COT 6930: Generative Artificial Intelligence and Software Development Lifecycles

- Instructor: Dr. Fernando Koch
- TA: Sumedh Krishna

Exercise 3 - Prompt Engineering Lab

Apply advanced Prompt Engineering techniques to later phases of the Software Development Life Cycle (SDLC).

In this exercise you will:

- Select 3 techniques from the list of 16 Prompt Engineering strategies.
- Use them to simulate prompts for System Design (requirements → architecture) and Solution Design (architecture → code structure).
- Construct Meta-Prompts that augment a raw user request into a structured Augmented Prompt.
- Run the augmented prompts through model_request with explicit parameters.
- Compare outputs across techniques — which produced the clearest, most actionable system or code design?

Notes:

- Working Environment Architectures
- We will can work with Ollama inside Google Collab.
- You can select any Model to work
- I strongly recommend you have this installation running either on your computer or on a VM for performance and to support future exercises.

(Part I) WORKING ENVIRONMENT

Assuming we will be using '(C1) All Google Collab' (see document above) as the working enviroment for this exercise:

- (Step 1) Install Ollama
- (Step 2) Install working Models
- (Step 3) Prepare Model Provider

Double-click (or enter) to edit

```
!pip install colab-xterm
%load_ext colabxterm
```

(Step 1) Install Ollama (of required)

To get started with Ollama, we will need to install it using the official installation script.

1. Launch the xterm terminal within our Colab cell using the command `%xterm`
2. Install Ollama using this command within the `%xterm`

```
curl https://ollama.ai/install.sh | sh
```

3, Then execute the Ollama server

```
ollama serve &
```

```
motd_content = """
Welcome to your Colab Environment!

Instructions:
1. Launch the xterm terminal using %xterm
2. Install Ollama: `curl https://ollama.ai/install.sh | sh`
3. Execute the Ollama server: `ollama serve &`
"""
print(motd_content)

%xterm
```

(Step 2) Install the Working Models

- `ollama pull [model_name]`: Downloads a model from the Ollama library.
- `ollama list`: Displays models available on the local Ollama server.

List of available models: https://ollama.com/library

```
# We will be working with `tinyllama`
# https://ollama.com/library/tinyllama
#
# INSTALL ANY OTHER MODEL YOU WANT TO TRY OUT HERE!
#
!ollama pull tinyllama
```

```
# This will list all installed models
!ollama list
```

(Step 3) Prepare Model Provider

```
##
# Install the Ollama Client
!pip install ollama
from ollama import Client
```

```
##
```

```
# Create the re-usable function ``model_request(.)``
from ollama import Client

# Default model
DEFAULT_MODEL = "tinyllama"

# Ollama client
client = Client()

def model_request(
    prompt,
    model=DEFAULT_MODEL,
    temperature=0.7,
    top_k=40,
    top_p=0.9,
    num_predict=200,
    context_window=2048
):
    """
    Call the model client with configurable parameters.

    Args:
        prompt (str | list[str]): The user prompt(s).
        model (str): Model name to call (defaults to DEFAULT_MODEL).
        temperature (float): Controls randomness (higher = more random).
        top_k (int): Number of candidates considered at each step.
        top_p (float): Nucleus sampling threshold.
        num_predict (int): Maximum number of tokens to generate.
        context_window (int): Max number of tokens considered from prior c

    Returns:
        tuple: (model's response text, number of tokens used).
    """

    # Wrap prompt(s) into messages
    messages = [{"role": "user", "content": p} for p in ([prompt] if isins

    response = client.chat(
        model=model,
        messages=messages,
        options={
            "temperature": temperature,
            "top_k": top_k,
            "top_p": top_p,
            "num_predict": num_predict,
```

```
                "num_ctx": context_window
            }
        )

        text = response["message"]["content"]
        tokens_used = response.get("eval_count", len(text.split()))

        return text, tokens_used
```

(PART II) EXPERIMENTS

🧪 Experiment 1 - Prompt Augmentation for Problem Ideation

Simulate a user talking to the Solution Design Bot for help with Problem Ideation about a product. Each variation uses a different prompt design technique from the fundamentals document.

Remember that you are creating Meta-Prompts for Prompt Augmentation:

```
User (Prompt)->
  Bot ->
  AI Pipeline ->
  Prompt Augmentation (Meta-Prompts, Augmented Prompt) ->
  Model Request (Augmentated Prompt, Parameters)
```

```
# --- Step 1: Define raw User Prompt ---
# NOTE: REPLACE the 'target solution' with your proposed project!

USER_PROMPT = "Help me brainstorm problems for an AI-powered note-taking a
#USER_PROMPT = "Help me brainstorm problems for an AI-powered {{YOUR SOLU
print("User Prompt:\n", USER_PROMPT)
```

```
User Prompt:
 Help me brainstorm problems for an AI-powered note-taking app for univers
```

```
# --- Step 2: Apply a Prompt Augmentation Technique ---
# Choose ONE technique (Persona / Reflection / Alternative Approaches)

# Example: Persona Prompting
META_PROMPT = f"""
Act as a senior product strategist.
Your task is to help a team brainstorm problem statements for this idea:
{USER_PROMPT}
List at least 5 problem statements, each with:
- The user pain point
- Assumptions made
- A clarifying question
""".strip()
```

```
# Example: Reflection Prompting
# META_PROMPT = f"""
# Generate 5 problem statements for:
# {USER_PROMPT}
#
# Then review your own output:
# - Identify weaknesses, gaps, or biases
# - Suggest 2 improvements or alternative framings
# """.strip()
```

```
# Example: Alternative Approaches Prompting
# META_PROMPT = f"""
# Frame problems for:
# {USER_PROMPT}
#
# Produce three alternative perspectives:
# 1. From students
# 2. From instructors
# 3. From administrators
#
# For each, list 2–3 problem statements.
# """.strip()
```

```
# --- Step 3: Construct Augmented Prompt ---
AUGMENTED_PROMPT = META_PROMPT
print("\n--- Augmented Prompt ---\n")
print(AUGMENTED_PROMPT)
```

```
--- Augmented Prompt ---

Act as a senior product strategist.
Your task is to help a team brainstorm problem statements for this idea:
Help me brainstorm problems for an AI-powered note-taking app for universi
List at least 5 problem statements, each with:
- The user pain point
- Assumptions made
- A clarifying question
```

```
# --- Step 4: Call the model with all parameters explicitly set ---
response, tokens = model_request(
    prompt=AUGMENTED_PROMPT,
    model=DEFAULT_MODEL,   # or override with a different model name
    temperature=0.7,       # controls randomness (0=deterministic, >1 = m
    top_k=40,              # number of candidates considered at each step
    top_p=0.9,             # nucleus sampling threshold
    num_predict=1000,      # max tokens to generate
    context_window=2048    # size of context window
)

print("\n--- Bot Response ---\n")
print(response)
print(f"\n[Tokens used: {tokens}]")
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call
last)
/tmp/ipython-input-2292146706.py in <cell line: 0>()
      1 # --- Step 4: Call the model with all parameters explicitly set --
-
----> 2 response, tokens = model_request(
      3     prompt=AUGMENTED_PROMPT,
      4     model=DEFAULT_MODEL,   # or override with a different model
name
      5     temperature=0.7,       # controls randomness (0=deterministic,
```

🧪 Experiment 2 — Prompt Augmentation for Solution Ideation

```
# --- Step 1: Define raw User Prompt ---
# NOTE: REPLACE the 'target solution' with your proposed project!
USER_PROMPT = "Help me brainstorm solution ideas for an AI-powered note-ta
#USER_PROMPT = "Help me brainstorm solution ideas for an AI-powered {{YOUR

print("User Prompt:\n", USER_PROMPT)
```

```
User Prompt:
 Help me brainstorm solution ideas for an AI-powered note-taking app for u
```

```
# --- Step 2: Apply a Prompt Augmentation Technique ---

# Example: Chain of Thought Prompting
META_PROMPT = f"""
You are a Solution Design Bot.
Think step by step before answering.

Task: Generate 2–3 solution concepts for this idea:
{USER_PROMPT}

Process:
1. Reason step by step about possible user needs, constraints, and opportu
2. Derive 2–3 concrete solution ideas from this reasoning.
3. For each idea, include: core concept, key features, risks, success meas
""".strip()
```

```
# Example: Template Prompting
#META_PROMPT = f"""
#You are a Solution Design Bot.
#
#Task: Produce a structured **one-pager solution concept** for this idea:
#{USER_PROMPT}
#
#Use the following template:
#Problem Summary:
#Solution Overview:
#Core Features:
#Differentiators:
#Risks & Open Questions:
#Success Metrics:
#""".strip()
```

```
# Example: Comparative Prompting
#META_PROMPT = f"""
#You are a Solution Design Bot.
#
#Task: Generate and compare **three different solution approaches** for tl
#{USER_PROMPT}
#
#For each approach, include:
#- Core idea
#- Key benefits
#- Trade-offs
#- Risk factors
#
#Then conclude with a recommendation: which approach is most viable and wl
#""".strip()
```

```
# --- Step 3: Construct Augmented Prompt ---
AUGMENTED_PROMPT = META_PROMPT
print("\n--- Augmented Prompt ---\n")
print(AUGMENTED_PROMPT)
```

```
# --- Step 4: Call the model with all parameters explicitly set ---
response, tokens = model_request(
    prompt=AUGMENTED_PROMPT,
    model=DEFAULT_MODEL,   # or override with a different model name
    temperature=0.7,       # controls randomness (0=deterministic, >1 = m
    top_k=40,              # number of candidates considered at each step
    top_p=0.9,             # nucleus sampling threshold
    num_predict=1000,      # max tokens to generate
    context_window=2048    # size of context window
)

print("\n--- Bot Response ---\n")
print(response)
print(f"\n[Tokens used: {tokens}]")
```

🧪 Experiment 3 — Prompt Augmentation for Requirement Analysis

```
# --- Step 1: Define raw User Prompt ---
# NOTE: REPLACE the 'target solution' with your proposed project!
USER_PROMPT = "Help me write requirements for an AI-powered note-taking a
#USER_PROMPT = "Help me write requirements for an AI-powered {{YOUR SOLUT
print("User Prompt:\n", USER_PROMPT)
```

```
User Prompt:
 Help me write requirements for an AI-powered note-taking app for universi
User Prompt:
 Help me brainstorm solution ideas for an AI-powered note-taking app for u
```

```
# --- Step 2: Apply a Prompt Augmentation Technique ---
# Choose ONE technique (Template / FactCheck / Reflection)

# Example: Template Prompting
META_PROMPT = f"""
You are a Requirements Engineer Bot.

Task: Write **user stories with acceptance criteria** for this idea:
{USER_PROMPT}

Use the following template for each story:
User Story: As a <role>, I want <capability>, so that <benefit>.
Acceptance Criteria:
- Given ...
- When ...
- Then ...
Notes:
""".strip()
```

```
# Example: Fact Check List Prompting
#META_PROMPT = f"""
#You are a Requirements Engineer Bot.

#Task: Write requirements for this idea:
#{USER_PROMPT}

#For each requirement:
#1. Provide the requirement statement.
#2. List factual claims (that can be verified).
#3. List assumptions (that need validation).
#4. Suggest one clarifying question.

#Format:
#Requirement: ...
#Facts: ...
#Assumptions: ...
#Question: ...
#""".strip()
```

```
# Example: Reflection Prompting
#META_PROMPT = f"""
#You are a Requirements Engineer Bot.
#
#Task: Write requirements for this idea:
#{USER_PROMPT}
#
#Step 1: Generate 5–7 functional and non-functional requirements.
#Step 2: Reflect on your own output:
#- Are all requirements testable?
#- Did you miss any critical constraints (e.g., privacy, accessibility, re
#- Rewrite or add 2 improved requirements if needed.
#
#Output format:
#Requirements:
#1) ...
#2) ...
#3) ...
#Reflection:
#- gaps_or_risks: ...
#- improvements:
#   A) ...
#   B) ...
#""".strip()
```

(Part III) EXERCISE - PROMPT AUGMENTATION FOR LATER SDLC PHASES

- **System Design**: translate requirements into system architecture
- **Solution Development**: adopt best practices for coding; generate code structure.

Challenge:

- Select 3 Prompt Engineering techniques below (check class presentation for details)
- Apply them to these phases.

1. Zero-Shot Prompting
2. Few-Shot Prompting
3. Chain-of-Thought (CoT)
4. Meta Prompting
5. Self-Consistency
6. Generated Knowledge Prompting
7. Prompt Chaining
8. Tree of Thoughts (ToT)
9. Automatic Reasoning
10. Automatic Prompt Engineering (APE)
11. Active-Prompt
12. Directional Stimulus Prompting
13. Reflexion
14. Graph Prompting

Exercise 1 - System Design

```
# --- Step 1: Define raw User Prompt ---
# Note: ADJUST for your solution scenario

USER_PROMPT = "Propose a backend code structure for an AI-powered {{YOUR S
print("User Prompt:\n", USER_PROMPT)
```

```
# --- Step 2: Apply a Prompt Augmentation Technique ---
# Technique: Few-Shot Prompting
# NOTE: PICK ANOTHER TECHNIQUE AND COMPLETE THE BELOW

META_PROMPT = f"""
You are a Solution Design Bot.

Here are examples of backend project structures:

Example :
project/
├── app/
│   ├── __init__.py
│   ├── models.py
│   ├── routes.py
│   └── services.py
├── tests/
│   └── test_app.py
├── requirements.txt
└── run.py

Now, using a similar style, propose a **backend code structure** for:
👉 {USER_PROMPT}

Include:
- Folder layout
- Key files
- A short description of each component
""".strip()
```

```
# Alternative 1
# YOUR TURN!
# TEST WITH A DIFFERENT PROMPT ENGINEERING TECHNIQUE
```

```
# Alternative 2
# YOUR TURN!
# TEST WITH A DIFFERENT PROMPT ENGINEERING TECHNIQUE
```

```
# --- Step 3: Construct Augmented Prompt ---
AUGMENTED_PROMPT = META_PROMPT
print("\n--- Augmented Prompt ---\n")
print(AUGMENTED_PROMPT)
```

```
# --- Step 4: Call the model with all parameters explicitly set ---
# NOTE: TEST WITH VARIATIONS OF MODEL PARAMTERS; FIND THE CONFIGURATION TH
response, tokens = model_request(
    prompt=AUGMENTED_PROMPT,
    model=DEFAULT_MODEL,   # override if desired
    temperature=0.7,
    top_k=40,
    top_p=0.9,
    num_predict=600,
    context_window=2048
)

print("\n--- Bot Response ---\n")
print(response)
print(f"\n[Tokens used: {tokens}]")
```

Exercise 2 - Solution Development

```
## YOUR TURN
# Alternative 1
# STRUCTURE THE CODE AS ABOVE
# TEST WITH 2-3 DIFFERENT PROMPT ENGINEERING TECHNIQUES
# NOTE: TEST WITH VARIATIONS OF MODEL PARAMTERS; FIND THE CONFIGURATION TH
```

(PART IV) Clean up

After your session ends:

- Click on `Run all -> Restart Session`
- Remove Ollama and models to free up space in your Google Account!

```
# Show where the binary is
!which ollama || true

# Remove the binary and library dir if present
!rm -f /usr/local/bin/ollama
!rm -rf /usr/local/lib/ollama
!rm -rf /root/.ollama

# Verify it is gone

!which ollama || echo "ollama binary not found (removed)"
!du -sh ~/.ollama 2>/dev/null || echo "~/.ollama directory not found (remo
```