SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**AcademicYear:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s)Name	1. Dr. Mohammed Ali Shaik 2. Dr. T Sampath Kumar 3. Mr. S Naresh Kumar 4. Dr. V. Rajesh 5. Dr. Brij Kishore 6. Dr Pramoda Patro 7. Dr. Venkataramana 8. Dr. Ravi Chander 9. Dr. Jagjeeth Singh		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	06-08-2025	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:4.5(Present assignment number)/**24**(Total number of assignments)			

Q. No.	Question	*ExpectedTime to complete*
1	**Lab 4: Advanced Prompt Engineering: Zero-shot, one-shot, and few-shot techniques** **Objective:** 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). 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. Tasks to be completed are as below 1. **Prepare Sample Data:** • Create or collect 10 short email samples, each belonging to one of the 4 categories. 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*	08.08.2025 EOD

for last month.’”

3. **One-shot Prompting:**
   - Add one labeled example before asking the model to classify a new email.
4. **Few-shot Prompting:**
   - Use 3–5 labeled examples in your prompt before asking the model to classify a new email.
5. **Evaluation:**
   - Run all three techniques on the same set of 5 test emails.
   - Compare and document the accuracy and clarity of responses.

PROMPT:

Write Python code that uses an LLM to classify customer emails into 4 categories: Billing, Technical Support, Feedback, and Others. Implement three techniques: 1) Zero-shot (no examples), 2) One-shot (1 labeled example), 3) Few-shot (4 labeled examples).

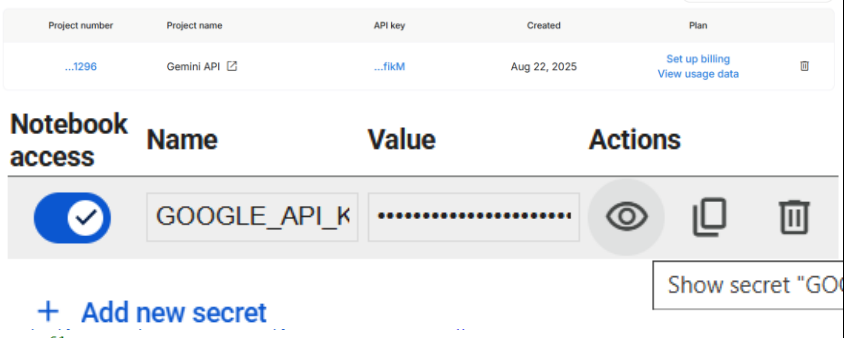
Code:

```
# Define the categories
categories = ["Billing", "Technical Support", "Feedback", "Others"]

# Define test emails
test_emails = [
    "I have a question about my last bill.",
    "My internet connection is not working.",
    "I love your service, it's amazing!",
    "Can I get a refund for my subscription?",
    "What are your business hours?"
]

# Define prompts for each technique

# Zero-shot prompt
zero_shot_prompt = """Classify the following email into one of these categories: {}.

Email: {}
Category:""".format(", ".join(categories), "{}")

# One-shot prompt
one_shot_prompt = """Classify the following email into one of these categories: {}.

Email: My account is locked.
Category: Technical Support

Email: {}
Category:""".format(", ".join(categories), "{}")
```

```
# Few-shot prompt
few_shot_prompt = """Classify the following email into one of these categories: {}.

Email: I can't log in to my account.
Category: Technical Support

Email: I want to change my payment method.
Category: Billing

Email: Your new feature is great!
Category: Feedback

Email: Where is your office located?
Category: Others

Email: {}
Category:""".format(", ".join(categories), "{}")

# Function to classify email using LLM
def classify_email(prompt):
    response = gemini_model.generate_content(prompt)
    return response.text.strip()

# Classify test emails using each technique
results = {
    "Email": test_emails,
    "Zero-shot": [],
    "One-shot": [],
    "Few-shot": []
}
```

```
for email in test_emails:
    results["Zero-shot"].append(classify_email(zero_shot_prompt.format(email)))
    results["One-shot"].append(classify_email(one_shot_prompt.format(email)))
    results["Few-shot"].append(classify_email(few_shot_prompt.format(email)))

# Create and display the comparison table
results_df = pd.DataFrame(results)
display(results_df)
```

OUTPUT:




```
     61
     62 for email in test_emails:
---> 63     results["Zero-shot"].append(classify_email(zero_shot_prompt.format(email)))
     64     results["One-shot"].append(classify_email(one_shot_prompt.format(email)))
     65     results["Few-shot"].append(classify_email(few_shot_prompt.format(email)))
```

```
    292                 self._initial, self._maximum, multiplier=self._multiplier
    293             )
--> 294             return retry_target(
    295                 target,
    296                 self._predicate,

/usr/local/lib/python3.12/dist-packages/google/api_core/retry/retry_unary.py in retry_target(target, predicate,
sleep_generator, timeout, on_error, exception_factory, **kwargs)
    154         except Exception as exc:
    155             # defer to shared logic for handling errors
--> 156             next_sleep = _retry_error_helper(
    157                 exc,
    158                 deadline,

/usr/local/lib/python3.12/dist-packages/google/api_core/retry/retry_base.py in _retry_error_helper(exc, deadline,
sleep_iterator, error_list, predicate_fn, on_error_fn, exc_factory_fn, original_timeout)
    212             original_timeout,
    213         )
--> 214         raise final_exc from source_exc
    215     if on_error_fn is not None:
    216         on_error_fn(exc)

/usr/local/lib/python3.12/dist-packages/google/api_core/retry/retry_unary.py in retry_target(target, predicate,
sleep_generator, timeout, on_error, exception_factory, **kwargs)
    145     while True:
    146         try:
--> 147             result = target()
    148             if inspect.isawaitable(result):
    149                 warnings.warn(_ASYNC_RETRY_WARNING)

/usr/local/lib/python3.12/dist-packages/google/api_core/timeout.py in func_with_timeout(*args, **kwargs)
```

Requirements:

- VS Code with Github Copilot or Cursor IDE and/or Google Colab with Gemini

Deliverables:

- A .txt or .md file showing prompts and model responses.
- A comparison table showing classification accuracy for each technique.
- A short reflection on which method was most effective and why

.