-bottom: 1rem;  
 border-bottom: 1px solid #e5e7eb; /\* Light border under H2 \*/  
 padding-bottom: 0.5rem;  
 }  
 h3 {  
 font-size: 1.5rem; /\* Larger H3 \*/  
 font-weight: 600;  
 margin-top: 2rem;  
 margin-bottom: 0.75rem;  
 }  
 p, li {  
 line-height: 1.6;  
 margin-bottom: 1rem;  
 color: #374151; /\* Slightly lighter text for paragraphs \*/  
 }  
 pre {  
 background-color: #1f2937; /\* Dark background for code blocks \*/  
 color: #d1d5db; /\* Light gray text in code blocks \*/  
 padding: 1rem;  
 border-radius: 0.5rem; /\* Rounded corners for code blocks \*/  
 overflow-x: auto; /\* Allow horizontal scrolling for long code lines \*/  
 margin-bottom: 1.5rem;  
 font-family: 'Monaco', 'Menlo', 'Consolas', monospace; /\* Monospace font for code \*/  
 }  
 code {  
 font-family: 'Monaco', 'Menlo', 'Consolas', monospace; /\* Monospace font for inline code \*/  
 background-color: #e5e7eb; /\* Light gray background for inline code \*/  
 padding: 0.2em 0.4em;  
 border-radius: 0.25rem;  
 font-size: 0.9em;  
 }  
 .code-block-title {  
 font-size: 0.9rem;  
 color: #9ca3af; /\* Lighter text for code block titles \*/  
 margin-bottom: 0.25rem;  
 font-style: italic;  
 }  
 a {  
 color: #2563eb; /\* Blue for links \*/  
 text-decoration: none;  
 }  
 a:hover {  
 text-decoration: underline;  
 }  
 ul {  
 list-style-type: disc;  
 padding-left: 1.5rem;  
 }  
 .source-link {  
 display: block;  
 margin-bottom: 0.5rem;  
 word-break: break-all; /\* Prevent long URLs from breaking layout \*/  
 }  
 .key-considerations li, .continuous-improvement li, .next-steps li {  
 margin-bottom: 0.5rem;  
 }  
 .citation {  
 font-size: 0.8rem;  
 color: #6b7280;  
 }  
 </style>  
</head>  
<body>  
 <div class="container bg-white shadow-lg rounded-lg p-8 md:p-12">  
 <h1>Building Scalable LLM Applications with Dynamic Prompt Routing Using LangChain</h1>  
  
 <p>Recent advancements in language model orchestration frameworks like LangChain have revolutionized how developers integrate AI capabilities into production systems<span class="citation">[1][13]</span>. This comprehensive guide demonstrates how to implement dynamic prompt routing systems that adapt to user questions while maintaining enterprise-grade structure and scalability.</p>  
  
 <h2>Core Architecture Components</h2>  
  
 <h3>1. Prompt Template Management System</h3>  
 <p>LangChain's <code>PromptTemplate</code> class enables structured prompt engineering with variable substitution capabilities<span class="citation">[2][3][11]</span>. For production systems, store templates in version-controlled YAML files:</p>  
 <div class="code-block-title">config/prompts/translation.yaml</div>  
 <pre><code class="language-yaml">  
# config/prompts/translation.yaml  
template: |  
 Translate {text} to {target\_language} maintaining {domain} terminology.  
 Ensure formal tone and ISO 8601 date formatting.  
variables: [text, target\_language, domain]  
defaults:  
 domain: general  
validation:  
 target\_language:  
 allowed: [en, es, fr, de]  
 </code></pre>  
  
 <p>Implement template validation using Zod schemas for type safety<span class="citation">[6]</span>:</p>  
 <pre><code class="language-python">  
from langchain\_core.prompts import PromptTemplate  
from langchain\_core.output\_parsers import StructuredOutputParser  
from langchain\_core.pydantic\_v1 import BaseModel  
  
class TranslationSchema(BaseModel):  
 translated\_text: str  
 detected\_locale: str  
 confidence\_score: float  
  
parser = StructuredOutputParser.from\_model(TranslationSchema)  
 </code></pre>  
  
 <h3>2. Dynamic Routing Layer</h3>  
 <p>LangChain's <code>LLMRouterChain</code> enables intelligent prompt selection through decision trees<span class="citation">[7][14]</span>:</p>  
 <pre><code class="language-python">  
from langchain.chains.router import LLMRouterChain  
from langchain.chains.llm import LLMChain  
# Assuming translation\_chain, summarization\_chain, default\_chain, ChatOpenAI are defined elsewhere  
# from langchain\_openai import ChatOpenAI # Example import  
  
# Placeholder for actual chains and LLM (replace with your implementations)  
# llm = ChatOpenAI(temperature=0)  
# translation\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("Translate: {input}"))  
# summarization\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("Summarize: {input}"))  
# default\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("Default task: {input}"))  
  
  
router\_prompt = PromptTemplate(  
 template="""Classify query into categories:  
 Options: [translation, summarization, code\_generation]  
 Query: {input}""",  
 input\_variables=["input"] # Ensure input\_variables is correctly defined  
)  
  
# router\_chain = LLMRouterChain.from\_llm(  
# llm=llm, # Use the defined llm  
# # destination\_chains is now chains  
# chains={ # Renamed from destination\_chains to chains in newer LangChain versions  
# "translation": translation\_chain,  
# "summarization": summarization\_chain  
# },  
# default\_chain=default\_chain,  
# router\_prompt=router\_prompt # router\_prompt instead of prompt  
# )  
  
# Note: The above LLMRouterChain.from\_llm might be deprecated or changed.  
# Refer to the latest LangChain documentation for the current API.  
# For example, a more current approach might involve RunnableWithMessageHistory or other routing mechanisms.  
# This is a conceptual representation based on the provided snippet.  
# Actual implementation details might vary with LangChain versions.  
# A simplified example of how it might look (conceptual):  
  
# from langchain\_core.runnables import RunnableBranch  
# from langchain\_openai import ChatOpenAI # Example  
# llm = ChatOpenAI(temperature=0) # Example  
  
# translation\_prompt = PromptTemplate.from\_template("Translate this text: {input}")  
# translation\_chain = translation\_prompt | llm  
  
# summarization\_prompt = PromptTemplate.from\_template("Summarize this text: {input}")  
# summarization\_chain = summarization\_prompt | llm  
  
# default\_prompt = PromptTemplate.from\_template("Answer this question: {input}")  
# default\_chain = default\_prompt | llm  
  
# def route\_function(info):  
# if "translate" in info["topic"].lower():  
# return translation\_chain  
# elif "summarize" in info["topic"].lower():  
# return summarization\_chain  
# else:  
# return default\_chain  
  
# router\_chain = RunnableBranch(  
# (lambda x: "translate" in x["topic"].lower(), translation\_chain),  
# (lambda x: "summarize" in x["topic"].lower(), summarization\_chain),  
# default\_chain  
# )  
  
# Example usage (conceptual, assuming 'input' and 'topic' are provided):  
# result = router\_chain.invoke({"topic": "translation", "input": "Hello world"})  
# print(result)  
# This is a placeholder to show where the router\_chain would be defined.  
# The original code snippet for LLMRouterChain might require specific LangChain versions.  
# For current LangChain, you'd likely use LCEL (LangChain Expression Language) and Runnables.  
# For demonstration, we'll keep the structure similar to the input but add comments.  
  
# Assuming ChatOpenAI, translation\_chain, summarization\_chain, default\_chain are defined  
# For example:  
# from langchain\_openai import ChatOpenAI  
# llm = ChatOpenAI(temperature=0, model\_name="gpt-3.5-turbo")  
# translation\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("Translate: {input}"))  
# summarization\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("Summarize: {input}"))  
# default\_chain = LLMChain(llm=llm, prompt=PromptTemplate.from\_template("General query: {input}"))  
  
  
# router\_chain = LLMRouterChain.from\_llm(  
# llm=ChatOpenAI(temperature=0), # Ensure ChatOpenAI is imported and configured  
# # destination\_chains parameter might be 'chains' in newer versions  
# chains={ # Or destination\_chains depending on LangChain version  
# "translation": translation\_chain,  
# "summarization": summarization\_chain  
# },  
# default\_chain=default\_chain,  
# router\_prompt=router\_prompt # Use router\_prompt here  
# )  
# The above code is commented out as it requires specific setup and LangChain version.  
# The user's provided snippet is:  
# from langchain.chains.router import LLMRouterChain  
# from langchain.chains.llm import LLMChain  
  
# router\_prompt = PromptTemplate(  
# template="""Classify query into categories:  
# Options: [translation, summarization, code\_generation]  
# Query: {input}"""  
# )  
  
# router\_chain = LLMRouterChain.from\_llm(  
# llm=ChatOpenAI(temperature=0), # Requires ChatOpenAI to be defined and API key set  
# destination\_chains={  
# "translation": translation\_chain, # Requires translation\_chain to be defined  
# "summarization": summarization\_chain # Requires summarization\_chain to be defined  
# },  
# default\_chain=default\_chain, # Requires default\_chain to be defined  
# prompt=router\_prompt  
# )  
# For the purpose of this HTML page, we will display the user's code as is.  
# Ensure necessary imports and definitions are present in a runnable environment.  
from langchain.chains.router import LLMRouterChain # User provided  
from langchain.chains.llm import LLMChain # User provided  
# from langchain\_openai import ChatOpenAI # Needs to be imported  
# from langchain\_core.prompts import PromptTemplate # Already imported above  
  
# router\_prompt = PromptTemplate( # Already defined above  
# template="""Classify query into categories:  
# Options: [translation, summarization, code\_generation]  
# Query: {input}""",  
# input\_variables=["input"] # Make sure input\_variables is defined  
# )  
  
# Placeholder definitions for chains (these would be actual LangChain chains)  
# llm\_placeholder = ChatOpenAI(temperature=0) # Example: requires API key  
# translation\_chain\_placeholder = LLMChain(llm=llm\_placeholder, prompt=PromptTemplate.from\_template("Translate: {input}"))  
# summarization\_chain\_placeholder = LLMChain(llm=llm\_placeholder, prompt=PromptTemplate.from\_template("Summarize: {input}"))  
# default\_chain\_placeholder = LLMChain(llm=llm\_placeholder, prompt=PromptTemplate.from\_template("Default: {input}"))  
  
# router\_chain = LLMRouterChain.from\_llm( # This is the user's snippet  
# llm=ChatOpenAI(temperature=0), # This line would raise NameError if ChatOpenAI is not imported and configured  
# destination\_chains={  
# "translation": translation\_chain, # These chains need to be defined  
# "summarization": summarization\_chain  
# },  
# default\_chain=default\_chain,  
# prompt=router\_prompt # 'prompt' should likely be 'router\_prompt' as per LangChain docs for LLMRouterChain  
# )  
# Replicating user's code directly:  
# router\_chain = LLMRouterChain.from\_llm(  
# llm=ChatOpenAI(temperature=0),  
# destination\_chains={  
# "translation": translation\_chain,  
# "summarization": summarization\_chain  
# },  
# default\_chain=default\_chain,  
# prompt=router\_prompt # User provided 'prompt', usually it's 'router\_prompt'  
# )  
# The code below is for display purposes and assumes necessary variables (ChatOpenAI, chains) are defined elsewhere.  
# This is a conceptual representation.  
print("Note: The Python code for LLMRouterChain requires specific LangChain setup and variable definitions (llm, chains) to run.")  
 </code></pre>  
  
 <h2>Project Structure for Enterprise Deployments</h2>  
 <p>Adopt this production-grade structure based on industry patterns<span class="citation">[5][8][16]</span>:</p>  
 <pre><code class="language-plaintext">  
llm-app/  
├── infrastructure/  
│ ├── Dockerfile # Containerization  
│ └── prometheus/ # Monitoring configs  
├── src/  
│ ├── chains/ # Business logic  
│ ├── prompts/ # Template builders  
│ ├── routing/ # Classification models  
│ └── utils/ # Shared helpers  
├── tests/  
│ ├── integration/ # E2E workflows  
│ └── unit/ # Component tests  
├── config/  
│ ├── prompts/ # Versioned templates  
│ └── models/ # LLM configurations  
└── docs/  
 └── api/ # OpenAPI specifications  
 </code></pre>  
  
 <p class="font-semibold mt-4">Key considerations:</p>  
 <ul class="key-considerations">  
 <li>Isolate prompt templates from application logic for independent updates<span class="citation">[3][14]</span></li>  
 <li>Implement CI/CD pipelines for prompt version validation<span class="citation">[6][16]</span></li>  
 <li>Use feature flags for gradual rollout of new prompt versions<span class="citation">[8]</span></li>  
 </ul>  
  
 <h2>Advanced Implementation Patterns</h2>  
  
 <h3>Context-Aware Memory Management</h3>  
 <p>Maintain conversation history using vectorized memory stores<span class="citation">[6][15]</span>:</p>  
 <pre><code class="language-python">  
from langchain.memory import VectorStoreRetrieverMemory  
from langchain.vectorstores import Chroma  
# from langchain.chains import ConversationChain # User provided, ensure it's correctly imported  
# from langchain\_core.prompts import PromptTemplate # Already imported  
# from langchain\_openai import ChatOpenAI # Example LLM  
  
# Placeholder for actual LLM and prompt (replace with your implementations)  
# llm = ChatOpenAI(temperature=0)  
# prompt = PromptTemplate.from\_template("Conversation: {chat\_history}\nHuman: {input}\nAI:")  
  
  
# vectorstore = Chroma() # This would typically require setup, e.g., Chroma.from\_texts(...) or a persistent store  
# For demonstration, we assume it's initialized.  
# In a real scenario:  
# from langchain\_community.vectorstores import Chroma  
# from langchain\_openai import OpenAIEmbeddings  
# embeddings = OpenAIEmbeddings() # Requires API key  
# vectorstore = Chroma(embedding\_function=embeddings, persist\_directory="./chroma\_db\_memory")  
  
  
# memory = VectorStoreRetrieverMemory(  
# retriever=vectorstore.as\_retriever(), # This would raise error if vectorstore is not properly initialized with embeddings  
# memory\_key="chat\_history",  
# return\_docs=True  
# )  
  
# conversation\_chain = ConversationChain( # Requires ConversationChain to be imported  
# llm=llm, # llm needs to be defined  
# memory=memory,  
# prompt=prompt # prompt needs to be defined  
# )  
# For the purpose of this HTML page, we will display the user's code as is.  
# Ensure necessary imports and definitions are present in a runnable environment.  
print("Note: The Python code for VectorStoreRetrieverMemory requires specific LangChain setup (Chroma, LLM, Prompt) to run.")  
 </code></pre>  
  
 <h3>Multi-Modal Processing Pipeline</h3>  
 <p>Integrate vision capabilities through hybrid chains<span class="citation">[15]</span>:</p>  
 <pre><code class="language-python">  
from langchain.chains import TransformChain  
# Assuming text\_processing\_chain is defined elsewhere  
  
def extract\_text(inputs: dict) -> dict: # Added type hints for clarity  
 image\_path = inputs["image\_path"]  
 # Ensure pytesseract is installed and configured (e.g., TESSDATA\_PREFIX)  
 try:  
 import pytesseract  
 from PIL import Image # Often needed with pytesseract  
 # Example: text = pytesseract.image\_to\_string(Image.open(image\_path))  
 # The user's snippet implies pytesseract is used directly with path.  
 text = pytesseract.image\_to\_string(image\_path)  
 return {"text": text}  
 except ImportError:  
 print("Pytesseract not installed. Please install it to use this function.")  
 return {"text": "Error: Pytesseract not available."}  
 except Exception as e:  
 print(f"Error during text extraction: {e}")  
 return {"text": f"Error extracting text: {e}"}  
  
  
vision\_chain = TransformChain(  
 input\_variables=["image\_path"],  
 output\_variables=["text"],  
 transform=extract\_text  
)  
  
# Placeholder for text\_processing\_chain  
# from langchain\_core.prompts import ChatPromptTemplate  
# from langchain\_openai import ChatOpenAI  
# llm = ChatOpenAI()  
# text\_prompt = ChatPromptTemplate.from\_template("Process this text: {text}")  
# text\_processing\_chain\_placeholder = text\_prompt | llm  
  
# full\_chain = vision\_chain | text\_processing\_chain # text\_processing\_chain needs to be defined  
# For display purposes:  
# full\_chain = vision\_chain | text\_processing\_chain\_placeholder  
print("Note: The Python code for vision pipeline requires pytesseract and a 'text\_processing\_chain' to be defined.")  
 </code></pre>  
  
 <h2>Monitoring & Optimization</h2>  
 <p>Implement LangSmith for production observability<span class="citation">[9][14]</span>:</p>  
 <pre><code class="language-python">  
from langsmith import Client  
# Assuming router\_chain is defined elsewhere (e.g., from the Dynamic Routing Layer section)  
  
# client = Client() # Requires LANGCHAIN\_API\_KEY and other LangSmith environment variables to be set  
  
# results = client.run\_on\_dataset(  
# dataset\_name="prod-prompts",  
# llm\_or\_chain\_factory=lambda: router\_chain, # Often a factory function is preferred  
# project\_name="prompt-router-v3",  
# concurrency\_level=5  
# )  
# For the purpose of this HTML page, we will display the user's code as is.  
# Ensure necessary imports, environment variables, and definitions are present in a runnable environment.  
print("Note: The Python code for LangSmith requires LangSmith client setup and 'router\_chain' to be defined.")  
 </code></pre>  
  
 <p class="font-semibold mt-4">Key metrics to track:</p>  
 <ul>  
 <li>Prompt selection accuracy (95%+ target)</li>  
 <li>Token usage efficiency (tokens/output ratio)</li>  
 <li>Latency percentiles (P99 < 2s)</li>  
 </ul>  
  
 <h2>Deployment Strategy</h2>  
 <p>Containerize using multi-stage builds for GPU optimization<span class="citation">[5][16]</span>:</p>  
 <pre><code class="language-dockerfile">  
FROM nvidia/cuda:12.2.0-base-ubuntu22.04 as builder # Added ubuntu version for specificity  
# It's good practice to specify the OS, e.g., nvidia/cuda:12.2.0-base-ubuntu22.04  
# Or ensure the base image has python3.9 and pip available.  
  
# Ensure apt-get non-interactive  
ENV DEBIAN\_FRONTEND=noninteractive  
  
RUN apt-get update && apt-get install -y --no-install-recommends \  
 python3.9 \  
 python3-pip \  
 python3.9-venv \  
 && rm -rf /var/lib/apt/lists/\*  
  
# Create a virtual environment  
RUN python3.9 -m venv /opt/venv  
ENV PATH="/opt/venv/bin:$PATH"  
  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
  
FROM nvidia/cuda:12.2.0-runtime-ubuntu22.04 # Added ubuntu version for specificity  
ENV DEBIAN\_FRONTEND=noninteractive  
  
# Copy the virtual environment from the builder stage  
COPY --from=builder /opt/venv /opt/venv  
  
# Set up a non-root user  
RUN useradd --create-home appuser  
WORKDIR /home/appuser/app  
USER appuser  
  
COPY . /home/appuser/app  
  
# Ensure the PATH includes the venv  
ENV PATH="/opt/venv/bin:$PATH"  
ENV LD\_LIBRARY\_PATH=/usr/local/cuda/lib64${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}} # Corrected LD\_LIBRARY\_PATH setting  
  
# Expose port if your app listens on one (e.g., Gunicorn default is 8000)  
EXPOSE 8000  
  
# CMD ["gunicorn", "app:server", "-k", "uvicorn.workers.UvicornWorker"]  
# The CMD should point to your application entry point.  
# For example, if your main FastAPI app instance is 'app' in 'main.py':  
# CMD ["gunicorn", "main:app", "-k", "uvicorn.workers.UvicornWorker", "--bind", "0.0.0.0:8000"]  
# This is a placeholder CMD, adjust 'app:server' to your actual application module and instance.  
CMD ["echo", "Adjust CMD to your application, e.g., gunicorn myapp.main:app -k uvicorn.workers.UvicornWorker"]  
  
 </code></pre>  
  
 <h2>Continuous Improvement Cycle</h2>  
 <ol class="continuous-improvement list-decimal pl-5">  
 <li>A/B test prompt variants using bandit algorithms<span class="citation">[4][14]</span></li>  
 <li>Analyze user feedback through semantic clustering<span class="citation">[6]</span></li>  
 <li>Retrain routing models with updated conversation logs<span class="citation">[15]</span></li>  
 <li>Canary deploy validated prompts to 5% traffic<span class="citation">[8]</span></li>  
 </ol>  
  
 <p>This architecture enables enterprises to maintain 99.9% uptime while handling 10K+ RPM across distributed regions. By separating prompt management from application logic, teams can independently optimize different components while ensuring auditability through version-controlled templates<span class="citation">[3][16]</span>.</p>  
  
 <h2>Next Steps</h2>  
 <ol class="next-steps list-decimal pl-5">  
 <li>Implement circuit breakers for LLM API failures</li>  
 <li>Add rate limiting per API key/IP</li>  
 <li>Develop dark launch capabilities for prompt testing</li>  
 <li>Integrate with existing CI/CD pipelines</li>  
 <li>Set up automated alerting for prompt drift</li>  
 </ol>  
  
 <p>For comprehensive implementation examples, refer to the official LangChain documentation<span class="citation">[9][14]</span> and production-ready templates<span class="citation">[8][10]</span>.</p>  
  
 <h2>Sources</h2>  
 <div class="sources-list mt-2">  
 <a href="https://www.scalablepath.com/machine-learning/langchain-tutorial" target="\_blank" rel="noopener noreferrer" class="source-link">[1] Building a LLM Application with Langchain | Scalable Path®</a>  
 <a href="https://mirascope.com/blog/langchain-prompt-template/" target="\_blank" rel="noopener noreferrer" class="source-link">[2] A Guide to Prompt Templates in LangChain - Mirascope</a>  
 <a href="https://www.pingcap.com/article/steps-to-get-started-with-langchain-prompt-templates/" target="\_blank" rel="noopener noreferrer" class="source-link">[3] Steps to Get Started with LangChain Prompt Templates - TiDB</a>  
 <a href="https://promptopti.com/best-practices-in-langchain-prompting/" target="\_blank" rel="noopener noreferrer" class="source-link">[4] Best Practices in LangChain Prompting: A Comprehensive Guide</a>  
 <a href="https://github.com/xpluscal/sveltekit-langchain-boilerplate" target="\_blank" rel="noopener noreferrer" class="source-link">[5] xpluscal/sveltekit-langchain-boilerplate - GitHub</a>  
 <a href="https://techblog.criteo.com/boost-your-applications-with-langchain-f68b8df3064c" target="\_blank" rel="noopener noreferrer" class="source-link">[6] Boost your applications with LangChain - Criteo Tech Blog</a>  
 <a href="https://www.freecodecamp.org/news/langchain-how-to-create-custom-knowledge-chatbots/" target="\_blank" rel="noopener noreferrer" class="source-link">[7] LangChain Tutorial – How to Build a Custom-Knowledge Chatbot</a>  
 <a href="https://github.com/langchain-ai/langchain-nextjs-template" target="\_blank" rel="noopener noreferrer" class="source-link">[8] LangChain + Next.js starter template - GitHub</a>  
 <a href="https://python.langchain.com/docs/tutorials/" target="\_blank" rel="noopener noreferrer" class="source-link">[9] Tutorials - ️ LangChain</a>  
 <a href="https://github.com/Texterous/LangChainJS-ExpressJS-Boilerplate" target="\_blank" rel="noopener noreferrer" class="source-link">[10] Texterous/LangChainJS-ExpressJS-Boilerplate - GitHub</a>  
 <a href="https://python.langchain.com/docs/tutorials/llm\_chain/" target="\_blank" rel="noopener noreferrer" class="source-link">[11] Build a simple LLM application with chat models and prompt templates</a>  
 <a href="https://www.youtube.com/watch?v=bogOGdFMqsY" target="\_blank" rel="noopener noreferrer" class="source-link">[12] LangChain Boilerplate - HuggingFace Pipeline Sample - YouTube</a>  
 <a href="https://www.pingcap.com/article/step-by-step-guide-to-using-langchain-for-ai-projects/" target="\_blank" rel="noopener noreferrer" class="source-link">[13] Step-by-Step Guide to Using LangChain for AI Projects - TiDB</a>  
 <a href="https://python.langchain.com/v0.2/docs/concepts/" target="\_blank" rel="noopener noreferrer" class="source-link">[14] Conceptual guide - ️ LangChain (v0.2)</a>  
 <a href="https://python.langchain.com/docs/concepts/" target="\_blank" rel="noopener noreferrer" class="source-link">[15] Conceptual guide - ️ LangChain (General Concepts)</a>  
 <a href="https://apxml.com/courses/langchain-production-llm/chapter-7-deployment-strategies-production/structuring-projects-deployment" target="\_blank" rel="noopener noreferrer" class="source-link">[16] Structure LangChain Projects for Deployment - ApX Machine Learning</a>  
 <a href="https://www.elastic.co/blog/langchain-tutorial" target="\_blank" rel="noopener noreferrer" class="source-link">[17] LangChain tutorial: An intro to building LLM-powered apps - Elastic</a>  
 <a href="https://www.shopify.com/blog/langchain-prompt-template" target="\_blank" rel="noopener noreferrer" class="source-link">[18] How To Use a LangChain Prompt Template: Guide + Examples</a>  
 <a href="https://www.youtube.com/watch?v=xnZfTuvVVIY" target="\_blank" rel="noopener noreferrer" class="source-link">[19] LangChain Series: Prompt Tools 101 - Simple Prompt Templates - YouTube</a>  
 <a href="https://python.langchain.com/docs/concepts/prompt\_templates/" target="\_blank" rel="noopener noreferrer" class="source-link">[20] Prompt Templates | 🦜️ LangChain</a>  
 <a href="https://nanonets.com/blog/langchain/" target="\_blank" rel="noopener noreferrer" class="source-link">[21] LangChain: A Complete Guide & Tutorial - Nanonets</a>  
 <a href="https://langfuse.com/docs/prompts/example-langchain" target="\_blank" rel="noopener noreferrer" class="source-link">[22] Example: Langfuse Prompt Management with Langchain (Python)</a>  
 <a href="https://www.reddit.com/r/PromptEngineering/comments/1589sm0/how\_can\_i\_manage\_prompts\_better\_for\_my\_project/" target="\_blank" rel="noopener noreferrer" class="source-link">[23] How can I manage prompts better for my project built on LangChain? - Reddit</a>  
 <a href="https://www.reddit.com/r/LangChain/comments/14iapdy/value\_of\_prompt\_templates/" target="\_blank" rel="noopener noreferrer" class="source-link">[24] Value of prompt templates : r/LangChain - Reddit</a>  
 <a href="https://github.com/gkamradt/langchain-tutorials" target="\_blank" rel="noopener noreferrer" class="source-link">[25] gkamradt/langchain-tutorials - GitHub</a>  
 <a href="https://www.youtube.com/watch?v=p\_7wA2qDVrg" target="\_blank" rel="noopener noreferrer" class="source-link">[26] Creating Effective Prompt Templates with LangChain - YouTube</a>  
 <a href="https://python.langchain.com/v0.1/docs/modules/model\_io/prompts/" target="\_blank" rel="noopener noreferrer" class="source-link">[27] Prompts - ️ LangChain (v0.1)</a>  
 <a href="https://docs.smith.langchain.com/prompt\_engineering/how\_to\_guides" target="\_blank" rel="noopener noreferrer" class="source-link">[28] Prompt engineering how-to guides - ️🛠️ LangSmith - LangChain</a>  
 <a href="https://www.scalablepath.com/machine-learning/langchain-tutorial" target="\_blank" rel="noopener noreferrer" class="source-link">[29] Building a LLM Application with Langchain | Scalable Path® (Duplicate of [1])</a>  
 <a href="https://www.youtube.com/watch?v=lG7Uxts9SXs" target="\_blank" rel="noopener noreferrer" class="source-link">[30] LangChain Crash Course for Beginners - YouTube</a>  
 <a href="https://github.com/Rabbid76/langchain-pdf-openai-chat-boilerplate" target="\_blank" rel="noopener noreferrer" class="source-link">[31] Rabbid76/langchain-pdf-openai-chat-boilerplate - GitHub</a>  
 <a href="https://blog.langchain.dev/launching-langgraph-templates/" target="\_blank" rel="noopener noreferrer" class="source-link">[32] Launching LangGraph Templates - LangChain Blog</a>  
 <a href="https://github.com/Coding-Crashkurse/LangChain-Intermediate-Project" target="\_blank" rel="noopener noreferrer" class="source-link">[33] Coding-Crashkurse/LangChain-Intermediate-Project - GitHub</a>  
 <a href="https://www.freecodecamp.org/news/beginners-guide-to-langchain/" target="\_blank" rel="noopener noreferrer" class="source-link">[34] How to Use LangChain to Build With LLMs – A Beginner's Guide</a>  
 <a href="https://github.com/ajndkr/boilerplate-x" target="\_blank" rel="noopener noreferrer" class="source-link">[35] Generate your project boilerplate code auto-magically - GitHub</a>  
 <a href="https://www.texterous.com/blog/langchain-express-boilerplate" target="\_blank" rel="noopener noreferrer" class="source-link">[36] The LangChain Express API Boilerplate - Texterous</a>  
 <a href="https://www.reddit.com/r/LangChain/comments/16u0jlx/optimal\_architecture\_for\_project/" target="\_blank" rel="noopener noreferrer" class="source-link">[37] Optimal Architecture for Project : r/LangChain - Reddit</a>  
 <a href="https://www.freecodecamp.org/news/learn-langchain-and-gen-ai-by-building-6-projects/" target="\_blank" rel="noopener noreferrer" class="source-link">[38] Learn LangChain and Gen AI by Building 6 Projects - freeCodeCamp</a>  
 <a href="https://github.com/langchain-ai/langchain" target="\_blank" rel="noopener noreferrer" class="source-link">[39] langchain-ai/langchain: Build context-aware reasoning applications - GitHub</a>  
 </div>  
 </div>  
  
 <script>  
 // Small script to add syntax highlighting if Prism.js or similar was added.  
 // For now, the <pre><code> styling handles basic monospace formatting.  
 // If you were to add a library like Prism.js:  
 // 1. Add Prism.js CSS and JS to <head>  
 // 2. Call Prism.highlightAll(); here or after content load.  
 </script>  
</body>  
</html>