Prompt Engineering Assignments

You will build **applications** purely by crafting and refining prompts, focusing on structured output, error handling (missing data), and domain specificity.

Each assignment illustrates how to request information from users and handle missing information before generating the final artifact.

Tasks:

Build interactive chatbot for the assignment assigned to you and publish it on Huggingface

1) You need to do **only ONE** of the below assignments as per allocation sheet: https://github.com/vijay-agrawal/genai-batch2/blob/main/assignments/Prompt%20Engineering%20Assignment%20allocations.pdf

2) You may use the promptperfect or any other chatbot to help you generate the prompt (https://promptperfect.jina.ai/)

2) Your chatbot should be context aware (have memory/history) for the session

3) Use Amazon Bedrock model as AI assistant

4) Deploy your chatbot on Huggingface (HF)
   Create a new huggingface space
   Name it as per your assignment, for example: insurance_form_generator
   HF Setup document for reference: https://github.com/vijay-agrawal/genai-batch2/blob/main/training-materials/Huggingface%20Setup.pdf

5) Bonus task:
Copy the prompt into Gemini or Copilot or ChatGpt and see how it behaves.
Try to use the audio capabilities of Gemni to see if you can interact with the application using voice.

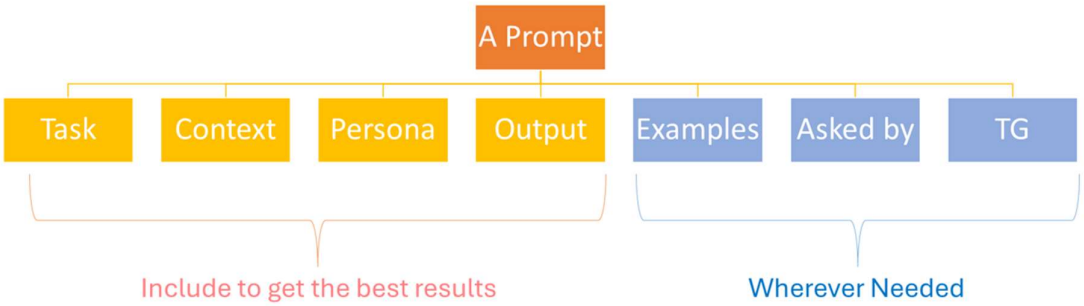
Submission

Create a Word or text document inside your shared assignments submission folder, under “Prompt Engineering” subfolder. Put the HF URL in it.

Send email to Vijay (vijay95051@gmail.com) and cc Saakshi to notify them about your submission.

General Tips to help with the assignments:

1. Use the prompt engineering best practices learnt in class, summarized below:



2. **Review Prompt Types**
   https://www.promptingguide.ai/
3. **Start with a Simple Prompt**
   - Begin with a single prompt that tries to gather all data and produce output.
   - They'll quickly see if the output is missing details or is disorganized.
4. **Iterative Prompting**
   - Encourage them to **iterate** on the prompt:
     - Add instructions to check for missing info.
     - Add formatting/style guidelines.
     - Add domain constraints (e.g., valid insurance policy numbers, feasible income ranges).
5. **Error Handling**
   - Make sure the prompt politely **asks for corrections** if a user supplies contradictory data (e.g., a date of birth that doesn't make sense).
6. **Be Creative**
   - Customize the final output style (letter, form, summary, bullet points) by specifying the desired format in the prompt.
   - Experiment with formality levels, professional jargon, or layman's terms.
7. **Time Management**
   - Each assignment can fit in a one-hour window if students:
     - Spend ~20 minutes crafting the initial prompt.
     - Spend ~20 minutes refining and testing with sample data.

        - Spend ~20 minutes finalizing the structured output and discussing improvements.

Assignment 1. Hospital Admission Letter Generator

Overview

Build a prompt-based "application" to generate a **Hospital Admission Letter** from **unstructured patient input**. The goal is to ensure the prompt captures all necessary details (patient name, date of birth, diagnosis, doctor's name, admission date, and reason for admission).

Steps

1. **Initial Prompt Design**
    - Start with a basic prompt (e.g., "Generate a hospital admission letter from the following information…").
    - Let the user provide some incomplete input (e.g., just the patient name and reason for admission).
2. **Identify Missing Information**
    - The prompt (or chain of prompts) should detect missing fields (like date of birth, doctor's name, etc.) and explicitly **ask** the user to supply them (e.g., "Please provide the doctor's name.").
3. **Refined Prompt with Conditional Logic**
    - Refine the prompt so that it repeatedly checks for completeness. If any field is still missing, it prompts the user again.
4. **Generate the Final Admission Letter**
    - Once all required information is provided, the prompt generates a structured, professional admission letter.

Learning Objectives

- Understand how to prompt for **missing fields** and handle incomplete data.
- Practice refining prompts to produce a **polished** document.
- Experience the back-and-forth nature of prompt-driven interactions.

Assignment 2. Medical Prescription Summary

Overview

This "application" will generate a **Medical Prescription Summary** based on partial patient and medication data. It demonstrates how to handle domain-specific terminology and ensure completeness (e.g., dosage, frequency).

Steps

1. **Set Up a Conversation Flow**
    - The first prompt instructs the model to **gather patient details** (name, age) and medication details (drug name, dosage, frequency, duration).
2. **Iterative Query**
    - If any piece of information is missing or unclear, the prompt automatically asks the user for clarification.
    - Example: If the user forgets to mention dosage, the model asks, "Please provide the prescribed dosage (e.g., 500 mg)."
3. **Output Format**
    - The final output is a **Prescription Summary** that clearly lists:
        - Patient name
        - Age and relevant medical history (if any)
        - Medication details (drug name, dosage, frequency)
        - Special instructions
4. **Role-Play Variation**
    - Practice different prompts simulating pharmacists or nurses needing a structured summary for patient records.

Learning Objectives

- Use **iterative prompting** to collect domain-specific data (medication dosage, schedule).
- Practice structuring the output in a **clear, professional** format.
- Experience creating domain-aware prompts for healthcare use cases.

Assignment 3. Insurance Claim Form Assistant (Healthcare-Finance Intersection)

Overview

This assignment blends **healthcare** and **finance** by creating an "application" that generates an **Insurance Claim Form**. The "application" will ask for policy details, claimant details, and treatment information, then produce a well-formatted claim form.

Steps

1. **Check Required Fields**
    - Policy number, claimant's name, date of treatment, reason for claim, amount claimed, etc.
2. **Craft a Prompt That Validates Inputs**
    - In the prompt, specify: "If any field is missing or incomplete, ask the user to provide it."
    - Example user input: "Policy #12345, my name is John Doe, and I had a knee surgery."
    - The model should respond: "What is the date of your surgery? Please provide the total cost of treatment. Also provide your date of birth."
3. **Generate the Structured Claim Form**
    - Once all fields are collected, the prompt instructs the model to generate a final claim document.
    - The output should look like a standard insurance form with labeled sections.
4. **Formatting Requirements**
    - Refine the prompt to **format** the output in a tabular or bullet-based layout suitable for submission to an insurance company.

Learning Objectives

- Combine **health and finance data** requirements in one prompt application.
- Implement **error handling** for incomplete or inconsistent fields.
- Learn to output complex multi-section documents through iterative prompt refinement.

Assignment 4. Bank Loan Application Assistant

Overview

Create a prompt-based “assistant” that guides a user through a **Bank Loan Application**. This covers personal and financial details required for the application, ensuring completeness before generating the final loan request form.

Steps

1. **Initial Prompt**
    - The model greets the user and states it will help fill out a loan application.
    - “I need some details. Please provide your name, address, and annual income.”
2. **Collect Additional Data**
    - The model recognizes missing fields (loan amount requested, duration, credit score, etc.) and prompts for them.
3. **Refining for Professional Tone**
    - Add instructions to generate a polite, professional final letter.
    - Example: “Use a formal but friendly tone. Include a short introduction, the main request details, and a concluding paragraph thanking the bank.”
4. **Final Loan Application Output**
    - The prompt merges collected details into a single structured form/letter.
    - You may add “If any part is unclear, ask me for more info before generating the letter” to ensure completeness.

Learning Objectives

- Learn how to build a **step-by-step data collection** conversation for finance applications.
- Control the **tone and style** of the output via prompt instructions.
- Experience how prompt engineering can eliminate coding for **basic form creation**.

Assignment 5. Personal Financial Plan Generator

Overview

This “application” helps a user generate a **personal financial plan** by walking them through their income, expenses, financial goals, and risk tolerance. It emphasizes the

use of **conditional logic** based on user inputs (e.g., if they have children, include education savings).

Steps

1. **Discovery Prompt**
    - The model starts by asking: “What is your monthly income? What are your monthly expenses? Do you have any specific financial goals (like retirement, buying a house, or saving for college)?”
2. **Conditional Paths**
    - If the user says they have children, the model follows up with, “Would you like to include a college savings plan?”
    - If the user has no children, it omits that section.
3. **Handling Missing or Conflicting Information**
    - The model carefully checks if the user’s stated expenses exceed income and might prompt further clarifications: “You mentioned your expenses are higher than your income. Do you have any additional sources of funds or debt details to consider?”
4. **Final Financial Plan**
    - The plan includes a breakdown of suggested savings, investments, and advice for future planning.
    - Refine prompts to **summarize** in bullet points or a structured outline.

Learning Objectives

- Incorporate **conditional logic** based on user responses.
- Showcase how to generate **customized outputs** for different user scenarios.
- Practice prompting for complex data to create a concise, actionable plan.
