WILLIAM & MARY

CHARTERED 1693

GenAI for SD

Prof. Antonio Mastropaolo
Assistant: Aya Garryyeva

Prompt Engineering for In-Context Learning

In this assignment, you will explore the power of **in-context learning** by designing effective prompts to perform various software engineering tasks. You will craft and compare different prompting strategies for a diverse set of problems using large language models (LLMs).

⚙ **Models to Use:**

- ChatGPT (GPT-4)
- Claude (Anthropic)
- Gemini (Google)
- Model of your choice (e.g., LLaMA, Mistral, Github Copilot, etc.)
- Please note that DeepSeek is not permitted on W&M's network and devices.

Tasks Overview and Requirements

You will be given 22 problems, each involving a code understanding or generation task (e.g., summarization, bug fixing, classification). For each problem:

- Apply at least **two different prompting strategies** (zero-shot, few-shot, chain-of-thought, self-consistency, or prompt chaining).
- Compare and report the outputs for at least **two different models**.
- Analyze why models responded the way they did and which approach performed best.

You are encouraged to reuse and adapt the example prompts provided as references.

Point Distribution (Total: 100 points)

- 30 points – Prompt design and execution across different techniques
- 30 points – Variety and difficulty of the selected tasks (ensure coverage across all task types)
- 20 points – Comparative analysis across different models and strategies

- 20 points – Report clarity, insightfulness, and format

What to Submit:

- GitHub repo containing:
  - **ReadMe** file outlining the purpose of the assignment
  - **Raw prompts and outputs** for all 22 problems
  - Any scripts used for automation or data formatting
  - **PDF report** (up to 10 pages) containing:
    - Prompt(s) used
    - Output(s) from models
    - Short analysis for each task
    - Comparison between the models and prompting techniques

Deadline: Thursday, April 24th, 2025 @ 12:00 PM

Late policy: 5 points/day up to 5 days.

Special Notes

- Focus on the model's reasoning, even when the answer is wrong.

Example Walkthrough: How to Complete an Assignment

Here is a complete example that shows you how to structure your submission. This task involves writing a function that checks if a number is prime — a classic problem used to evaluate code understanding and generation.

Task: Check if a Number is Prime (Python)

Given the following incomplete function:

```python
def is_prime(n):
    # TODO: Return True if n is a prime number, else False
    pass
```

Prompting Strategy 1: Zero-Shot Prompt

`Prompt: Write a Python function that checks if a number is a prime number`

Model Output:

```python
def is_prime(n):
  if n <= 1:
      return False
  for i in range(2, n):
      if n % i == 0:
          return False
  return True
```

Comment: This solution is correct for most inputs. However, it's not the most efficient because it checks all numbers from 2 to $n-1$. There's room for optimization.

—

Prompting Strategy 2: Few-Shot Prompt

```
Prompt:
"Example 1:
Input: 7
Output: True

Example 2:
Input: 10
Output: False

Write a Python function that checks if a number is prime."
```

Model Output:

```
def is_prime(n):
  if n <= 1:
     return False
  for i in range(2, int(n ** 0.5) + 1):
     if n % i == 0:
       return False
   return True
```

Comment: This version improves the runtime by only checking divisibility up to the square root of n. It's more efficient and shows better reasoning from the model due to the structured prompt.

—

Model Comparison and Analysis

The zero-shot prompt produced a correct and readable solution but lacked optimization. The few-shot version used example-based prompting, which led the model to generate a better algorithm — leveraging the square root property of primes. In short, the Prompt quality significantly impacts code quality. Adding examples can guide the model toward more efficient and robust solutions — even for simple problems.

Problems to Explore

Students must develop prompts for the following tasks:

1. **Code Summarization (Java)** Summarize the functionality of the following method:

```
public Map<String, Integer> countWordFrequency(List<String> words) {
    Map<String, Integer> freqMap = new HashMap<>();
    for (String word : words) {
        freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);
    }
    return freqMap;
}
```

2. **Bug Fixing (Python – Off-by-One)** Identify and fix the off-by-one error in this function:

```
def sum_range(start, end):
    total = 0
    for i in range(start, end):
        total += i
    return total
```

3. **Bug Classification (C++)** Classify the type of bug in the following C++ function:

```
int* getArray(int size) {
    int arr[size]; // Warning: local array
    return arr; // Bug: returning pointer to local variable
}
```

4. **Generating Email Validators (Python + Regex)** Complete the function using regex to validate basic email addresses:

```python
def is_valid_email(email):
    # TODO: Complete using regex
    pass
```

5. **Generating Flask APIs (Python)** Create a '/greet/<username>' endpoint that returns a JSON greeting:

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # TODO: Return a JSON greeting
    pass
```

6. **SQL Schema Design (SQL)** Write the schema for a review app with users, books, and reviews:

```sql
-- TODO: Design schema with appropriate keys and constraints
-- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id,
    rating)
```

7. **Null Dereference Detection (Java)** Identify any null dereference risk:

```java
public int getLength(String s) {
    return s.length(); // What if s is null?
}
```

8. **CSV Parser Variants (Python)** Improve the parser to support quoted fields:

```python
def parse_csv_line(line):
    return line.split(',')  # Incomplete: doesn't handle quoted fields
```

9. **Data Class to API Conversion (Kotlin)** Convert the data class to a REST API using Ktor:

```kotlin
data class Product(val id: Int, val name: String, val price: Double)

// TODO: Create GET and POST endpoints using Ktor
```

10. **Function Summarization (Python)** Write a brief summary of the function:

```python
def reverse_words(sentence):
    return ' '.join(sentence.split()[::-1])
```

11. **Prompt from Code Comments (Python)** Write a prompt that could generate the code:

```python
# This function checks if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
```

12. **Fixing Factorial Bug (Python)** Fix the bug when input is 0:

```
def factorial(n):
    result = 1
    for i in range(1, n):
        result *= i
    return result
```

13. **Linked List Node Deletion (C)** Implement node deletion by value:

```
struct Node {
    int data;
    struct Node* next;
};

void deleteNode(struct Node** head, int key) {
    // TODO: Implement node deletion
}
```

14. **Recursive Function Completion (Python)** Complete the recursive function for Fibonacci:

```
def fibonacci(n):
    # TODO: Base cases and recursive call
    pass
```

15. **Constructor Completion (Python)** Complete the class constructor:

```
class Person:
    def __init__(self):
        # TODO: Add name, age, and optional email
        pass
```

16. **Binary Search Completion (Java)** Complete the binary search implementation:

```
public int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        // TODO: Compare and adjust bounds
    }
    return -1;
}
```

17. **Self-Consistency Bug Fixing (C++)** Resolve inconsistency between function name and logic:

```
// Supposed to return true if x is even
bool isOdd(int x) {
    return x % 2 == 0; // Logic contradicts function name
}
```

18. **Prompt Chaining: Bug Identification → Fix (JavaScript)** Identify and fix the bug:

```
function isEven(n) {
    return n % 2; // Returns 1 or 0, not true/false
}
```

19. **Summary Decomposition (C++)** Decompose the high-level comment/summary into logical steps:

```
// Function that validates an input, calculates square, and returns result
int process(int x) {
    if (x < 0) return -1;
    return x * x;
}
```

20. **Purpose Inference → Completion (Python)** Complete the function based on intent:

```
def calculate_average(scores):
    total = 0
    # TODO: Complete to return average
    pass
```

21. **Full-File Bug Detection and Refactoring (Python)** Analyze the following utility script. Identify any potential logic issues or design flaws. Then, refactor it for better readability, correctness, and safety.
Bonus Question — Worth up to 2.5 points toward the total grade

```
# utils.py - Script to parse and summarize numeric CSV files

import csv

def read_csv(filepath):
  with open(filepath, ’r’) as f:
     return [row for row in csv.reader(f)]

def summarize_column(data, index):
  values = [float(row[index]) for row in data[1:]]  # skip header
  total = sum(values)
  avg = total / len(values)
  return total, avg

def main():
  filepath = ’data.csv’
  data = read_csv(filepath)
  total, avg = summarize_column(data, 1)
  print("Total:", total)
  print("Average:", avg)

if __name__ == ’__main__’:
  main()
```

22. **Code Completion and Robustness Enhancement (Python)** Complete the following file-processing script. The goal is to clean each line, remove punctuation, and count word frequencies correctly.
Bonus Question — Worth up to 2.5 points toward the total grade

```
# file_processor.py - Incomplete script for processing text files

import string

def load_file(filepath):
   with open(filepath, ’r’) as f:
      return f.readlines()

def clean_line(line):
   # TODO: Remove punctuation and make lowercase
  pass

def count_words(lines):
   word_counts = {}
   for line in lines:
   clean = clean_line(line)
   for word in clean.split():
     word_counts[word] = word_counts.get(word, 0) + 1
  return word_counts

def main():
  filepath = 'input.txt'
  lines = load_file(filepath)
  counts = count_words(lines)
  for word, count in sorted(counts.items()):
     print(f"{word}: {count}")

if __name__ == '__main__':
main()
```