# AI ASSISTED CODING

**Name: Gattu Varshith**

**Roll: 2403A510E3**

**Batch: 05**

**QUESTION 1: Zero-shot Classification**

**Task 1:**

Scenario: Classify tweets into “Positive,” “Negative,” or “Neutral.”  
Write a zero-shot prompt to classify sentiment without any examples.

**PROMPT:**

Classify the sentiment of the following tweet into Positive, Negative, or Neutral.

Tweet: "My Python code finally ran without any errors after 3 hours of debugging!"

**CODE GENERATED:**

tweet = "My Python code finally ran without any errors after 3 hours of debugging!"

positive\_keywords = ["finally", "ran without any errors", "success", "solved", "fixed", "great", "happy", "relieved"]

negative\_keywords = ["error", "fail", "problem", "issue", "debugging", "struggle", "frustrated", "bad"]

tweet\_lower = tweet.lower()

sentiment = "Neutral"

if any(word in tweet\_lower for word in positive\_keywords):

    sentiment = "Positive"

elif any(word in tweet\_lower for word in negative\_keywords):

    sentiment = "Negative"

print(f"Sentiment: {sentiment}")

**OUTPUT:**

Sentiment: Positive

**OBSERVATION:**

The code checks the tweet for positive and negative keywords. If it finds any positive word, it marks the tweet as *Positive*. If it finds a negative word, it marks it as *Negative*. If no keywords match, it gives *Neutral*. In this case, since the tweet has the word *“finally”*, the output is *Positive*.

**TASK 2:**

Create a scenario where an AI assistant needs to help a student solve math problems. Write two prompts: one without context and one with detailed context (e.g., grade level, topic, difficulty).

**Without context:**

**PROMPT:**

Generate a python code for finding prime numbers between a range.

**Code Generated:**

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

return False

return True

start = 1

end = 10

print(f"Prime numbers between {start} and {end}:")

for number in range(start, end + 1):

if is\_prime(number):

print(number)

**Output:**

**Prime numbers between 1 and 10:**

**2**

**3**

**5**

**7**

**Observation:**

The code checks each number from 1 to 10 using the is\_prime function. If a number is prime, it prints it. The output shows the prime numbers between 1 and 10. This is a basic code because I did not give any context.

**WITH CONTEXT:**

**PROMPT:**

You are a coding assistant helping a BTech 2nd year CSE student. The student is learning Python basics and wants to write a program that prints all prime numbers between a given range. Provide step-by-step explanation and generate the Python code for checking prime numbers between 1 and 10.

**CODE GENERATED:**

for num in range(1, 11):

if num > 1:

is\_prime = True

for i in range(2, num):

if num % i == 0:

is\_prime = False

break

if is\_prime:

print(num)

**OUTPUT:**

1. **Understand what a prime number is:**  
   A prime number is a number greater than 1 that has no divisors other than 1 and itself.
2. **Loop through each number in the range:**  
   Check each number from 1 to 10.
3. **Check if the number is prime:**  
   For each number, check if it is divisible by any number between 2 and itself minus 1.  
   If it is not divisible by any of these, it is prime.
4. **Print the prime numbers:**  
   Print the numbers that are prime.

**CODE OUTPUT:**

2

3

5

7

**OBSERVATION:**

The code loops from 1 to 10 and checks each number by dividing it with smaller numbers. If no exact division is found, the number is prime and gets printed. This prints the prime numbers between 1 and 10. This code is more basic and it also explained every step line by line and made easy for the student to understand.

**Q2. One-shot vs Few-shot**

**ONE SHOT PROMPT:**

Generate a python code to give Fibonacci series in a specified range as output

Example

Range(input): 10

Output:1 2 3 5 8

**CODE GENERATED:**

n = int(input("Enter the range: "))

a, b = 1, 2

while a <= n:

    print(a, end=" ")

    a, b = b, a + b

**OUTPUT:**

Enter the range: 20

1 2 3 5 8 13

**OBSERVATION:**

The code starts with two numbers 1 and 2, then adds them repeatedly to generate the Fibonacci series. It prints all Fibonacci numbers up to the given range value and it was very basic.

**FEW SHOT PROMPT:**

Generate a python code to give fibinocci series using functions in a specified range as output

Example

Range(input): 10

Output:1 2 3 5 8

Input: 20

Output: 1 2 3 5 8 13

Input: 30

Output: 1 2 3 5 8 13 21

Input: 40

Output: 1 2 3 5 8 13 21 34

**CODE GENERATED:**

def fibonacci\_upto\_n(n):

a, b = 1, 2

result = []

while a <= n:

result.append(a)

a, b = b, a + b

return result

n = int(input("Enter the range: "))

fib\_series = fibonacci\_upto\_n(n)

print(" ".join(str(num) for num in fib\_series))

**OUTPUT:**

Enter the range: 40

1 2 3 5 8 13 21 34

**OBSERVATION:**

In one-shot prompting, the AI is shown only **one example** before solving the task, so its understanding may be limited. In few-shot prompting, the AI is given **multiple examples**, which helps it learn the pattern better and usually produce more accurate results and the code generated is more functional and correct.

**TASK 2:**

**One shot prompt:**

Classify the sentiment of tweets as Positive, Negative, or Neutral.

Example:

Tweet: "Python is so easy to understand!"

Sentiment: Positive

Now classify:

Tweet: "Indentation errors in Python are so annoying."

**CODE GENERATED:**

def classify\_sentiment(tweet):

    positive\_keywords = ["easy", "love", "great", "awesome", "good", "understand", "enjoy", "fun"]

    negative\_keywords = ["annoying", "error", "hate", "bad", "problem", "difficult", "frustrating"]

    tweet\_lower = tweet.lower()

    if any(word in tweet\_lower for word in positive\_keywords):

        return "Positive"

    elif any(word in tweet\_lower for word in negative\_keywords):

        return "Negative"

    else:

        return "Neutral"

# Example usage

tweets = [

    "Indentation errors in Python are so annoying."

]

for t in tweets:

    print(f'Sentiment: {classify\_sentiment(t)}\n')

**OUTPUT:**

Sentiment: Negative

**OBSERVATION:**

The code is simple and functional. It uses keyword matching to classify a tweet as Positive, Negative, or Neutral. While it works for basic cases, it is limited because it only looks for specific words and may not handle complex sentences correctly.

**Few shot prompt:**

Classify the sentiment of tweets as Positive, Negative, or Neutral.

Examples:

Tweet: "Python is so easy to understand!"

Sentiment: Positive

Tweet: "Debugging Python code can be very stressful."

Sentiment: Negative

Tweet: "Just installed Python, let’s see how it goes."

Sentiment: Neutral

Tweet: "I solved my first coding problem in Python today!"

Sentiment: Positive

Now classify:

Tweet: "Indentation errors in Python are so annoying."

**CODE GENERATED:**

def classify\_sentiment(tweet):

    positive\_keywords = ["easy", "love", "great", "awesome", "good", "understand", "enjoy", "fun", "solved", "success"]

    negative\_keywords = ["annoying", "error", "hate", "bad", "problem", "difficult", "frustrating", "stressful"]

    tweet\_lower = tweet.lower()

    if any(word in tweet\_lower for word in positive\_keywords):

        return "Positive"

    elif any(word in tweet\_lower for word in negative\_keywords):

        return "Negative"

    else:

        return "Neutral"

# Example tweets

tweets = [

    "Indentation errors in Python are so annoying."

]

for t in tweets:

    print(f'Sentiment: {classify\_sentiment(t)}\n')

tweets = [

    "Indentation errors in Python are so annoying."

]

for t in tweets:

    print(f'Sentiment: {classify\_sentiment(t)}\n')

**OUTPUT:**

Sentiment: Negative

**OBSERVATION:**

The code is straightforward and works for basic sentiment analysis. It classifies tweets by checking for positive or negative keywords and labels the rest as Neutral. It is functional but limited since it depends only on fixed words

**Comparison Explanation**

**One-shot**: Only one example or a small set of keywords, sometimes misclassifies Neutral or Positive tweets.

**Few-shot**: Multiple examples/keywords, provides better context, so classification is more accurate and balanced.