2303A52366

Batch-31

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assignment-3.1** | | | | |
|  | **Q.No.** | **Question** | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab Experiment: Prompt Engineering – Improving Prompts and Context Management**  **Lab Objectives**   1. To understand and apply different prompt engineering techniques for generating Python programs using AI-assisted tools. 2. To analyze the impact of context and examples on the accuracy and efficiency of AI-generated code. 3. To develop and refine real-world Python applications through iterative prompt improvement.   **Lab Outcomes**   1. Students will be able to design effective prompts to generate correct and optimized Python code. 2. Students will be able to compare and evaluate AI-generated solutions produced using different prompting strategies. 3. Students will be able to implement and document real-world Python applications using AI-assisted coding tools.   **Experiment – Prompt Engineering Techniques**  **Task Description**  Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.  **Question 1: Zero-Shot Prompting (Palindrome Number Program)**  Write a **zero-shot prompt** (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.  **Task:**   * Record the AI-generated code.      * Test the code with multiple inputs.      * Identify any logical errors or missing edge-case handling.   The program does not work correctly when the number starts with 010.    **Question 2: One-Shot Prompting (Factorial Calculation)**  Write a **one-shot prompt** by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.  **Example:** Input: 5 → Output: 120  **Task:**   * Compare the generated code with a zero-shot solution.      * Examine improvements in clarity and correctness.   The recursive and iterative implementations are both correct for non-negative inputs, but they lack input validation, handling for negative numbers, and explanatory comments, which reduces robustness and clarity. While the recursive version risks errors for large inputs due to recursion limits, the iterative version is more scalable, though the code overall would benefit from better structure, documentation, and basic testing.  **Question 3: Few-Shot Prompting (Armstrong Number Check)**  Write a **few-shot prompt** by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.  **Examples:**   * Input: 153 → Output: Armstrong Number * Input: 370 → Output: Armstrong Number * Input: 123 → Output: Not an Armstrong Number   **Task:**   * Analyze how multiple examples influence code structure and accuracy.   Uses a while loop to extract digits, compute cubes, and compare the sum to the original number.     * Test the function with boundary values and invalid inputs.     ***(Optional Extension)***  **Question 4: Context-Managed Prompting (Optimized Number Classification)**  Design a **context-managed prompt** with clear instructions and constraints to generate an optimized Python program that classifies a number as **prime, composite, or neither**.  **Task:**   * Ensure proper input validation.      * Optimize the logic for efficiency.      * Compare the output with earlier prompting strategies. * Input: 2 → "2 is prime." * Input: 4 → "4 is composite." * Input: 1 → "1 is neither prime nor composite." * Input: 29 → "29 is prime." (Efficient, ~3 iterations in loop.) * Input: abc → "Invalid input. Please enter an integer." * Vs. Zero-Shot: Might output "2 is prime" but crash on abc. Vs. Few-Shot: Similar accuracy but slower for large n (e.g., 1000003 takes more time without even-check).   **Question 5: Zero-Shot Prompting (Perfect Number Check)**  Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.  Task:   * Record the AI-generated code.      * Test the program with multiple inputs.      * Identify any missing conditions or inefficiencies in the logic.   No edge-case handling:  For num ≤ 0, the logic is invalid (perfect numbers are positive integers).  num = 1 should explicitly be treated as not a perfect number.  Inefficient loop:  The loop checks all numbers from 1 to num-1; it can be optimized by checking divisors only up to num // 2 or √num.  No input validation:  Non-integer or invalid input will cause a runtime error.  **Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)**  Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.  Examples:   * Input: 8 → Output: Even * Input: 15 → Output: Odd * Input: 0 → Output: Even   Task:   * Analyze how examples improve input handling and output clarity.   Examples help explain:  n % 2 == 0 → even number  n % 2 != 0 → odd number     * Test the program with negative numbers and non-integer inputs. | Week2 - Monday |  |