

LAB(3.6) – AI FOR CODING

2303a51731

Lab 3: Prompt Engineering: Improving prompt and context management Objective:

To explore how variations in prompt structure affect the quality, completeness, and accuracy of responses from a large language model.

Requirements:

- VS Code with GitHub Copilot or Cursor API and/or Google Colab with Gemini
- Tasks to be completed are as below.

Task 1: Conceptual Understanding in Physics

Scenario

Suppose that you are a data assistant developer for an EdTech company that uses ChatGPT to answer student queries related to introductory physics.

Tasks to be completed

1. Baseline Prompt Testing

Choose 5 typical user queries, for example:

Week-1

Saturday

“Explain, What, Define, Why, What”

- Run these prompts in a chat-based AI model and record the raw responses.

```
def answer_physics_query(query):
    physics_faq = {
        "What is Newton's First Law?": "Newton's First Law states that an object at rest stays at rest and an object in motion stays in motion with the same speed and in the same direction unless acted upon by an unbalanced force.",
        "Define kinetic energy.": "Kinetic energy is the energy that an object possesses due to its motion, calculated as 0.5 * mass * velocity squared.",
        "Why does an object fall to the ground?": "An object falls to the ground due to the force of gravity acting upon it.",
        "Explain the concept of momentum.": "Momentum is the product of an object's mass and its velocity, representing the quantity of motion it has.",
        "What is the difference between speed and velocity?": "Speed is a scalar quantity representing how fast an object is moving, while velocity is a vector quantity representing both speed and direction."
    }

    return physics_faq.get(query, "I'm sorry, I don't have an answer for that question.")

# Example usage
if __name__ == "__main__":
    user_query = input("Enter your physics question: ")
    answer = answer_physics_query(user_query)
    print(f"Answer: {answer}")

# Explanation of the code.
# The function uses a predefined dictionary of common physics questions
# and their answers to respond to user queries.
# Limitations include the inability to handle questions outside the
# predefined set and lack of detailed explanations for complex topics.
# sample input output pairs
```

Output:

```
Enter your physics question: What is Newton's First Law?  
Answer: Newton's First Law states that an object at rest stays at rest and an object in motion stays in motion unless acted upon by an external force.  
PS C:\Users\Ajay Kumar\Desktop\Ai> & "C:/Program Files/Python39/python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab(3.6).py"  
Enter your physics question: Define kinetic energy.
```

2. Prompt Refinement

Rewrite each query using the following strategies:

- Add 5 different contexts (school level, competitive exam, real-life application, mathematical focus, conceptual focus).
- Make the task explicit (e.g., “List and explain Newton’s three laws with one real-world example each.”).
- Break the query into subtasks (definition → explanation → example).

Run these prompts in a chat-based AI model and record the raw responses.

3. Evaluate Outputs

Score AI responses on a scale of 1–5 using:

- Completeness
- Accuracy
- Relevance • Clarity

Run these prompts in a chat-based AI model and record the raw responses.

Present results in a comparative table.

4. Reflection

Discuss how contextual and structured prompts influenced the depth and correctness of responses.

```

def classify_post(content):
    offensive_keywords = ["hate", "violence", "abuse"]
    spam_keywords = ["buy now", "click here", "subscribe"]

    content_lower = content.lower()

    if any(keyword in content_lower for keyword in offensive_keywords):
        return "Offensive"
    elif any(keyword in content_lower for keyword in spam_keywords):
        return "Spam"
    else:
        return "Acceptable"
# Example usage
if __name__ == "__main__":
    user_input = input("Enter the social media post content: ")
    classification = classify_post(user_input)
    print(f"The post is classified as: {classification}.")
# Explanation of the code.
# The function checks the content for offensive and spam keywords
# and classifies the post accordingly.
# Zero-shot prompting challenges include lack of context, ambiguity
# in language, and difficulty in handling nuanced content without
# giving sample input-output pairs.

```

Output :

```

Enter the social media post content: I hate you
The post is classified as: Offensive.
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:/Program Files/Python39/python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab(3.6).py"
Enter the social media post content: hello everyone
The post is classified as: Acceptable.

```

Task 2: Programming Fundamentals (Python)

Scenario

Suppose that you are a data assistant developer for an EdTech platform that supports beginner programming students.

Tasks to be completed

1. Baseline Prompt Testing

Choose 5 common user queries, such as:

- “What, Explain, How, What, Difference”

Run these prompts in a chat-based AI model and record the raw responses.

2. Prompt Refinement

Rewrite each query by:

- Adding 5 contexts (beginner, exam-oriented, real-world analogy, syntax-focused, performance-focused).

- Making instructions explicit (e.g., “Define a Python loop and show one example for for-loop and while-loop.”).
- Breaking into subtasks (definition → syntax → example → use case).

Run these prompts in a chat-based AI model and record the raw responses.

3. Evaluate Outputs

- Evaluate responses using completeness, accuracy, relevance, and clarity.
- Summarize findings in a table.

4. Reflection

Analyze, how explicit subtasks improve code correctness and explanation quality.

```
def baseline_prompt_testing(queries):
    responses = {}
    for query in queries:
        # Simulate AI model response (replace with actual AI model call)
        responses[query] = f"Raw response for: {query}"
    return responses

def prompt_refinement(queries):
    refined_responses = {}
    contexts = [
        "beginner",
        "exam-oriented",
        "real-world analogy",
        "syntax-focused",
        "performance-focused"
    ]
    for query in queries:
        refined_query = f"{query} with contexts: {''.join(contexts)}. "
        refined_query += "Define, explain syntax, provide example, and use case."
        # Simulate AI model response (replace with actual AI model call)
        refined_responses[refined_query] = f"Refined response for: {refined_query}"
    return refined_responses

def evaluate_outputs(baseline_responses, refined_responses):
    evaluation = {}
    for query in baseline_responses:
        evaluation[query] = {
            "baseline": {
                "completeness": "Good",
                "accuracy": "Good",
                "relevance": "Good",
                "clarity": "Good"
            },
            "refined": {
                "completeness": "Excellent",
                "accuracy": "Excellent",
                "relevance": "Excellent",
                "clarity": "Excellent"
            }
        }
    return evaluation
```

```

def main():
    queries = [
        "What is a loop in Python?",
        "Explain the difference between list and tuple.",
        "How to write a function in Python?",
        "What is recursion?",
        "Difference between Python 2 and Python 3."
    ]
    baseline_responses = baseline_prompt_testing(queries)
    refined_responses = prompt_refinement(queries)
    evaluation = evaluate_outputs(baseline_responses, refined_responses)

    # Print evaluation summary
    for query, eval_data in evaluation.items():
        print(f"Query: {query}")
        print("Baseline Evaluation:", eval_data["baseline"])
        print("Refined Evaluation:", eval_data["refined"])
        print()

if __name__ == "__main__":
    main()
# Explanation of the code.
# The code defines functions to simulate baseline prompt testing,
# prompt refinement, and evaluation of outputs. It processes a list
# of common user queries, refines them with additional context and
# explicit instructions, and evaluates the responses based on
# completeness, accuracy, relevance, and clarity.

```

Output

```

Query: Explain the difference between list and tuple.
Baseline Evaluation: {'completeness': 'Good', 'accuracy': 'Good', 'relevance': 'Good', 'clarity': 'Good'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}

Query: How to write a function in Python?
Baseline Evaluation: {'completeness': 'Good', 'accuracy': 'Good', 'relevance': 'Good', 'clarity': 'Good'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}

Query: What is recursion?
Baseline Evaluation: {'completeness': 'Good', 'accuracy': 'Good', 'relevance': 'Good', 'clarity': 'Good'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}

Query: Difference between Python 2 and Python 3.
Baseline Evaluation: {'completeness': 'Good', 'accuracy': 'Good', 'relevance': 'Good', 'clarity': 'Good'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}

Query: Difference between Python 2 and Python 3.
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}
Refined Evaluation: {'completeness': 'Excellent', 'accuracy': 'Excellent', 'relevance': 'Excellent', 'clarity': 'Excellent'}

```

Task 3: Data Science and Machine Learning Concepts

Scenario

Suppose that you are a data assistant developer for an EdTech company offering data science courses.

Tasks to be completed

1. Baseline Prompt Testing

Select 5 typical queries, for example:

- “What, Explain, What, Define, What”

Run these prompts in a chat-based AI model and record the raw responses.

2. Prompt Refinement

Refine each query by:

- Adding 5 contexts (academic, industry, beginner-friendly, mathematical, interview-focused).
- Making tasks explicit (e.g., “Define supervised learning and explain it with one real-world example.”).
- Breaking into subtasks (definition → types → example → limitation).

Run these prompts in a chat-based AI model and record the raw responses.

3. Evaluate Outputs

Score outputs using the given metrics and present results in a table.

4. Reflection

Reflect on how context management affects conceptual clarity in technical domains.

```
20 # 4. Reflection
21 # Reflect on how context management affects conceptual clarity in technical
22 # domains.
23 #generate the python code for above task
24 # Since the tasks involve running prompts in a chat-based AI model and recording
25 # responses, we can create a Python script that outlines the steps to be taken.
26
27 def run_prompt_testing(queries, contexts):
28     results = []
29     for query in queries:
30         for context in contexts:
31             refined_prompt = f"Context: {context}\nTask: {query}"
32             # Here we would normally call the chat-based AI model API
33             # For demonstration, we'll just simulate a response
34             response = f"Simulated response for prompt: '{refined_prompt}'"
35             results.append((refined_prompt, response))
36     return results
37 if __name__ == "__main__":
38     typical_queries = [
39         "What is supervised learning?",
40         "Explain the bias-variance tradeoff.",
41         "Define clustering in machine learning.",
42         "What are decision trees?",
43         "What is overfitting?"
44     ]
```

Output:

Prompt:
Context: industry
Task: What is supervised learning?
Response:
Simulated response for prompt: 'Context: industry
Task: What is supervised learning?'

Prompt:
Context: beginner-friendly
Task: What is supervised learning?
Response:
Simulated response for prompt: 'Context: beginner-friendly
Task: What is supervised learning?'

Prompt:
Context: mathematical
Task: What is supervised learning?
Response:
Simulated response for prompt: 'Context: mathematical
Task: What is supervised learning?'

Prompt:
Context: interview-focused
Task: What is supervised learning?
Response:
Simulated response for prompt: 'Context: interview-focused
Task: What is supervised learning?'

Task 4: Database and SQL Queries

Scenario

Suppose that you are a data assistant developer supporting students learning database systems.

Tasks to be completed

1. Baseline Prompt Testing

Choose 5 common queries, such as:

- “Explain, What, Difference, where, how”

Run these prompts in a chat-based AI model and record the raw responses.

2. Prompt Refinement

Rewrite each prompt by:

- Adding 5 contexts (theory exam, practical lab, interview prep, real-world database, optimization focus).
- Making instructions explicit (e.g., “Explain SQL JOIN types with syntax and examples.”).
- Breaking into subtasks (definition → syntax → example → use case).

Run these prompts in a chat-based AI model and record the raw responses.

3. Evaluate Outputs

Evaluate responses using the four metrics and summarize results in a comparison table.

4. Reflection

Discuss how refined prompts reduce ambiguity in technical explanations.

```
def explain_sql_query(query_type):  
    explanations = {  
        "difference": {  
            "definition": "The SQL DIFFERENCE function compares the SOUNDEX values of two strings and returns an integer value indicating the difference between them.",  
            "syntax": "DIFFERENCE(string1, string2)",  
            "example": "SELECT DIFFERENCE('smith', 'smyth'); -- Returns 4",  
            "use_case": "Used to find similar-sounding strings in databases."  
        },  
        "where": {  
            "definition": "The WHERE clause is used to filter records in a SQL query based on specified conditions.",  
            "syntax": "SELECT column1, column2 FROM table_name WHERE condition;",  
            "example": "SELECT * FROM Employees WHERE Age > 30;",  
            "use_case": "Used to retrieve specific records that meet certain criteria."  
        },  
        "how": {  
            "definition": "The HOW keyword is not a standard SQL keyword but can refer to methods of performing operations in SQL.",  
            "syntax": "N/A",  
            "example": "N/A",  
            "use_case": "Understanding different methods to achieve database tasks."  
        },  
        "what": {  
            "definition": "The WHAT keyword is not a standard SQL keyword but can refer to understanding the purpose of SQL command.",  
            "syntax": "N/A",  
            "example": "N/A",  
            "use_case": "Clarifying the intent behind SQL queries."  
        },  
        "explain": {  
            "definition": "The EXPLAIN statement provides information about how SQL statements are executed by the database engine.",  
            "syntax": "EXPLAIN SELECT * FROM table_name;",  
            "example": "EXPLAIN SELECT * FROM Employees;",  
            "use_case": "Used for query optimization and understanding execution plans."  
        }  
    }  
    return explanations[query_type]
```

```

# Example usage
if __name__ == "__main__":
    query = input("Enter the SQL query type (difference, where, how, what, explain): ")
    explanation = explain_sql_query(query)

    if isinstance(explanation, dict):
        print(f"Definition: {explanation['definition']}")
        print(f"Syntax: {explanation['syntax']}")
        print(f"Example: {explanation['example']}")
        print(f"Use Case: {explanation['use_case']}")
    else:
        print(explanation)

```

Output:

```

Enter the SQL query type (difference, where, how, what, explain): difference
Definition: The SQL DIFFERENCE function compares the SOUNDEX values of two strings and returns an integer value indicating the
Enter the SQL query type (difference, where, how, what, explain): difference
Definition: The SQL DIFFERENCE function compares the SOUNDEX values of two strings and returns an integer value indicating the
Syntax: DIFFERENCE(string1, string2)
Example: SELECT DIFFERENCE('smith', 'Smyth'); -- Returns 4
Example: SELECT DIFFERENCE('smith', 'Smyth'); -- Returns 4
Use Case: Used to find similar-sounding strings in databases.
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:/Program Files/Python39/python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/
Enter the SQL query type (difference, where, how, what, explain): where
Definition: The WHERE clause is used to filter records in a SQL query based on specified conditions.
Syntax: SELECT column1, column2 FROM table_name WHERE condition;
Example: SELECT * FROM Employees WHERE Age > 30;
Use Case: Used to retrieve specific records that meet certain criteria.

```

Task 5: General Aptitude and Logical Reasoning

Scenario

Suppose that you are a data assistant developer for an EdTech company focused on aptitude and competitive exam preparation.

Tasks to be completed

1. Baseline Prompt Testing

Select 5 user queries, for example:

- “Explain, What, Difference, where, how”

Run these prompts in a chat-based AI model and record the raw responses.

2. Prompt Refinement

Rewrite each query by:

- Adding 5 contexts (school exams, competitive exams, real-life analogy, formula-based, step-by-step solving).
- Making tasks explicit (e.g., “Define probability and solve one simple numerical example.”).

- Breaking into subtasks (definition → formula → example → common mistakes).

Run these prompts in a chat-based AI model and record the raw responses.

3. Evaluate Outputs

- Score responses using completeness, accuracy, relevance, and clarity.
- Present findings in a table.

4. Reflection

Reflect on how structured prompts improve step-by-step reasoning and learner understanding.

```
def baseline_prompt_testing(queries):
    responses = {}
    for query in queries:
        # Simulate AI model response (replace with actual model call)
        response = f"Raw response for query: {query}"
        responses[query] = response
    return responses

def prompt_refinement(queries):
    refined_responses = {}
    for query in queries:
        refined_query = (f"Context: school exams, competitive exams, real-life analogy, "
                        f"formula-based, step-by-step solving. "
                        f"Task: Define the concept, provide formula, solve an example, "
                        f"highlight common mistakes for the query: {query}")
        # Simulate AI model response (replace with actual model call)
        response = f"Refined response for query: {refined_query}"
        refined_responses[refined_query] = response
    return refined_responses

def evaluate_outputs(baseline_responses, refined_responses):
    evaluation_table = []
    for query in baseline_responses:
        baseline_response = baseline_responses[query]
        refined_query = (f"Context: school exams, competitive exams, real-life analogy, "
                        f"formula-based, step-by-step solving. "
                        f"Task: Define the concept, provide formula, solve an example, "
                        f"highlight common mistakes for the query: {query}")
        refined_response = refined_responses[refined_query]

        # Simulate scoring (replace with actual scoring logic)
        baseline_score = {"completeness": 2, "accuracy": 2, "relevance": 2, "clarity": 2}
        refined_score = {"completeness": 4, "accuracy": 4, "relevance": 4, "clarity": 4}

        evaluation_table.append({
```

```

def reflect_on_improvements(evaluation_table):
    refined_score = entry["Refined Score"]

    improvement = {
        "Query": query,
        "Baseline Total Score": sum(baseline_score.values()),
        "Refined Total Score": sum(refined_score.values()),
        "Improvement": sum(refined_score.values()) - sum(baseline_score.values())
    }
    reflections.append(improvement)
return reflections

# Example usage
if __name__ == "__main__":
    user_queries = [
        "Explain probability",
        "What is the difference between mean and median?",
        "Where is the formula for area of circle used?",
        "How to solve quadratic equations?",
        "Define permutations and combinations"
    ]

    baseline_responses = baseline_prompt_testing(user_queries)
    refined_responses = prompt_refinement(user_queries)
    evaluation_table = evaluate_outputs(baseline_responses, refined_responses)
    reflections = reflect_on_improvements(evaluation_table)

    # Print evaluation table
    for entry in evaluation_table:
        print(entry)

    # Print reflections
    for reflection in reflections:
        print(reflection)

```

Output:

```

{'Query': 'Explain probability', 'Baseline Response': 'Raw response for query: Explain probability', 'Refined Response': 'Refined response for query: Context: school exams, competitive exams, real-life analogy, formula-based, step-by-step solving. Task: Define the concept, provide formula, solve an example, highlight common mistakes for the query: Explain probability', 'Baseline Score': {'completeness': 2, 'accuracy': 2, 'relevance': 2, 'clarity': 2}, 'Refined Score': {'completeness': 4, 'accuracy': 4, 'relevance': 4, 'clarity': 4}}
{'Query': 'What is the difference between mean and median?', 'Baseline Response': 'Raw response for query: What is the difference between mean and median?', 'Refined Response': 'Refined response for query: Context: school exams, competitive exams, real-life analogy, formula-based, step-by-step solving. Task: Define the concept, provide formula, solve an example, highlight common mistakes for the query: What is the difference between mean and median?', 'Baseline Score': {'completeness': 2, 'accuracy': 2, 'relevance': 2, 'clarity': 2}, 'Refined Score': {'completeness': 4, 'accuracy': 4, 'relevance': 4, 'clarity': 4}}
{'Query': 'Where is the formula for area of circle used?', 'Baseline Response': 'Raw response for query: Where is the formula for area of circle used?', 'Refined Response': 'Refined response for query: Context: school exams, competitive exams, real-life analogy, formula-based, step-by-step solving. Task: Define the concept, provide formula, solve an example, highlight common mistakes for the query: Where is the formula for area of circle used?', 'Baseline Score': {'completeness': 2, 'accuracy': 2, 'relevance': 2, 'clarity': 2}, 'Refined Score': {'completeness': 4, 'accuracy': 4, 'relevance': 4, 'clarity': 4}}
{'Query': 'How to solve quadratic equations?', 'Baseline Response': 'Raw response for query: How to solve quadratic equations?', 'Refined Response': 'Refined response for query: Context: school exams, competitive exams, real-life analogy, formula-based, step-by-step solving. Task: Define the concept, provide formula, solve an example, highlight common mistakes for the query: How to solve quadratic equations?', 'Baseline Score': {'completeness': 2, 'accuracy': 2, 'relevance': 2, 'clarity': 2}, 'Refined Score': {'completeness': 4, 'accuracy': 4, 'relevance': 4, 'clarity': 4}}
{'Query': 'Define permutations and combinations', 'Baseline Response': 'Raw response for query: Define permutations and combinations', 'Refined Response': 'Refined response for query: Context: school exams, competitive exams, real-life analogy, formula-based, step-by-step solving. Task: Define the concept, provide formula, solve an example, highlight common mistakes for the query: Define permutations and combinations', 'Baseline Score': {'completeness': 2, 'accuracy': 2, 'relevance': 2, 'clarity': 2}, 'Refined Score': {'completeness': 4, 'accuracy': 4, 'relevance': 4, 'clarity': 4}}
{'Query': 'Explain probability', 'Baseline Total Score': 8, 'Refined Total Score': 16, 'Improvement': 8}
{'Query': 'What is the difference between mean and median?', 'Baseline Total Score': 8, 'Refined Total Score': 16, 'Improvement': 8}
{'Query': 'Where is the formula for area of circle used?', 'Baseline Total Score': 8, 'Refined Total Score': 16, 'Improvement': 8}
{'Query': 'How to solve quadratic equations?', 'Baseline Total Score': 8, 'Refined Total Score': 16, 'Improvement': 8}
{'Query': 'Define permutations and combinations', 'Baseline Total Score': 8, 'Refined Total Score': 16, 'Improvement': 8}

```