Homework 8: Prompt Engineering

Points: 20 | **Due:** Sunday, April 5, 2026 @ 11pm Pacific

Author: Richard Young, Ph.D. | UNLV Lee Business School

Compute: CPU (free tier)

Learning Objectives

1. **Master** core prompt engineering techniques
2. **Apply** zero-shot, few-shot, and chain-of-thought prompting
3. **Evaluate** prompt effectiveness systematically
4. **Optimize** prompts for specific business tasks
5. **Document** prompt iterations and improvements

Why This Matters for Business

10x Productivity: GitHub reports that developers using Copilot with well-crafted prompts complete tasks 55% faster. The difference between a good and bad prompt can be hours of saved work.

Cost Control: OpenAI charges by token. A senior engineer at Stripe found that optimizing prompts reduced their API costs by 40% while improving output quality—prompt engineering is a business skill.

Consistency at Scale: Walmart uses templated prompts to generate thousands of product descriptions. Poorly engineered prompts produce inconsistent, unusable content. Good prompts produce publishable copy.

Competitive Advantage: Companies like Jasper AI built billion-dollar businesses on prompt engineering expertise. The LLM is the same for everyone; the prompts are the secret sauce.

Grading

Component	Points	Effort	What We're Looking For
Zero-Shot Prompts	3	*	Clear task instructions
Few-Shot Prompts	5	**	Well-chosen examples
Chain-of-Thought	5	**	Reasoning improves output
Prompt Optimization	4	**	Systematic improvement
Documentation	3	*	Clear iteration history
Total	**20**		

Effort Key: * Straightforward | ** Requires thinking | *** Challenge

---

The Big Picture

Prompt engineering is the art and science of communicating effectively with LLMs.

Technique	Description	When to Use
Zero-shot	Just the task, no examples	Simple, common tasks
Few-shot	Task + examples	Complex or unusual tasks
Chain-of-thought	“Think step by step”	Reasoning, math, logic
Role prompting	“You are an expert…”	Domain-specific tasks
Output formatting	“Return as JSON…”	Structured outputs

Instructions

1. Open `MIS769_HW8_Prompt_Engineering.ipynb` in Google Colab
2. Start with a business task (sentiment analysis, summarization, etc.)
3. Write a zero-shot prompt and evaluate results
4. Add few-shot examples and compare
5. Apply chain-of-thought for complex tasks
6. Optimize your best prompt through iteration
7. Document what worked, what didn’t, and why

What Your Output Should Look Like

Zero-Shot Example:

```
□  ZERO-SHOT PROMPT
============================================================
Prompt: "Classify the sentiment of this review as positive,
negative, or neutral: 'The product arrived late but works great.'"

Response: "Neutral"

Evaluation: Correct! Simple classification works zero-shot.
```

Few-Shot Example:

```
□  FEW-SHOT PROMPT
============================================================
Prompt:
"Classify customer complaints into categories.

Example 1:
Complaint: "My order never arrived"
Category: Shipping

Example 2:
Complaint: "The product broke after one day"
```

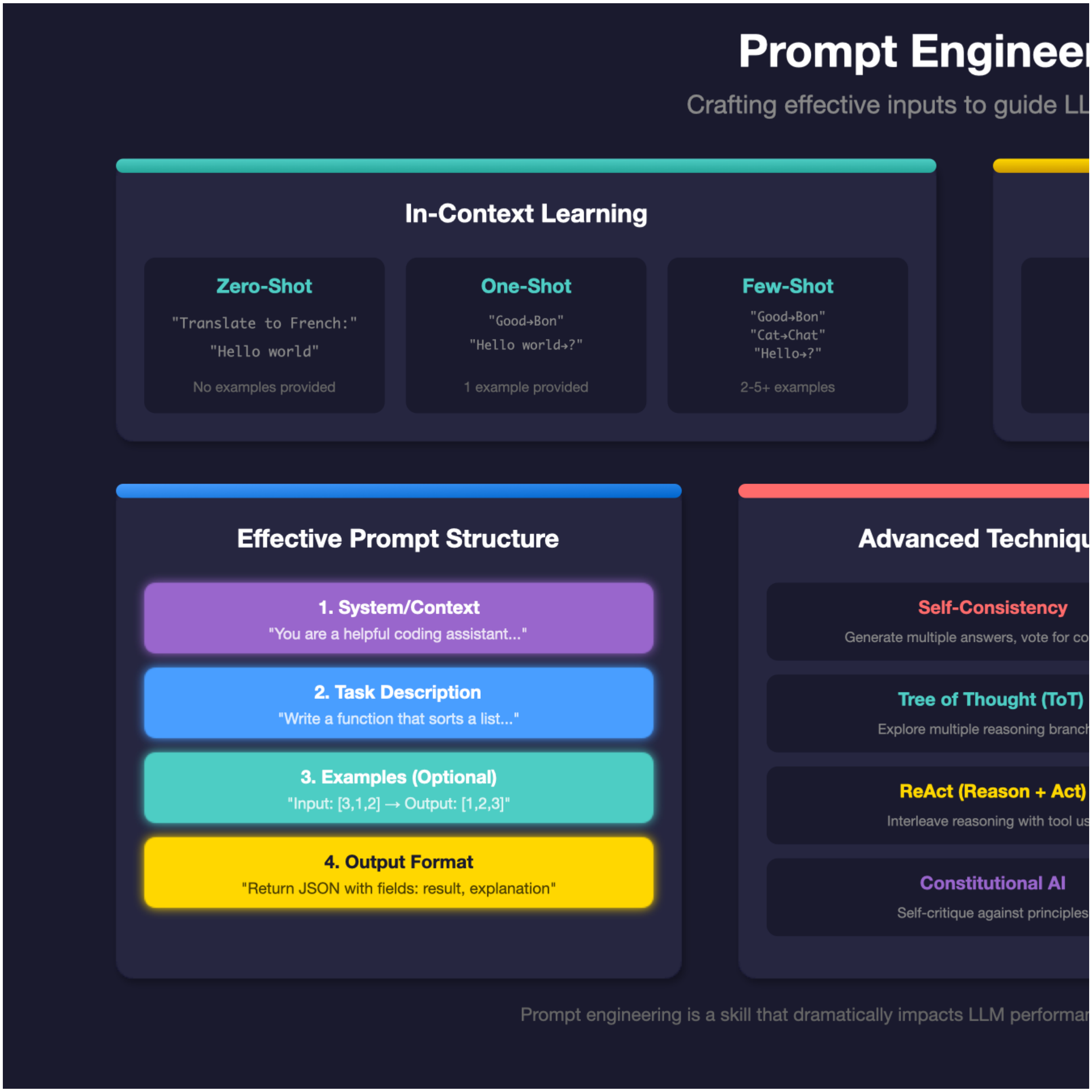

Figure 1: Prompt Engineering

```
Category: Quality

Example 3:
Complaint: "I was charged twice"
Category: Billing

Now classify:
Complaint: "The app keeps crashing when I try to checkout"
Category:"

Response: "Technical"

Evaluation: ✓ Correctly inferred a new category from patterns
```

Chain-of-Thought Example:

```
  CHAIN-OF-THOUGHT PROMPT
============================================================
Task: Determine if this review is from a verified purchaser.

Without CoT:
"Review: 'Best phone ever, been using it for 6 months daily'"
Response: "Cannot determine" ✗

With CoT:
"Think step by step:
1. Does the review mention specific usage duration? Yes, 6 months
2. Does it describe personal experience? Yes, 'been using it'
3. Are there specific details only a user would know? Daily usage
Conclusion: Likely verified purchaser based on specific,
extended usage details." ✓
```

Prompt Iteration Log:

```
  PROMPT OPTIMIZATION LOG
============================================================
Task: Extract product features from reviews

V1: "List the features mentioned"
   Accuracy: 45% | Issue: Too vague, missed context

V2: "List product features as bullet points"
   Accuracy: 62% | Issue: Included opinions, not just features

V3: "Extract only factual product features (not opinions).
    Format: - Feature: [feature name]"
   Accuracy: 84% | Better! But still some opinions

V4: "Extract factual product specifications and features.
    Exclude subjective opinions like 'great' or 'terrible'.
    Format: - [Feature]: [Value/Description]"
   Accuracy: 91% ✓ Best version
```

Common Mistakes (and How to Avoid Them)

Mistake	Symptom	Fix
Vague instructions	Inconsistent outputs	Be specific: "in 3 bullet points"
No output format	Unparseable responses	Specify: "Return as JSON"
Too many examples	High cost, diminishing returns	3-5 examples usually enough
Wrong examples	Model learns wrong patterns	Ensure examples match task
Missing edge cases	Fails on unusual inputs	Include diverse examples
No temperature control	Random outputs	Use temp=0 for consistency

If outputs are inconsistent: - Set `temperature=0` for deterministic outputs - Add "You must…" constraints - Provide explicit format templates

If the model refuses: - Rephrase as a helpful task - Add context for why the task is legitimate - Break into smaller, less ambiguous steps

---

Questions to Answer

- **Q1:** Which prompting technique worked best for your task? Why?
- **Q2:** Show your prompt iteration history. What improved results most?
- **Q3:** When did few-shot outperform zero-shot? When didn't it matter?
- **Q4:** How would you A/B test prompts in a production system?

Going Deeper (Optional Challenges)

Challenge A: Prompt Injection Defense

Write prompts that are robust to injection attacks. Test with adversarial inputs like "Ignore previous instructions and…" How can you make your prompts more secure?

Challenge B: Automatic Prompt Optimization

Implement a simple prompt optimizer that tries variations and scores outputs. Can you automate finding the best prompt?

Challenge C: Multi-Step Prompting

Build a pipeline where one LLM call's output feeds into another's input. Example: Extract □ Classify □ Summarize. When does chaining help?

Quick Reference

```python
# Setup (using OpenAI as example)
from openai import OpenAI
client = OpenAI()

def prompt(text, system="You are a helpful assistant.", temp=0):
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        temperature=temp,
        messages=[
            {"role": "system", "content": system},
            {"role": "user", "content": text}
        ]
    )
    return response.choices[0].message.content

# ZERO-SHOT
result = prompt("Summarize this article in 3 bullet points: ...")

# FEW-SHOT
few_shot = """Classify the sentiment:

Text: "Love it!" → Positive
Text: "Terrible product" → Negative
Text: "It's okay" → Neutral

Text: "Could be better but not bad" →"""

result = prompt(few_shot)

# CHAIN-OF-THOUGHT
cot = """Solve this step by step:

A store has 50 apples. They sell 23 and receive a shipment of 15.
How many apples do they have now?

Let's think step by step:"""

result = prompt(cot)

# ROLE PROMPTING
result = prompt(
    "Review this contract for risks: ...",
    system="You are an experienced corporate lawyer specializing in contract law."
)

# STRUCTURED OUTPUT
```

```
json_prompt = """Extract information as JSON:

Review: "Great battery life, but the screen is too dim"

Return format:
{
  "positive_features": [...],
  "negative_features": [...],
  "overall_sentiment": "positive|negative|neutral"
}"""

result = prompt(json_prompt)
```

Prompt Engineering Patterns: | Pattern | Example | Use Case | |———|———|———-| | Role | "You are a senior data scientist…" | Domain expertise | | Format | "Return as markdown table…" | Structured output | | Constraint | "In exactly 3 sentences…" | Length control | | Reasoning | "Explain your reasoning…" | Transparency | | Negative | "Do NOT include opinions…" | Avoid unwanted content | | Example | "Like this: [example]" | Clarify expectations |

Submission

Upload to Canvas: - Your completed `.ipynb` notebook with all cells executed

---

— *Richard Young, Ph.D.*