PHASE - 2

Tasks:

- Task 1: Code Summarization (Information Synthesis)
- Task 2: Technical FAQ Answering (Question-Answering)
- Task 3: API Documentation Creation (Content Generation)

Prompting Techniques:

- CLEAR prompt
- Few-shot prompt
- Chain-of-Thought (CoT) prompt

Task-1: Code Summarization (Information Synthesis)

CLEAR Prompt
Few-shot prompt
Chain-of-Thought (CoT) prompt

Task 2: Technical FAQ Answering (Question-Answering)

CLEAR Prompt
Few-shot prompt
Chain-of-Thought (CoT) prompt

Task 3: API Documentation Creation (Content Generation)

CLEAR Prompt
Few-shot prompt
Chain-of-Thought (CoT) prompt

Prompt Portfolio
Task: 1 Code Summarization (Information Synthesis)

CLEAR Prompt

You are a senior software engineer tasked with reviewing a code snippet and providing a clear, concise summary for other developers.
Context:
You will be given:
1. The code snippet
2. A little description about the project

Instructions:
- Summarize the code's purpose in 2–3 sentences.
- List the key functions or classes and their roles.
- Describe the inputs and outputs clearly.
- Mention any dependencies or external libraries used.
- Use simple, professional language appropriate for internal documentation.
- Limit the response to 500 words

Code:
{Code}

Project Description:
{Description}

Rationale:
The CLEAR structure ensures that the context is well defined, the role is explicit, and there is a step-by-step breakdown of expectations and the results.

Expected Output:
1. 2-3 sentences in summary of the program.
2. Inputs and outputs stated.
3. Listing of key functions.
4. Professional language for internal documentation.

Failure Models and Mitigation Strategies:
1. Hallucination: If the code is too long, it would start hallucinating and starts to invent things that aren’t even there. To mitigate, we can explicitly say it to describe what is in the code.
2. Overly verbose: Having the word limit would be a good option as we’d want to have to-the-point answers, instead of explanations that are not needed.

Few-shot Prompt

You are tasked with summarizing code snippets for documentation.
Here are two examples:
Example 1:
Code:

```
def add(a, b):
    return a + b
```

Summary:
This function `add` takes two numbers and returns their sum. Inputs: a, b (numbers). Output: number (sum).

Example 2:
Code:

```
class Order:
    def __init__(self, id, amount):
        self.id = id
        self.amount = amount
```

Summary:
The `Order` class represents a purchase order, including its ID and amount. Constructor initializes these attributes. Inputs: id, amount. Output: instantiated Order object.

Now, summarize the following code using the same style:

Code:
{Code}

Rationale:
Few-shot learning anchors the style and prevents hallucinations by showing ideal examples.

Expected Output:

1. Matches the example style of output.
2. Structured, clear, and Minimal.

Failure Models:

1. Style Drift: Keep examples consistent with the final desired output format.
2. Overfitting: Model copies the examples word-for-word. To mitigate this, we should replace the examples regularly.

Chain-of-Thought Prompt

You are analyzing a code snippet to create a summary for documentation.
First, follow these reasoning steps:
1. Identify the main purpose of the code.
2. List each function or class and briefly describe what it does.
3. Determine the inputs and outputs for the code.
4. Check for external libraries or dependencies.
5. Combine this information into a concise 2–3 sentence summary. //Tested with and without this line to check the changes

Code:
{Code}

Final Summary (concise and professional)

Rationale:
This model method thinks step-by-step internally, increasing the accuracy and completeness, especially for complex code structures.

Expected Output:

1. Clear breakdown of functions or classes.
2. Highly accurate summary.
3. Minimal Hallucinations.

Failure Modes:

1. The only problem with this type of prompting is that sometimes the answers become too verbose. To mitigate that, we can mention in the prompt that, we want to have only the final summary.

Task 2: Technical FAQ Answering (Question-Answering)

CLEAR Prompt

You are an experienced API support engineer. Your job is to answer developer questions accurately and clearly. Answer using ONLY the provided documentation.

Context:
You will receive:
1. Developer's question
2. Relevant section of the documentation

Instructions:
- Provide a direct, actionable answer.
- Include a minimal example if helpful.
- Cite the exact section or heading from the documentation.
- Do NOT invent information beyond the provided documentation.
- Limit the response to 500 words

Question:
{Question}

Documentation:
{Documentation}

Rationale:
It would prevent hallucination since we explicitly stated that we want to use only the provided docs. Also, it sets the role as a support engineer.

Expected Output:
1. Concise answer
2. Citation of the documentation followed

Failure Modes:
1. It can make up its own documentation or APIs, to mitigate, we specifically wrote that use only the provided docs.

Few-shot Prompting

You are answering developer questions based on documentation.

Example 1:
Question: How do I authenticate with the API?
Docs: "To authenticate, include your API key in the Authorization header."
Answer: Include your API key in the `Authorization` header as follows:
Authorization: Bearer <your_api_key>

Example 2:
Question: How do I create a new user?
Docs: "POST /users creates a new user. Required fields: email, password."
Answer: Use the `POST /users` endpoint with `email` and `password` fields:
{
  "email": "test@example.com",
  "password": "mypassword"
}

Now, answer the following question using the documentation:
Question:
{Question}

Docs:
{Documentation}

Rationale:
Shows the desired answer format and tone via examples.

Expected Output:

1. Mirror the example structure.
2. Would include a citation or example of a code snippet.

Failure Modes:

1. Overgeneralization or Overfitting. To mitigate this, I would change the examples regularly or whenever the model starts repeating verbatim.

Chain-of-Thought Prompt

You are answering a developer question. Follow these reasoning steps:
1. Read the question carefully.
2. Identify key action or entity being asked about.
3. Scan the provided documentation for relevant information.
4. Match the question to the exact documentation content.
5. Formulate a clear, concise answer with a code example if applicable.

Question:
{Question}

Docs:
{Documentation}

Final Answer:

Rationale:
Improves the reasoning by forcing the model to think and walk us through the searching and mapping process.

Expected Output:

1. Clear walk-through or thinking process of the model.
2. Highly accurate and less hallucination.

Failure Modes:
Can be verbose, so have to explicitly state how and which type of answer we want.

Task 3: API Documentation Creation (Content Generation)

CLEAR Prompt

You are a technical writer and DevRel expert creating professional API documentation.

Instructions:
Use ONLY the given API specification.
Your output must include these sections in this exact order:

1. Title
2. Description
3. Endpoint & Method
4. Headers
5. Parameters (Markdown Table: Name | Type | Required | Description)
6. Example Request
7. Example Response
8. Errors
9. Notes / Best Practices

- If a field or error is not in the spec, write: "Not specified in the spec."
- Do NOT invent information.
- Limit response to 400-500 words.

API Specification:
{Specification}

Rationale:
CLEAR method ensures a consistent structure.

Expected Output:

1. Polished and clear documentation.

Failure Modes:

1. Made up fields or data. To mitigate, remind the model to stick to a given spec.

Few-shot Prompt

You are writing structured API documentation. Use ONLY the given spec.

Example:
Spec:
Endpoint: POST /orders
Headers: Authorization, Content-Type
Fields: customer_id, items[]
Description: Creates a new order.

Output:
POST /orders
Creates a new order with one or more line items.

Headers
Name	Required	Description
Authorization	Yes	Bearer token for authentication
Content-Type	Yes	Must be application/json

Parameters
Name	Type	Required	Description
customer_id	string	Yes	Unique customer identifier
items	array	Yes	Array of order items

Example Request
POST /orders
```
{
  "customer_id": "12345",
  "items": [
    { "sku": "SKU-001", "quantity": 2 }
  ]
}
```

Errors
- 400 Bad Request – Invalid parameters
- 401 Unauthorized – Missing or invalid token

Now, generate documentation for the following:

Spec:
{Specification}

Rationale:
Examples would guide the model towards developer-grade formatting.

Failure modes:
Output formatting may drift, to mitigate, ask the model to keep the examples clean and consistent throughout.

Chain-of-thought Prompting

You are documenting an API. Think silently through these steps:
1. Identify purpose of the endpoint.
2. Extract headers, request fields, and response fields.
3. Identify all errors or missing fields.
4. Plan how to structure the output cleanly.

Then OUTPUT ONLY the final documentation with these sections:
Title
Description
Endpoint & Method
Headers
Parameters (Markdown table)
Example Request
Example Response
Errors
Notes / Best Practices

Rules:
- If a field or error is not present in the spec, write: "Not specified in the spec."
- Do NOT invent information or endpoints.
- Limit total length to 500 words.

API Specification:
{Specification}

Rationale:
Ensures full coverage by guiding the model through each documentation element systematically.

Failure modes:
Can be overly verbose, so limit the section lengths with explicit constraints.

Pilot Testing:

Pilot testing was done with each prompt x model to check the prompts and make them more airtight and better. The common problem seen with them was giving out information that was not much useful, or that was making the answers or responses unnecessarily longer.

Thus, after the pilot testing, all the prompts were refined. The examples in few-shot were changed for the API documentation to give a proper example as to what and how the responses are needed. Specific instructions were added to make the responses shorter, ie, I specifically added instructions that summarize the answers in 2-3 sentences, or 300-500 words, limiting the answer length.

PS: Another thing I noticed was that some LLMs, even GPT, were repeating the same responses even though I deleted the line to summarize the answer within 2-3 sentences. Up until I added the word “explain” or “In-detail”, the 2-3 line answer was there and no explanations or details were provided.