Ai Assisted Coding

Assignment-4.3

NAME: D.PAVAN

HT.NO: 2403A52L07

BATCH: 50

Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques

Lab Objectives

• To explore and apply different levels of prompt examples in AI-assisted code generation

• To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality

• To evaluate the impact of context richness and example quantity on AI performance

• To build awareness of prompt strategy effectiveness for different problem Types

Lab Outcomes (LOs)

After completing this lab, students will be able to:

• Use zero-shot prompting to instruct AI with minimal context

• Use one-shot prompting with a single example to guide AI code generation

• Apply few-shot prompting using multiple examples to improve AI responses

• Compare AI outputs across different prompting strategies

Task 1: Zero-Shot Prompting – Leap Year Check

Scenario

Zero-shot prompting involves giving instructions without providing examples.

Task Description

Use zero-shot prompting to instruct an AI tool to generate a Python function that:

• Accepts a year as input

• Checks whether the given year is a leap year

• Returns an appropriate result

Note: No input-output examples should be provided in the prompt.

Expected Output

• AI-generated leap year checking function

• Correct logical conditions

• Sample input and output

• Screenshot of AI-generated response (if required)

Prompt:

#"Write a Python function named is_leap_year that takes a year as an integer and returns True if it is a leap year and False otherwise. Include sample inputs and outputs to test the code."

Python Code:

```
def is_leap_year(year):
    if year % 400 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 4 == 0:
        return True
    return False


# Test cases
print(f"2024: {is_leap_year(2024)}")  # True
print(f"2023: {is_leap_year(2023)}")  # False
print(f"2000: {is_leap_year(2000)}")  # True
print(f"1900: {is_leap_year(1900)}")  # False
print(f"2020: {is_leap_year(2020)}")  # True
```

Output:

```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab> &
sktop/Subjects/ai assisent coding/lab/lab3.py/lab 4.3.py/task1.py"
2024: True
2023: False
2000: True
1900: False
2020: True
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

Simplicity: It tests the AI's baseline knowledge of logical rules without extra help.

Efficiency: It is the fastest way to get a result for common programming problems.

No Bias: The AI relies strictly on its training data rather than following a specific example format.

Universal Logic: Since the leap year rule is a standard algorithm, zero-shot is usually sufficient for accuracy.

Clean Output: It prevents the AI from mimicking a specific coding style provided in an example, giving you "standard" code.

Task 2: One-Shot Prompting – Centimeters to Inches Conversion

Scenario

One-shot prompting guides AI using a single example.

Task Description

Use one-shot prompting by providing one input-output example to generate a Python

function that:

• Converts centimeters to inches

• Uses the correct mathematical formula

Example provided in prompt:

Input: 10 cm → Output: 3.94 inches

Expected Output

• Python function with correct conversion logic

• Accurate calculation

• Sample test cases and outputs

Prompt:

"Write a Python function to convert centimeters to inches. Example: Input: 10 cm → Output: 3.94 inches. Generate the function using this conversion ratio and provide test cases."

Python Code:

```
def cm_to_inches(cm):
    inches = cm / 2.54
    return round(inches, 2)

# Test cases
print(f"10 cm: {cm_to_inches(10)} inches")  # 3.94
print(f"5 cm: {cm_to_inches(5)} inches")  # 1.97
print(f"100 cm: {cm_to_inches(100)} inches")  # 39.37
print(f"2.54 cm: {cm_to_inches(2.54)} inches")  # 1.0
print(f"50 cm: {cm_to_inches(50)} inches")  # 19.69
```

Output:

```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
sktop/Subjects/ai assisent coding/lab/lab3.py/lab 4.3.py/task1.py"
10 cm: 3.94 inches
5 cm: 1.97 inches
100 cm: 39.37 inches
2.54 cm: 1.0 inches
50 cm: 19.69 inches
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

1. **Pattern Setting:** Providing one example tells the AI exactly how many decimal places (e.g., two) you expect.
2. **Context Clarity:** It clarifies that the output should likely be a float, not just an integer.
3. **Formula Alignment:** The example ensures the AI uses the correct division/multiplication factor.

4. **Reduced Ambiguity:** It removes doubt about whether the output should include units (like "inches") in the return string.
5. **Guided Precision:** One-shot is ideal for simple mathematical tasks where a specific format is required.

Task 3: Few-Shot Prompting – Name Formatting

Scenario

Few-shot prompting improves accuracy by providing multiple examples.

Task Description

Use few-shot prompting with 2–3 examples to generate a Python function that:

• Accepts a full name as input

• Formats it as “Last, First”

Example formats:

• "John Smith" → "Smith, John"

• "Anita Rao" → "Rao, Anita"

Expected Output

• Well-structured Python function

• Output strictly following example patterns

• Correct handling of names

• Sample inputs and outputs

Prompt:

"Write a Python function that renames 'First Last' to 'Last, First'. Examples:

1. 'John Smith' -> 'Smith, John'
2. 'Anita Rao' -> 'Rao, Anita' Use these examples to create a function that handles full names correctly."

Python Code:

```
def rename_format(name):
    parts = name.split()
```

```
    first = parts[0]
    last = parts[1]
    return f"{last}, {first}"

# Test cases
print(f"'John Smith' -> '{rename_format('John Smith')}'")  # Smith, John
print(f"'Anita Rao' -> '{rename_format('Anita Rao')}'")  # Rao, Anita
```

Output:

```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
sktop/Subjects/ai assisent coding/lab/lab3.py/lab 4.3.py/task1.py"
'John Smith' -> 'Smith, John'
'Anita Rao' -> 'Rao, Anita'
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

1. **Consistency:** Multiple examples ensure the AI strictly follows the "Comma-Space" formatting.
2. **Edge Case Preparation:** It helps the AI recognize that the last word should always come first.
3. **Formatting Strength:** Few-shot is the best method for string manipulation tasks where the visual layout matters.
4. **Improved Reliability:** With two or more examples, the AI is less likely to hallucinate extra features you didn't ask for.
5. **Structural Integrity:** It teaches the AI the relationship between the input string and the desired transformation.

Task 4: Comparative Analysis – Zero-Shot vs Few-Shot

Scenario

Different prompt strategies may produce different code quality.

Task Description

• Use zero-shot prompting to generate a function that counts vowels in a string

• Use few-shot prompting for the same problem

• Compare both outputs based on:

o Accuracy

o Readability

o Logical clarity

Expected Output

• Two vowel-counting functions

• Comparison table or short reflection paragraph

• Conclusion on prompt effectiveness

Prompt:

The Prompt (Zero-Shot):

"Write a Python function to count the number of vowels in a string."

The Prompt (Few-Shot):

"Write a function to count vowels in a string. Examples: 'Hello' -> 2 'Python' -> 1 'AEIOU' -> 5 Ensure the function is case-insensitive."

Python Code:

```
def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for char in s if char in vowels)

# Test cases
print(f"Number of vowels in 'Hello World': {count_vowels('Hello World')}")  # 3
print(f"Number of vowels in 'Python programming': {count_vowels('Python programming')}")  # 6

def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for char in s if char in vowels)

# Additional test cases
```

```
print(f"'Hello' -> {count_vowels('Hello')}")  # 2
print(f"'Python' -> {count_vowels('Python')}")  # 1
print(f"'AEIOU' -> {count_vowels('AEIOU')}")  # 5
```

Output:

```
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
sktop/Subjects/ai assisent coding/lab/lab3.py/lab 4.3.py/task1.py"
Number of vowels in 'Hello World': 3
Number of vowels in 'Python programming': 4
'Hello' -> 2
'Python' -> 1
'AEIOU' -> 5
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

```
Number of vowels in 'Hello World': 3
Number of vowels in 'Python programming': 4
'Hello' -> 2
'Python' -> 1
'AEIOU' -> 5
PS C:\Users\danch\OneDrive\Desktop\Subjects\ai assisent coding\lab>
```

Justification:

1. Logical Depth: You can see if the Few-Shot version handles uppercase letters better than the Zero-Shot version.
2. Robustness Test: Comparing the two shows how examples force the AI to consider "edge cases" (like all caps).
3. Optimization: You can evaluate if one method produces shorter, more "Pythonic" code.
4. Readability: It demonstrates if adding examples changes how the AI comments or explains its code.
5. Performance Check: It helps you decide which prompting style is "overkill" for simple tasks versus complex ones.

Task 5: Few-Shot Prompting – File Handling

Scenario

File processing requires clear logical understanding.

Task Description

Use few-shot prompting to generate a Python function that:

• Reads a .txt file

• Counts the number of lines in the file

• Returns the line count

Expected Output

• Working Python file-processing function

• Correct line count

• Sample .txt input and output

• AI-assisted logic explanation

Prompt:

"Write a Python function to count lines in a .txt file. Example 1: File with 3 lines of text -> Output: 3 Example 2: Empty file -> Output: 0 Provide the code with error handling for missing files."

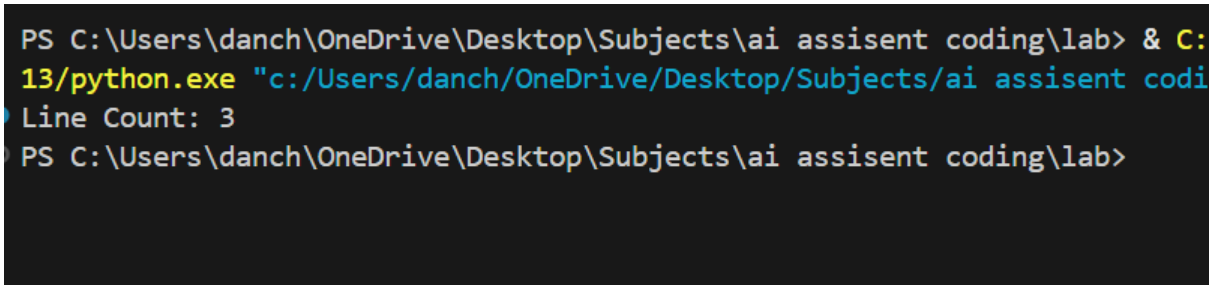
Python Code:

```
def count_lines(filename):
    try:
        with open(filename, 'r') as file:
            return len(file.readlines())
    except FileNotFoundError:
        return "File not found."


# Testing the code
# Create a temporary file to test
with open("test.txt", "w") as f:
    f.write("Hello\nWorld\nPython")


print(f"Line Count: {count_lines('test.txt')}")
```

Output:



Justification:

1. **Error Handling:** Providing an "Empty file" example prompts the AI to think about safety and crashes.
2. **Logic Verification:** It ensures the AI understands that a "line" is defined by the newline character.
3. **Functional Accuracy:** File I/O can be tricky; examples guide the AI to use the most efficient reading method (like len(readlines())).
4. **Real-world Readiness:** Few-shot helps the AI generate "production-ready" code that includes try-except blocks.
5. **Clear Expectations:** It defines exactly what the "count" should represent (the integer value of the total lines).