|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hall:2303A51812 J.Tejesh Batach-26** | | | | |
| **Assignment-3.4** | | | | |
|  | **Q.No.** | **Question** | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques  **Task 1: Zero-shot Prompt – Fibonacci Series Generator**  **Task Description #1**  • Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.  **Expected Output #1**  **•** A complete Python function generated by Copilot without any example provided.  • Correct output for sample input N = 7 ➝ 0 1 1 2 3 5 8  • Observation on how Copilot understood the instruction with zero context.  Promt:// Write a Java function that prints the first N Fibonacci numbers  Code:  public class Fibonacci {  public static void printFibonacci(int n) {  int a = 0, b = 1;  for (int i = 0; i < n; i++) {  System.out.print(a + " ");  int next = a + b;  a = b;  b = next;  }  }  public static void main(String[] args) {  printFibonacci(7);  }  }  Output:    Understanding:  In this task, a **zero-shot prompt** was used, which means the instruction was given **without any example**. Only a single comment describing the task (“print first N Fibonacci numbers”) was provided to GitHub Copilot.  **Task 2: One-shot Prompt – List Reversal Function**  **Task Description #2**  • Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.  **Expected Output #2**  • Copilot-generated function to reverse a list using slicing or loop.  • Output: [3, 2, 1] for input [1, 2, 3]  • Observation on how adding a single example improved Copilot’s accuracy.  Promt:  // Write a Java function to reverse a list.  // Example:  // Input: [1, 2, 3]  // Output: [3, 2, 1]  Code:  import java.util.\*;  public class ReverseList {      public static List<Integer> reverseList(List<Integer> list) {          List<Integer> reversed = new ArrayList<>();          for (int i = list.size() - 1; i >= 0; i--) {              reversed.add(list.get(i));          }          return reversed;      }      public static void main(String[] args) {          List<Integer> nums = Arrays.asList(1, 2, 3);          System.out.println(reverseList(nums));      }  }  Output:    Understanding:  helped Copilot clearly understand:   * The direction of reversal * The expected output format   **Task 3: Few-shot Prompt – String Pattern Matching**  **Task Description #3**  • Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.  **Expected Output #3**  • A function is\_valid() that checks the pattern.  • Output: True or False based on input.  • Students reflect on how multiple examples guide Copilot to generate more accurate code.  Promt:  // Write a Java function isValid(String s) that checks:  // "Hello." → true  // "hello." → false  // "Hello" → false  // The string must start with a capital letter and end with a period.  Code:  public class StringPattern {      public static boolean isValid(String s) {          if (s == null || s.length() < 2) {              return false;          }          return Character.isUpperCase(s.charAt(0)) && s.endsWith(".");      }      public static void main(String[] args) {          System.out.println(isValid("Hello.")); // true          System.out.println(isValid("hello.")); // false          System.out.println(isValid("Hello"));  // false      }  }  Output:    Understanding:  In this task, **few-shot prompting** was applied by giving **multiple examples** showing valid and invalid cases.  Examples such as:   * "Hello." → true * "hello." → false * "Hello" → false   helped Copilot infer the **exact rules**:   * The string must start with a capital letter * The string must end with a period   **Task 4: Zero-shot vs Few-shot – Email Validator**  Task Description #4  • First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).  • Then, rewrite the prompt using few-shot examples.  **Expected Output #4**  • Compare both outputs:  Zero-shot may result in basic or generic validation.  Few-shot gives detailed and specific logic (e.g., @ and domain checking).  • Submit both code versions and note how few-shot improves reliability.  Promt:  // Write a Java function to validate an email address.  public class EmailValidatorZeroShot {      public static boolean isValidEmail(String email) {          return email.contains("@") && email.contains(".");      }      public static void main(String[] args) {          System.out.println(isValidEmail("test@gmail.com"));      }  }  Output:    **Understanding:**  **Understanding**  **This task compares zero-shot and few-shot prompting.**  **Zero-shot:**   * **Only a general instruction was given** * **Copilot generated a basic email check using @ and .**   **Few-shot:**   * **Valid and invalid email examples were provided** * **Copilot generated a regex-based validation, which is more accurate and reliable**   **This comparison proves that:**   * **Zero-shot prompts may result in incomplete or weak logic** * **Few-shot prompts improve correctness, robustness, and real-world usability**   **Task 5: Prompt Tuning – Summing Digits of a Number**  **Task Description #5**  • Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.  Style 1: Generic task prompt  Style 2: Task + Input/Output example  **Expected Output #5**  • Two versions of the sum\_of\_digits() function.  • Example Output: sum\_of\_digits(123) ➝ 6  • Short analysis: which prompt produced cleaner or more optimized code and why?  /\* =========================================================  TASK 5: PROMPT TUNING – SUM OF DIGITS  ========================================================= \*/  /\* -------- STYLE 1: GENERIC PROMPT --------  Write a Java function to return the sum of digits of a number.  ------------------------------------------ \*/  class SumOfDigitsGeneric {  public static int sumOfDigits(int n) {  int sum = 0;  while (n > 0) {  sum += n % 10;  n /= 10;  }  return sum;  }  public static void main(String[] args) {  System.out.println(sumOfDigits(123)); // Output: 6  }  }  /\* -------- STYLE 2: PROMPT WITH EXAMPLE --------  Write a Java function to return the sum of digits of a number.  Input: 123  Output: 6  ---------------------------------------------- \*/  class SumOfDigitsExample {  public static int sumOfDigits(int n) {  return String.valueOf(n)  .chars()  .map(Character::getNumericValue)  .sum();  }  public static void main(String[] args) {  System.out.println(sumOfDigits(123)); // Output: 6  }  }  Output:    **Understanding:**  **In this task, two different prompt styles were tested for the same problem.**  **Style 1 (Generic Prompt):**   * **Copilot generated a traditional loop-based solution** * **Code was correct but basic**   **Style 2 (Prompt with Example):**   * **Providing an input-output example guided Copilot to generate a more concise and optimized solution** * **Used modern Java features (streams)**   **This shows that prompt tuning directly affects code quality. Including examples encourages cleaner, more efficient, and readable code.**  **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots** | Week2 |  |