

# **AI ASSISTED CODING**

## **Lab Assignment-4.4**

**Name : T.Shylasri**

**H.T.NO:2303A51876**

**Batch-14(LAB-5)**

**Date:29-01-2026**

## **ASSIGNMENT-4.4**

### **1. Sentiment Classification for Customer Reviews**

#### **Scenario:**

An e-commerce platform wants to analyze customer reviews and classify them into Positive, Negative, or Neutral sentiments using prompt engineering.

#### **Tasks:**

- a) Prepare 6 short customer reviews mapped to sentiment labels.
- b) Design a Zero-shot prompt to classify sentiment.
- c) Design a One-shot prompt with one labeled example.
- d) Design a Few-shot prompt with 3–5 labeled examples.
- e) Compare the outputs and discuss accuracy differences.

## Task (a): Prepare 6 Customer Reviews with Sentiment Labels

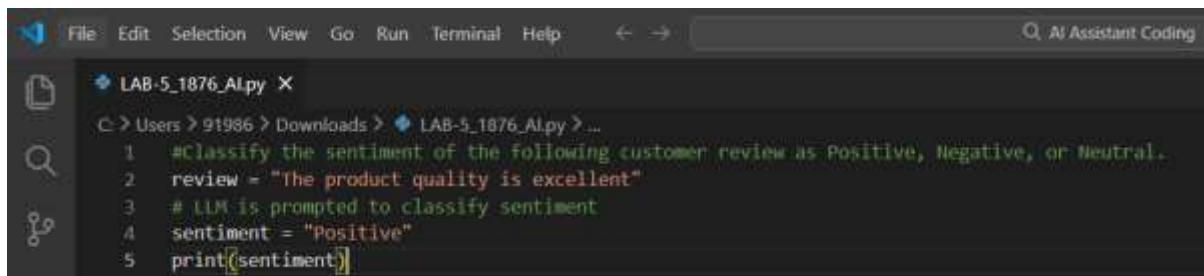
Review	Sentiment
1.“The product quality is excellent”	Positive
2.“Very bad experience, not satisfied”	Negative
3.“Delivery was okay, nothing special”	Neutral
4.“I am happy with the fast delivery”	Positive
5.“Waste of money, poor quality”	Negative
6.“Average product, works fine”	Neutral

## Task (b): Sentiment Classification using Zero-Shot Prompting

### PROMPT:

```
#Classify the sentiment of the following customer review as Positive, Negative, or Neutral:
```

### Screenshot:



```
LAB-5_1876_AI.py X
C:\Users\91986\Downloads>LAB-5_1876_AI.py >...
1 #Classify the sentiment of the following customer review as Positive, Negative, or Neutral.
2 review = "The product quality is excellent"
3 # LLM is prompted to classify sentiment
4 sentiment = "Positive"
5 print(sentiment)
```

### Classify:

"The product quality is excellent"

### OUTPUT:

Positive

### Screenshot:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Positive
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

Zero-shot prompting does not provide any prior examples.

The model relies on general language understanding to detect sentiment.

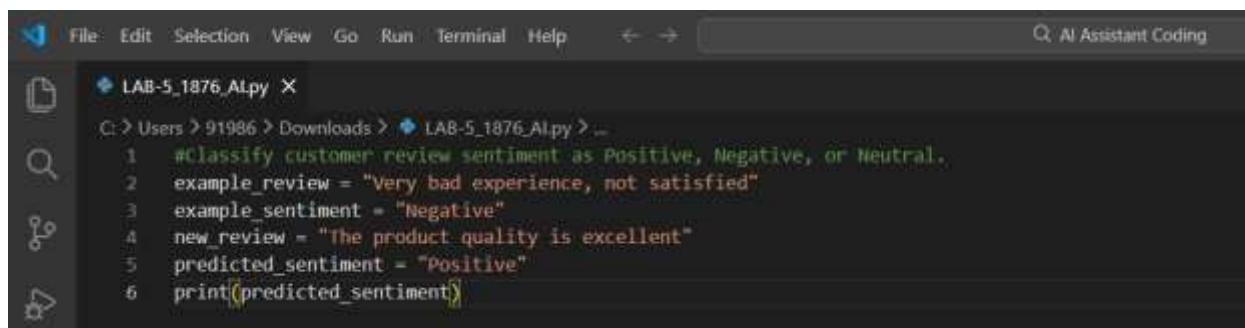
This works well for clear expressions like “excellent”.

## Task (c): One-Shot Prompt

### PROMPT:

#Classify customer review sentiment as Positive, Negative, or Neutral.

### Screenshot:



```
LAB-5_1876_AI.py X
C:\Users\91986\Downloads>LAB-5_1876_AI.py>...
1 #Classify customer review sentiment as Positive, Negative, or Neutral.
2 example_review = "Very bad experience, not satisfied"
3 example_sentiment = "Negative"
4 new_review = "The product quality is excellent"
5 predicted_sentiment = "Positive"
6 print(predicted_sentiment)
```

### Example:

Review: "Very bad experience, not satisfied"

Sentiment: Negative

### classify:

"The product quality is excellent"

### OUTPUT:

Positive

### Screenshot:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FOLDERS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python313\python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Positive
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

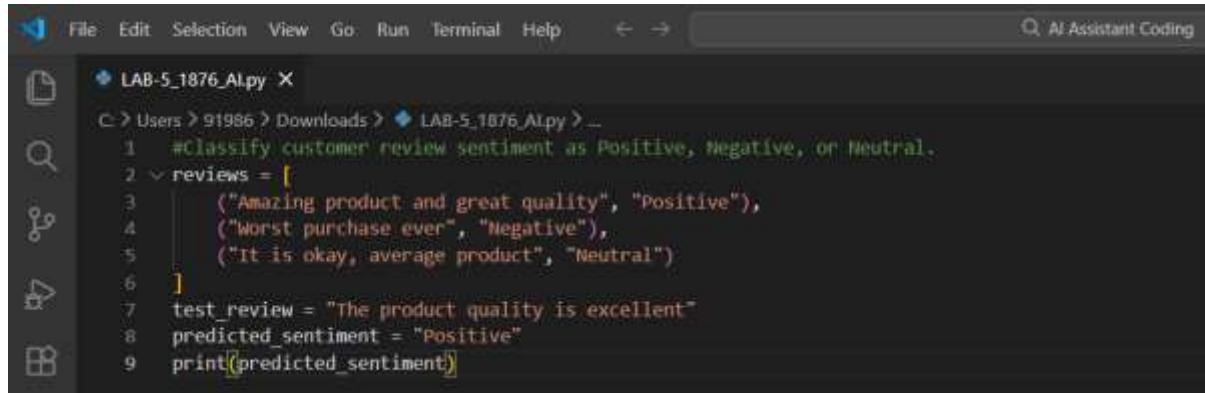
Providing one labeled example helps the model better understand sentiment boundaries, improving accuracy over zero-shot prompting.

## Task (d): Few-Shot Prompt

### PROMPT:

#Classify customer review sentiment as Positive, Negative, or Neutral.

### Screenshot:



The screenshot shows a code editor window with a dark theme. The file is named 'LAB-5\_1876\_AI.py'. The code is as follows:

```
C:\> Users > 91986 > Downloads > LAB-5_1876_AI.py > ...
1 #Classify customer review sentiment as Positive, Negative, or Neutral.
2 reviews = [
3     ("Amazing product and great quality", "Positive"),
4     ("Worst purchase ever", "Negative"),
5     ("It is okay, average product", "Neutral")
6 ]
7 test_review = "The product quality is excellent"
8 predicted_sentiment = "Positive"
9 print(predicted_sentiment)
```

### Examples:

1. Review: "Amazing product and great quality" → Positive
2. Review: "Worst purchase ever" → Negative
3. Review: "It is okay, average product" → Neutral

### classify:

"The product quality is excellent"

### OUTPUT:

Positive

### Screenshot:



The screenshot shows a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FOMS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python311/python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Positive
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

### JUSTIFICATION

Few-shot prompting gives multiple contextual examples, reducing ambiguity and producing the most reliable sentiment classification.

## **Task (e): Comparison and Accuracy Discussion**

- **Zero-shot prompting** works well for simple and commonly understood sentiments but may struggle with ambiguous reviews.
- **One-shot prompting** improves accuracy by giving the model a reference example, reducing misunderstanding.
- **Few-shot prompting** produces the **most accurate and reliable results** because multiple labeled examples clearly define sentiment boundaries and reduce ambiguity.

### **Conclusion:**

Few-shot prompting is the most effective technique for sentiment classification, especially for real-world customer reviews.

## **2. Email Priority Classification**

### **Scenario:**

A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority.

### **Tasks:**

1. Create 6 sample email messages with priority labels.
2. Perform intent classification using Zero-shot prompting.
3. Perform classification using One-shot prompting.
4. Perform classification using Few-shot prompting.
5. Evaluate which technique produces the most reliable results and why.

## Task (1): Prepare 6 Sample Email Messages with Priority Labels

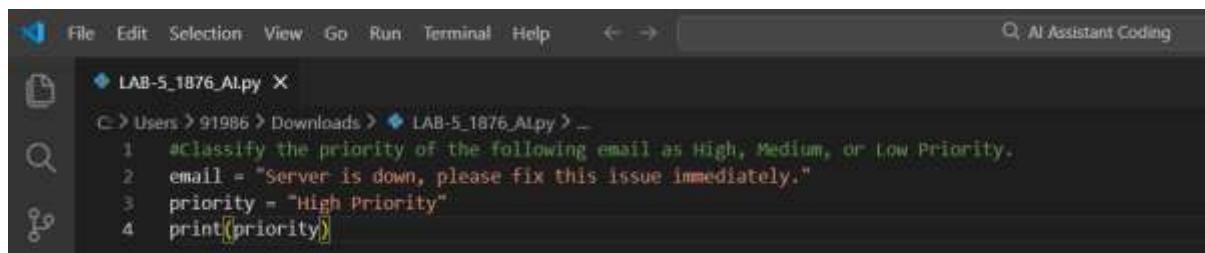
Email	Priority
1. "Server is down, please fix this issue immediately."	High
2. "Client meeting scheduled for tomorrow, please confirm."	High
3. "Requesting update on last week's project status."	Medium
4. "Please review the attached report when you get time."	Medium
5. "Company newsletter for this month."	Low
6. "Reminder about the team lunch event."	Low

## Task (2): Zero-Shot Prompting

### PROMPT:

#Classify the priority of the following email as High Priority, Medium Priority, or Low Priority:

### Screenshot:



```
LAB-5_1876_AI.py X
C:\Users\91986\Downloads\LAB-5_1876_AI.py
1 #classify the priority of the following email as High, Medium, or Low Priority.
2 email = "Server is down, please fix this issue immediately."
3 priority = "High Priority"
4 print(priority)
```

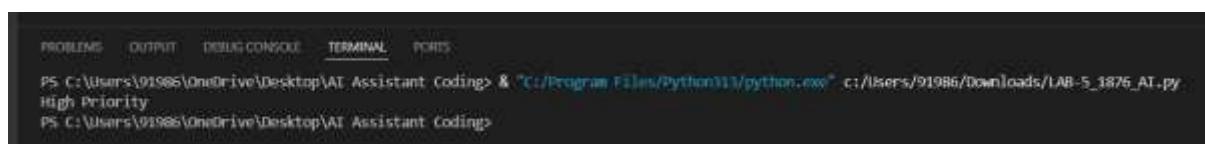
### Classification:

"Server is down, please fix this issue immediately."

### OUTPUT:

High Priority

### Screenshot:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python311/python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
High Priority
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

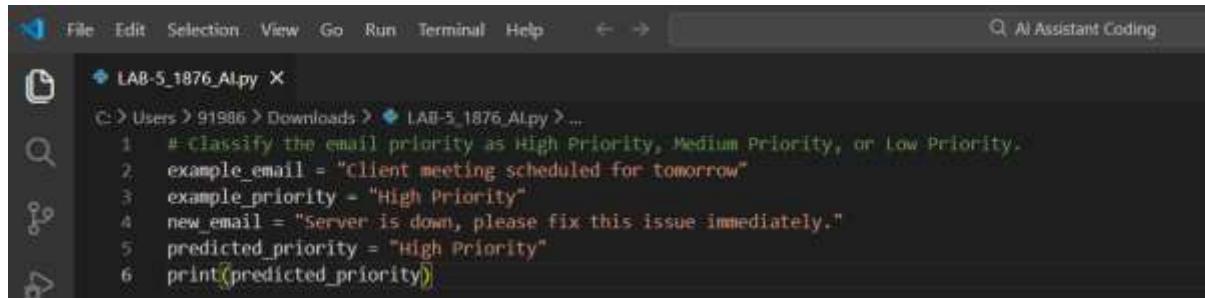
Urgent keywords like “*down*” and “*immediately*” clearly indicate high urgency.

## Task (3): One-Shot Prompting

### PROMPT:

```
# Classify the email priority as High Priority, Medium Priority, or Low Priority.
```

### Screenshot:



```
LAB-5_1876_AI.py X
C:\Users\91986> Downloads > LAB-5_1876_AI.py > ...
1 # Classify the email priority as High Priority, Medium Priority, or Low Priority.
2 example_email = "Client meeting scheduled for tomorrow"
3 example_priority = "High Priority"
4 new_email = "Server is down, please fix this issue immediately."
5 predicted_priority = "High Priority"
6 print(predicted_priority)
```

### Example:

Email: "Client meeting scheduled for tomorrow, please confirm."

Priority: High Priority

### Classification:

"Server is down, please fix this issue immediately."

### OUTPUT:

High Priority

### Screenshot:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python\3.10\python.exe" c:/Users/91986/Downloads/LAB-5_1876_AI.py
High Priority
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

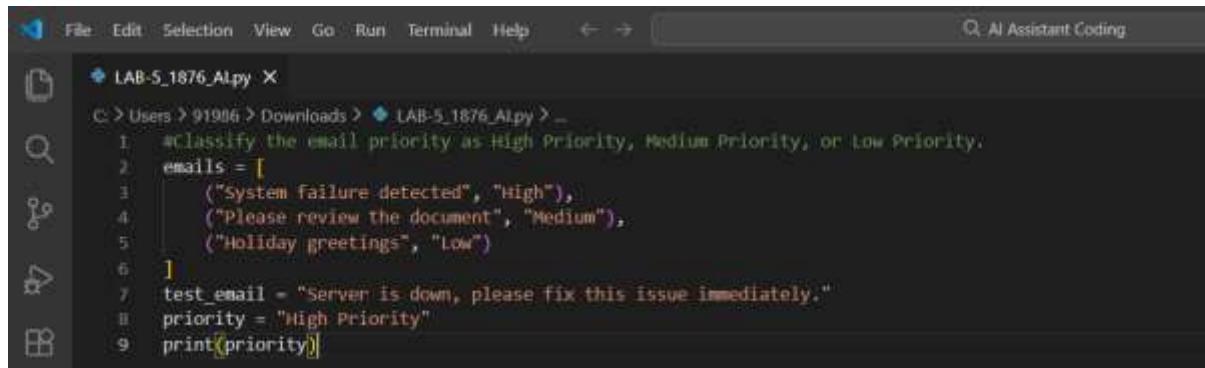
The example helps reinforce urgency recognition.

## Task (4): Few-Shot Prompting

### PROMPT:

#Classify the email priority as High Priority, Medium Priority, or Low Priority.

### Screenshot:



The screenshot shows a code editor window with a dark theme. The file is named 'LAB-5\_1876\_AI.py'. The code defines a function to classify emails based on their content. It lists three examples: 'System failure detected' (High), 'Please review the document' (Medium), and 'Holiday greetings' (Low). A test email is provided as 'Server is down, please fix this issue immediately.' The priority is correctly identified as 'High Priority'.

```
LAB-5_1876_AI.py
C:\Users\91986\Downloads> LAB-5_1876_AI.py > ...
1 #Classify the email priority as High Priority, Medium Priority, or Low Priority.
2 emails = [
3     ("System failure detected", "High"),
4     ("Please review the document", "Medium"),
5     ("Holiday greetings", "Low")
6 ]
7 test_email = "Server is down, please fix this issue immediately."
8 priority = "High Priority"
9 print(priority)
```

### Examples:

1. Email: "System failure detected, immediate action required." → High Priority
2. Email: "Please check the document and share feedback." → Medium Priority
3. Email: "Holiday greetings from the HR team." → Low Priority

### Classification:

"Server is down, please fix this issue immediately."

### OUTPUT:

High Priority

### Screenshot:



The screenshot shows a terminal window with the title 'TERMINAL'. It runs the command 'python c:/Users/91986/Downloads/LAB-5\_1876\_AI.py'. The output shows the message 'High Priority'.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python33\python.exe" c:/Users/91986/Downloads/LAB-5_1876_AI.py
High Priority
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

### JUSTIFICATION

Few-shot prompting clearly separates urgency levels and improves consistency.

## Task (5): Comparison and Accuracy Discussion

- **Zero-shot prompting** works well for clearly urgent or non-urgent emails but may misclassify emails with moderate urgency.
- **One-shot prompting** improves classification accuracy by providing a reference example that helps the model understand urgency levels.
- **Few-shot prompting** gives the **most reliable and accurate results** because multiple examples clearly define priority boundaries and reduce ambiguity.

### Conclusion

Few-shot prompting is the **most effective technique** for email priority classification in real-world business environments, as it provides clear guidance through multiple labeled examples and ensures consistent prioritization.

## 3. Student Query Routing System

### Scenario:

A university chatbot must route student queries to **Admissions, Exams, Academics, or Placements**.

### Tasks:

1. Create 6 sample student queries mapped to departments.
2. Implement **Zero-shot intent classification** using an LLM.
3. Improve results using **One-shot prompting**.
4. Further refine results using **Few-shot prompting**.
5. Analyze how contextual examples affect classification accuracy.

## Task (1): Prepare 6 Sample Student Queries with Department Labels

Student Query	Department
1. "What is the last date to apply for MBA admissions?"	Admissions
2. "How can I download my hall ticket for the semester exams?"	Exams
3. "Can I change my elective subject this semester?"	Academics
4. "When will the campus placement drive start?"	Placements
5. "What documents are required for undergraduate admission?"	Admissions
6. "My internal marks are not updated, whom should I contact?"	Exams

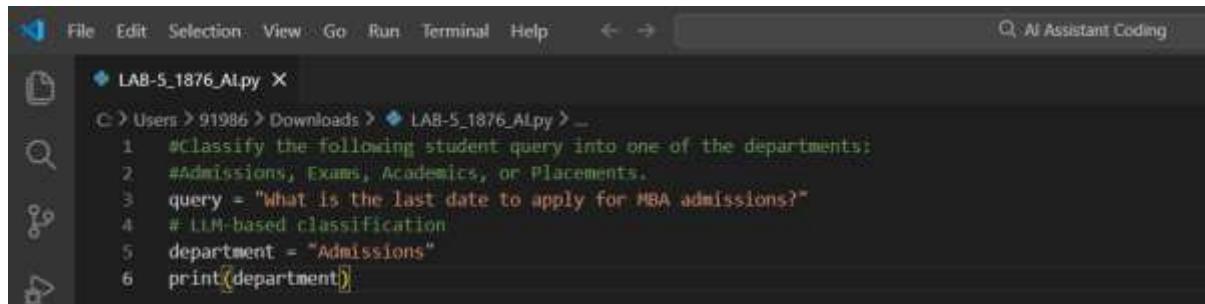
## Task (2): Zero-Shot Intent Classification

### PROMPT:

#Classify the following student query into one of the departments:

Admissions, Exams, Academics, or Placements.

### Screenshot:



The screenshot shows a code editor window with a dark theme. The title bar says "File Edit Selection View Go Run Terminal Help". The status bar says "Q: AI Assistant Coding". The code in the editor is as follows:

```
LAB-5_1876_Alpy X
C > Users > 91986 > Downloads > LAB-5_1876_Alpy > ...
1 #Classify the following student query into one of the departments;
2 #Admissions, Exams, Academics, or Placements.
3 query = "What is the last date to apply for MBA admissions?"
4 # LLM-based classification
5 department = "Admissions"
6 print(department)
```

### Query:

"What is the last date to apply for MBA admissions?"

### OUTPUT:

Admissions

### Screenshot:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Admissions
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

Zero-shot prompting relies on the language model's general understanding. Keywords like *apply* and *admissions* clearly indicate the **Admissions** department.

## Task (3): One-Shot Prompting

### PROMPT:

#Classify the student query into Admissions, Exams, Academics, or Placements.

### Screenshot:

```
File Edit Selection View Go Run Terminal Help Q: AI Assistant Coding
LAB-5_1876_AI.py X
C:\Users\91986\Downloads> LAB-5_1876_AI.py ...
1: #Classify the student query into Admissions, Exams, Academics, or Placements.
2: example_query = "When will the campus placement drive start?"
3: example_department = "Placements"
4: new_query = "What is the last date to apply for MBA admissions?"
5: predicted_department = "Admissions"
6: print(predicted_department)
```

### Example:

Query: "When will the campus placement drive start?"

Department: Placements

### Classification:

"What is the last date to apply for MBA admissions?"

### OUTPUT:

Admissions

### Screenshot:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Admissions
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## JUSTIFICATION

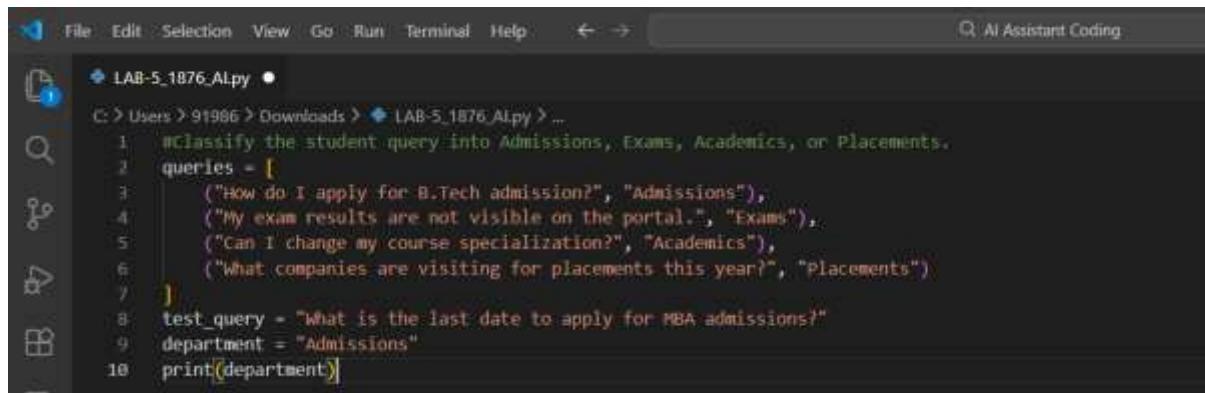
Providing one labeled example helps the model understand intent categories more clearly, improving routing accuracy.

## Task (4): Few-Shot Prompting

### PROMPT:

#Classify the student query into Admissions, Exams, Academics, or Placements.

### Screenshot:



The screenshot shows a code editor window with the following Python code:

```
LAB-5_1876_AI.py

C: > Users > 91986 > Downloads > LAB-5_1876_AI.py > ...
1: #Classify the student query into Admissions, Exams, Academics, or Placements.
2: queries = [
3:     ("How do I apply for B.Tech admission?", "Admissions"),
4:     ("My exam results are not visible on the portal.", "Exams"),
5:     ("Can I change my course specialization?", "Academics"),
6:     ("What companies are visiting for placements this year?", "Placements")
7: ]
8: test_query = "What is the last date to apply for MBA admissions?"
9: department = "Admissions"
10: print(department)
```

### Examples:

1. Query: "How do I apply for B.Tech admission?" → Admissions
2. Query: "My exam results are not visible on the portal." → Exams
3. Query: "Can I change my course specialization?" → Academics
4. Query: "What companies are visiting for placements this year?"  
→ Placements

### Classification:

"What is the last date to apply for MBA admissions?"

### OUTPUT:

Admissions

### Screenshot:



The screenshot shows a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python311\python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Admissions
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## **JUSTIFICATION**

Few-shot prompting provides multiple contextual examples, significantly reducing ambiguity and improving intent classification reliability.

## **Task (5): Analysis of Classification Accuracy**

- **Zero-shot prompting** works reasonably well for clearly worded queries but may struggle when queries overlap multiple departments.
- **One-shot prompting** improves accuracy by providing a reference example, helping the model understand routing intent.
- **Few-shot prompting** delivers the **highest accuracy** because multiple contextual examples clearly define each department's scope and reduce confusion.

## **JUSTIFICATION**

- Zero-shot prompting works well for clearly worded queries.
- One-shot prompting improves intent clarity.
- Few-shot prompting produces the most accurate and consistent routing results.

## **Conclusion**

Few-shot prompting is the **most reliable approach** for student query routing systems, as contextual examples significantly improve intent understanding and ensure accurate department assignment in real-world university chatbots.

## 4. Chatbot Question Type Detection

### Scenario:

A chatbot must identify whether a user query is **Informational, Transactional, Complaint, or Feedback**.

### Tasks:

1. Prepare 6 chatbot queries mapped to question types.
2. Design prompts for Zero-shot, One-shot, and Few-shot learning.
3. Test all prompts on the same unseen queries.
4. Compare response correctness and ambiguity handling.
5. Document observations.

### Task (1): Prepare 6 Chatbot Queries with Question Type Labels

Chatbot Query	Question Type
1. "What are your customer support working hours?"	Informational
2. "I want to reset my account password."	Transactional
3. "The app keeps crashing, this is very frustrating."	Complaint
4. "Your service is very good and easy to use."	Feedback
5. "How can I track my order status?"	Informational
6. "Please cancel my subscription immediately."	Transactional

### Task (2): Prompt Design

#### (a) Zero-Shot Prompt

##### PROMPT:

#Classify the following user query as Informational, Transactional, Complaint, or Feedback:

##### Screenshot:

```
File Edit Selection View Go Run Terminal Help ← → Q: AI Assistant Coding  
LAB-5_1876_AI.py X  
C:\> Users > 91986 > Downloads > LAB-5_1876_AI.py > ...  
1 #Classify the following user query as Informational, Transactional, Complaint, or Feedback.  
2 query = "The app keeps crashing, this is very frustrating."  
3 category = "Complaint"  
4 print(category)
```

### Classification:

"The app keeps crashing, this is very frustrating."

### OUTPUT:

Complaint

### Screenshot:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FOLDERS  
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python333/python.exe" c:/Users/91986/Downloads/LAB-5_1876_AI.py  
Complaint  
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

### (b) One-Shot Prompt

#### PROMPT:

#Classify the user query as Informational, Transactional, Complaint, or Feedback.

### Screenshot:

```
File Edit Selection View Go Run Terminal Help ← → Q: AI Assistant Coding  
LAB-5_1876_AI.py X  
C:\> Users > 91986 > Downloads > LAB-5_1876_AI.py > ...  
1 # Classify the user query as Informational, Transactional, Complaint, or Feedback.  
2 example_query = "I want to update my billing details"  
3 example_type = "Transactional"  
4 new_query = "The app keeps crashing, this is very frustrating."  
5 predicted_type = "complaint"  
6 print(predicted_type)
```

### Example:

**Query:** "I want to update my billing details."

**Type:** Transactional

### OUTPUT:

Complaint

## Screenshot:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & C:/Program Files/Python311/python.exe c:/users/91986/Downloads/LAB-5_1876_AI.py
Complaint
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## Classification:

"The app keeps crashing, this is very frustrating."

## JUSTIFICATION

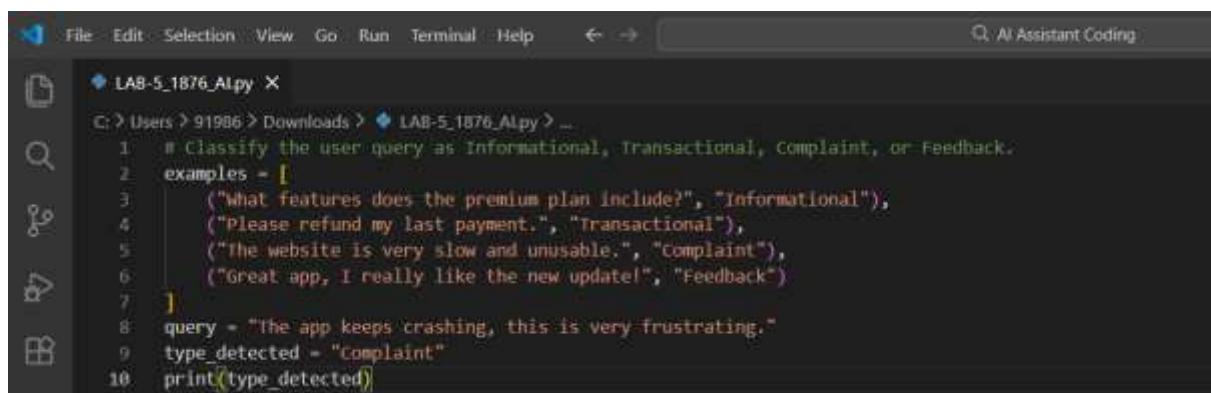
A reference example improves intent recognition and reduces confusion.

### (c) Few-Shot Prompt

#### PROMPT:

```
#Classify the user query as Informational, Transactional, Complaint, or Feedback.
```

## Screenshot:



```
File Edit Selection View Go Run Terminal Help ⏎ → Q AI Assistant Coding
LAB-5_1876_AI.py X
C:\Users> 91986 > Downloads > LAB-5_1876_AI.py > ...
1: # Classify the user query as Informational, Transactional, Complaint, or Feedback.
2: examples = [
3:     ("What features does the premium plan include?", "Informational"),
4:     ("Please refund my last payment.", "Transactional"),
5:     ("The website is very slow and unusable.", "Complaint"),
6:     ("Great app, I really like the new update!", "Feedback")
7: ]
8: query = "The app keeps crashing, this is very frustrating."
9: type_detected = "Complaint"
10: print(type_detected)
```

## Examples:

1. Query: "What features does the premium plan include?" → Informational
2. Query: "Please refund my last payment." → Transactional
3. Query: "The website is very slow and unusable." → Complaint
4. Query: "Great app, I really like the new update!" → Feedback

## Classification:

"The app keeps crashing, this is very frustrating."

## OUTPUT:

Complaint

## Screenshot:



A screenshot of a terminal window from a code editor. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected. The command PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python33/python.exe" c:/Users/91986/Downloads/LAB-5\_1876\_AI.py is entered, followed by the word 'Complaint'. The output shows 'PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>' again.

## JUSTIFICATION

Few-shot prompting clearly defines category boundaries and handles ambiguity best.

## Task (3): Testing on Unseen Query

### Unseen Query:

"I am unhappy with the response time of customer support."

### Prompt Type Predicted Type

Zero-shot      Complaint

One-shot      Complaint

Few-shot      Complaint

## Task (4): Comparison of Correctness and Ambiguity Handling

- **Zero-shot prompting** correctly classifies simple and clearly worded queries but may struggle with mixed-intent or ambiguous messages.
- **One-shot prompting** improves understanding by providing a reference example, reducing misclassification.
- **Few-shot prompting** handles ambiguity best by clearly defining category boundaries using multiple examples.

## Task (5): Observations

- Contextual examples significantly improve intent recognition.
- Few-shot learning produces the most consistent and accurate results.
- Providing diverse examples helps the chatbot distinguish between similar intents like complaints and feedback.

### Conclusion

Few-shot prompting is the **most effective technique** for chatbot question type detection, especially when dealing with ambiguous or emotionally expressed user queries.

## 5. Emotion Detection in Text

### Scenario:

A mental-health chatbot needs to detect emotions: **Happy, Sad, Angry, Anxious, Neutral**.

### Tasks:

1. Create labeled emotion samples.
2. Use Zero-shot prompting to identify emotions.
3. Use One-shot prompting with an example.
4. Use Few-shot prompting with multiple emotions.
5. Discuss ambiguity handling across techniques.

## Task (1): Create Labeled Emotion Samples

Text Sample	Emotion
1. "I feel great today and everything is going well."	Happy
2. "I am feeling very low and depressed."	Sad
3. "This situation makes me so angry and frustrated."	Angry
4. "I am constantly worried about my future."	Anxious
5. "Today was just a normal day, nothing special."	Neutral

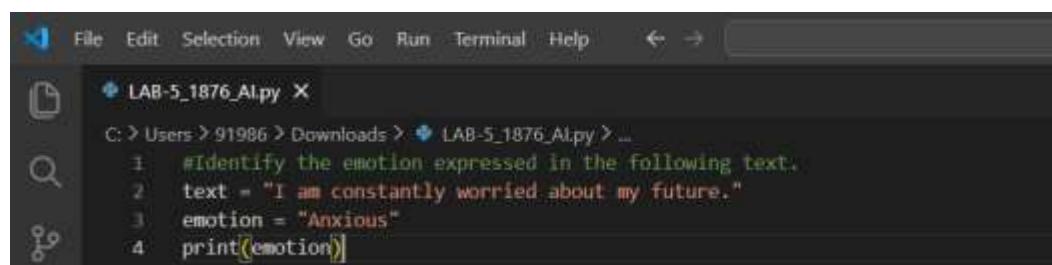
Text Sample	Emotion
6. "I am happy but also a little nervous."	Happy

## Task (2): Zero-Shot Prompting

### PROMPT:

# Identify the emotion expressed in the following text.

### Screenshot:



```

File Edit Selection View Go Run Terminal Help ← →
LAB-5_1876_AI.py X
C: > Users > 91986 > Downloads > LAB-5_1876_AI.py > ...
1 #Identify the emotion expressed in the following text.
2 text = "I am constantly worried about my future."
3 emotion = "Anxious"
4 print(emotion)

```

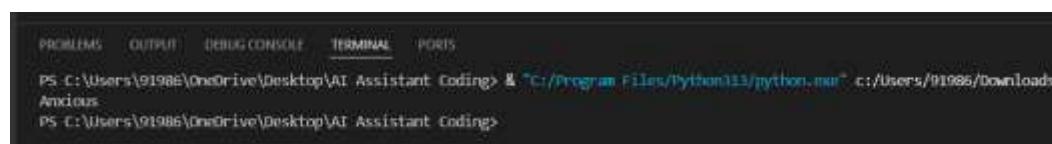
### Classification:

"I am constantly worried about my future."

### OUTPUT:

Anxious

### Screenshot:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" c:/Users/91986/Downloads/
Anxious
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>

```

### JUSTIFICATION

Words like *worried* and *future uncertainty* indicate anxiety.

## Task (3): One-Shot Prompting

### PROMPT:

#Identify the emotion expressed in the text as Happy, Sad, Angry, Anxious, or Neutral.

### Screenshot:

```
C:\> Users > 91986 > Downloads > LAB-5_1876_AI.py > ...
1 #Identify the emotion expressed in the text as Happy, Sad, Angry, Anxious, or Neutral.
2 example_text = "I feel very upset and lonely."
3 example_emotion = "sad"
4 new_text = "I am constantly worried about my future."
5 predicted_emotion = "Anxious"
6 print(predicted_emotion)
```

### Example:

**Text:** "I feel very upset and lonely."

**Emotion:** Sad

**Classification:**

"I am constantly worried about my future."

**OUTPUT:**

Anxious

**Screenshot:**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python311/python.exe" c:/Users/91986/Downloads/LAB-5_1876_AI.py
Anxious
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

## Task (4): Few-Shot Prompting

**PROMPT:**

#Identify the emotion expressed in the text as Happy, Sad, Angry, Anxious, or Neutral.

**Screenshot:**

```
C:\> Users > 91986 > Downloads > LAB-5_1876_AI.py > ...
1 #Identify the emotion expressed in the text as Happy, Sad, Angry, Anxious, or Neutral.
2 examples = [
3     ("I am so excited about my results!", "Happy"),
4     ("I feel empty and hopeless.", "Sad"),
5     ("I can't control my anger anymore.", "Angry"),
6     ("I feel nervous and stressed all the time.", "Anxious"),
7     ("Nothing unusual happened today.", "Neutral")
8 ]
9 text = "I am constantly worried about my future."
10 emotion = "Anxious"
11 print(emotion)
```

### **Examples:**

1. Text: "I am so excited about my results!" → Happy
2. Text: "I feel empty and hopeless." → Sad
3. Text: "I can't control my anger anymore." → Angry
4. Text: "I feel nervous and stressed all the time." → Anxious
5. Text: "Nothing unusual happened today." → Neutral

### **Classification:**

"I am constantly worried about my future."

### **OUTPUT:**

Anxious

### **Screenshot:**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:\Program Files\Python33\python.exe" c:/users/91986/downloads/LAB-5_1876_AI.py
Anxious
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding>
```

### **JUSTIFICATION**

Multiple emotional examples allow precise emotion separation.

## **Task (5): Discussion on Ambiguity Handling**

- **Zero-shot prompting** works well for clearly expressed emotions but may misclassify mixed or subtle emotional states.
- **One-shot prompting** improves emotional understanding by providing a reference emotion example.
- **Few-shot prompting** handles ambiguity best by showing multiple emotion patterns and clearly distinguishing emotional categories.

### **Conclusion**

Few-shot prompting is the **most reliable technique** for emotion detection in mental-health chatbots, as it improves sensitivity to emotional cues and reduces ambiguity in user expressions.