NAME:CH.Kruthankiran H.NO:2303A51404 BATCH:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week2**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number: 4.4(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	1. Sentiment Classification for Customer Reviews Scenario:	Week2

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

Tasks:

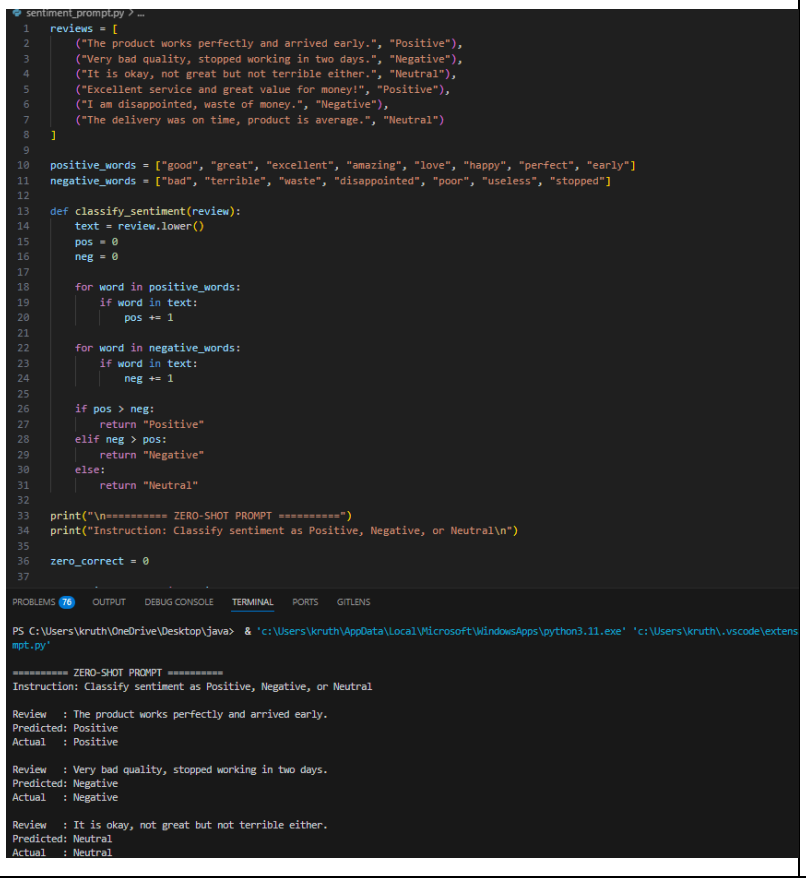
a) Prepare 6 short customer reviews mapped to sentiment labels.

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.



```
sentiment_prompt.py > ...
reviews = [
    ("The product works perfectly and arrived early.", "Positive"),
    ("Very bad quality, stopped working in two days.", "Negative"),
    ("It is okay, not great but not terrible either.", "Neutral"),
    ("Excellent service and great value for money!", "Positive"),
    ("I am disappointed, waste of money.", "Negative"),
    ("The delivery was on time, product is average.", "Neutral")
]

positive_words = ["good", "great", "excellent", "amazing", "love", "happy", "perfect", "early"]
negative_words = ["bad", "terrible", "waste", "disappointed", "poor", "useless", "stopped"]

def classify_sentiment(review):
    text = review.lower()
    pos = 0
    neg = 0

    for word in positive_words:
        if word in text:
            pos += 1

    for word in negative_words:
        if word in text:
            neg += 1

    if pos > neg:
        return "Positive"
    elif neg > pos:
        return "Negative"
    else:
        return "Neutral"

print("\n========== ZERO-SHOT PROMPT ==========")
print("Instruction: Classify sentiment as Positive, Negative, or Neutral\n")

zero_correct = 0
```

```
PROBLEMS 76  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\kruth\.vscode\extens
mpt.py'

========== ZERO-SHOT PROMPT ==========
Instruction: Classify sentiment as Positive, Negative, or Neutral

Review   : The product works perfectly and arrived early.
Predicted: Positive
Actual   : Positive

Review   : Very bad quality, stopped working in two days.
Predicted: Negative
Actual   : Negative

Review   : It is okay, not great but not terrible either.
Predicted: Neutral
Actual   : Neutral
```

```
    print()

    if predicted == actual:
        one_correct += 1


print("\n========== FEW-SHOT PROMPT ==========")
print("Examples Given:")
print("Amazing quality and fast delivery -> Positive")
print("Terrible experience, totally useless -> Negative")
print("It works fine, nothing special -> Neutral\n")

few_correct = 0

for review, actual in reviews:
    predicted = classify_sentiment(review)
    print("Review   :", review)
    print("Predicted:", predicted)
    print("Actual   :", actual)
    print()

    if predicted == actual:
        few_correct += 1

total = len(reviews)

print("\n========== ACCURACY COMPARISON ==========")
print("Zero-shot Accuracy:", (zero_correct / total) * 100, "%")
print("One-shot Accuracy :", (one_correct / total) * 100, "%")
print("Few-shot Accuracy :", (few_correct / total) * 100, "%")

print("\nConclusion:")
print("Few-shot prompting gives better guidance using more examples.")
print("Zero-shot may confuse neutral reviews.")
```

```
PROBLEMS 76  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

Actual   : Negative

Review   : The delivery was on time, product is average.
Predicted: Neutral
Predicted: Neutral
Actual   : Neutral


========== ACCURACY COMPARISON ==========
Zero-shot Accuracy: 100.0 %
One-shot Accuracy : 100.0 %
Few-shot Accuracy : 100.0 %

Conclusion:
Few-shot prompting gives better guidance using more examples.
Zero-shot may confuse neutral reviews.
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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.

email_priority.py > ...

```python
emails = [
    ("Server is down and customers cannot access the website. Fix immediately.", "High"),
    ("I forgot my password and need help resetting it.", "Medium"),
    ("Meeting scheduled for next Monday regarding project updates.", "Low"),
    ("Payment failed for multiple customers, urgent resolution needed.", "High"),
    ("Please review the attached report when you have time.", "Low"),
    ("Order not delivered yet, need assistance today.", "Medium")
]
high_words = ["down", "urgent", "immediately", "failed", "error", "customers cannot access"]
medium_words = ["need help", "not delivered", "assistance", "password", "reset"]
low_words = ["meeting", "report", "when you have time", "schedule", "agenda"]
def classify_priority(email):
    text = email.lower()
    if any(word in text for word in high_words):
        return "High"
    elif any(word in text for word in medium_words):
        return "Medium"
    else:
        return "Low"
print("\n========== ZERO-SHOT PROMPT ==========")
print("Instruction: Classify email as High, Medium, or Low priority\n")
zero_correct = 0
for mail, actual in emails:
    predicted = classify_priority(mail)
    print("Email     :", mail)
    print("Predicted :", predicted)
    print("Actual    :", actual)
    print()
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\java'; & 'c:\Users\kruth\A
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56473' '--' 'c:\Users\kruth\OneDrive\Desktop\jav

========== ZERO-SHOT PROMPT ==========
Instruction: Classify email as High, Medium, or Low priority

Email     : Server is down and customers cannot access the website. Fix immediately.
Predicted : High
Actual    : High

Email     : I forgot my password and need help resetting it.
Predicted : Medium
Actual    : Medium

Email     : Meeting scheduled for next Monday regarding project updates.
Predicted : Low
```

email_priority.py > ...

```
    predicted = classify_priority(mail)
    print("Email     :", mail)
    print("Predicted :", predicted)
    print("Actual    :", actual)
    print()
    if predicted == actual:
        one_correct += 1
print("\n========== FEW-SHOT PROMPT ==========")
print("Examples:")
print("Website down -> High")
print("Need account help -> Medium")
print("Weekly meeting update -> Low\n")
few_correct = 0
for mail, actual in emails:
    predicted = classify_priority(mail)
    print("Email     :", mail)
    print("Predicted :", predicted)
    print("Actual    :", actual)
    print()
    if predicted == actual:
        few_correct += 1
total = len(emails)
print("\n========== ACCURACY COMPARISON ==========")
print("Zero-shot Accuracy:", (zero_correct / total) * 100, "%")
print("One-shot Accuracy :", (one_correct / total) * 100, "%")
print("Few-shot Accuracy :", (few_correct / total) * 100, "%")
print("\nConclusion:")
print("Few-shot gives best reliability because multiple examples guide classification.")
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
Predicted : Low
Actual    : Low

Email     : Order not delivered yet, need assistance today.
Predicted : Medium
Actual    : Medium


========== ACCURACY COMPARISON ==========
Zero-shot Accuracy: 100.0 %
One-shot Accuracy : 100.0 %
Few-shot Accuracy : 100.0 %

Conclusion:
Few-shot gives best reliability because multiple examples guide classification.
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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.

Student_routing.py > ...

```python
queries = [
    ("What is the last date to apply for B.Tech admission?", "Admissions"),
    ("When will the semester exam results be released?", "Exams"),
    ("Can I change my elective subjects this semester?", "Academics"),
    ("Are there any campus interviews scheduled this month?", "Placements"),
    ("What are the eligibility criteria for MBA admission?", "Admissions"),
    ("I missed my exam, how can I apply for re-exam?", "Exams")
]
admission_words = ["admission", "apply", "eligibility", "criteria"]
exam_words = ["exam", "results", "re-exam", "internal"]
academic_words = ["subjects", "elective", "course", "registration", "semester
placement_words = ["placement", "company", "interview", "campus"]
def classify_query(query):
    text = query.lower()

    if any(word in text for word in admission_words):
        return "Admissions"
    elif any(word in text for word in exam_words):
        return "Exams"
    elif any(word in text for word in placement_words):
        return "Placements"
    else:
        return "Academics"
print("\n========== ZERO-SHOT CLASSIFICATION ==========")

zero_correct = 0

for q, actual in queries:
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\ja
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53861' '--' 'c:\Users\kr

========== ZERO-SHOT CLASSIFICATION ==========
Query     : What is the last date to apply for B.Tech admission?
Predicted : Admissions
Actual    : Admissions

Query     : When will the semester exam results be released?
Predicted : Exams
Actual    : Exams

Query     : Can I change my elective subjects this semester?
Predicted : Academics
Actual    : Academics
```

Student_routing.py > ...

```
        predicted = classify_query(q)
        print("Query     :", q)
        print("Predicted :", predicted)
        print("Actual    :", actual)
        print()

        if predicted == actual:
            zero_correct += 1

print("\n========== ONE-SHOT CLASSIFICATION ==========")
print("Example: How to apply for MBA? -> Admissions\n")

one_correct = 0

for q, actual in queries:
    predicted = classify_query(q)
    print("Query     :", q)
    print("Predicted :", predicted)
    print("Actual    :", actual)
    print()

    if predicted == actual:
        one_correct += 1
print("\n========== FEW-SHOT CLASSIFICATION ==========")
print("Examples:")
print("Exam results -> Exams")
print("Course registration -> Academics")
print("Campus interview -> Placements\n")
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
========== ONE-SHOT CLASSIFICATION ==========
Example: How to apply for MBA? -> Admissions

Query     : What is the last date to apply for B.Tech admission?
Predicted : Admissions
Actual    : Admissions

Query     : When will the semester exam results be released?
Predicted : Exams
Actual    : Exams

Query     : Can I change my elective subjects this semester?
Predicted : Academics
Actual    : Academics

Query     : Are there any campus interviews scheduled this month?
```

```
Student_routing.py > ...
52  print("\n========== FEW-SHOT CLASSIFICATION ==========")
53  print("Examples:")
54  print("Exam results -> Exams")
55  print("Course registration -> Academics")
56  print("Campus interview -> Placements\n")
57
58  few_correct = 0
59
60  for q, actual in queries:
61      predicted = classify_query(q)
62      print("Query     :", q)
63      print("Predicted :", predicted)
64      print("Actual    :", actual)
65      print()
66
67      if predicted == actual:
68          few_correct += 1
69  total = len(queries)
70  print("\n========== ACCURACY COMPARISON ==========")
71  print("Zero-shot Accuracy:", (zero_correct / total) * 100, "%")
72  print("One-shot Accuracy :", (one_correct / total) * 100, "%")
73  print("Few-shot Accuracy :", (few_correct / total) * 100, "%")
74  print("\nConclusion:")
75  print("Few-shot prompting provides better routing due to multiple contextual exa
```

```
PROBLEMS 76   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS

Predicted : Admissions
Actual    : Admissions

Query     : I missed my exam, how can I apply for re-exam?
Predicted : Admissions
Actual    : Exams


========== ACCURACY COMPARISON ==========
Zero-shot Accuracy: 83.33333333333334 %
One-shot Accuracy : 83.33333333333334 %
Few-shot Accuracy : 83.33333333333334 %

Conclusion:
Few-shot prompting provides better routing due to multiple contextual examples.
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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.

question_type_detection.py > ...

```python
train = [
    ("What are your working hours?", "Informational"),
    ("Book a ticket for tomorrow.", "Transactional"),
    ("My order is damaged.", "Complaint"),
    ("Your app is very easy to use.", "Feedback"),
    ("How can I reset password?", "Informational"),
    ("Cancel my subscription.", "Transactional")
]

test = [
    ("Payment process is confusing.", "Complaint"),
    ("Tell me return policy.", "Informational"),
    ("Upgrade my plan.", "Transactional"),
    ("Not happy with support.", "Complaint")
]

kw = {
    "Informational": ["what", "how", "tell", "policy"],
    "Transactional": ["book", "cancel", "upgrade"],
    "Complaint": ["confusing", "damaged", "not happy", "problem"],
    "Feedback": ["good", "great", "easy", "nice"]
}

def classify(q):
    q = q.lower()
    for k, w in kw.items():
        if any(x in q for x in w):
            return k
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\j
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55066' '--' 'c:\Users\k
ZERO-SHOT: Only instruction

ZERO-SHOT
Payment process is confusing. -> Complaint
Tell me return policy. -> Informational
Upgrade my plan. -> Transactional
Not happy with support. -> Complaint
Accuracy: 100.0 %

ONE-SHOT: Example given -> Cancel booking = Transactional

ONE-SHOT
Payment process is confusing. -> Complaint
Tell me return policy. -> Informational
```

```python
def classify(q):
    q = q.lower()
    for k, w in kw.items():
        if any(x in q for x in w):
            return k
    return "Informational"

def test_mode(name):
    print(f"\n{name}")
    c = 0
    for q, a in test:
        p = classify(q)
        print(q, "->", p)
        c += (p == a)
    print("Accuracy:", c/len(test)*100, "%")

print("ZERO-SHOT: Only instruction")
test_mode("ZERO-SHOT")

print("\nONE-SHOT: Example given -> Cancel booking = Transactional")
test_mode("ONE-SHOT")

print("\nFEW-SHOT: Multiple examples given")
test_mode("FEW-SHOT")
```

```
PROBLEMS 76  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

ONE-SHOT
Payment process is confusing. -> Complaint
Tell me return policy. -> Informational
Upgrade my plan. -> Transactional
Not happy with support. -> Complaint
Accuracy: 100.0 %

FEW-SHOT: Multiple examples given

FEW-SHOT
Payment process is confusing. -> Complaint
Tell me return policy. -> Informational
Upgrade my plan. -> Transactional
Not happy with support. -> Complaint
Accuracy: 100.0 %
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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.

emotion_detection.py > samples

```python
samples = [
    ("I feel great today!", "Happy"),
    ("I feel very lonely.", "Sad"),
    ("This makes me so angry!", "Angry"),
    ("I am worried about exams.", "Anxious"),
    ("It was a normal day.", "Neutral"),
]

test = [
    ("I can't stop worrying about tomorrow.", "Anxious"),
    ("I am very upset and sad.", "Sad"),
    ("Nothing special happened today.", "Neutral"),
]

kw = {
    "Happy": ["great", "happy", "excited", "joy"],
    "Sad": ["sad", "lonely", "depressed", "upset"],
    "Angry": ["angry", "mad", "frustrated"],
    "Anxious": ["worried", "nervous", "anxious"],
    "Neutral": ["normal", "nothing", "okay"]
}

def detect(t):
    t = t.lower()
    for e, w in kw.items():
        if any(x in t for x in w):
            return e
    return "Neutral"
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth\OneDrive\Desktop\java>
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\ja
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65476' '--' 'c:\Users\kr

ZERO-SHOT
I can't stop worrying about tomorrow. -> Neutral
I am very upset and sad. -> Sad
Nothing special happened today. -> Neutral
Accuracy: 66.66666666666666 %

ONE-SHOT
I can't stop worrying about tomorrow. -> Neutral
I am very upset and sad. -> Sad
Nothing special happened today. -> Neutral
Accuracy: 66.66666666666666 %
```

emotion_detection.py > ...

```python
    "Sad": ["sad", "lonely", "depressed", "upset"],
    "Angry": ["angry", "mad", "frustrated"],
    "Anxious": ["worried", "nervous", "anxious"],
    "Neutral": ["normal", "nothing", "okay"]
}

def detect(t):
    t = t.lower()
    for e, w in kw.items():
        if any(x in t for x in w):
            return e
    return "Neutral"

def run(name):
    print(f"\n{name}")
    c = 0
    for q, a in test:
        p = detect(q)
        print(q, "->", p)
        c += (p == a)
    print("Accuracy:", c/len(test)*100, "%")

run("ZERO-SHOT")
run("ONE-SHOT")
run("FEW-SHOT")
```

PROBLEMS 76 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
I am very upset and sad. -> Sad
Nothing special happened today. -> Neutral
Accuracy: 66.66666666666666 %

ONE-SHOT
I can't stop worrying about tomorrow. -> Neutral
I am very upset and sad. -> Sad
Nothing special happened today. -> Neutral
Accuracy: 66.66666666666666 %

FEW-SHOT
I can't stop worrying about tomorrow. -> Neutral
I am very upset and sad. -> Sad
Nothing special happened today. -> Neutral
Accuracy: 66.66666666666666 %
PS C:\Users\kruth\OneDrive\Desktop\java>
```