

| SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE   |   | DEPARTMENT OF COMPUTER SCIENCE ENGINEERING   |                    |
|--|---|--|--------------------|
| <b>Program Name:</b> B. Tech   |   | <b>Assignment Type:</b> Lab  |                    |
| <b>Course Coordinator Name</b>   |   | Venkataramana Veeramsetty  |                    |
| <b>Instructor(s)Name</b>   |   | 1. Dr. Mohammed Ali Shaik<br>2. Dr. T Sampath Kumar<br>3. Mr. S Naresh Kumar<br>4. Dr. V. Rajesh<br>5. Dr. Brij Kishore<br>6. Dr Pramoda Patro<br>7. Dr. Venkataramana<br>8. Dr. Ravi Chander<br>9. Dr. Jagjeeth Singh |                    |
| <b>Course Code</b>   | 24CS002PC215  | <b>Course Title</b>  | AI Assisted Coding |
| <b>Year/Sem</b>  | II/I  | <b>Regulation</b>  | R24                |
| <b>Date and Day of Assignment</b>  | 06-08-2025  | <b>Time(s)</b>   |                    |
| <b>Duration</b>  | 2 Hours   | <b>Applicable to Batches</b>   |                    |
| <b>AssignmentNumber:</b> 4.5(Present assignment number)/ <b>24</b> (Total number of assignments) |   |  |                    |
|  |   |  |                    |
| <b>Q. No.</b>  | <b>Question</b>   | <b>Expected Time to complete</b>   |                    |
| 1  | <p><b>Lab 4: Advanced Prompt Engineering: Zero-shot, one-shot, and few-shot techniques</b></p> <p><b>Objective:</b> To explore and compare Zero-shot, One-shot, and Few-shot prompting techniques for classifying emails into predefined categories using a large language model (LLM).</p> <p>Suppose that you work for a company that receives hundreds of customer emails daily. Management wants to automatically classify emails into categories like "Billing", "Technical Support", "Feedback", and "Others" before assigning them to appropriate departments. Instead of training a new model, your task is to use prompt engineering techniques with an existing LLM to handle the classification.</p> <p>Tasks to be completed are as below</p> <p><b>1. Prepare Sample Data:</b></p> <ul style="list-style-type: none"> <li>• Create or collect 10 short email samples, each belonging to one of the 4 categories.</li> </ul> <p><b>SAMPLE EMAILS:</b></p> <ol style="list-style-type: none"> <li>1. Requesting a meeting: "Could we schedule a meeting next week to discuss the project details?"</li> <li>2. Following up: "Just following up to see if you had a chance to review my previous email."</li> <li>3. Thanking: "Thank you for your assistance with the recent issue; it was greatly appreciated."</li> </ol> | 08.08.2025 EOD   |                    |

4. Confirming appointment: "This is to confirm our appointment scheduled for Tuesday at 3 PM."
5. Providing information: "Please find attached the requested report for your review."
6. Apologizing for delay: "Apologies for the delay in response; I am looking into your request now."
7. Requesting feedback: "Could you please provide feedback on the proposal by Friday?"
8. Introducing yourself: "I'm reaching out to introduce myself as your new account manager."
9. Announcing update: "We have updated our privacy policy; please review the changes at your convenience."
10. Closing a deal: "We're excited to move forward; please sign the attached contract to get started."

## 2. Zero-shot Prompting:

- Design a prompt that asks the LLM to classify a single email without providing any examples.
- Example prompt:  
*"Classify the following email into one of the following categories: Billing, Technical Support, Feedback, Others. Email: I have not received my invoice for last month."*

## PROMPT :

Give similar output expected code to classify among billing,technical support,feedback,others.

```

❸ import renn.py > ...
1  import re
2
3  def classify_email(message):
4      billing_keywords = ['invoice', 'payment', 'charge', 'refund', 'billing', 'credit card', 'receipt']
5      tech_keywords = ['error', 'bug', 'issue', 'problem', 'crash', 'technical', 'support', 'not working', 'fail']
6      feedback_keywords = ['suggestion', 'feedback', 'recommend', 'like', 'dislike', 'improve', 'feature request']
7      msg = message.lower()
8
9      if any(word in msg for word in billing_keywords):
10         return "Billing"
11     elif any(word in msg for word in tech_keywords):
12         return "Technical Support"
13     elif any(word in msg for word in feedback_keywords):
14         return "Feedback"
15     else:
16         return "Others"
17
18 if __name__ == "__main__":
19     print("Type your email message:")
20     user_message = ""
21     while True:
22         try:
23             line = input()
24             if line.strip() == "":
25                 break
26             user_message += line + " "
27         except EOFError:
28             break
29
30     email_type = classify_email(user_message)
31     print(f"\nThis email is classified as: {email_type}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
 This email is classified as: Technical Support  
 PS D:\AI> & C:/Users/H/AppData/Local/Programs/Python/Python313/python.exe "d:/AI/import\_renn .py"  
 Type your email message:

I really appreciate the prompt support I received-thank you!

This email is classified as: Technical Support  
 PS D:\AI> []

### 3. One-shot Prompting:

- Add one labeled example before asking the model to classify a new email.

## Prompt:

Here is an example:

Type your email message: "The app crashes every time I try to open it—can you assist?"

Email classified as: Technical Support

Give similar output expected code to classify among billing,technical support,feedback,others.

```
email v.py > ...
1  def classify_email(message):
2      message_lower = message.lower()
3      billing_keywords = ['charged', 'billing', 'invoice', 'payment', 'refund', 'subscription', 'price']
4      tech_keywords = ['error', 'issue', 'problem', 'bug', 'not working', 'crash', 'technical', 'login', 'reset']
5      feedback_keywords = ['feedback', 'suggestion', 'recommend', 'like', 'dislike', 'improve', 'love', 'hate']
6
7      if any(word in message_lower for word in billing_keywords):
8          return 'Billing'
9      elif any(word in message_lower for word in tech_keywords):
10         return 'Technical Support'
11     elif any(word in message_lower for word in feedback_keywords):
12         return 'Feedback'
13     else:
14         return 'Others'
15
16 # Example usage
17 email_message = "I was charged twice for my subscription. Please fix this."
18 classification = classify_email(email_message)
19 print(f'Email: "{email_message}"')
20 print(f'Classified as: {classification}')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
This email is classified as: Technical Support
PS D:\AI> & C:/Users/HI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI/email .v.py"
Email classified as: Billing
PS D:\AI> & C:/Users/HI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI/email .v.py"
Email: "I was charged twice for my subscription. Please fix this."
Classified as: Billing
PS D:\AI> []
```

Q Ln 20, Col 42 Spac

### 4. Few-shot Prompting:

- Use 3–5 labeled examples in your prompt before asking the model to classify a new email.

## PROMPT:

Type your email message:

The program will read your message and tell you what kind it is:

Billing, Technical Support, Feedback, or Others.

**Example:**

Type your email message: "The app crashes every time I try to open it—can you assist?"

Email classified as: Technical Support.

2. "Could you confirm receipt of my job application submitted last week?

Email classified as: others.

3. "I was charged twice for my subscription. Please fix this."

Email classified as: Billing

Give me a code such that based on above examples output is based on classification of billing,technical support,feedback,others.

```
email.v.py > ...
1  def classify_email(message):
2      message_lower = message.lower()
3      billing_keywords = ['charged', 'billing', 'invoice', 'payment', 'refund', 'subscription', 'price']
4      tech_keywords = ['error', 'issue', 'problem', 'bug', 'not working', 'crash', 'technical', 'login']
5      feedback_keywords = ['feedback', 'suggestion', 'recommend', 'like', 'dislike', 'improve', 'love',
6
7          if any(word in message_lower for word in billing_keywords):
8              return 'Billing'
9          elif any(word in message_lower for word in tech_keywords):
10             return 'Technical Support'
11         elif any(word in message_lower for word in feedback_keywords):
12             return 'Feedback'
13         else:
14             return 'Others'
15
16 # Example usage
17 email_message = "I was charged twice for my subscription. Please fix this."
18 classification = classify_email(email_message)
19 print(f'Email: "{email_message}"')
20 print(f'Classified as: {classification}')


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

This email is classified as: Technical Support
PS D:\AI> & C:/Users/Hi/AppData/Local/Programs/Python/Python313/python.exe "d:/AI/email .v.py"
Email classified as: Billing
PS D:\AI> & C:/Users/Hi/AppData/Local/Programs/Python/Python313/python.exe "d:/AI/email .v.py"
Email: "I was charged twice for my subscription. Please fix this."
Classification as: Billing
PS D:\AI> []
\△ 0
```

##### 5. Evaluation:

- Run all three techniques on the same set of 5 test emails.
- Compare and document the accuracy and clarity of responses.

RESPONSE:

- Accuracy depends on up-to-date, relevant data for each item being compared.
- Clear, objective criteria enhance trustworthiness; subjective or incomplete information reduces accuracy.
- Tables that include detailed performance metrics or verified user ratings tend to offer higher accuracy.
- Interactive tables that allow filtering or sorting can improve user experience but do not inherently increase accuracy of data itself.

|  |   |  |
|--|---|--|
|  | <p><b>SUMMARY:</b></p> <p>Comparison tables are effective tools for simplifying decision-making by presenting key attributes of products or services side by side. To ensure accuracy and usefulness, comparison tables should contain consistent, complete, and up-to-date content, focusing on relevant features and limiting the number of items (ideally up to five) for clarity. They should be designed for easy scanning, using concise text, clear labels, visual cues like color coding, and a simple layout. Interactivity such as filtering or collapsible sections can enhance usability. Keeping the table unbiased and regularly updated maintains trust and accuracy. Overall, the best comparison tables are concise, clear, and user-friendly, enabling quick and informed choices without overwhelming the user.</p> <p><b>Requirements:</b></p> <ul style="list-style-type: none"> <li>• VS Code with Github Copilot or Cursor IDE and/or Google Colab with Gemini</li> </ul> <p><b>Deliverables:</b></p> <ul style="list-style-type: none"> <li>• A .txt or .md file showing prompts and model responses.</li> <li>• A comparison table showing classification accuracy for each technique.</li> <li>• A short reflection on which method was most effective and why</li> </ul> |  |
|--|---|--|