1.2 PROMPT PATTERNS & TEMPLATES

Understanding how to structure prompts effectively is crucial for getting reliable, high-quality outputs from LLMs. This module covers common prompt patterns and how to build reusable templates.

LEARNING OBJECTIVES

By the end of this module, you will:

- Recognize common prompt patterns and when to use them
- Understand why structure matters in prompt engineering
- Know how to create reusable prompt templates
- See how the accelerator uses prompts in production

CORE PROMPT PATTERNS

1. INSTRUCTION PROMPTS

The simplest pattern: give the model a clear instruction.

Structure:

```
[Instruction]
```

Examples:

```
Summarize this text in 3 sentences.

Extract all email addresses from the following document.

Translate this paragraph to French.
```

Best for: - Simple, well-defined tasks - When the model already knows what to do - Single-step operations

Tips: - Be specific and direct - Use action verbs (summarize, extract, translate, list) - Specify output format if needed

2. FEW-SHOT EXAMPLES

Provide examples of the task before asking the model to do it.

Structure:

```
[Instruction]
```

PROMPT DESIGN PRINCIPLES

1. CLARITY AND SPECIFICITY

Bad:

```
Tell me about AI.
```

Good:

```
Explain the difference between supervised and unsupervised machine learning
in 3 paragraphs, with one example of each.
```

Why it matters: Vague prompts lead to vague, unfocused responses.

2. ROLE AND PERSONA

Give the model a role to frame its responses.

Structure:

```
You are a [role] with expertise in [domain].

[Task]
```

Examples:

```
You are a senior software architect with 15 years of experience in distributed systems.

Design a scalable architecture for a real-time chat application.
```

PROMPT TEMPLATES

WHAT IS A TEMPLATE?

A **prompt template** is a reusable pattern with placeholders for variable content.

Benefits: - **Consistency**: Same structure every time - **Maintainability**: Update once, apply everywhere - **Testability**: Easier to evaluate prompt changes - **Scalability**: Supports batch processing

SIMPLE PYTHON TEMPLATES

Using f-strings:

```
def summarize(text: str, length: int = 3) -> str:
    prompt = f"""Summarize the following text in {length} sentences:

{text}

Summary:"""
    return llm.generate(prompt)
```

Using str.format():

```
TEMPLATE = """You are a helpful assistant.

Task: {task}

Input: {input_text}
```

HOW THE ACCELERATOR USES PROMPTS

The accelerator centralizes prompt logic in `rag/prompt.py`:

CURRENT STRUCTURE

```
# accelerator/rag/prompt.py

SYSTEM = """You are a careful and accurate assistant.
You answer questions based on provided context.
If you cannot find the answer in the context, say so."""

USER_TEMPLATE = """Context:
{context}

Question: {question}

Answer:"""
```

ON DAY 2-3, YOU'LL EXTEND THIS

Multi-turn conversations:

```
CHAT_TEMPLATE = """You are a helpful assistant. Use the following context to answer questions.

Context:
{context}

Conversation history:
{history}
```

REFERENCE NOTEBOOKS

RAG PROMPT EXAMPLES

`use-watsonx-chroma-and-langchain-to-answer-questions-rag.ipynb`:

```
# Example from the notebook
from langchain.prompts import PromptTemplate

rag_prompt = PromptTemplate(
    template="""Use the following pieces of context to answer the question at the end.
If you don't know the answer, just say that you don't know, don't try to make up an answer.

{context}

Question: {question}
Helpful Answer:""",
    input_variables=["context", "question"],
)
```

`QnA_with_RAG.ipynb` (accelerator): - Shows how context is concatenated - Demonstrates citation patterns - Includes error handling for edge cases

KEY OBSERVATIONS

1. **Context injection**: Always happens before the question
2. **Structured formats**: JSON/XML for tool calling
3. **Safety prompts**: Explicitly state boundaries

EXAMPLES TO REUSE IN LAB 1.2

EXAMPLE 1: SUMMARIZATION

Template:

```
SUMMARIZE_TEMPLATE = """Summarize the following text in {num_sentences} sentences.
Focus on the main points and key takeaways.

Text:
{text}

Summary:"""
```

Usage:

```
prompt = SUMMARIZE_TEMPLATE.format(
    num_sentences=3,
    text="[Your long text here]"
)
```

EXAMPLE 2: STYLE TRANSFER

Template:

```
REWRITE_TEMPLATE = """Rewrite the following text in a {target_tone} tone:

Original text:
{original}
```

CONNECTION TO LABS

LAB 1.2: PROMPT TEMPLATES

In this lab, you'll:

1. **Create templates** for:
   - Summarization
   - Style rewrite
   - Q&A with context
2. **Implement in both environments**:
   - `prompt_patterns_ollama.ipynb` (local)
   - `prompt_patterns_watsonx.ipynb` (managed)
3. **Compare results**:
   - Same template, different models
   - Measure quality and consistency

LOOKING AHEAD

Day 2: These templates become the foundation for RAG prompts **Day 3**: Templates extended for multi-turn agents and tool calling

BEST PRACTICES SUMMARY

✅ DO

- Start with simple, clear instructions
- Use few-shot examples for structured outputs
- Specify output format explicitly
- Test prompts with edge cases
- Version control your templates
- Measure prompt performance systematically

❌ DON’T

- Use vague or ambiguous language
- Mix multiple tasks in one prompt
- Assume the model knows your context
- Forget to handle error cases
- Over-engineer prompts prematurely

KEY TAKEAWAYS

- **Prompts are code**: Treat them with the same rigor as application code
- **Structure matters**: Well-structured prompts are more reliable
- **Templates enable scale**: Reusable patterns save time and improve consistency
- **Test and iterate**: Prompt engineering is empirical—what works for one task may not work for another

Next: Time to build these patterns hands-on in Lab 1.2!