AI-Assisted Coding

Name: P. Archith   Roll Number: 2303A52258   Batch: 45

Lab 4: 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

Week2 -

Wednesday

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: Generate a Python Code that accepts a year as input and checks whether the given year is a leap year.

The function should return an appropriate result indicating whether the year is a leap year or not.

Also include a sample input and its corresponding output.

Code:

```python
def is_leap_year(year: int) -> bool:
    Leap year rules:
      - divisible by 4
      - except years divisible by 100, unless also divisible by 400
    """
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)


def check_leap_from_input() -> None:
    """Read a year from stdin and print whether it's a leap year."""
    try:
        year = int(input("Enter year: ").strip())
    except ValueError:
        print("Invalid input. Please enter an integer year.")
        return

    if is_leap_year(year):
        print(f"{year} is a leap year.")
    else:
        print(f"{year} is not a leap year.")


if __name__ == "__main__":
    # Example usage
    # Sample Input: 2000
    # Sample Output: 2000 is a leap year.
    check_leap_from_input()
```

Output:

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: Convert centimeters to inches using Python.

Example:

Input: 10 cm

Output: 3.94 inches

Now generate a Python code that:

- Accepts a value in centimeters as input

- Converts it to inches using the correct formula

- Returns the converted value

- Includes a few sample test cases with outputs

Code:

```python
def cm_to_inches(cm: float, ndigits: int = 2) -> float:
    """
    Convert centimeters to inches. Returns the value rounded to ndigits.
    1 inch = 2.54 cm
    """
    return round(cm / 2.54, ndigits)


def parse_cm_input(s: str) -> float:
    """Parse input like '10', '10 cm', or '  10.5 CM ' into a float centimeters value."""
    s = s.strip().lower().replace("cm", "").strip()
    return float(s)


if __name__ == "__main__":
    # Interactive example
    try:
        user = input("Enter length in centimeters (e.g. '10 cm'): ")
        cm_value = parse_cm_input(user)
        inches = cm_to_inches(cm_value)
        print(f"{cm_value} cm = {inches:.2f} inches")
    except Exception as e:
        print("Invalid input. Please enter a numeric value, optionally followed by 'cm'.")

    # Sample test cases with outputs
    print("\nSample conversions:")
    samples = [10, 0, 2.54, 30.48]
    for s in samples:
        print(f"{s} cm -> {cm_to_inches(s):.2f} inches")
```

Output:

```
Enter length in centimeters (e.g. '10 cm'): 25
25.0 cm = 9.84 inches

Sample conversions:
10 cm -> 3.94 inches

Sample conversions:
10 cm -> 3.94 inches
Sample conversions:
10 cm -> 3.94 inches
0 cm -> 0.00 inches
2.54 cm -> 1.00 inches
10 cm -> 3.94 inches
0 cm -> 0.00 inches
2.54 cm -> 1.00 inches
0 cm -> 0.00 inches
2.54 cm -> 1.00 inches
2.54 cm -> 1.00 inches
30.48 cm -> 12.00 inches
PS D:\3-2\AI Assitant Coding> 
```

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: Format a person's full name into the format "Last, First" using Python.

Examples:

Input: John Smith

Output: Smith, John

Input: Anita Rao

Output: Rao, Anita

Input: Rahul Verma

Output: Verma, Rahul

Now generate a Python Code that:

- Accepts a full name as input

- Formats the name as "Last, First"

- Returns the formatted name

- Includes sample inputs and outputs

Code:

```python
def format_name(full_name: str) -> str:
    """
    Format a full name into "Last, First".
    - Uses the first token as First and the last token as Last.
    - Returns empty string for empty input, single name returned as-is.
    """
    parts = full_name.strip().split()
    if not parts:
        return ""
    if len(parts) == 1:
        return parts[0]
    return f"{parts[-1]}, {parts[0]}"
if __name__ == "__main__":
    # Interactive usage
    name = input("Enter full name: ")
    print(format_name(name))

    # Sample inputs and outputs
    samples = ["John Smith", "Anita Rao", "Rahul Verma", "Madonna", "  Alice   Bob   Carol  "]
    print("\nSamples:")
    for s in samples:
        print(f"Input: {s!r} -> Output: {format_name(s)}")
```

Output:

```
Samples:
Input: 'John Smith' -> Output: Smith, John
Input: 'Anita Rao' -> Output: Rao, Anita
Input: 'Rahul Verma' -> Output: Verma, Rahul
Input: 'Madonna' -> Output: Madonna
Samples:
Input: 'John Smith' -> Output: Smith, John
Input: 'Anita Rao' -> Output: Rao, Anita
Input: 'Rahul Verma' -> Output: Verma, Rahul
Input: 'Madonna' -> Output: Madonna
Input: 'Rahul Verma' -> Output: Verma, Rahul
Input: 'Madonna' -> Output: Madonna
Input: '  Alice   Bob   Carol  ' -> Output: Carol, Alice
PS D:\3-2\AI Assitant Coding> ▮
```

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:

1. Zero Shot: Generate a Python Code that accepts a string as input and counts the number of vowels in the string.

The function should return the total vowel count.

Also include a sample input and output.

Code:

```python
def count_vowels(s: str) -> int:
    """Return the number of vowels (a, e, i, o, u) in the given string (case-insensitive)."""
    vowels = set("aeiouAEIOU")
    return sum(1 for ch in s if ch in vowels)


if __name__ == "__main__":
    # Interactive usage
    user_input = input("Enter a string: ")
    total = count_vowels(user_input)
    print(f"Vowel count: {total}")

    # Sample input/output
    # Sample Input: Hello World
    # Sample Output: Vowel count: 3

    # Additional sample runs
    samples = ["Hello World", "AEIOU", "bcdfg", "The quick brown fox"]
    for s in samples:
        print(f"Input: {s!r} -> Vowel count: {count_vowels(s)}")
```

Output:

2. Few Shot: Count the number of vowels in a string using Python.

Examples:

Input: "hello"

Output: 2

Input: "Education"

Output: 5

Input: "sky"

Output: 0

Now generate a Python Code that:

- Accepts a string as input

- Counts the number of vowels in the string

- Returns the total count

- Includes sample inputs and outputs

Code:

```python
import re
def count_vowels(text: str) -> int:
    """Return the number of vowels (a, e, i, o, u) in text (case-insensitive)."""
    return len(re.findall(r"[aeiouAEIOU]", text))

if __name__ == "__main__":
    # Read a string from stdin and print the vowel count
    s = input("Input: ")
    print("Output:", count_vowels(s))

    # Sample inputs and outputs
    samples = ["hello", "Education", "sky"]
    print("\nSamples:")
    for sample in samples:
        print(f"Input: {sample!r} -> Output: {count_vowels(sample)}")
```

Output:

```
Samples:
Input: 'hello' -> Output: 2
Input: 'Education' -> Output: 5
Input: 'sky' -> Output: 0
PS D:\3-2\AI Assitant Coding>
```

Comparative Analysis: Compare the zero-shot and few-shot vowel counting functions based on accuracy, readability, and logical clarity.

Present the comparison in a table or a short reflection paragraph.

Conclude which prompting technique is more effective and why.

Code:

```python
# Comparison of two vowel-counting implementations (set-based vs regex-based)
rows = [
    ("Criterion", "Set-based (zero-shot)", "Regex-based (few-shot)"),
    ("Accuracy", "Equivalent for ASCII vowels; both count a,e,i,o,u correctly", "Equivalent for ASCII vowels; both count a,e,i,o,u correctl
    ("Readability", "Very clear: explicit set and iteration; easy for newcomers", "More concise but requires regex familiarity; pattern may
    ("Logical clarity", "Intent obvious, minimal dependencies, easy to modify", "Intent clear if you know regex; flexible for patterns but
    ("Performance", "Fast and low overhead for short strings", "Comparable for typical inputs; regex has compilation overhead but negligibl
    ("Extensibility", "Easier to adapt for Unicode or custom rules programmatically", "Easier to extend via patterns (e.g., Unicode classes
]

# Print table
col_widths = [max(len(str(r[i])) for r in rows) + 2 for i in range(3)]
for r in rows:
    print(r[0].ljust(col_widths[0]) + r[1].ljust(col_widths[1]) + r[2].ljust(col_widths[2]))

# Conclusion
print("\nConclusion: For most uses, the set-based (zero-shot) approach is preferable for readability and logical clarity while matching the
```

Output:

```
s\SNEHITH\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53206' '--' 'D:\3-2\AI Assitant Coding\Task4-3.p
y'
Criterion        Set-based (zero-shot)                              Regex-based (few-shot)

Accuracy         Equivalent for ASCII vowels; both count a,e,i,o,u correctly   Equivalent for ASCII vowels; both count a,e,i,o,u correctly

Readability      Very clear: explicit set and iteration; easy for newcomers    More concise but requires regex familiarity; pattern may be opaque t
o some
Logical clarity  Intent obvious, minimal dependencies, easy to modify          Intent clear if you know regex; flexible for patterns but less expli
cit
Performance      Fast and low overhead for short strings                       Comparable for typical inputs; regex has compilation overhead but ne
gligible amortized
Extensibility    Easier to adapt for Unicode or custom rules programmatically  Easier to extend via patterns (e.g., Unicode classes) but needs care
ful regex design

Conclusion: For most uses, the set-based (zero-shot) approach is preferable for readability and logical clarity while matching the regex version in
 accuracy. Use the regex (few-shot) approach when you need compact code or advanced pattern matching (e.g., Unicode classes).
PS D:\3-2\AI Assitant Coding> []
```

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: Read a text file and count the number of lines using Python.

Examples:

Example 1:

File content:

Hello

Welcome to Python

File handling is easy

Output:

Number of lines: 3

Example 2:

File content:

AI

Prompt Engineering

Few-shot Learning

Output:

Number of lines: 3

Now generate a Python Code that:

- Reads a .txt file given its filename

- Counts the total number of lines in the file

- Returns the line count

- Includes a sample .txt file input and its output

- Explains the logic used in the function

Code:

```python
def count_lines(filename: str) -> int:
    """
    Return the number of lines in the given text file.

    Logic: open the file and iterate over the file object (which yields one line at a time)
    and sum 1 for each yielded line. This avoids loading the whole file into memory.
    """
    with open(filename, "r", encoding="utf-8") as f:
        return sum(1 for _ in f)
if __name__ == "__main__":
    # Sample file 1 (Example 1)
    sample1 = "sample1.txt"
    with open(sample1, "w", encoding="utf-8") as f:
        f.write("Hello\nWelcome to Python\nFile handling is easy\n")

    # Sample file 2 (Example 2)
    sample2 = "sample2.txt"
    with open(sample2, "w", encoding="utf-8") as f:
        f.write("AI\nPrompt Engineering\nFew-shot Learning\n")

    # Run and print outputs
    print("Sample 1 file content:")
    print("Hello\nWelcome to Python\nFile handling is easy\n")
    print("Output:")
    print(f"Number of lines: {count_lines(sample1)}\n")

    print("Sample 2 file content:")
    print("AI\nPrompt Engineering\nFew-shot Learning\n")
    print("Output:")
    print(f"Number of lines: {count_lines(sample2)}")
```

Output:

```
y'
Sample 1 file content:
Hello
Welcome to Python
File handling is easy

Output:
Number of lines: 3

Sample 2 file content:
AI
Prompt Engineering
Few-shot Learning

Output:
Number of lines: 3
PS D:\3-2\AI Assitant Coding> 
```