**Notes**

- All models were set to a temperature of 0.7 and 768 max tokens unless stated otherwise
- Some of the outputs are REALLY lengthy. For convenience, I highlighted the parts that directly addressed the task (for the long outputs).
- Code BLEU scores are calculated only on the fixed/completed/created functions for the task and with new lines removed from the functions

## Task 1: Code Summarization (Java)

### Prompting Strategy 1: Zero-Shot

| | | |
|---|---|---|
| **Task** | Code Summarization (Java) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Summarize what this code does in one sentence:
```java
public Map < String , Integer > countWordFrequency ( List < String > words ) {
    Map < String , Integer > freqMap = new HashMap < >() ;
    for ( String word : words ) {
        freqMap . put ( word , freqMap . getOrDefault ( word , 0) + 1) ;
    }
    return freqMap ;
}
```
~~~

| | | |
|---|---|---|
| **Output** | The code defines a method that counts the frequency of each word in a list and returns a map where the keys are the words and the values are their respective counts. | This code defines a method that takes a list of words and returns a map with each word as the key and its frequency in the list as the value. |

### Prompting Strategy 2: Self-Consistency

| | | |
|---|---|---|
| **Task** | Code Summarization (Java) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Summarize what this code does in one sentence:
```java
public Map < String , Integer > countWordFrequency ( List <
~~~

~~~
String > words ) {
    Map < String , Integer > freqMap = new HashMap < >() ;
    for ( String word : words ) {
        freqMap . put ( word , freqMap . getOrDefault ( word , 0) + 1) ;
    }
    return freqMap ;
}
```
~~~

| | | |
|---|---|---|
| **Output Attempt 1** | The code defines a method that counts the frequency of each word in a list and returns a map where the keys are the words and the values are their respective counts. | This code defines a method that takes a list of words and returns a map with each word as the key and its frequency in the list as the value. |
| **Output Attempt 2** | This code defines a method that takes a list of words and returns a map that counts the frequency of each word in the list. | This Java method `countWordFrequency` takes a list of strings and returns a map where the keys are the strings and the values are their respective frequencies in the list. |
| **Output Attempt 3** | This code defines a method that takes a list of words and returns a map containing the frequency count of each word. | The code defines a method that takes a list of words and returns a map where the keys are words and the values are their respective frequencies in the list. |

**Analysis of Models**

BLEU Scores (between models)

- Output Attempt 1 (same as zero shot): 24.08
- Output Attempt 2: 22.78
- Output Attempt 3: 42.34

Although the BLEU scores are not very high, both models semantically responded with similar things across both prompting strategies. However, one thing to note is that Codestral 25.01 defined the keys and values in all three runs while gpt-4o mini only defined them in 1.

**Analysis of Prompting Strategies**

The zero shot approach was sufficient for the task at hand. Since it is the more time-saving approach, I think zero shot prompting was the better strategy for task 1.

# Task 2: Bug Fixing (Python – Off-by-One)

### Prompting Strategy 1: Zero-Shot

| **Task** | Bug Fixing (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Identify and fix the bug in the following Python function:
```python
def sum_range ( start , end ) :
    total = 0
    for i in range ( start , end ) :
        total += i
    return total
```
~~~

**Output**

gpt-4o mini:

~~~
The provided function
`sum_range` is intended to
calculate the sum of all
integers from `start` to `end -
1`. However, if the intention
is to include `end` in the
summation (i.e., to sum from
`start` to `end` inclusive),
the range should be modified.

Here's the corrected version of
the function:

```python
def sum_range(start, end):
    total = 0
    for i in range(start, end +
1):  # Change end to end + 1 to
include 'end'
        total += i
    return total
```

If the original intention was
to sum from `start` to `end -
1`, then the function is
correct as it is. Please
clarify the expected behavior,
and choose the appropriate
version based on whether you
~~~

Codestral 25.01:

~~~
The bug in the given Python
function `sum_range` is
related to the range of
numbers being summed. The
`range` function in Python
includes the start value but
excludes the end value.
Therefore, the function
currently calculates the sum
of all integers from `start`
to `end-1`. If you want to
include the `end` value in
the sum, you need to adjust
the range to include `end`.

Here is the corrected
function:

```python
def sum_range(start, end):
    total = 0
    for i in range(start, end
+ 1):
        total += i
    return total
```

With this change, the
function will correctly sum
all integers from `start` to
~~~

| | | |
|---|---|---|
| | want to include `end` or not. | `end` inclusive.<br><br>Let's see an example to verify the correctness:<br><br>```python<br>print(sum_range(1, 5))  # Output should be 1 + 2 + 3 + 4 + 5 = 15<br>print(sum_range(0, 0))  # Output should be 0<br>print(sum_range(3, 7))  # Output should be 3 + 4 + 5 + 6 + 7 = 25<br>```<br><br>These examples demonstrate that the function now correctly sums the range of numbers from `start` to `end` inclusive. |

**Prompting Strategy 2: Prompt Chaining**

| | | |
|---|---|---|
| **Task** | Bug Fixing (Python) | |
| **Model** | gpt-4.1 mini | Codestral 25.01 |

**Prompt**

Step 1 Prompt:

~~~
Identify the bug in the following Python function:
```python
def sum_range ( start , end ) :
    total = 0
    for i in range ( start , end ) :
        total += i
    return total
```
Give only the identified bug back as an answer.
~~~

Step 2 Prompt:

~~~
Fix the bug in the following Python function:
```python
def sum_range ( start , end ) :
    total = 0
    for i in range ( start , end ) :
~~~

~~~
        total += i
    return total
```
~~~

**Output**

Response 1:
The bug in the function is that the range function does not include the `end` value.

Response 2:
You can fix the bug by changing the range function to include the `end` value. You can do this by adding 1 to the `end` value in the range function. Here is the corrected function:

~~~
```python
def sum_range ( start , end ) :
    total = 0
    for i in range ( start ,
end + 1 ) :
        total += i
    return total
```
~~~

Response 1:
The function does not include the `end` value in the sum because `range(start, end)` is exclusive of `end`.

Response 2:

~~~
```python
def sum_range(start, end):
    total = 0
    for i in range(start, end
+ 1):
        total += i
    return total
```
~~~

**Analysis of Models**

BLEU scores:

- Zero shot (between **gpt-4o** mini and Codestral 25.01): 29.36
- Prompt Chaining (between **gpt-4.1** mini and Codestral 25.01)
    - Result 1: 31.58
    - Result 2: 33.11

Code BLEU scores:

- Zero shot (between **gpt-4o** mini and Codestral 25.01): 76.07
- Prompt Chaining (between **gpt-4.1** mini and Codestral 25.01): 71.01

The significant difference in BLEU score and Code BLEU score indicates that the fixed functions themselves are more similar to each other than the overall outputs.

**Analysis of Prompting Strategies**

Both prompting strategies correctly identified the bug and fixed it. However, the runs using prompt chaining are more concise, and I think therefore shows better results. This is likely because I instructed the model to identify the bug in one sentence which greatly reduced verbosity. Furthermore, the results from prompt chaining seems more sure about what bug is, whereas the results from zero shot question the user's intent with phrases like "if you want" or "if the intention is."

## Task 3: Bug Classification (C++)

### Prompting Strategy 1: Zero Shot

| | | |
|---|---|---|
| **Task** | Bug Classification (C++) | |
| **Model** | gpt-4.1-mini | Codestral 25.01 |

**Prompt**

~~~
Classify the bug in the following C++ function:

```cpp
int * getArray (int size ) {
    int arr [ size ];
    return arr ;
}
```

Do not include a solution.
~~~

| | | |
|---|---|---|
| **Output** | The bug in the function is a **returning pointer to a local stack-allocated array**, which results in **undefined behavior** due to returning a pointer to memory that goes out of scope once the function exits. | To classify the bug in the given C++ function, consider the following:<br>1. **Array Lifetime**: The array \`arr\` is created within the function \`getArray\`. This array has automatic storage duration, meaning it is allocated on the stack and will be destroyed when the function ends.<br>2. **Returning a Pointer**: The function returns a pointer to the local array \`arr\`. After the function returns, the array \`arr\` no longer exists, making the returned pointer invalid.<br>3. **Use-After-Free**: Attempting to use the returned pointer will result in undefined behavior because |

| | | |
|---|---|---|
| | | it points to memory that has been deallocated.<br><br>Therefore, the bug in this function is related to:<br>- **Lifetime of Local Variables**: Returning a pointer to a local array that goes out of scope.<br>- **Invalid Pointer Usage**: Using a pointer to memory that has been deallocated or is no longer valid.<br><br>This is an example of a common bug in C++ related to the lifetime of local variables and the dangers of returning pointers to such variables. |

### Prompting Strategy 2: Chain of Thought

| **Task** | Bug Classification (C++) | |
|---|---|---|
| **Model** | gpt-4.1 mini | Codestral 25.01 |

**Prompt**

~~~
Classify the bug in the following C++ function. Think
step-by-step:

```cpp
int * getArray (int size ) {
    int arr [ size ];
    return arr ;
}
```


Do not include a solution.
~~~

| **Output** | Let's analyze the code step-by-step:<br><br>1. The function \`getArray\` takes an integer \`size\` as input.<br>2. Inside the function, an array \`arr\` of \`int\` with | Let's think through the function step-by-step to identify any potential issues:<br><br>1. **Function Definition**: The function \`getArray\` is defined to take an integer |
|---|---|---|

| | | |
|---|---|---|
| | length `size` is declared as a local variable.<br>3. The function returns a pointer to the first element of the array `arr`.<br><br>The key point here is that `arr` is a local array with automatic storage duration. Once the function returns, the lifetime of `arr` ends, and the memory it occupied becomes invalid.<br><br>Therefore, returning a pointer to this local array results in a pointer to memory that is no longer valid once the function exits.<br><br>**Bug classification:** Returning a pointer to a local variable (stack-allocated memory) that goes out of scope, leading to undefined behavior due to a dangling pointer. | `size` and return a pointer to an integer.<br><br>2. **Local Array**: Inside the function, an array `arr` of integers is declared with a size specified by the `size` parameter: `int arr[size];`.<br><br>3. **Return Statement**: The function returns a pointer to this local array: `return arr;`.<br><br>Now, let's consider what could go wrong:<br><br>- **Lifetime of the Local Array**: The array `arr` is a local variable. Local variables in C++ have automatic storage duration, meaning they are allocated on the stack and their lifetime ends when the function returns. Returning a pointer to a local variable that goes out of scope is dangerous because the memory it points to may be reused or deallocated.<br><br>- **Dangling Pointer**: After the function `getArray` returns, the pointer returned will become a dangling pointer because it points to memory that is no longer valid.<br><br>- **Memory Leaks**: Since the function does not dynamically allocate memory using `new`, there are no memory leaks to worry about, but returning a pointer to a local array still poses a problem.<br><br>- **Undefined Behavior**: Accessing the memory through the returned pointer after the function has exited |

|  |  |  |
|---|---|---|
|  |  | results in undefined behavior, which can lead to crashes or other unpredictable behavior.<br><br>Would you like to proceed to the next step or focus on a specific aspect of the function? |

**Analysis of Models**

BLEU scores (across models)

- Zero shot: 6.27
- Chain of thought: 21.08

In general Codestral 25.01 had longer responses than gpt-4.1 mini.

**Analysis of Prompting Strategies**

In general, the chain of thought prompts provided more detailed explanations about the bug than the zero shot approach. If someone was unfamiliar with C, I could see the chain of thought runs being more helpful as it explains what is happening in the code step by step. However, in regard to the task of "classifying," the responses were too wordy. It may have been better to tell the models to limit the response to a few words.

## Task 4: Generating Email Validators (Python + Regex)

### Prompting Strategy 1: Prompt Chaining

|  |  |
|---|---|
| **Task** | Write a Python function to determine if an email is valid using Regex |
| **Prompt** | Step 1 Prompt:<br>`Give a concise list of criteria for a valid email address`<br><br>Step 2 Prompt:<br>`Given the following criteria:`<br>`{response 1 (see below)}`<br><br>`Complete the Python function using regex to validate email addresses:`<br><br>`` ```python `` |

~~~~
def is_valid_email ( email ):
  # TODO : Complete using regex
  pass
```


Please respond with only the completed function.
~~~~

| | |
|---|---|
| **Model 1** | gpt-4.1 mini |

**Output for model 1**

Response 1:

~~~~
Criteria for a valid email address:

1. **Format:** `local-part@domain`
2. **Local-part:**
   - Contains letters (A-Z, a-z), digits (0-9), and allowed special characters (`!#$%&'*+/=?^_`{|}~-`)
   - Dots (`.`) allowed but not at start/end or consecutively
3. **Domain:**
   - Consists of labels separated by dots (`.`)
   - Each label starts and ends with a letter or digit
   - Labels may contain hyphens (`-`) but not start/end with them
   - Top-level domain (last label) is at least 2 characters, only letters
4. **No spaces or unallowed special characters**
5. **Overall length:** Typically up to 254 characters maximum

Note: This aligns with RFC 5322 and domain name rules.
~~~~

Response 2:

~~~~
```python
import re

def is_valid_email(email):
    pattern = re.compile(
        r'^(?!.*\.\.)'                               # no consecutive dots in local-part
        r'[A-Za-z0-9!#$%&\'*+/=?^_`{|}~-]+'        # local-part start
        r'(?:\.[A-Za-z0-9!#$%&\'*+/=?^_`{|}~-]+)*' # local-part continued with dots
        r'@'
        r'(?:(?:[A-Za-z0-9](?:[A-Za-z0-9-]*[A-Za-z0-9])?\.)+' # domain labels
        r'[A-Za-z]{2,})$'                            # TLD with at least 2 letters
    )
    if len(email) > 254:
        return False
    return bool(pattern.match(email))
~~~~

<table>
<tr><td></td><td>```</td></tr>
<tr><td><b>Model 2</b></td><td>Codestral 25.01</td></tr>
<tr><td><b>Output for model 2</b></td><td>Response 1:<br>Give a concise list of criteria for a valid email address<br><br>RESPONSE 1:<br>Here is a concise list of criteria for a valid email address:<br><br>1. **Local Part**:<br>   - Can contain letters (a-z, A-Z), digits (0-9), and special characters (!#$%&'*+-/=?^_`{|}~).<br>   - Cannot contain spaces or certain special characters (e.g., commas, colons, semicolons).<br>   - Must be at least one character long.<br><br>2. **At Symbol (@)**:<br>   - Must be present and separate the local part from the domain part.<br><br>3. **Domain Part**:<br>   - Must contain at least one dot (.) to separate the domain name from the top-level domain (TLD).<br>   - Can include letters, digits, and hyphens (-).<br>   - Must end with a valid top-level domain (e.g., .com, .org, .net).<br><br>4. **General Length**:<br>   - The entire email address must be no longer than 254 characters.<br><br>5. **No Consecutive or Leading/Trailing Dots**:<br>   - The local part and domain part cannot have consecutive dots or start/end with a dot.<br><br>6. **No Quoted Strings**:<br>   - The local part cannot contain quoted strings (e.g., "john.doe"@example.com).<br><br>7. **No Comments**:<br>   - The email address cannot contain comments, which are enclosed in parentheses (e.g., john(doe)@example.com).<br><br>Examples of valid email addresses:<br>- john.doe@example.com<br>- jane_doe123@sub.example.co.uk<br><br>Examples of invalid email addresses:<br>- john..doe@example.com (consecutive dots)<br>- "john doe"@example.com (quoted string in local part)<br>- john.doe(@example.com (missing @ symbol)</td></tr>
</table>

Response 2:

~~~
```python
import re

def is_valid_email(email):
    # Define the regex pattern for a valid email address
    pattern =
r'^[a-zA-Z0-9!#$%&\'*+\-/=?^_`{|}~]+(\.[a-zA-Z0-9!#$%&\'*+\-/=?
^_`{|}~]+)*@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(\.[a-
zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)+$'

    # Match the email against the pattern
    if re.match(pattern, email):
        return True
    else:
        return False
```
~~~

## Prompting Strategy 2: Few Shot

| | |
|---|---|
| **Task** | Write a Python function to determine if an email is valid using Regex |

**Prompt**

```
Example 1:
Input: x@example.com
Output: True

Example 2:
Input: Abc..123@example.com
Output: False

Example 3:
Input: user.name+tag+sorting@example.com
Output: True

Example 4:
Input: user@.com
Output: False

Example 5:
Input: angh__hgj@domain.com
Output: False

Complete the Python function using regex to validate email
addresses:
```

~~~~

```python
def is_valid_email ( email ):
  # TODO : Complete using regex
  pass
```


Please respond with only the completed function.
~~~~

| | |
|---|---|
| **Model 1** | gpt-4o mini |

**Output for model 1**

~~~~
```python
import re

def is_valid_email(email):
    pattern = 
r'^(?!.*\.\.)(?!.*\.$)(?!.*@.*@)(?!.*@-)(?!.*-@)(?!.*@\.)(?!.*\
.$)[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
```
~~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

**Output for model 2**

~~~~
```python
import re

def is_valid_email(email):
    # Define the regex pattern for validating an email
    pattern = re.compile(
        r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
    )
    return bool(pattern.match(email))
```
~~~~

**Analysis of Models**

BLEU scores:

- Prompt Chaining (between gpt-4.1 mini and Codestral 25.01)
    - Result 1: 15.47

Code BLEU scores:

- Prompt Chaining (between gpt-4.1 mini and Codestral 25.01): 20.04
- Few Shot (between gpt-4o mini and Codestral 25.01): 21.22

**Analysis of Prompting Strategies**

# Task 5: Generating Flask APIs (Python)

### Prompting Strategy 1: Zero Shot

| **Task** | Generating Flask APIs (Python) |
|---|---|

**Prompt**

~~~
Create a ‘/greet/<username>‘ endpoint that returns a JSON
greeting given the following:

```python
from flask import Flask , jsonify

app = Flask ( __name__ )

@app . route (’/ greet / < username >’)
def greet ( username ) :
  # TODO : Return a JSON greeting
  pass
```
~~~

| **Model 1** | gpt-4o mini |
|---|---|

**Output for Model 1**

~~~
To create a `/greet/<username>` endpoint in your Flask
application that returns a JSON greeting, you'll need to follow
these steps:

1. **Import Required Libraries**: You already have the
necessary imports for Flask and `jsonify`.

2. **Define the Route**: You need to ensure the route is
defined correctly, handling the URL parameter for the username.

3. **Create the JSON Response**: Inside the `greet` function,
you'll create a dictionary with a greeting message that
includes the username, and then return it as a JSON response.

4. **Run the Application**: Ensure the Flask app runs if it is
executed as the main module.

Here’s how you can implement this:

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
~~~

~~~
    # Create a JSON response with a greeting message
    response = {
        'greeting': f'Hello, {username}!'
    }
    return jsonify(response)

if __name__ == '__main__':
    app.run(debug=True)
```

### Explanation of the Code:

- **Flask App Initialization**: `app = Flask(__name__)`
initializes the Flask application.

- **Route Definition**: `@app.route('/greet/<username>')`
defines a route that captures a username from the URL.

- **Greeting Function**: Inside the `greet` function:
  - A dictionary is created containing a greeting message that
incorporates the username.
  - `jsonify(response)` converts the dictionary to a JSON
response.

- **Running the App**: The block `if __name__ == '__main__':`
ensures that the Flask app runs when the script is executed
directly. `app.run(debug=True)` starts the server in debug
mode, which helps during development by providing detailed
error messages and automatic reloads.

### Testing the Endpoint:

You can test the endpoint using a web browser or a tool like
`curl` or Postman by navigating to
`http://127.0.0.1:5000/greet/<username>`, replacing
`<username>` with an actual name, such as `Alice`. You should
receive a JSON response like this:

```json
{
    "greeting": "Hello, Alice!"
}
```

This implementation provides a simple and effective way to
greet users using a RESTful API.
~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |
| **Output for Model 2** | To create a `/greet/<username>` endpoint that returns a JSON greeting, we need to follow these steps:<br><br>1. Define the route in the Flask application. |

2. Create a function that handles the request and generates the JSON response.
3. Use the `jsonify` function from Flask to create the JSON response.

Here is the complete code:

~~~
```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # Create a greeting message
    greeting_message = f"Hello, {username}! Welcome to our service."
    # Use jsonify to create a JSON response
    return jsonify({"message": greeting_message})

if __name__ == '__main__':
    app.run(debug=True)
```
~~~

### Explanation:

1. **Importing Required Modules**:
   - `Flask` is imported to create the Flask application.
   - `jsonify` is imported to create JSON responses.

2. **Creating the Flask Application**:
   - `app = Flask(__name__)` initializes the Flask application.

3. **Defining the Route**:
   - `@app.route('/greet/<username>')` sets up the route. The `<username>` part indicates that this part of the URL will be captured as a variable named `username`.

4. **Greet Function**:
   - The `greet` function takes `username` as a parameter.
   - It creates a greeting message using an f-string.
   - `jsonify({"message": greeting_message})` creates a JSON response with the greeting message.

5. **Running the Application**:
   - `if __name__ == '__main__':` ensures that the app runs only if the script is executed directly.
   - `app.run(debug=True)` starts the Flask development server with debug mode enabled.

With this setup, when you access the URL `/greet/John`, the response will be:

~~~
```json
{
    "message": "Hello, John! Welcome to our service."
}
```
~~~

### Prompting Strategy 2: Chain of Thought

| **Task** | Generating Flask APIs (Python) |
|---|---|

**Prompt**

~~~
Create a ‘/greet/<username>‘ endpoint that returns a JSON
greeting given the following:

```python
from flask import Flask , jsonify

app = Flask ( __name__ )

@app . route (’/ greet / < username >’)
def greet ( username ) :
  # TODO : Return a JSON greeting
  pass
```


Think step-by-step.
~~~

| **Model 1** | gpt-4o mini |
|---|---|

**Output for model 1**

~~~
To create a `/greet/<username>` endpoint that returns a JSON
greeting in a Flask application, we can follow these steps:

1. **Import Necessary Libraries**: We already have `Flask` and
`jsonify` imported.
2. **Define the Route**: We need to ensure the route is
correctly formatted and remove any unnecessary spaces or
incorrect characters.
3. **Create the Greeting Message**: We will create a message
using the `username` passed in the URL.
4. **Return the JSON Response**: We need to use `jsonify` to
return the greeting in JSON format.

Here's the complete implementation of the `/greet/<username>`
endpoint:

```python
from flask import Flask, jsonify
~~~

~~~
app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # Create a greeting message
    greeting_message = f"Hello, {username}!"

    # Return the greeting as a JSON response
    return jsonify({'greeting': greeting_message})

if __name__ == '__main__':
    app.run(debug=True)
```

### Explanation of the Code:
- **Route Definition**: The route is defined with
`@app.route('/greet/<username>')`, where `<username>` is a
variable that will capture the username from the URL.
- **Greeting Message**: Inside the `greet` function, we use an
f-string to create a personalized greeting message using the
`username`.
- **JSON Response**: We return the greeting message wrapped in
a dictionary and pass it to `jsonify`, which converts it to a
JSON response.
- **Running the App**: The `if __name__ == '__main__':` block
ensures that the Flask app runs when the script is executed
directly.

### Testing the Endpoint:
You can test the endpoint by running the Flask application and
then visiting `http://127.0.0.1:5000/greet/YourUsername` in
your web browser or using a tool like Postman. It should return
a JSON response like:

```json
{
  "greeting": "Hello, YourUsername!"
}
```

Make sure to replace `YourUsername` with any username you wish
to test.
~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

**Output for model 2**

~~~
Sure! Let's break down the task step-by-step to create the
`/greet/<username>` endpoint in your Flask application.

### Step 1: Set Up the Flask Application
You've already set up the basic structure of your Flask
application. Here’s what you have so far:

```python
~~~

~~~~
from flask import Flask, jsonify

app = Flask(__name__)
```

### Step 2: Define the Route
You need to define the route `/greet/<username>`. This route
will accept a username as a part of the URL.

### Step 3: Create the View Function
The view function should return a JSON greeting message. We'll
use the `jsonify` function from Flask to create a JSON
response.

### Step 4: Return the JSON Greeting
Inside the view function, you'll construct the greeting message
and return it as a JSON response.

### Complete Code
Here is the complete code with the `/greet/<username>` endpoint
implemented:

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # Create a greeting message
    message = f"Hello, {username}!"

    # Return the message as a JSON response
    return jsonify(greeting=message)

if __name__ == '__main__':
    app.run(debug=True)
```

### Explanation
1. **Importing Required Modules**: You import `Flask` and
`jsonify` from the `flask` module.
2. **Creating the Flask App**: You create an instance of the
Flask class and assign it to the variable `app`.
3. **Defining the Route**: The
`@app.route('/greet/<username>')` decorator defines the route.
The `<username>` part is a variable part of the URL.
4. **View Function (`greet`)**: This function takes the
`username` as an argument. It constructs a greeting message
using an f-string and returns it as a JSON response using
`jsonify`.

### Running the Application
~~~~

~~~
To run the application, save the code to a file (e.g.,
`app.py`) and execute it using Python:

```sh
python app.py
```

You can then access the `/greet/<username>` endpoint by
navigating to `http://127.0.0.1:5000/greet/your_username` in
your web browser or using a tool like `curl` or Postman.

For example:
- Accessing `http://127.0.0.1:5000/greet/Alice` will return:
  ```json
  {
    "greeting": "Hello, Alice!"
  }
  ```

That's it! You've successfully created a `/greet/<username>`
endpoint that returns a JSON greeting.
~~~

**Analysis of Models**

BLEU scores:

- Zero shot: 46.24
- Chain of thought: 40.58

**Analysis of Prompting Strategies**

Overall, all the runs performed very similarly. They all responded with similar functions, explanations of the code, and ways to test the code. It seems telling the models to "think step-by-step" doesn't make a huge difference. In fact, the models were already thinking step-by-step without being prompted too.

## Task 6: SQL Schema Design (SQL)

**Prompting Strategy 1: Zero Shot**

| | | |
|---|---|---|
| **Task** | SQL schema design | |
| **Model** | gpt-4o mini | Codestral 25.01 |
| **Prompt** | | |

```
Write the schema for a review app with users, books, and
reviews:
-- TODO : Design schema with appropriate keys and constraints
-- Tables : users (id , name ), books (id , title ), reviews
```

| | | |
|---|---|---|
| | (id , user_id , book_id ,<br>rating )<br>Respond with only the schema. | |

**Output**

gpt-4o mini:

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >=
1 AND rating <= 5),
    FOREIGN KEY (user_id)
REFERENCES users(id) ON DELETE
CASCADE,
    FOREIGN KEY (book_id)
REFERENCES books(id) ON DELETE
CASCADE
);
```
~~~

Codestral 25.01:

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT
NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT
NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating
>= 1 AND rating <= 5),
    FOREIGN KEY (user_id)
REFERENCES users(id) ON
DELETE CASCADE,
    FOREIGN KEY (book_id)
REFERENCES books(id) ON
DELETE CASCADE
);
```
~~~

### Prompting Strategy 2: Self Consistency

| | | |
|---|---|---|
| **Task** | SQL schema design | |
| **Model** | gpt-4o mini | Codestral 25.01 |
| **Prompt** | Write the schema for a review app with users, books, and reviews:<br>-- TODO : Design schema with appropriate keys and constraints<br>-- Tables : users (id , name ), books (id , title ), reviews (id , user_id , book_id ,<br>rating )<br>Respond with only the schema. | |

**Output Attempt**

gpt-4o mini:

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
~~~

Codestral 25.01:

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
~~~

**1**

~~~
    name VARCHAR(255) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```
~~~

~~~
    name VARCHAR(255) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```
~~~

**Output Attempt 2**

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```
~~~

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE,
    UNIQUE (user_id, book_id)
);
```
~~~

**Output Attempt 3**

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating
BETWEEN 1 AND 5),
    FOREIGN KEY (user_id)
REFERENCES users(id) ON DELETE
CASCADE,
    FOREIGN KEY (book_id)
REFERENCES books(id) ON DELETE
CASCADE,
    UNIQUE (user_id, book_id)
-- Ensures a user can only
review a book once
);
```
~~~

~~~
```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT
NULL
);

CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT
NULL
);

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating
>= 1 AND rating <= 5),
    FOREIGN KEY (user_id)
REFERENCES users(id) ON
DELETE CASCADE,
    FOREIGN KEY (book_id)
REFERENCES books(id) ON
DELETE CASCADE
);
```
~~~

**Analysis of Models**

Code BLEU scores:

- Output Attempt 1: 100
- Output Attempt 2: 87.54
- Output Attempt 3: 79.67

The Code BLEU scores are all high which demonstrates that the two different models perform similarly on the given prompt.

**Analysis of Prompting Strategies**

All the responses are really similar. However, with more runs on the same prompt (self consistency) we see the model suggest making user_id and book_id unique in the reviews table. If the intention was to ensure that the user can only review a book once, then running the prompts again would have been beneficial.

## Task 7: Null Dereference Detection (Java)

### Prompting Strategy 1: Zero Shot

| | | |
|---|---|---|
| **Task** | Null dereference detection (Java) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~~
Identify the most important issues (at most two) in the
following Java function:

```java
public int getLength ( String s ) {
  return s . length () ;
}
```
Only focus on the problems, do not respond with solutions.
~~~~

| | | |
|---|---|---|
| **Output** | 1. **Null Pointer Exception**: The function does not handle the case where the input string \`s\` could be \`null\`, which would cause a \`NullPointerException\` when attempting to call \`length()\`.<br><br>2. **Lack of Input Validation**: The function does not validate the input before proceeding, which could lead to unexpected behavior if the function is used in a context where input validation is necessary. | The most important issues in the provided Java function are:<br><br>1. **Lack of Input Validation**: The function does not check if the input string \`s\` is \`null\`. Calling the \`length()\` method on a \`null\` string will result in a \`NullPointerException\`.<br><br>2. **Inconsistent Naming Convention**: The function name \`getLength\` uses camelCase, which is a common naming convention in Java. However, the method name \`length\` in the \`String\` class is in lowercase, which might be slightly inconsistent if the intention is to mirror the behavior of the \`String\` class closely. |

### Prompting Strategy 2: Self Consistency

| | | |
|---|---|---|
| **Task** | Null dereference detection (Java) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~~
Identify the most important issues (at most two) in the
following Java function:
~~~~

~~~
```java
public int getLength ( String s ) {
  return s . length () ;
}
```
Only focus on the problems, do not respond with solutions.
~~~

| | | |
|---|---|---|
| **Output Attempt 1** | 1. **Null Pointer Exception**: The function does not handle the case where the input string `s` could be `null`, which would cause a `NullPointerException` when attempting to call `length()`.<br>2. **Lack of Input Validation**: The function does not validate the input before proceeding, which could lead to unexpected behavior if the function is used in a context where input validation is necessary. | The most important issues in the provided Java function are:<br>1. **Lack of Input Validation**: The function does not check if the input string `s` is `null`. Calling the `length()` method on a `null` string will result in a `NullPointerException`.<br>2. **Inconsistent Naming Convention**: The function name `getLength` uses camelCase, which is a common naming convention in Java. However, the method name `length` in the `String` class is in lowercase, which might be slightly inconsistent if the intention is to mirror the behavior of the `String` class closely. |
| **Output Attempt 2** | 1. **Null Pointer Exception**: The function does not handle the case where the input string `s` is `null`, which would lead to a `NullPointerException` when attempting to call `s.length()`.<br>2. **Lack of Input Validation**: The function does not validate the input string, potentially leading to unexpected behavior or results in cases where the input is not suitable (e.g., if the input type is not a string). | The most important issues in the given Java function are:<br>1. **Lack of Parameter Validation**: The function does not check whether the input parameter `s` is null. If `s` is null, calling `s.length()` will result in a `NullPointerException`.<br>2. **Potential Confusion with Method Naming**: The method name `getLength` suggests that it returns the length of a string. However, the method signature and body are straightforward and don't |

| | | |
|---|---|---|
| | | indicate any potential confusion. This issue is less critical compared to the null pointer risk. |
| **Output Attempt 3** | 1. **Null Pointer Exception**: The function does not handle the case where the input string `s` is `null`, which would result in a `NullPointerException` when attempting to call `s.length()`.<br>2. **Lack of Input Validation**: The function does not provide any validation or error handling for empty strings or other potential issues related to the input, such as ensuring that `s` is not modified or that it meets certain criteria if needed. | The most important issues in the provided Java function are:<br>1. **Lack of Null Check**: The function does not check whether the input string `s` is `null`. Attempting to call `length()` on a `null` string will result in a `NullPointerException`.<br>2. **Potential Performance Issue**: The function directly calls `s.length()`, which is an `O(1)` operation. However, if the function is part of a larger context where `s` is repeatedly accessed, it might become a performance bottleneck if `s.length()` is called frequently within a loop or in a high-performance section of the code. |

**Analysis of Models**

BLEU scores:

- Output Attempt 1: 26.49
- Output Attempt 2: 26.67
- Output Attempt 3: 27.88

The model Codestral 25.01 demonstrated more creativity in its responses. It mentioned potential issues with performance and naming conventions while gpt-4o mini only mentioned issues related if s was null.

**Analysis of Prompting Strategies**

One run (zero shot) was sufficient to detect the null dereference issue. There was no need for more runs (self consistency), although it does show that the models consistently identify the potential Null Pointer Exception as an important issue.

# Task 8: CSV Parser Variants (Python)

## Prompting Strategy 1: Zero Shot

| | |
|---|---|
| **Task** | Improve Python CSV parser to support quoted fields |
| **Prompt** | |
| **Model 1** | gpt-4o mini |
| **Output for model 1** | |

**Prompt**

~~~
Improve the following Python function to support quoted fields:

```python
def parse_csv_line ( line ) :
  return line . split (’,’) # Incomplete : doesn ’t handle
quoted fields
```
~~~

**Output for model 1**

~~~
To improve the `parse_csv_line` function to support quoted
fields, you can use the `csv` module from Python's standard
library, which is specifically designed to handle CSV parsing,
including quoted fields and other complexities. Here’s an
improved version of your function:

```python
import csv
from io import StringIO

def parse_csv_line(line):
    # Use StringIO to treat the string as a file
    f = StringIO(line)
    reader = csv.reader(f)
    return next(reader)  # Read the first (and only) row
```

### Explanation:
1. **`csv.reader`**: This function handles the parsing of CSV
lines, taking care of quoted fields, commas within quotes, and
other CSV formatting issues.
2. **`StringIO`**: This allows us to treat a string as a file,
which is necessary because `csv.reader` expects a file-like
object.

### Example Usage:
```python
line = 'value1,"value, with, commas",value3'
parsed = parse_csv_line(line)
print(parsed)  # Output: ['value1', 'value, with, commas',
'value3']
```

This approach is robust and will correctly parse lines with
~~~

| | various complexities in CSV formatting. |
|---|---|
| **Model 2** | Codestral 25.01 |
| **Output for model 2** | |

~~~~
To support quoted fields in a CSV line, you need to handle cases where fields may contain commas, newline characters, or other special characters. This involves parsing the line character by character and accounting for quoted sections.

Here is an improved version of the function that handles quoted fields:

```python
def parse_csv_line(line):
    fields = []
    field = []
    in_quote = False
    escape_next = False

    for char in line:
        if escape_next:
            field.append(char)
            escape_next = False
        elif char == '"':
            if in_quote:
                if field and field[-1] == '"':  # Handle double quotes
                    field.append(char)
                else:
                    in_quote = False
            else:
                in_quote = True
        elif char == ',' and not in_quote:
            fields.append(''.join(field))
            field = []
        elif char == '\\':
            escape_next = True
        else:
            field.append(char)

    # Append the last field
    fields.append(''.join(field))

    return fields

# Example usage:
line = 'John Doe,"123, Main St",50,"He said, ""Hello!""","New\nLine"'
print(parse_csv_line(line))
```

### Explanation:
~~~~

````
1. **Initialization**:
   - `fields` is a list to store the parsed fields.
   - `field` is a list to accumulate characters of the current
field.
   - `in_quote` is a flag to track whether we are inside a
quoted section.
   - `escape_next` is a flag to handle escaped characters.

2. **Character Iteration**:
   - If `escape_next` is `True`, the character is appended to
`field` and `escape_next` is reset.
   - If a double quote (`"`) is encountered:
     - If `in_quote` is `True`, it toggles `in_quote` off,
ensuring that the closing quote is part of the field.
     - If `in_quote` is `False`, it toggles `in_quote` on.
   - If a comma (`,`) is encountered and `in_quote` is `False`,
the current `field` is added to `fields`, and `field` is reset.
   - If a backslash (`\`) is encountered, `escape_next` is set
to `True` to handle the next character as escaped.
   - Otherwise, the character is appended to `field`.

3. **Final Field**:
   - After the loop, the last field is added to `fields`.

This function now properly handles quoted fields, including
those containing commas, double quotes, and escaped characters.
````

**Prompting Strategy 2: Chain of Thought**

| **Task** | Improve Python CSV parser to support quoted fields |
|---|---|

**Prompt**

~~~
Improve the following Python function to support quoted fields:
Think step-by-step:

```python
def parse_csv_line ( line ) :
  return line . split (’,’) # Incomplete : doesn ’t handle
quoted fields
```
~~~

| **Model 1** | gpt-4o mini |
|---|---|

**Output for model 1**

````
To improve the `parse_csv_line` function to handle quoted
fields, we need to consider the following steps:

1. **Understanding Quoted Fields**: In CSV files, fields can be
enclosed in quotes. If a field contains a comma, it should be
treated as part of the field rather than as a delimiter. For
example, the string `"Doe, John",25` should be split into
````

`["Doe, John", "25"]`.

2. **Using Regular Expressions**: To handle the complexity of quoted fields, we can use the `re` module in Python, which allows for more sophisticated string manipulation.

3. **Implementing the Function**: We need to create a regex pattern that accounts for:
   - Fields enclosed in double quotes (which may contain commas).
   - Unquoted fields (which are separated by commas).
   - Handling escape quotes if necessary (e.g., `""""` should be interpreted as a single `"`).

Here’s an updated version of the `parse_csv_line` function:

```python
import re

def parse_csv_line(line):
    # Regular expression to match fields in a CSV line.
    # This pattern accounts for:
    # - Quoted fields: "field, with, commas"
    # - Unquoted fields: field1,field2
    # - Optional spaces around commas
    pattern = r'''(
        "([^"]*)"          # Match a quoted field (with any content inside)
        |                   # OR
        ([^,"]+)           # Match an unquoted field (any content except commas or quotes)
    )'''

    # Find all matches in the line
    matches = re.findall(pattern, line.strip())

    # Extract the matched groups into a list
    result = []
    for match in matches:
        # Either the first group will be filled (quoted) or the second (unquoted)
        quoted_field, unquoted_field = match[1], match[2]
        if quoted_field:
            # Remove surrounding quotes and return the field
            result.append(quoted_field)
        elif unquoted_field:
            result.append(unquoted_field.strip())

    return result

# Example usage
line = '"Doe, John",25,"New York, NY"'
parsed_line = parse_csv_line(line)
```

~~~~
print(parsed_line)  # Output: ['Doe, John', '25', 'New York, NY']
```

### Explanation of the Code:

1. **Regex Pattern**:
   - `r'''(...)'''`: This is a raw string that allows multiline regex patterns.
   - `("([^"]*)")`: This captures a quoted field, allowing any character except a quote inside.
   - `|` (OR): This allows for matching unquoted fields.
   - `([^,"]+)`: Captures any sequence of characters that are not commas or quotes.

2. **Finding Matches**: `re.findall()` is used to return all non-overlapping matches of the pattern in the string.

3. **Processing Matches**: The function iterates through the matches and appends the corresponding fields to the result list. If a field is quoted, it is added as is. If it's unquoted, it is stripped of any leading or trailing spaces.

This function can now properly parse CSV lines with quoted fields, handling commas within those fields correctly.
~~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

**Output for model 2**

~~~~
To improve the `parse_csv_line` function to handle quoted fields correctly, we need to consider the following steps:

1. **Identify Quoted Fields**: Determine when a field is enclosed in quotes.
2. **Handle Leading and Trailing Spaces**: Ensure that spaces inside quoted fields are preserved.
3. **Handle Embedded Commas**: Allow commas within quoted fields without splitting them.

Here's a step-by-step approach to achieve this:

### Step 1: Identify Quoted Fields
We need to identify fields that are enclosed in quotes. This can be done by iterating through the characters of the line.

### Step 2: Handle Leading and Trailing Spaces
Quoted fields may have leading and trailing spaces that should be preserved.

### Step 3: Handle Embedded Commas
Commas inside quoted fields should not be considered as field separators.

### Step 4: Implement the Logic
~~~~

We will use a state machine approach to handle the parsing:
- **Inside Quoted Field**: When we encounter a quote, we toggle between inside and outside a quoted field.
- **Inside Non-Quoted Field**: When we encounter a comma, we split the field.

Here is the improved function:

```python
def parse_csv_line(line):
    fields = []
    field = []
    inside_quotes = False
    escape_char = False

    for char in line:
        if char == '"' and not escape_char:
            # Toggle the inside_quotes state
            inside_quotes = not inside_quotes
        elif char == ',' and not inside_quotes:
            # End of a field
            fields.append(''.join(field).strip())
            field = []
        elif char == '\\' and inside_quotes:
            # Handle escape character
            escape_char = True
        else:
            # Regular character
            field.append(char)
            escape_char = False

    # Add the last field
    fields.append(''.join(field).strip())

    return fields

# Example usage
line = 'John "Doe, Jr.", 25, "New York, NY"'
print(parse_csv_line(line))
```

### Explanation:
1. **Initialization**: We initialize an empty list `fields` to store the parsed fields, an empty list `field` to build the current field, and boolean flags `inside_quotes` and `escape_char`.

2. **Iterate through Characters**: We iterate through each character in the line.
    - **Quotes**: If we encounter a quote (`"`), we toggle the `inside_quotes` state unless we are inside an escape sequence.
    - **Commas**: If we encounter a comma (`,`) and we are not inside quotes, we end the current field, strip any

```
leading/trailing spaces, and add it to `fields`.
    - **Escape Character**: If we encounter an escape character
(`\`), we set the `escape_char` flag to handle the next
character as part of the field.
    - **Regular Characters**: We add the character to the
`field` and reset the `escape_char` flag.

3. **Final Field**: After the loop, we add the last field to
`fields`.

This approach ensures that quoted fields are handled correctly,
preserving spaces and embedded commas.
```

**Analysis of Models**

BLEU scores:

- Zero Shot: 6.29
- Chain of Thought: 22.6

Code BLEU scores:

- Zero Shot: 0.64
- Chain of Thought: 0.45

The gpt-4o mini model used libraries to solve the task, while Codestral 25.01 handled the quoted fields without use of an external library. This also explains why the Code BLEU scores are so low. The regex solution (chain of thought, gpt-4o mini) did not work though.

**Analysis of Prompting Strategies:**

For the responses to gpt-4o-mini, there was a major change in implementation from zero shot to chain of thought. I think since the model was told to list out the steps to improve the function, it made the code more complicated to correspond to those steps. On the other hand, the responses from Codestral followed similar logic although the implementation of the if statements was a little different. Overall, I would choose the solution from gpt-4o-mini using zero shot for conciseness or the solution from Codestral using chain of thought to understand how parsing of quoted fields is done.

## Task 9: Data Class to API Conversion (Kotlin)

### Prompting Strategy 1: Zero Shot

| | |
|---|---|
| **Task** | Data Class to API Conversion (Kotlin) |
| **Prompt** | `Convert the data class to a REST API using Ktor:` |

```
data class Product ( val id : Int , val name : String , val
price : Double )
// TODO : Create GET and POST endpoints using Ktor
```

| | |
|---|---|
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |

**Output for model 1**

~~~
To create a REST API using Ktor with a `Product` data class,
you'll need to follow several steps. Below is a complete
example that includes setting up Ktor, defining the data class,
and implementing GET and POST endpoints.

First, ensure you have the necessary dependencies in your
`build.gradle.kts` file:

```kotlin
plugins {
    kotlin("jvm") version "1.6.21"
    id("io.ktor.plugin") version "2.0.0"
}

repositories {
    mavenCentral()
}

dependencies {
    implementation("io.ktor:ktor-server-core:2.0.0")
    implementation("io.ktor:ktor-server-netty:2.0.0")

implementation("io.ktor:ktor-server-content-negotiation:2.0.0")

implementation("io.ktor:ktor-serialization-kotlinx-json:2.0.0")

implementation("org.jetbrains.kotlinx:kotlinx-serialization-jso
n:1.3.2")
}
```

Next, create the `Product` data class and implement the Ktor
server with GET and POST endpoints.

```kotlin
import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.features.StatusPages
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.serialization.*
~~~

```
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json

@Serializable
data class Product(
    val id: Int,
    val name: String,
    val price: Double
)

fun main() {
    val products = mutableListOf(
        Product(1, "Product 1", 10.0),
        Product(2, "Product 2", 20.0)
    )

    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json(Json { prettyPrint = true })
        }

        install(StatusPages) {
            exception<Throwable> { cause ->

call.respond(HttpStatusCode.InternalServerError,
cause.localizedMessage)
            }
        }

        routing {
            get("/products") {
                call.respond(products)
            }

            post("/products") {
                val product = call.receive<Product>()
                products.add(product)
                call.respond(HttpStatusCode.Created,
product.id)
            }
        }
    }.start(wait = true)
}
```

### Explanation:

1. **Dependencies**: The `build.gradle.kts` file includes the necessary Ktor and serialization dependencies.

2. **Product Data Class**: The `Product` data class is

~~~
annotated with `@Serializable` to enable
serialization/deserialization with `kotlinx.serialization`.

3. **Main Function**:
   - An embedded Netty server is created on port 8080.
   - `ContentNegotiation` is installed to handle JSON
serialization/deserialization.
   - `StatusPages` is installed to handle exceptions.
   - Routing is configured to handle GET and POST requests:
     - `GET /products`: Responds with the list of products.
     - `POST /products`: Receives a new product from the
request body, adds it to the list, and responds with the ID of
the new product.

### Running the Server:

To run the server, execute the `main` function. The server will
start and listen on port 8080. You can test the endpoints using
tools like `curl`, Postman, or any HTTP client.

- **GET /products**:
  ```sh
  curl -X GET http://localhost:8080/products
  ~~~
- **POST /products**:
  ```sh
  curl -X POST -H "Content-Type: application/json" -d '{"id":
3, "name": "Product 3", "price": 30.0}'
http://localhost:8080/products
  ```

This example provides a basic structure for a REST API using
Ktor. You can expand it with additional features like
validation, error handling, and more complex business logic as
needed.
```

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

**Output for model 2**

````
Certainly! Here's a simple example of how to create a Ktor REST
API with GET and POST endpoints for the `Product` data class.

```kotlin
import io.ktor.application.*
import io.ktor.http.*
import io.ktor.response.*
import io.ktor.request.*
import io.ktor.routing.*
import io.ktor.features.*
import io.ktor.serialization.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

data class Product(val id: Int, val name: String, val price:
````

```
Double)

fun main() {
    // In-memory storage for products
    val products = mutableListOf<Product>()

    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }
        routing {
            route("/products") {
                // GET all products
                get {
                    call.respond(products)
                }

                // POST a new product
                post {
                    val product = call.receive<Product>()
                    products.add(product)
                    call.respond(HttpStatusCode.Created, product)
                }

                // Optional: GET a product by id
                get("{id}") {
                    val id = call.parameters["id"]?.toIntOrNull()
                    if (id == null) {
                        call.respond(HttpStatusCode.BadRequest, "Missing or invalid id")
                        return@get
                    }
                    val product = products.find { it.id == id }
                    if (product == null) {
                        call.respond(HttpStatusCode.NotFound, "Product not found")
                    } else {
                        call.respond(product)
                    }
                }
            }
        }
    }.start(wait = true)
}
```

### Explanation:
- The app listens on port 8080.
- The `products` list holds the products in memory.
- `GET /products` returns all products.
- `POST /products` accepts a Product JSON and adds it to the

~~~
list.
- `GET /products/{id}` returns a product by its id (optional
but commonly useful).
- `ContentNegotiation` with `json` is installed to handle JSON
serialization/deserialization.

### Usage example:

**POST** `/products` with JSON body:

```json
{
  "id": 1,
  "name": "Laptop",
  "price": 999.99
}
```

**GET** `/products` returns:

```json
[
  {
    "id": 1,
    "name": "Laptop",
    "price": 999.99
  }
]
```

Let me know if you want me to include other HTTP methods or use
a persistent database!
~~~

**Prompting Strategy 2: Chain of Thought**

| **Task** | Data Class to API Conversion (Kotlin) |
|---|---|
| **Prompt** | `Convert the following data class to a REST API using Ktor.`<br>`Think step-by-step:`<br>`data class Product ( val id : Int , val name : String , val price : Double )`<br>`// TODO : Create GET and POST endpoints using Ktor` |
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |
| **Output for** | ``Sure! Let's convert your Kotlin `data class` into a simple REST API using Ktor that supports both GET and POST endpoints`` |

**model 1**

~~~
step-by-step.

---

### Step 1: Setup Ktor project

Make sure you have a Ktor project set up with the necessary
dependencies. Here's an example of what your `build.gradle.kts`
might include:

```kotlin
plugins {
    kotlin("jvm") version "1.8.10"
    application
    id("io.ktor.plugin") version "2.3.0"
}

repositories {
    mavenCentral()
}

dependencies {
    implementation("io.ktor:ktor-server-core:2.3.0")
    implementation("io.ktor:ktor-server-netty:2.3.0")

implementation("io.ktor:ktor-server-content-negotiation:2.3.0")

implementation("io.ktor:ktor-serialization-kotlinx-json:2.3.0")
    testImplementation("io.ktor:ktor-server-tests:2.3.0")
    testImplementation(kotlin("test"))
}

application {
    mainClass.set("io.ktor.server.netty.EngineMain")
}
```

---

### Step 2: Define your `Product` data class

```kotlin
data class Product(val id: Int, val name: String, val price:
Double)
```

---

### Step 3: Create a simple in-memory storage for Products

For demonstration, we'll store products in a mutable list.

```kotlin
~~~

~~~
val products = mutableListOf<Product>()
```

---

### Step 4: Configure Ktor Application with routes

We'll create GET and POST endpoints:

- **GET /products** - returns all products
- **POST /products** - accepts a Product in the request body
and adds it to the list

---

### Step 5: Complete Ktor Application code

```kotlin
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import io.ktor.server.application.*
import io.ktor.server.plugins.contentnegotiation.*
import io.ktor.serialization.kotlinx.json.*
import io.ktor.server.response.*
import io.ktor.server.request.*
import io.ktor.server.routing.*
import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json

@Serializable
data class Product(val id: Int, val name: String, val price:
Double)

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json(Json {
                prettyPrint = true
                isLenient = true
            })
        }
        routing {
            val products = mutableListOf<Product>()

            get("/products") {
                call.respond(products)
            }

            post("/products") {
                val product = call.receive<Product>()
                products.add(product)
                call.respond(product)
            }
~~~

~~~
        }
    }.start(wait = true)
}
```

---

### Explanation:

- We use `ContentNegotiation` with `kotlinx.serialization` for
JSON handling.
- The `GET /products` endpoint returns the current list of
products.
- The `POST /products` endpoint reads a `Product` from the
request body and adds it to the list, then returns the added
product.
- The server runs on port 8080.

---

### Step 6: Test your API

**Example POST request:**

```bash
curl -X POST http://localhost:8080/products \
-H "Content-Type: application/json" \
-d '{"id":1,"name":"Apple","price":0.99}'
```

**Example GET request:**

```bash
curl http://localhost:8080/products
```

Expected response:

```json
[
  {
    "id": 1,
    "name": "Apple",
    "price": 0.99
  }
]
```

---

Let me know if you want me to help with more features like PUT,
DELETE or persistence!
~~~

| Model 2 | Codestral 25.01 |
|---|---|

**Output for model 2**

~~~~
Sure! To create a REST API using Ktor for the `Product` data
class, we'll need to follow several steps. We'll use Ktor,
which is a framework for building asynchronous servers and
clients in connected systems. Here’s a step-by-step guide:

### Step 1: Add Dependencies
First, ensure you have the necessary dependencies in your
`build.gradle.kts` file:
```kotlin
dependencies {
    implementation("io.ktor:ktor-server-core:2.0.0")
    implementation("io.ktor:ktor-server-netty:2.0.0")
    implementation("io.ktor:ktor-server-auth:2.0.0")

implementation("io.ktor:ktor-server-content-negotiation:2.0.0")

implementation("io.ktor:ktor-serialization-kotlinx-json:2.0.0")
    implementation("io.ktor:ktor-server-status-pages:2.0.0")
    implementation("io.ktor:ktor-client-core:2.0.0")
    implementation("io.ktor:ktor-client-serialization:2.0.0")
}
```

### Step 2: Create the Data Class
Define your `Product` data class:
```kotlin
data class Product(
    val id: Int,
    val name: String,
    val price: Double
)
```

### Step 3: Set Up Ktor Application
Create a Ktor application and configure it to handle JSON
serialization:
```kotlin
import io.ktor.application.*
import io.ktor.features.*
import io.ktor.http.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.serialization.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }
~~~~

~~~
        routing {
            // Define routes here
        }
    }.start(wait = true)
}
```

### Step 4: Implement In-Memory Storage
For simplicity, we'll use an in-memory list to store our
products:
```kotlin
val products = mutableListOf<Product>()
```

### Step 5: Create GET Endpoint
Create an endpoint to retrieve the list of products:
```kotlin
routing {
    get("/products") {
        call.respond(products)
    }
}
```

### Step 6: Create POST Endpoint
Create an endpoint to add a new product:
```kotlin
routing {
    post("/products") {
        val product = call.receive<Product>()
        products.add(product)
        call.respond(HttpStatusCode.Created, product)
    }
}
```

### Step 7: Complete the Ktor Application
Combine all the steps into a complete Ktor application:
```kotlin
import io.ktor.application.*
import io.ktor.features.*
import io.ktor.http.*
import io.ktor.response.*
import io.ktor.request.*
import io.ktor.routing.*
import io.ktor.serialization.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

data class Product(
    val id: Int,
    val name: String,
~~~

~~~
    val price: Double
)

val products = mutableListOf<Product>()

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }

        routing {
            get("/products") {
                call.respond(products)
            }

            post("/products") {
                val product = call.receive<Product>()
                products.add(product)
                call.respond(HttpStatusCode.Created, product)
            }
        }
    }.start(wait = true)
}
```

### Step 8: Test Your API
Run your application and test the API using tools like `curl`, Postman, or any HTTP client.

- **GET Request**:
    ```sh
    curl -X GET http://localhost:8080/products
    ```

- **POST Request**:
    ```sh
    curl -X POST http://localhost:8080/products -H "Content-Type: application/json" -d '{"id": 1, "name": "Sample Product", "price": 19.99}'
    ```

With these steps, you have created a simple REST API using Ktor that can handle GET and POST requests for the `Product` data class.
~~~

**Analysis of Models**

BLEU scores

- Zero shot: 35.49
- Chain of thought: 48.96

Code BLEU score:

- Zero shot: 57.69
- Chain of thought: 55

**Analysis of Prompting Strategies**

The output and code implementation across strategies look pretty similar. They all mention the necessary dependencies and include the implementation and how to test the API. In terms of performance on the code, they’re all similar. However, I personally prefer the chain of thoughts strategy since I’ve never used Kotlin and it breaks the code down in smaller chunks.

## Task 10: Function Summarization (Python)

**Prompting Strategy 1: Zero Shot**

| **Task** | Function Summarization (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Summarize what this code does in one sentence:

```python
def reverse_words ( sentence ) :
  return ’ ’. join ( sentence . split () [:: -1])
```
~~~

| **Output** | `The code defines a function that takes a sentence as input and returns the sentence with its words reversed.` | `This code defines a function that reverses the order of words in a given sentence.` |
|---|---|---|

**Prompting Strategy 2: Self Consistency**

| **Task** | Function Summarization (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Summarize what this code does in one sentence:

```python
def reverse_words ( sentence ) :
  return ’ ’. join ( sentence . split () [:: -1])
```
~~~

| **Output** | `The code defines a function` | `This code defines a function` |
|---|---|---|

| | | |
|---|---|---|
| **Attempt 1** | `that takes a sentence as input and returns the sentence with its words reversed.` | `that reverses the order of words in a given sentence.` |
| **Output Attempt 2** | `The code defines a function that takes a sentence as input and returns the sentence with its words reversed.` | `This code defines a function that reverses the order of words in a given sentence.` |
| **Output Attempt 3** | `The code defines a function that reverses the order of words in a given sentence.` | `This code defines a function that reverses the order of words in a given sentence.` |

**Analysis of Models**

BLEU scores:

- Output Attempt 1: 21.31
- Output Attempt 2: 21.31
- Output Attempt 3: 93.06

The summaries across both models had the same meaning even if they weren't exact matches.

**Analysis of Prompting Strategies**

One run of the prompt was enough to complete the task so zero shot is the more efficient strategy in this case.

## Task 11: Prompt from Code Comments (Python)

**Prompting Strategy 1: Zero Shot**

| | | |
|---|---|---|
| **Task** | Write a prompt to generate some given Python code | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

```
Write a prompt to generate the given Python function:

# This function checks if a number is prime
def is_prime ( n ) :
  if n <= 1:
    return False
  for i in range (2 , int( n **0.5) +1) :
    if n % i == 0:
      return False
  return True
```

| | | |
|---|---|---|
| **Output** | Generate a Python function named `is_prime` that checks if a given integer `n` is a prime number. The function should return `False` for numbers less than or equal to 1, and for any number that is divisible by any integer from 2 up to the square root of `n`. It should return `True` if the number is prime. | Generate a Python function to check if a number is prime. |

### Prompting Strategy 2: Self Consistency

| | | |
|---|---|---|
| **Task** | Write a prompt to generate some given Python code | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

```
Write a prompt to generate the given Python function:


# This function checks if a number is prime
def is_prime ( n ) :
  if n <= 1:
    return False
  for i in range (2 , int( n **0.5) +1) :
    if n % i == 0:
      return False
  return True
```

| | | |
|---|---|---|
| **Output Attempt 1** | Generate a Python function named `is_prime` that checks if a given integer `n` is a prime number. The function should return `False` for numbers less than or equal to 1, and for any number that is divisible by any integer from 2 up to the square root of `n`. It should return `True` if the number is prime. | Generate a Python function to check if a number is prime. |
| **Output Attempt 2** | Create a Python function named `is_prime` that takes a single integer `n` as input and returns `True` if `n` is a prime number and `False` otherwise. The function should handle cases where `n` is less than or equal to 1 by returning `False`, and for other values, it should check for | Generate a Python function that checks if a number is prime. |

| | | |
|---|---|---|
| | `divisibility from 2 up to the square root of `n`.` | |
| **Output Attempt 3** | `Generate a Python function named `is_prime` that checks if a given number `n` is a prime number. The function should return `False` if `n` is less than or equal to 1. For numbers greater than 1, it should iterate from 2 to the square root of `n`, returning `False` if any number divides `n` evenly. If no divisors are found, it should return `True`.` | `Generate a Python function that checks if a given number is prime.` |

**Analysis of Models**
BLEU scores:

- Output Attempt 1: 6.87
- Output Attempt 2: 2.18
- Output Attempt 3: 8.08

The responses from gpt-4o mini were very specific, almost as if outlining what each line of the code does. On the other hand, the responses from Codestral 25.01 just state the intent of the function.

**Analysis of Prompting Strategies**
The models were highly consistent across the three runs. Therefore, one run was sufficient and zero shot is the more efficient strategy for this task.

## Task 12: Fixing Factorial Bug (Python)

**Prompting Strategy 1: Zero Shot**

| | | |
|---|---|---|
| **Task** | Fixing factorial bug (Python) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Fix the bug when input is 0 in the following Python function:

```python
def factorial ( n ) :
  result = 1
  for i in range (1 , n ) :
~~~

~~~
    result *= i
  return result
```


Only reply with the fixed function.
~~~

**Output**

gpt-4o mini:

~~~
```python
def factorial(n):
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```
~~~

Codestral 25.01:

~~~
```python
def factorial(n):
  result = 1
  for i in range(1, n + 1):
    result *= i
  return result
```
~~~

### Prompting Strategy 2: Few Shot

| | | |
|---|---|---|
| **Task** | Fixing factorial bug (Python) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Example 1:
Input: 0
Output: 1



Example 2:
Input: 1
Output: 1



Example 3:
Input: 5
Output: 120

Fix the bug when input is 0 in the following Python function:

```python
def factorial ( n ) :
  result = 1
  for i in range (1 , n ) :
    result *= i
~~~

~~~
  return result
```


Only reply with the fixed function.
~~~

**Output**

~~~
```python
def factorial(n):
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```
~~~

~~~
```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```
~~~

**Analysis of Models**

Code BLEU scores:

- Zero shot: 53.53
- Few shot: 53.53

The response by gpt-4o mini explicitly returns the factorial of 0 while the response by Codestral 25.01 lets the logic calculate it.

**Analysis of Prompting Strategies**

There is no difference across the prompting strategies.

## Task 13: Linked List Node Deletion (C)

### Prompting Strategy 1: Zero Shot

| | |
|---|---|
| **Task** | Linked list node deletion in C |

**Prompt**

~~~
Given the following, implement node deletion by value in C:

```c
struct Node {
  int data ;
  struct Node * next ;
};

void deleteNode ( struct Node ** head , int key ) {
~~~

~~~
  // TODO : Implement node deletion
}
```


Do not return more functions than was originally provided.
~~~

| **Model 1** | gpt-4.1 mini |
|---|---|

**Output for model 1**

~~~
```c
void deleteNode(struct Node **head, int key) {
    if (head == NULL || *head == NULL) return;

    struct Node *temp = *head;
    struct Node *prev = NULL;

    // If head node itself holds the key to be deleted
    if (temp != NULL && temp->data == key) {
        *head = temp->next; // Changed head
        free(temp);         // free old head
        return;
    }

    // Search for the key to be deleted, keep track of the
previous node
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // If key was not present in linked list
    if (temp == NULL) return;

    // Unlink the node from linked list
    prev->next = temp->next;

    free(temp);
}
```
~~~

| **Model 2** | Codestral 25.01 |
|---|---|

**Output for model 2**

~~~
Certainly! Below is the implementation of the `deleteNode`
function to delete a node with a specific value from a singly
linked list in C:

```c
struct Node {
  int data;
  struct Node *next;
};
~~~

~~~~
void deleteNode(struct Node** head, int key) {
  // Store the head node
  struct Node* temp = *head, *prev = NULL;

  // If head node itself holds the key to be deleted
  if (temp != NULL && temp->data == key) {
    *head = temp->next; // Change head
    free(temp);         // Free old head
    return;
  }

  // Search for the key to be deleted, keep track of the
previous node as we need to change 'prev->next'
  while (temp != NULL && temp->data != key) {
    prev = temp;
    temp = temp->next;
  }

  // If key was not present in linked list
  if (temp == NULL) return;

  // Unlink the node from linked list
  prev->next = temp->next;

  free(temp); // Free memory
}
```

### Explanation:
1. **Handle the Head Node**:
   - If the node to be deleted is the head node, update the
head to the next node and free the old head node.

2. **Traverse the List**:
   - Use a `temp` pointer to traverse the list and a `prev`
pointer to keep track of the previous node.
   - Traverse the list until you find the node with the key or
reach the end of the list.

3. **Delete the Node**:
   - If the key is found, update the `next` pointer of the
previous node to skip the current node.
   - Free the memory of the node to be deleted.

4. **Edge Cases**:
   - If the key is not found in the list, the function simply
returns without making any changes.

This implementation ensures that the node with the specified
value is correctly removed from the linked list.
~~~~

### Prompting Strategy 2: Chain of Thought

| | |
|---|---|
| **Task** | Linked list node deletion in C |

**Prompt**

~~~
Given the following, implement node deletion by value in C:

```c
struct Node {
  int data ;
  struct Node * next ;
};

void deleteNode ( struct Node ** head , int key ) {
  // TODO : Implement node deletion
}
```

Think step-by-step. Do not return more functions than was
originally provided.
~~~

| | |
|---|---|
| **Model 1** | gpt-4.1 mini |

**Output for model 1**

~~~
To delete a node by value (`key`) from a singly linked list in
C, you need to consider the following steps carefully:

1. **Check if the list is empty**. If `*head` is `NULL`,
there's nothing to delete.
2. **Check if the head node contains the key**. If yes, remove
it and update the head pointer.
3. **Traverse the list to find the node to delete**. Keep track
of the previous node, so you can update its `next` pointer when
you find the node.
4. **If the node is found**, adjust the previous node's `next`
pointer to skip the deleted node.
5. **Free the memory** of the deleted node.
6. If the node is not found, do nothing.

Here is the implementation:

```c
void deleteNode(struct Node **head, int key) {
    if (head == NULL || *head == NULL) {
        // List is empty, nothing to delete
        return;
    }

    struct Node *temp = *head;
    struct Node *prev = NULL;
~~~

~~~
    // If head node itself holds the key
    if (temp->data == key) {
        *head = temp->next;  // Change head
        free(temp);          // Free old head
        return;
    }

    // Search for the key to be deleted, keep track of previous node
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // If key was not present in linked list
    if (temp == NULL) {
        return;
    }

    // Unlink the node from linked list
    prev->next = temp->next;

    free(temp);  // Free memory
}
```

This function deletes the first occurrence of `key` in the linked list. If the key is not present, it does nothing.
~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |
| **Output for model 2** | |

```
Sure, let's implement the `deleteNode` function step-by-step. The goal is to delete a node with a specific value from a linked list.

### Steps to Implement `deleteNode`

1. **Handle Empty List**: If the list is empty (i.e., `head` is `NULL`), there is nothing to delete.
2. **Delete Head Node**: If the node to be deleted is the head node, update the head to point to the next node.
3. **Traverse the List**: Traverse the list to find the node to be deleted. Keep track of the previous node to update its `next` pointer.
4. **Update Links**: Once the node to be deleted is found, update the `next` pointer of the previous node to skip the node to be deleted.
5. **Free Memory**: Free the memory of the node to be deleted to avoid memory leaks.

Here is the implementation of the `deleteNode` function:
```

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
  int data;
  struct Node *next;
};

void deleteNode(struct Node **head, int key) {
  // Check if the list is empty
  if (*head == NULL) {
    printf("The list is empty.\n");
    return;
  }

  // Check if the node to be deleted is the head node
  struct Node *temp = *head;
  if (temp->data == key) {
    *head = temp->next; // Update head to the next node
    free(temp);          // Free the old head node
    return;
  }

  // Traverse the list to find the node to be deleted
  struct Node *prev = NULL;
  while (temp != NULL && temp->data != key) {
    prev = temp;
    temp = temp->next;
  }

  // If the key was not found in the list
  if (temp == NULL) {
    printf("The key %d is not present in the list.\n", key);
    return;
  }

  // Update the next pointer of the previous node
  prev->next = temp->next;

  // Free the node to be deleted
  free(temp);
}

// Function to print the list (for testing purposes)
void printList(struct Node *node) {
  while (node != NULL) {
    printf("%d -> ", node->data);
    node = node->next;
  }
  printf("NULL\n");
}
```

```
// Helper function to create a new node (for testing purposes)
struct Node* createNode(int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}

int main() {
  // Create a sample list: 1 -> 2 -> 3 -> 4
  struct Node *head = createNode(1);
  head->next = createNode(2);
  head->next->next = createNode(3);
  head->next->next->next = createNode(4);

  printf("Original list: ");
  printList(head);

  // Delete node with key 3
  deleteNode(&head, 3);
  printf("After deleting 3: ");
  printList(head);

  // Delete head node with key 1
  deleteNode(&head, 1);
  printf("After deleting 1: ");
  printList(head);

  // Delete a non-existing node with key 5
  deleteNode(&head, 5);
  printf("After trying to delete 5: ");
  printList(head);

  // Free the remaining nodes
  while (head != NULL
```

**Analysis of Models**

BLEU scores:

- Zero shot: 37.96
- Chain of thought: 26.98

Code BLEU scores:

- Zero Shot: 54.5
- Chain of Thought: 36.71

The implementations of deleteNode were very similar across models. However, the code BLEU scores do not reflect that. I think this is mainly due to the differences in comments in the code. While the code wasn't that different, the responses from the model Codestral 25.01 are wordier.

For example, the response from Codestral using the chain of thought strategy had extra functions even though the prompt stated not to include any extra functions.

**Analysis of Prompting Strategies**

The implementations of deleteNode were very similar across prompting strategies as well. If anything, the chain of thought outputs seemed too wordy as the steps outlined by the models are also reflected in the comments in the code.

## Task 14: Recursive Function Completion (Python)

### Prompting Strategy 1: Zero Shot

| **Task** | Recursive Function Completion (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Complete the recursive Python function to return the n-th
Fibonacci number:

```python
def fibonacci ( n ) :
  # TODO : Base cases and recursive call
  pass
```

Return only the completed function.
~~~

**Output**

~~~
```python
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```
~~~

~~~
```python
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```
~~~

### Prompting Strategy 2: Few Shot

| **Task** | Recursive Function Completion (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Example 1:
Input: 1
Output: 0

Example 2:
Input: 5
Output: 3

Example 3:
Input: 9
Output: 21

Complete the recursive Python function to return the n-th Fibonacci number:

```python
def fibonacci ( n ) :
  # TODO : Base cases and recursive call
  pass
```

Return only the completed function.
~~~

**Output**

~~~
```python
def fibonacci(n):
    # Base cases
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # Recursive call
    return fibonacci(n - 1) + fibonacci(n - 2)
```
~~~

~~~
Here is the completed recursive Python function to return the n-th Fibonacci number:

```python
def fibonacci(n):
    # Base cases
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive call
        return fibonacci(n - 1) + fibonacci(n - 2)
```

This function uses recursion to compute the n-th Fibonacci number. The base cases are set for `n == 0` and `n ==
~~~

| | | |
|---|---|---|
| | | 1`, which return 0 and 1 respectively. For other values of `n`, the function calls itself with `n - 1` and `n - 2` and adds the results. |

**Analysis of Models**

BLEU scores:

- Zero shot: 100
- Few shot: 17.78

Code BLEU scores:

- Zero shot: 75 (no data flows)
- Few shot: 70.43 (no data flows)

The code implementation across models is very similar. However, Codestral's response with few shot is wordy and ignores the instruction in the prompt to return only the function.

**Analysis of Prompting Strategies**

The few shot attempt doesn't seem to have taken the examples into account. In the examples, 0 is considered the first number in the fibonacci sequence whereas in the code, 1 is. This could be because the standard implementation of fibonacci is the version that the models returned and the models have been trained on so many of that version.

## Task 15: Constructor Completion (Python)

### Prompting Strategy 1: Zero Shot

| **Task** | Constructor Completion (Python) | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |
| **Prompt** | | |

~~~~
Complete the following class constructor:

```python
class Person :
  def __init__ ( self ) :
    # TODO : Add name , age , and optional email
    pass

Reply with only the completed code.
~~~~

**Output**

~~~
```python
class Person:
    def __init__(self, name, age, email=None):
        self.name = name
        self.age = age
        self.email = email
```
~~~

~~~
```python
class Person:
  def __init__(self, name, age, email=None):
    self.name = name
    self.age = age
    self.email = email
```
~~~

### Prompting Strategy 2: Few Shot

| | | |
|---|---|---|
| **Task** | Constructor Completion (Python) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Example 1:
```python
class Book:
    def __init__(self, title, author, year):
        self.title = title
        self.author = author
        self.year = year
```

Example 2:
```python
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
```

Example 3:
```python
class Rectangle:
    def __init__(self, width=1, height=1):
        self.width = width
        self.height = height
```

Complete the following class constructor:
~~~

~~~
```python
class Person :
  def __init__ ( self ) :
    # TODO : Add name , age , and optional email
    pass

Reply with only the completed code.
~~~

**Output**

~~~
```python
class Person:
    def __init__(self, name, age, email=None):
        self.name = name
        self.age = age
        self.email = email
```
~~~

~~~
```python
class Person :
  def __init__ ( self , name , age , email=None) :
    self.name = name
    self.age = age
    self.email = email
```
~~~

**Analysis of Models and Prompting Strategies**

Code BLEU scores (across models):

- Zero shot: 100
- Few shot: 68.37

There are either no or barely noticeable differences across all four runs. No approach was better than the others.

## Task 16: Binary Search Completion (Java)

### Prompting Strategy 1: Zero Shot

| **Task** | Binary Search Completion (Java) |
|---|---|

**Prompt**

~~~
Complete the binary search Java implementation:

```java
public int binarySearch ( int [] arr , int target ) {
  int left = 0 , right = arr . length - 1;
  while ( left <= right ) {
    int mid = ( left + right ) / 2;
    // TODO : Compare and adjust bounds
  }
  return -1;
~~~

~~~
}
```
~~~

| | |
|---|---|
| **Model 1** | gpt-4o mini |

**Output for model 1**

~~~
Sure! Below is the completed implementation of the binary
search algorithm in Java, including the comparison and
adjustment of bounds.

```java
public int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;

        // Compare the middle element with the target
        if (arr[mid] == target) {
            return mid;  // Target found, return the index
        } else if (arr[mid] < target) {
            left = mid + 1;  // Target is in the right half
        } else {
            right = mid - 1; // Target is in the left half
        }
    }

    return -1; // Target not found
}
```

### Explanation:
1. **Initialization**: The `left` pointer is set to the start
of the array, and the `right` pointer is set to the end of the
array.
2. **Loop**: As long as `left` is less than or equal to
`right`, the loop continues.
3. **Mid Calculation**: The middle index `mid` is calculated.
4. **Comparison**:
   - If the middle element (`arr[mid]`) is equal to the
`target`, the index `mid` is returned.
   - If the middle element is less than the `target`, it means
the target must be in the right half of the array, so we adjust
`left` to `mid + 1`.
   - If the middle element is greater than the `target`, it
means the target must be in the left half of the array, so we
adjust `right` to `mid - 1`.
5. **Completion**: If the loop exits without finding the
target, `-1` is returned to indicate that the target is not
present in the array.

This implementation assumes that the input array `arr` is
sorted in ascending order, which is a requirement for binary
search to work correctly.
~~~

| **Model 2** | Codestral 25.01 |
|---|---|

**Output for model 2**

~~~
To complete the binary search implementation, you need to
compare the middle element of the array with the target and
adjust the bounds (`left` and `right`) accordingly. If the
middle element is equal to the target, you return the middle
index. If the target is less than the middle element, you
adjust the right bound; otherwise, you adjust the left bound.
Here's the completed implementation:

```java
public int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;  // To avoid
potential overflow

        // Check if the target is present at mid
        if (arr[mid] == target) {
            return mid;
        }

        // If target is greater, ignore the left half
        if (arr[mid] < target) {
            left = mid + 1;
        }
        // If target is smaller, ignore the right half
        else {
            right = mid - 1;
        }
    }

    // Target was not found in the array
    return -1;
}
```

### Explanation:
1. **Initialization**: `left` is set to the start of the array,
and `right` is set to the end of the array.
2. **Mid Calculation**: `int mid = left + (right - left) / 2;`
is used to calculate the middle index to avoid potential
overflow issues that can occur with `(left + right) / 2` when
`left` and `right` are large.
3. **Comparison**:
   - If `arr[mid]` is equal to `target`, the function returns
`mid`.
   - If `target` is greater than `arr[mid]`, it means the
target must be in the right half of the array, so `left` is
updated to `mid + 1`.
   - If `target` is less than `arr[mid]`, it means the target
must be in the left half of the array, so `right` is updated to
~~~

~~~~
`mid - 1`.
4. **Termination**: If `left` exceeds `right`, it means the
target is not in the array, and the function returns `-1`.

This implementation ensures that the binary search algorithm
efficiently finds the target in a sorted array.
~~~~

**Prompting Strategy 2: Chain of Thought**

| | |
|---|---|
| **Task** | Binary Search Completion (Java) |

**Prompt**

~~~~
Complete the binary search Java implementation. Think
step-by-step:

```java
public int binarySearch ( int [] arr , int target ) {
  int left = 0 , right = arr . length - 1;
  while ( left <= right ) {
    int mid = ( left + right ) / 2;
    // TODO : Compare and adjust bounds
  }
  return -1;
}
```
~~~~

| | |
|---|---|
| **Model 1** | gpt-4o mini |

**Output for model 1**

~~~~
Sure! Let's complete the binary search implementation
step-by-step. In a binary search, you need to compare the
target value with the middle element of the current search
range (defined by `left` and `right`). Depending on the
comparison result, you will either find the target, or you will
adjust the bounds to continue the search.

Here’s the full implementation:

```java
public int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;

    while (left <= right) {
        int mid = (left + right) / 2;

        // Compare the middle element with the target
        if (arr[mid] == target) {
            return mid; // Target found, return the index
        } else if (arr[mid] < target) {
~~~~

~~~
            left = mid + 1; // Target is in the right half
        } else {
            right = mid - 1; // Target is in the left half
        }
    }

    return -1; // Target not found
}
```

### Explanation of the Code:

1. **Initialization**:
   - `left` is initialized to the start of the array (index 0).
   - `right` is initialized to the end of the array (index
`arr.length - 1`).

2. **While Loop**:
   - The loop continues as long as `left` is less than or equal
to `right`, which means there are still elements in the search
range.

3. **Finding the Middle Index**:
   - The middle index `mid` is calculated by averaging `left`
and `right`.

4. **Comparison**:
   - If the element at the middle index (`arr[mid]`) is equal
to the target, we return `mid` as the index where the target is
found.
   - If `arr[mid]` is less than the target, it means the target
must be in the right half of the array (so we adjust `left` to
`mid + 1`).
   - If `arr[mid]` is greater than the target, it means the
target must be in the left half of the array (so we adjust
`right` to `mid - 1`).

5. **Return Statement**:
   - If the loop ends without finding the target, we return
`-1` to indicate that the target is not present in the array.

This complete code will work correctly for a sorted array.
~~~

| **Model 2** | Codestral 25.01 |
|---|---|

**Output for model 2**

````
To complete the binary search implementation, we need to add
the logic to compare the middle element with the target and
adjust the bounds of the search accordingly.

Here's the step-by-step completion:

1. **Initialize the bounds**: We start with `left` at the
beginning of the array and `right` at the end.
````

2. **Calculate the middle index**: `mid` is calculated as the average of `left` and `right`.
3. **Compare the middle element with the target**:
   - If the middle element is equal to the target, return the `mid` index.
   - If the middle element is less than the target, move the `left` bound to `mid + 1` (since the target must be in the right half).
   - If the middle element is greater than the target, move the `right` bound to `mid - 1` (since the target must be in the left half).
4. **Repeat until the bounds cross each other**: The loop continues until `left` is greater than `right`.
5. **Return -1 if the target is not found**: If the bounds cross each other and the target is not found, return `-1`.

Here's the complete implementation:

```java
public int binarySearch(int[] arr, int target) {
  int left = 0, right = arr.length - 1;

  while (left <= right) {
    int mid = left + (right - left) / 2; // This avoids potential overflow

    // Check if the target is present at mid
    if (arr[mid] == target) {
      return mid;
    }

    // If the target is greater, ignore the left half
    if (arr[mid] < target) {
      left = mid + 1;
    }
    // If the target is smaller, ignore the right half
    else {
      right = mid - 1;
    }
  }

  // Target was not found in the array
  return -1;
}
```

### Explanation of the Code:
- **Initialization**: `left` starts at the beginning of the array, and `right` starts at the end.
- **Mid Calculation**: `int mid = left + (right - left) / 2;` avoids overflow that can occur with `(left + right) / 2` for large arrays.
- **Comparison and Adjustment**:

```
  - If `arr[mid] == target`, the target is found, and `mid` is
returned.
  - If `arr[mid] < target`, it means the target is in the right
half, so `left` is updated to `mid + 1`.
  - If `arr[mid] > target`, it means the target is in the left
half, so `right` is updated to `mid - 1`.
- **Loop Termination**: The loop continues until `left` is
greater than `right`.
- **Return -1**: If the loop terminates and the target is not
found, `-1` is returned.

This implementation ensures that the binary search runs in
\(O(\log n)\) time, making it efficient for large datasets.
```

**Analysis of Models**
BLEU scores:

- Zero shot: 51.2
- Few shot: 39.36

Code BLEU scores:

- Zero shot: 67.05
- Chain of thought: 66.43

The models respond with implementations of binary search that follow the same logic. However, Codestral 25.01 has taken the extra step to make sure there are no overflows with large arrays.

**Analysis of Prompting Strategies**
The responses are largely similar across the two prompting strategies. The models will explain the code step-by-step whether it is prompted to or not. However, the prompting to "think step-by-step," can lead to repetitiveness in responses as seen in Codestral 25.01's response. Therefore, I think the better strategy here is zero shot.

## Task 17: Self-Consistency Bug Fixing (C++)

### Prompting Strategy 1: Zero Shot

| | | |
|---|---|---|
| **Task** | Self consistency bug fixing (C++) | |
| **Model** | gpt-4o mini | Codestral 25.01 |
| **Prompt** | | |

~~~
Resolve the inconsistency between the function name and logic:

```cpp
// Supposed to return true if x is even
~~~

~~~~
bool isOdd (int x ) {
  return x % 2 == 0;
}
```


Only include the solution
~~~~

**Output**

~~~~cpp
```cpp
// Corrected function to return true if x is odd
bool isOdd(int x) {
  return x % 2 != 0;
}
```
~~~~

~~~~cpp
```cpp
// Supposed to return true if x is even
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~~

**Prompting Strategy 2: Self Consistency**

| | | |
|---|---|---|
| **Task** | Self consistency bug fixing (C++) | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~~
Resolve the inconsistency between the function name and logic:


```cpp
// Supposed to return true if x is even
bool isOdd (int x ) {
  return x % 2 == 0;
}
```


Only include the solution
~~~~

**Output Attempt 1**

~~~~cpp
```cpp
// Corrected function to return true if x is odd
bool isOdd(int x) {
  return x % 2 != 0;
}
```
~~~~

~~~~cpp
```cpp
// Supposed to return true if x is even
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~~

**Output Attempt 2**

~~~~cpp
```cpp
// Corrected function name and logic
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~~

~~~~
To resolve the inconsistency between the function name and its logic, you should change the function name to accurately reflect its behavior. Since the function currently returns `true` if
~~~~

| | | |
|---|---|---|
| | | `x` is even, but its name suggests it returns `true` if `x` is odd, you should rename the function to `isEven`.<br>Here is the corrected code: |

~~~
```cpp
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~

| | | |
|---|---|---|
| **Output Attempt 3** | | To resolve the inconsistency between the function name and its logic, you should change the function name to reflect the intended functionality. Since the logic checks if `x` is even, the function name should be `isEven`. Here is the corrected code: |

~~~
```cpp
// Corrected function name to reflect the logic
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~

~~~
```cpp
// Returns true if x is even
bool isEven(int x) {
  return x % 2 == 0;
}
```
~~~

**Analysis of Models and Prompting Strategies**

BLEU scores (across models):

- Output attempt 1: 66.88
- Output attempt 2: 8.92
- Output attempt 3: 18.25

Code BLEU scores (across models, disregarding comment at the top):

- Output attempt 1: 78.02
- Output attempt 2: 100
- Output attempt 3: 100

Running the prompt multiple times definitely helped with performance. For example, in the first attempt gpt-4o mini inferred the intent of the function wrong, but in the following two runs it guessed correctly. Therefore, the better prompting strategy was self consistency.

## Task 18: Prompt Chaining: Bug Identification → Fix (JavaScript)

### Prompting Strategy 1: Zero Shot

| **Task** | Identify and fix bug in JavaScript | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Identify and fix the bug in the following JavaScript function:

```js
function isEven ( n ) {
  return n % 2;
}
```

Do not provide examples on how to test the function.
~~~

**Output**

gpt-4o mini:

~~~
The bug in the function
`isEven` is that it returns the
result of `n % 2`, which will
be `0` for even numbers and `1`
for odd numbers. However, to
properly indicate whether a
number is even, the function
should return `true` for even
numbers and `false` for odd
numbers.

Here is the corrected version
of the function:

```js
function isEven(n) {
  return n % 2 === 0;
}
```

This way, the function
correctly returns `true` for
even numbers and `false` for
odd numbers.
~~~

Codestral 25.01:

~~~
The bug in the `isEven`
function is that it does not
correctly determine whether a
number is even or odd. The
expression `n % 2` returns
`0` for even numbers and a
non-zero value for odd
numbers. However, the
function should return `true`
for even numbers and `false`
for odd numbers.

To fix the bug, you should
compare the result of `n % 2`
to `0`. Here is the corrected
function:

```js
function isEven(n) {
  return n % 2 === 0;
}
```

This ensures that the
function returns `true` if
`n` is even and `false` if
`n` is odd.
~~~

### Prompting Strategy 2: Prompt Chaining

| **Task** | Identify and fix bug in JavaScript | |
|---|---|---|
| **Model** | gpt-4.1 mini | Codestral 25.01 |

**Prompt**

Step 1 Prompt:

~~~~
Identify the bug in the following JavaScript function in at
most two sentences:

```js
function isEven ( n ) {
  return n % 2;
}
```
~~~~

Step 2 Prompt:

~~~~
Fix the above JavaScript function. Do not provide examples on
how to test the function.
~~~~

**Output**

Result 1:

~~~~
The function returns the
remainder of `n % 2`, which is
`0` for even numbers and `1`
for odd numbers, but since `0`
is falsy and `1` is truthy in
JavaScript, it incorrectly
treats even numbers as `false`.
To fix it, the function should
return `n % 2 === 0` to
explicitly return a boolean
indicating evenness.
~~~~

Result 2:

~~~~
```js
function isEven(n) {
  return n % 2 === 0;
}
```
~~~~

Result 1:

~~~~
The function `isEven` returns
`n % 2`, which is `0` for
even numbers and a non-zero
value for odd numbers. This
is the opposite of what the
function name suggests. The
function should return `true`
for even numbers and `false`
for odd numbers.
~~~~

Result 2:

~~~~
```js
function isEven(n) {
  return n % 2 === 0;
}
```
~~~~

**Analysis of Models and Prompting Strategies**

BLEU scores:

- Zero shot (between gpt-4o mini and Codestral 25.01): 54.8
- Prompt Chaining (between gpt-4.1 mini and Codestral 25.01)
  - Result 1: 28.76

Code BLEU scores:

- Zero shot (between gpt-4o mini and Codestral 25.01): 100
- Prompt Chaining (between gpt-4.1 mini and Codestral 25.01)
  - Result 2: 100

All the runs performed the task correctly. There are no significant differences between them.

## Task 19: Summary Decomposition (C++)

### Prompting Strategy 1: Zero Shot

| **Task** | Decompose a high-level code summary | |
|---|---|---|
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Decompose the high-level comment/summary into logical steps:

```cpp
// Function that validates an input , calculates square , and
returns result
int process (int x ) {
  if ( x < 0) return -1;
    return x * x ;
}
```
~~~

**Output**

gpt-4o mini

```
To decompose the high-level
comment/summary into logical
steps based on the provided C++
function, we can break it down
as follows:

1. **Input Validation**:
   - Check if the input value
`x` is less than 0.
   - If `x` is less than 0,
return -1 to indicate invalid
input.

2. **Calculation**:
   - If the input is valid
(i.e., `x` is 0 or greater),
calculate the square of `x` by
multiplying `x` by itself.

3. **Return Result**:
   - Return the calculated
square of `x` as the output of
the function.

To summarize, the steps are:
```

Codestral 25.01

```
To decompose the high-level
comment/summary into logical
steps for the given C++
function, we can break it
down as follows:

1. **Input Validation:**
   - Check if the input `x`
is less than 0.
   - If `x` is less than 0,
return -1 to indicate an
invalid input.

2. **Calculation:**
   - If the input `x` is
valid (i.e., not less than
0), calculate the square of
`x`.

3. **Return Result:**
   - Return the calculated
square of `x`.

Here is the function with
these steps explicitly
```

| | | |
|---|---|---|
| | 1. Check if `x` is negative:<br>   - If yes, return -1.<br>2. If `x` is non-negative, compute `x * x`.<br>3. Return the result of the calculation. | outlined: |

~~~
```cpp
int process(int x) {
    // Step 1: Input Validation
    if (x < 0) {
        return -1; // Invalid input, return -1
    }

    // Step 2: Calculation
    int result = x * x; // Calculate the square of x

    // Step 3: Return Result
    return result; // Return the calculated square
}
```
~~~

This structure ensures that the function first checks for valid input, performs the necessary calculation if the input is valid, and then returns the result.

**Prompting Strategy 2: Chain of Thought**

| | | |
|---|---|---|
| **Task** | Decompose a high-level code summary | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Decompose the high-level comment/summary into logical steps.
Think step-by-step:

```cpp
// Function that validates an input , calculates square , and returns result
int process (int x ) {
  if ( x < 0) return -1;
    return x * x ;
}
```
~~~

| | | |
|---|---|---|
| **Output** | To decompose the high-level comment/summary into logical | To break down the high-level comment/summary of the |

| | | |
|---|---|---|
| | steps, we can break it down as follows:<br><br>1. **Function Definition**: Define a function named `process` that takes a single integer parameter `x`.<br><br>2. **Input Validation**:<br>   - Check if the input `x` is less than 0.<br>   - If `x` is less than 0, return -1. This indicates that the input is invalid.<br><br>3. **Calculate Square**:<br>   - If the input `x` is valid (i.e., non-negative), calculate the square of `x` by multiplying `x` by itself.<br><br>4. **Return Result**:<br>   - Return the calculated square of `x` as the result of the function.<br><br>In summary, the steps outline the process of validating the input, computing the square if valid, and returning the appropriate result based on the input value. | `process` function into logical steps, we need to follow the steps explicitly mentioned in the comment:<br><br>1. **Validate the Input:**<br>   - Check if the input `x` is less than 0.<br>   - If `x` is less than 0, return `-1`.<br><br>2. **Calculate the Square:**<br>   - If `x` is not less than 0, calculate the square of `x`.<br><br>3. **Return the Result:**<br>   - Return the calculated square of `x`.<br><br>Here's the step-by-step breakdown of the code: |

```cpp
int process(int x) {
  // Step 1: Validate the input
  if (x < 0) {
    return -1; // Return -1 if x is less than 0
  }

  // Step 2: Calculate the square
  int result = x * x;

  // Step 3: Return the result
  return result;
}
```

Let's go through the steps in more detail:

1. **Input Validation:**
   ```cpp
   if (x < 0) {
     return -1;
   }
   ```
   - This condition checks if `x` is negative.

~~~~
   - If `x` is negative, the
function immediately returns
`-1`, ending further
execution.

2. **Square Calculation:**
   ```cpp
   int result = x * x;
   ```
   - If the input `x` is not
negative, the function
proceeds to calculate the
square of `x`.
   - The result of `x * x` is
stored in the variable
`result`.

3. **Return the Result:**
   ```cpp
   return result;
   ```
   - Finally, the function
returns the calculated square
of `x`.

By following these steps, the
function `process` validates
the input, calculates the
square if the input is valid,
and returns the appropriate
result.
~~~~

**Analysis of Models**

BLEU scores:

- Zero shot: 42.19
- Chain of thought: 15.64

The responses from Codestral 25.01 are generally longer. The responses from the gpt-4o mini model only breaks down the comment, whereas the responses from Codestral also puts the broken down comments back into the function.

**Analysis of Prompting Strategies**

Using a chain of thought generally leads to longer responses. For this task, the extra length leads to repetitiveness rather than helpful details. Therefore, the better strategy here is zero shot.

## Task 20: Purpose Inference → Completion (Python)

### Prompting Strategy 1: Zero Shot

| | | |
|---|---|---|
| **Task** | Completed a Python function based on inferred intent | |
| **Model** | gpt-4o mini | Codestral 25.01 |

**Prompt**

~~~
Complete the function based on intent:

```python
def calculate_average ( scores ) :
  total = 0
  # TODO : Complete to return average
  pass
```

Respond with only the completed function.
~~~

**Output**

gpt-4o mini

~~~
```python
def calculate_average(scores):
    total = sum(scores)
    return total / len(scores) if scores else 0
```
~~~

Codestral 25.01

~~~
```python
def calculate_average(scores):
    total = 0
    for score in scores:
        total += score
    return total / len(scores)
```
~~~

### Prompting Strategy 2: Prompt Chaining

| | | |
|---|---|---|
| **Task** | Infer the intent of a partially completed Python function and then complete it | |
| **Model** | gpt-4.1 mini | Codestral 25.01 |

**Prompt**

Step 1 Prompt:

~~~
Identify the intended purpose of the following function in one sentence:

```python
def calculate_average ( scores ) :
  total = 0
  # TODO : Complete to return average
  pass
```
~~~

Step 2 Prompt:

```
Complete the function based on the intent identified above.
Respond with only the completed function.
```

**Output**

Result 1:

```
The function is intended to calculate and return the average of the numerical values in the `scores` list.
```

Result 2:

~~~
```python
def calculate_average(scores):
    total = sum(scores)
    return total / len(scores) if scores else 0
```
~~~

Result 1:

```
The intended purpose of the function is to calculate and return the average of a list of scores.
```

Result 2:

~~~
```python
def calculate_average ( scores ) :
  total = 0
  for score in scores:
    total += score
  return total / len(scores)
```
~~~

**Analysis of Models**

Code BLEU scores:

- Zero-shot (b/w gpt-4o mini and Codestral 25.01): 39.09
- Prompt chaining (b/w gpt-4.1 mini and Codestral 25.01): 31.12

The code implementation by the gpt models is better since it handles the case where scores is an empty list. On the other hand, the implementation by Codestral 25.01 will cause a ZeroDivisionError when the input is an empty list.

**Analysis of Prompting Strategies**

There is no difference in the implementation of the function calculate_average across strategies.

## Task 21: Full-File Bug Detection and Refactoring (Python)

### Prompting Strategy 1: Zero Shot

| | |
|---|---|
| **Task** | Identify the bugs in a full Python script and refactor |

**Prompt**

~~~
Analyze the following utility script:

```python
# utils .py - Script to parse and summarize numeric CSV files

import csv
~~~

~~~~
def read_csv ( filepath ) :
  with open ( filepath , ’r’) as f :
    return [ row for row in csv . reader ( f ) ]

def summarize_column ( data , index ) :
  values = [ float ( row [ index ]) for row in data [1:]] #
skip header
  total = sum( values )
  avg = total / len ( values )
  return total , avg

def main () :
  filepath = ’data .csv ’
  data = read_csv ( filepath )
  total , avg = summarize_column ( data , 1)
  print (" Total :", total )
  print (" Average :", avg )

if __name__ == ’__main__ ’:
  main ()
```

Identify any potential logic issues or design flaws. Then,
refactor it for better readability, correctness, and safety.
~~~~

| | |
|---|---|
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |

**Output for model 1**

~~~~
Let's analyze the provided script first, then identify issues,
and finally refactor it.

---

## Original Script Analysis

```python
# utils .py - Script to parse and summarize numeric CSV files

import csv

def read_csv ( filepath ) :
~~~~

~~~~
  with open ( filepath , ’r’)as f :
    return [ row for row in csv . reader ( f ) ]

def summarize_column ( data , index ) :
  values = [ float ( row [ index ]) for row in data [1:]] # skip header
  total = sum( values )
  avg = total / len ( values )
  return total , avg

def main () :
  filepath = ’data .csv ’
  data = read_csv ( filepath )
  total , avg = summarize_column ( data , 1)
  print (" Total :", total )
  print (" Average :", avg )

if __name__ == ’__main__ ’:
  main ()
```
~~~~

### Issues and potential improvements

1. **Syntax/Formatting Issues:**

   - The code uses non-standard whitespace: spaces inside parentheses, e.g. `read_csv ( filepath )` instead of `read_csv(filepath)`.
   - The script has inconsistent spacing and indentation style.
   - Quotes are inconsistent and sometimes incorrect:
     - The script uses `’` (right single quotation mark) instead of `'` (ASCII single quote).
     - This will cause syntax errors.
   - The filename `'data .csv '` has extra spaces.
   - The `if __name__ == '__main__':` line has extra spaces and wrong quote characters.

2. **Error Handling:**

   - No error handling when opening the file or parsing floats.
   - If the CSV file is missing, or if non-numeric data appears in the target column, the script will crash.

3. **Assumptions:**

   - Assumes the first row is a header; skips it.
   - Assumes the specified column index exists.
   - Assumes all rows have enough columns.
   - Assumes all values in the column are convertible to float.

4. **Design Flaws:**

   - `read_csv` returns a list of rows without any structure.

~~~
   - No flexibility in choosing which column to summarize —
hardcoded to index 1.
   - The script could use `csv.DictReader` to improve
readability and safety by using column names.
   - No command-line arguments support — file path and column
index are hardcoded.
   - No documentation or docstrings.

5. **Miscellaneous:**

   - The script name `utils.py` is generic and might cause
import conflicts.
   - The script prints results but does not return values or
allow reuse of functions.

---

## Refactored Version

Below is a refactored version that addresses these issues:

- Fixes syntax and formatting.
- Adds error handling.
- Uses `csv.DictReader` for clarity.
- Supports command-line arguments for filename and column
name/index.
- Adds docstrings and comments.
- Uses logging for better error reporting (simple print
statements used here for brevity).
- Uses a main guard properly.
- Improved readability and PEP8 compliance.

```python
import csv
import sys

def read_csv(filepath):
    """
    Reads a CSV file and returns a list of rows as
dictionaries.

    Args:
        filepath (str): Path to the CSV file.

    Returns:
        list of dict: List of rows with column names as keys.

    Raises:
        FileNotFoundError: If file does not exist.
        csv.Error: If CSV parsing error occurs.
    """
    with open(filepath, 'r', newline='') as f:
        reader = csv.DictReader(f)
~~~

```
        return list(reader)

def summarize_column(data, column):
    """
    Summarizes a numeric column from CSV data.

    Args:
        data (list of dict): CSV data rows.
        column (str): Column name to summarize.

    Returns:
        tuple: (total, average) of the column values.

    Raises:
        ValueError: If column values are not convertible to
float or column missing.
    """
    values = []
    for i, row in enumerate(data, start=1):
        try:
            value = float(row[column])
            values.append(value)
        except KeyError:
            raise KeyError(f"Column '{column}' not found in row
{i}.")
        except ValueError:
            raise ValueError(f"Non-numeric value in column
'{column}' at row {i}: {row[column]}")

    if not values:
        raise ValueError(f"No data found in column
'{column}'.")

    total = sum(values)
    average = total / len(values)
    return total, average

def main():
    if len(sys.argv) != 3:
        print("Usage: python utils.py <csv_file>
<column_name>")
        sys.exit(1)

    filepath = sys.argv[1]
    column =
```

| | |
|---|---|
| **Model 2** | Codestral 25.01 |
| **Output for model 2** | Let's analyze the utility script for potential issues and design flaws:<br><br>### Potential Logic Issues and Design Flaws: |

~~~
1. **Character Encoding Issues**:
   - The script uses non-standard single quotes (`’`) instead
of standard single quotes (`'`). This will cause a
`SyntaxError`.

2. **File Handling**:
   - The script does not handle exceptions that may arise from
file operations, such as file not found or permission issues.

3. **Data Validation**:
   - There is no validation to ensure that the file is a valid
CSV and that the specified column index exists in the data.

4. **Type Errors**:
   - The script does not handle cases where the values in the
specified column cannot be converted to floats.

5. **Magic Strings**:
   - The filename `data.csv` is hardcoded, which reduces
flexibility and testability.

6. **Lack of Documentation**:
   - There are no docstrings or comments explaining the purpose
of the functions.

7. **Column Indexing**:
   - The script assumes that the column index is valid and that
the data is numeric. There is no validation for these
assumptions.

### Refactored Script:

```python
# utils.py - Script to parse and summarize numeric CSV files

import csv
import sys

def read_csv(filepath):
    """
    Reads a CSV file and returns its content as a list of rows.

    :param filepath: Path to the CSV file.
    :return: List of rows from the CSV file.
    """
    try:
        with open(filepath, 'r') as f:
            return list(csv.reader(f))
    except FileNotFoundError:
        print(f"Error: The file {filepath} was not found.")
        sys.exit(1)
    except Exception as e:
        print(f"Error reading file: {e}")
~~~

```
        sys.exit(1)

def summarize_column(data, index):
    """
    Summarizes a column in the CSV data by calculating the total and average of its numeric values.

    :param data: List of rows from the CSV file.
    :param index: Index of the column to summarize.
    :return: Tuple containing the total and average of the column values.
    """
    if not data:
        return 0, 0

    try:
        values = [float(row[index]) for row in data[1:]]  # skip header
    except (ValueError, IndexError) as e:
        print(f"Error processing column {index}: {e}")
        sys.exit(1)

    total = sum(values)
    avg = total / len(values) if values else 0
    return total, avg

def main():
    """
    Main function to execute the script.
    """
    filepath = 'data.csv'
    data = read_csv(filepath)

    if len(data) < 2:
        print("Error: The CSV file is too short to contain data rows.")
        sys.exit(1)

    try:
        total, avg = summarize_column(data, 1)
        print("Total:", total)
        print("Average:", avg)
    except Exception as e:
        print(f"Error summarizing column: {e}")
        sys.exit(1)

if __name__ == '__main__':
    main()
```

### Improvements Made:

1. **Character Encoding Issues**:

```
   - Replaced non-standard single quotes with standard single
quotes.

2. **File Handling**:
   - Added exception handling for file operations.

3. **Data Validation**:
   - Added checks to ensure the file is a valid CSV and that
the column index exists.

4. **Type Errors**:
   - Added exception handling for converting column values to
floats.

5. **Magic Strings**:
   - The filename `data.csv` is now a variable, making it
easier to change for different files.

6. **Documentation**:
   - Added docstrings to explain the purpose of each function.

7. **Column Indexing**:
   - Added validation to ensure the column index is valid and
the data is numeric.

8. **Error Messages**:
   - Improved error messages to provide more context.

By making these improvements, the script is more robust,
readable, and easier to maintain.
```

**Prompting Strategy 2: Prompt Chaining**

| **Task** | Identify the bugs in a full Python script and refactor |
|---|---|

**Prompt**

Step 1 Prompt:

~~~
Identify any potential logic issues or design flaws in the
following:

```python
# utils .py - Script to parse and summarize numeric CSV files

import csv

def read_csv ( filepath ) :
  with open ( filepath , ’r’) as f :
    return [ row for row in csv . reader ( f ) ]
~~~

~~~
def summarize_column ( data , index ) :
  values = [ float ( row [ index ]) for row in data [1:]] #
skip header
  total = sum( values )
  avg = total / len ( values )
  return total , avg

def main () :
  filepath = ’data .csv ’
  data = read_csv ( filepath )
  total , avg = summarize_column ( data , 1)
  print (" Total :", total )
  print (" Average :", avg )

if __name__ == ’__main__ ’:
  main ()
```

Please format the response as a list. Do no provide solutions.
~~~

Step 2 Prompt:

`Fix the issues identified above`

Step 3 Prompt:

`Refactor the code for better readability, correctness, and safety`

| | |
|---|---|
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |
| **Output for model 1** | Result 1:<br>1. Inconsistent and incorrect use of quotation marks (e.g., use of smart quotes ’ ’ instead of standard single quotes ' ').<br>2. Extra spaces inside function calls and definitions that reduce code readability (e.g., `read_csv ( filepath )` instead of `read_csv(filepath)`).<br>3. Hardcoded filepath with spaces (`’data .csv ’`) which may cause file not found errors.<br>4. No error handling for file operations or CSV parsing. |

5. Assumes all rows have a valid numeric value at the specified index without validation.
6. No handling for empty data or empty columns which may cause division by zero in average calculation.
7. Assumes the CSV file has a header and skips the first row without verifying.
8. No checks for the index being within the range of the columns in rows.
9. The script reads the entire CSV into memory which may be inefficient for large files.
10. Function and variable naming is generic and may not clearly convey their purpose.
11. The script does not support command-line arguments for file input, reducing flexibility.
12. No documentation or comments explaining function behaviors or parameters.
13. The `main` function calls `summarize_column` with a hardcoded column index (`1`) without validation.
14. The print statements include inconsistent spacing in output strings.

Result 2:

```python
# utils.py - Script to parse and summarize numeric CSV files

import csv
import sys

def read_csv(filepath):
    """
    Reads a CSV file and returns its content as a list of rows.
    Each row is a list of strings.
    """
    try:
        with open(filepath, 'r', newline='', encoding='utf-8')
as f:
            return [row for row in csv.reader(f)]
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
        return None
    except Exception as e:
        print(f"Error reading '{filepath}': {e}")
        return None

def summarize_column(data, index):
    """
    Calculates the total and average of numeric values in a
specified column index.
    Skips the header row and rows with invalid numeric data.
    Returns a tuple (total, average) or None if no valid data
is found.
```

```
    """
    if not data or len(data) < 2:
        print("Error: CSV data is empty or missing rows.")
        return None

    if index < 0 or index >= len(data[0]):
        print(f"Error: Column index {index} is out of range.")
        return None

    values = []
    for row_num, row in enumerate(data[1:], start=2):  #
start=2 for user-friendly row number
        if len(row) <= index:
            print(f"Warning: Row {row_num} does not have column
index {index}. Skipping.")
            continue
        try:
            value = float(row[index])
            values.append(value)
        except ValueError:
            print(f"Warning: Non-numeric value '{row[index]}'
at row {row_num}, column {index}. Skipping.")

    if not values:
        print("Error: No valid numeric data found in the
specified column.")
        return None

    total = sum(values)
    avg = total / len(values)
    return total, avg

def main():
    if len(sys.argv) < 2:
        print("Usage: python utils.py <csv_filepath>
[column_index]")
        sys.exit(1)

    filepath = sys.argv[1]
    column_index = 1  # default column index

    if len(sys.argv) >= 3:
        try:
            column_index = int(sys.argv[2])
        except ValueError:
            print("Error: Column index must be an integer.")
            sys.exit(1)

    data = read_csv(filepath)
    if data is None:
        sys.exit(1)

    result = summarize_column(data, column_index)
```

~~~
    if result is None:
        sys.exit(1)

    total, avg = result
    print(f"Total: {total}")
    print(f"Average: {avg}")

if __name__ == '__main__':
    main()
```
~~~

Result 3:

~~~
```python
"""
utils.py - Script to parse and summarize numeric CSV files.

Usage:
    python utils.py <csv_filepath> [column_index]

Description:
    Reads a CSV file, extracts numeric values from a specified column,
    and computes the total and average of those values.
"""

import csv
import sys


def read_csv(filepath):
    """
    Reads a CSV file and returns its content as a list of rows.

    Args:
        filepath (str): Path to the CSV file.

    Returns:
        list[list[str]]: List of rows, where each row is a list of strings.
        None if file cannot be read.
    """
    try:
        with open(filepath, 'r', newline='', encoding='utf-8') as file:
            return list(csv.reader(file))
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
    except Exception as error:
        print(f"Error reading file '{filepath}': {error}")
    return None


def parse_numeric_column(data, column_index):
~~~

```
    """
    Parses numeric values from a specified column in CSV data, skipping the header.

    Args:
        data (list[list[str]]): CSV data as a list of rows.
        column_index (int): Index of the column to parse.

    Returns:
        list[float]: List of parsed numeric values.
    """
    values = []

    for row_num, row in enumerate(data[1:], start=2):  # Skip header, start counting rows at 2 for user clarity
        if len(row) <= column_index:
            print(f"Warning: Row {row_num} missing column {column_index}. Skipping.")
            continue

        cell = row[column_index].strip()
        if not cell:
            print(f"Warning: Empty cell at row {row_num}, column {column_index}. Skipping.")
            continue

        try:
            value = float(cell)
            values.append(value)
        except ValueError:
            print(f"Warning: Non-numeric value '{cell}' at row {row_num}, column {column_index}. Skipping.")

    return values


def summarize(values):
    """
    Calculates the total and average of a list of numeric values.

    Args:
        values (list[float]): List of numeric values.

    Returns:
        tuple[float, float]: Total and average of the values.
    """
    if not values:
        return None, None

    total = sum(values)
    average = total / len(values)
    return total, average
```

~~~
def main():
    if len(sys.argv) < 2:
        print("Usage: python utils.py <csv_filepath> [column_index]")
        sys.exit(1)

    filepath = sys.argv[1]

    # Default column index to 1 if not provided
    column_index = 1
    if len(sys.argv) >= 3:
        try:
            column_index = int(sys.argv[2])
            if column_index < 0:
                raise ValueError()
        except ValueError:
            print("Error: Column index must be a non-negative integer.")
            sys.exit(1)

    data = read_csv(filepath)
    if data is None:
        sys.exit(1)

    if len(data) < 2:
        print("Error: CSV file must contain a header and at least one data row.")
        sys.exit(1)

    if column_index >= len(data[0]):
        print(f"Error: Column index {column_index} out of range. CSV has {len(data[0])} columns.")
        sys.exit(1)

    values = parse_numeric_column(data, column_index)
    if not values:
        print("Error: No valid numeric data found in the specified column.")
        sys.exit(1)

    total, average = summarize(values)
    print(f"Total: {total}")
    print(f"Average: {average}")


if __name__ == "__main__":
    main()
```
~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

| | |
|---|---|
| **Output for model 2** | Result 1:<br>1. **Syntax Errors**: The code contains syntax errors due to incorrect usage of single quotes (`‘` and `’`) instead of regular single quotes (`'`).<br>2. **File Path Hardcoding**: The file path `data.csv` is hardcoded in the `main` function, which makes the script less flexible and reusable for different file paths.<br>3. **Error Handling**: There is no error handling for file operations or CSV parsing. If the file does not exist or is not accessible, the script will raise an exception.<br>4. **CSV Format Assumptions**: The script assumes that the CSV file has a header row and that the specified column index (in this case, `1`) contains numeric data. There is no validation to check if these assumptions are true.<br>5. **Empty Column Handling**: If the specified column contains no data (i.e., all rows have empty values in that column), the script will raise a `ZeroDivisionError` when trying to calculate the average.<br>6. **Column Index Validation**: There is no validation to check if the provided column index is within the bounds of the data. If the index is out of range, the script will raise an `IndexError`.<br>7. **Magic Numbers**: The script uses magic numbers, such as `1` for the column index, which makes the code less readable and maintainable.<br>8. **Missing Docstrings and Comments**: The script lacks docstrings and comments, making it difficult to understand the purpose and functionality of the functions and the overall script.<br>9. **Code Formatting**: The code formatting is inconsistent, with extra spaces and inconsistent indentation, which can make the code harder to read and maintain.<br>10. **Data Type Checking**: The script assumes that the values in the specified column are numeric. There is no type checking or conversion to ensure that the values are indeed numbers, which can lead to runtime errors if the data contains non-numeric values.<br>11. **Skipping Header Row**: The script skips the header row implicitly by starting the iteration from the second row. This assumes that the header is always present and does not provide a way to handle cases where the header is missing or needs to be included in the summary. |

Result 2:

~~~
Here is the revised version of the script with the identified
issues addressed:

```python
# utils.py - Script to parse and summarize numeric CSV files

import csv

def read_csv(filepath):
    """Reads a CSV file and returns its content as a list of
rows."""
    try:
        with open(filepath, 'r') as f:
            return [row for row in csv.reader(f)]
    except FileNotFoundError:
        print(f"Error: The file {filepath} does not exist.")
        return None
    except Exception as e:
        print(f"Error: An unexpected error occurred while
reading the file {filepath}.\n{e}")
        return None

def summarize_column(data, index):
    """Summarizes a column in the CSV data by calculating the
total and average."""
    if not data:
        print("Error: No data to summarize.")
        return None, None

    if index < 0 or index >= len(data[0]):
        print(f"Error: Column index {index} is out of bounds.")
        return None, None

    try:
        values = [float(row[index]) for row in data[1:]]  #
skip header
        total = sum(values)
        avg = total / len(values)
        return total, avg
    except ValueError:
        print(f"Error: Non-numeric data found in column
{index}.")
        return None, None
    except ZeroDivisionError:
        print(f"Error: Column {index} is empty.")
        return None, None

def main(filepath, column_index):
    """Main function to read CSV and summarize a column."""
    data = read_csv(filepath)
    if data is None:
        return
~~~

~~~~
    total, avg = summarize_column(data, column_index)
    if total is not None and avg is not None:
        print(f"Total: {total}")
        print(f"Average: {avg}")

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
        print("Usage: python utils.py <filepath>
<column_index>")
    else:
        filepath = sys.argv[1]
        column_index = int(sys.argv[2])
        main(filepath, column_index)
```

### Changes Made:

1. **Syntax Errors**: Corrected the use of single quotes.
2. **File Path Hardcoding**: Removed hardcoding and used
command-line arguments to specify the file path and column
index.
3. **Error Handling**: Added error handling for file operations
and CSV parsing.
4. **CSV Format Assumptions**: Added validation to check if the
file is empty and if the column index is within bounds.
5. **Empty Column Handling**: Added error handling for empty
columns.
6. **Column Index Validation**: Added validation to ensure the
column index is within the bounds of the data.
7. **Magic Numbers**: Removed magic numbers by using
command-line arguments.
8. **Docstrings and Comments**: Added docstrings to explain the
purpose of functions.
9. **Code Formatting**: Improved code formatting for
consistency.
10. **Data Type Checking**: Added type checking and error
handling for non-numeric data.
11. **Skipping Header Row**: Ensured the header row is skipped
and provided a way to handle cases where the header is missing.
~~~~

Result 3:

~~~~
Certainly! Here is a refactored version of the script with
improved readability, correctness, and safety:

```python
# utils.py - Script to parse and summarize numeric CSV files

import csv
import sys

def read_csv(filepath):
~~~~

```
    """
    Reads a CSV file and returns its content as a list of rows.

    :param filepath: Path to the CSV file.
    :return: List of rows from the CSV file or None if an error
occurs.
    """
    try:
        with open(filepath, mode='r', newline='') as file:
            reader = csv.reader(file)
            return list(reader)
    except FileNotFoundError:
        print(f"Error: The file {filepath} does not exist.")
    except Exception as e:
        print(f"Error: An unexpected error occurred while
reading the file {filepath}.\n{e}")
    return None

def summarize_column(data, index):
    """
    Summarizes a column in the CSV data by calculating the
total and average.

    :param data: List of rows from the CSV file.
    :param index: Index of the column to summarize.
    :return: Tuple (total, average) or (None, None) if an error
occurs.
    """
    if not data:
        print("Error: No data to summarize.")
        return None, None

    if index < 0 or index >= len(data[0]):
        print(f"Error: Column index {index} is out of bounds.")
        return None, None

    try:
        values = [float(row[index]) for row in data[1:]]  #
skip header
        total = sum(values)
        avg = total / len(values)
        return total, avg
    except ValueError:
        print(f"Error: Non-numeric data found in column
{index}.")
    except ZeroDivisionError:
        print(f"Error: Column {index} is empty.")
    return None, None

def main():
    """
    Main function to read CSV and summarize a column.
    """
```

~~~
    if len(sys.argv) != 3:
        print("Usage: python utils.py <filepath> <column_index>")
        return

    filepath = sys.argv[1]
    try:
        column_index = int(sys.argv[2])
    except ValueError:
        print("Error: Column index must be an integer.")
        return

    data = read_csv(filepath)
    if data is None:
        return

    total, avg = summarize_column(data, column_index)
    if total is not None and avg is not None:
        print(f"Total: {total}")
        print(f"Average: {avg}")

if __name__ == '__main__':
    main()
```

### Changes Made:

1. **Error Handling**: Enhanced error handling to provide clearer error messages.
2. **Function Documentation**: Added docstrings to all functions to explain their purpose and parameters.
3. **Code Structure**: Improved the structure of the script for better readability.
4. **Input Validation**: Added input validation for the file path and column index.
5. **Magic Numbers**: Removed magic numbers and used command-line arguments effectively.
6. **Code Formatting**: Ensured consistent code formatting for better readability.
7. **CSV Reader**: Enhanced the CSV reader to handle different newline formats gracefully.
8. **Command-Line Interface**: Improved the command-line interface for better user experience.

This refactored script should be more robust, readable, and maintainable.
~~~

**Analysis of Models**

BLEU scores:

- Zero-shot: 30.02
- Prompt Chaining

  - Result 1: 9.31
  - Result 2: 47.26
  - Result 3: 37.38

**Analysis of Prompting Strategies**

## Task 22: Code Completion and Robustness Enhancement (Python)

**Prompting Strategy 1: Zero Shot**

| | |
|---|---|
| **Task** | Complete a Python script for file-processing |

**Prompt**

~~~
Complete the following file-processing script:

```python
# file_processor .py - Incomplete script for processing text
files

import string

def load_file ( filepath ) :
  with open ( filepath , ’r’) as f :
    return f . readlines ()

def clean_line ( line ) :
  # TODO : Remove punctuation and make lowercase
  pass
def count_words ( lines ) :
  word_counts = {}
  for line in lines :
  clean = clean_line ( line )
  for word in clean . split () :
    word_counts [ word ] = word_counts . get ( word , 0) + 1
  return word_counts

def main () :
  filepath = ’input .txt ’
  lines = load_file ( filepath )
  counts = count_words ( lines )
  for word , count in sorted ( counts . items () ) :
~~~

~~~
    print ( f"{ word }: { count }")

if __name__ == ’__main__ ’:
main ()
```
.
The goal is to clean each line (by removing punctuation and
making the line lowercase) and count word frequencies
~~~

| | |
|---|---|
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |
| **Output for model 1** | Here is the completed script with the `clean_line` function implemented properly, and some formatting fixes for consistent Python style and syntax errors (like quotes and indentation): |

~~~
```python
# file_processor.py - Completed script for processing text
files

import string

def load_file(filepath):
    with open(filepath, 'r') as f:
        return f.readlines()

def clean_line(line):
    # Remove punctuation and make lowercase
    translator = str.maketrans('', '', string.punctuation)
    return line.translate(translator).lower()

def count_words(lines):
    word_counts = {}
    for line in lines:
        clean = clean_line(line)
        for word in clean.split():
            word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts

def main():
    filepath = 'input.txt'
    lines = load_file(filepath)
    counts = count_words(lines)
    for word, count in sorted(counts.items()):
        print(f"{word}: {count}")

if __name__ == '__main__':
    main()
~~~

~~~~text
```

### Explanation of the fix:
- `clean_line` uses `str.translate` with a translation table
that removes all punctuation characters from the line.
- Converts the line to lowercase using `.lower()`.
- Fixed indentation and quote characters (use straight quotes
`'` instead of curly ones).
- Fixed the indentation for the `count_words` function and the
`main` function call under `if __name__ == '__main__':`.

This script will now properly read the file, clean each line,
count the words, and print the word frequencies sorted
alphabetically.
~~~~

| | |
|---|---|
| **Model 2** | Codestral 25.01 |

**Output for model 2**

~~~~text
Sure! Let's complete the script by implementing the
`clean_line` function and correcting any syntax issues. Here is
the complete and functional script:

```python
# file_processor.py - Complete script for processing text files

import string

def load_file(filepath):
    with open(filepath, 'r') as f:
        return f.readlines()

def clean_line(line):
    # Remove punctuation and make lowercase
    translator = str.maketrans('', '', string.punctuation)
    cleaned_line = line.translate(translator).lower()
    return cleaned_line

def count_words(lines):
    word_counts = {}
    for line in lines:
        clean = clean_line(line)
        for word in clean.split():
            word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts

def main():
    filepath = 'input.txt'
    lines = load_file(filepath)
    counts = count_words(lines)
    for word, count in sorted(counts.items()):
        print(f"{word}: {count}")

if __name__ == '__main__':
    main()
~~~~

|  |  |
|---|---|

~~~~
```

### Explanation:

1. **`load_file` Function**: Reads the file and returns a list of lines.
2. **`clean_line` Function**:
   - Removes punctuation using `str.maketrans` and `translate`.
   - Converts the line to lowercase using `lower()`.
3. **`count_words` Function**:
   - Initializes an empty dictionary to store word counts.
   - Iterates through each line, cleans it, splits it into words, and updates the word count in the dictionary.
4. **`main` Function**:
   - Specifies the file path.
   - Loads the file.
   - Counts the words.
   - Prints the word counts in sorted order.

### Usage:
- Ensure you have an `input.txt` file in the same directory as the script.
- Run the script using Python: `python file_processor.py`.

This will process the text file, count the frequency of each word, and print the results.
~~~~

### Prompting Strategy 2: Chain of Thought

|  |  |
|---|---|
| **Task** | Complete a Python script for file-processing |
| **Prompt** | Complete the following file-processing script: |

~~~~
```python
# file_processor .py - Incomplete script for processing text files

import string

def load_file ( filepath ) :
  with open ( filepath , ’r’) as f :
    return f . readlines ()

def clean_line ( line ) :
  # TODO : Remove punctuation and make lowercase
  pass
~~~~

~~~
def count_words ( lines ) :
  word_counts = {}
  for line in lines :
  clean = clean_line ( line )
  for word in clean . split () :
    word_counts [ word ] = word_counts . get ( word , 0) + 1
  return word_counts

def main () :
  filepath = ’input .txt ’
  lines = load_file ( filepath )
  counts = count_words ( lines )
  for word , count in sorted ( counts . items () ) :
    print ( f"{ word }: { count }")

if __name__ == ’__main__ ’:
main ()
```
.
The goal is to clean each line (by removing punctuation and
making the line lowercase) and count word frequencies. Think
step-by-step.
~~~

| | |
|---|---|
| **Max Tokens** | 1024 |
| **Model 1** | gpt-4.1 mini |
| **Output for model 1** | |

```
Let's carefully complete and correct the script step-by-step.

---

### Step 1: Fix syntax issues

- Replace curly quotes `’` with straight quotes `'`.
- Fix indentation.
- Remove extra spaces inside parentheses, e.g., `open(filepath,
'r')`
- Correct the filepath string: `'input.txt'` instead of `'input
.txt '`
- Fix the `if __name__ == '__main__':` line and indent the
`main()` call.

---
```

### Step 2: Implement `clean_line()`

Goal: For a given string line,
- Remove all punctuation.
- Convert to lowercase.

We can use `str.translate()` with `str.maketrans()` and `string.punctuation`.

---

### Step 3: Verify the logic of counting words.

No changes needed; just fix indentation.

---

### Final corrected and completed script:

```python
# file_processor.py - Complete script for processing text files

import string

def load_file(filepath):
    with open(filepath, 'r') as f:
        return f.readlines()

def clean_line(line):
    # Remove punctuation and make lowercase
    translator = str.maketrans('', '', string.punctuation)
    return line.translate(translator).lower()

def count_words(lines):
    word_counts = {}
    for line in lines:
        clean = clean_line(line)
        for word in clean.split():
            word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts

def main():
    filepath = 'input.txt'
    lines = load_file(filepath)
    counts = count_words(lines)
    for word, count in sorted(counts.items()):
        print(f"{word}: {count}")

if __name__ == '__main__':
    main()
```

---

````
### Explanation:

- `clean_line` uses `str.translate` to remove punctuation.
- The line is lowercased.
- `count_words` splits the cleaned line into words and counts
frequencies.
- `main` loads the file, counts words, prints sorted counts.

This should fulfill the goal completely.
````

| | |
|---|---|
| **Model 2** | Codestral 25.01 |
| **Output for model 2** | |

~~~
Sure! Let's complete the script step-by-step.

### Step-by-Step Solution

1. **Import Necessary Libraries**: We will need `string` for
punctuation removal.
2. **Load File**: The `load_file` function is already defined
correctly. It reads all lines from the file.
3. **Clean Line**: The `clean_line` function needs to remove
punctuation and make the line lowercase.
4. **Count Words**: The `count_words` function needs to call
`clean_line` and update word counts.
5. **Main Function**: The `main` function should handle the
overall flow of the script.

### Detailed Implementation

1. **Import Necessary Libraries**:
   ```python
   import string
   ```

2. **Load File**: Already implemented correctly.
   ```python
   def load_file(filepath):
       with open(filepath, 'r') as f:
           return f.readlines()
   ```

3. **Clean Line**:
   - Remove punctuation using `str.translate` and
`str.maketrans`.
   - Convert the line to lowercase using `str.lower()`.
   ```python
   def clean_line(line):
       translator = str.maketrans('', '', string.punctuation)
       clean_line = line.translate(translator).lower()
       return clean_line
   ```
~~~

~~~
4. **Count Words**:
   - Call `clean_line` for each line.
   - Split the cleaned line into words.
   - Update word counts in a dictionary.
   ```python
   def count_words(lines):
       word_counts = {}
       for line in lines:
           clean = clean_line(line)
           for word in clean.split():
               word_counts[word] = word_counts.get(word, 0) + 1
       return word_counts
   ```

5. **Main Function**:
   - Load the file.
   - Count the words.
   - Print the word counts sorted alphabetically.
   ```python
   def main():
       filepath = 'input.txt'
       lines = load_file(filepath)
       counts = count_words(lines)
       for word, count in sorted(counts.items()):
           print(f"{word}: {count}")

   if __name__ == '__main__':
       main()
   ```

### Complete Script

Here is the complete script with all the steps implemented:

```python
import string

def load_file(filepath):
    with open(filepath, 'r') as f:
        return f.readlines()

def clean_line(line):
    translator = str.maketrans('', '', string.punctuation)
    clean_line = line.translate(translator).lower()
    return clean_line

def count_words(lines):
    word_counts = {}
    for line in lines:
        clean = clean_line(line)
        for word in clean.split():
            word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts
~~~

~~~
def main():
    filepath = 'input.txt'
    lines = load_file(filepath)
    counts = count_words(lines)
    for word, count in sorted(counts.items()):
        print(f"{word}: {count}")

if __name__ == '__main__':
    main()
```

### Explanation

- **Loading the File**: The `load_file` function reads all
lines from the specified file.
- **Cleaning the Line**: The `clean_line` function removes
punctuation and converts the line to lowercase.
- **Counting Words**: The `count_words` function processes each
line, cleans it, splits it into words, and updates the word
count dictionary.
- **Main Function**: The `main` function orchestrates the
loading of the file, counting of words, and printing of the
results.

This script should now correctly process a text file, count the
word frequencies, and print them in alphabetical order.
~~~

**Analysis of Models**

BLEU scores:

- Zero-shot: 56.86
- Chain of thought: 32.82

Code BLEU scores:

- Zero-shot: 70.61
- Chain of thought: 52.55

**Analysis of Strategies**

The implementation of clean_line is basically the same across all 4 runs. Telling the models to “think step-by-step” did not have an impact on the completion of the script. There is just more explanation of the code in the response.