Assignment 3 – Prompt Engineering Table

Task 1: Code Summarization (Java)

Few-Shot (Strategy 2)

Codestral:
The provided Java method counts the frequency of each word in a given list of words and returns a map where the keys are the words and the values are their respective frequencies.

Here's a summary of the method:

```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;
}
```

Summary: Counts the frequency of each word in a list of words and returns a map with words as keys and their frequencies as values.

GPT-4o-mini:
Summary: Counts the frequency of each word in a list and returns a map with words as keys and their corresponding counts as values.

Zero-Shot (Strategy 1)

Codestral:
The given Java method `countWordFrequency` takes a list of strings (`words`) as input and returns a `Map` where the keys are the words from the list and the values are the frequencies of those words in the list.

Here is a summary:

- **Input:** A list of strings.
- **Output:** A map where keys are the strings from the input list and values are their respective frequencies.
- **Process:** The method iterates through each word in the list, updating a frequency map with the count of each word's occurrences using `getOrDefault`.

Example:
- If the input list is `["apple", "banana", "apple", "orange", "banana", "apple"]`, the output map would be `{"apple": 3, "banana": 2, "orange": 1}`.

GPT-4o-mini:
The provided Java method, `countWordFrequency`, is designed to count the frequency of words in a list and return the results in a map. Let's break down how the method works step-by-step.

### Method Signature
```java
public Map<String, Integer> countWordFrequency(List<String> words)
```
- **Input:** A `List<String>` called `words`, which contains the words whose frequencies need to be counted.
- **Output:** A `Map<String, Integer>` where the keys are the unique words from the input list, and the values are the counts of how many times each word appears.

### Method Explanation
1. **Initialization of the frequency map:**
   ```java
   Map<String, Integer> freqMap = new HashMap<>();
   ```
   - A new `HashMap` named `freqMap` is created. This map will store the words as keys and their corresponding frequencies as values.

2. **Iterating through the list of words:**
   ```java
   for (String word : words) {
   ```
   - A `for-each` loop is used to iterate over each `String` in the `words` list one by one.

3. **Updating the frequency count:**

```java
freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);
```
  - For each `word`, the method checks if the word is already present in `freqMap`.
  - The `getOrDefault` method is used to retrieve the current count of the word:
    - If the word exists in the map, it returns its current count.
    - If the word does not exist, it returns `0`.
  - The count is then incremented by `1` (to account for the current occurrence of the word), and the new count is put back into `freqMap` using `put`.

4. **Returning the frequency map:**
```java
return freqMap;
```
  - After processing all the words, the method returns the `freqMap`, which now contains the frequency of each word in the input list.

### Example
Consider the following example to illustrate how the method works:

```java
List<String> words = Arrays.asList("apple", "banana", "apple", "orange", "banana", "apple");
Map<String, Integer> result = countWordFrequency(words);
```
- The final `result` would be:
```
{
  "apple" : 3,
  "banana" : 2,
  "orange" : 1
}
```
This indicates that "apple" appears 3 times, "banana" appears 2 times, and "orange" appears once in the provided list of words.

### Summary
The `countWordFrequency` method efficiently counts the occurrences of each word in a list and returns the result in a map format, allowing for quick lookups of word frequencies.

Analysis:

Zero-Shot missed key steps explaining the countWordFrequency method, while Few-Shot walked through initialization and updating clearly. Few-Shot's explanation made the process feel more structured and easier to follow. The BLEU score (around 0.29) also reflected how much closer it stayed to a full natural description.

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

Chain-of-Thought (Strategy 3)	Zero-Shot (Strategy 1)
Codestral: Let's break down the method step by step to understand what it is trying to do and identify any issues. ### Step-by-Step Analysis	Codestral: The method you provided has a few issues that need to be fixed: 1. The indentation of the `total` variable declaration and the `return` statement is incorrect.

1. **Function Definition**:
  ```python
  def sum_range(start, end):
  ```
  This defines a function named `sum_range` that takes two parameters: `start` and `end`.

2. **Initialization**:
  ```python
  total = 0
  ```
  This initializes a variable `total` to 0. This variable will hold the sum of the numbers in the range.

3. **Loop**:
  ```python
  for i in range(start, end):
  ```
  This loop iterates over the range of numbers from `start` to `end - 1`.

4. **Sum Calculation**:
  ```python
  total += i
  ```
  This line adds the current value of `i` to `total` in each iteration of the loop.

5. **Return Statement**:
  ```python
  return total
  ```
  This returns the accumulated sum stored in `total`.

### Identifying Issues

1. **Indentation**:
  The code is missing proper indentation. In Python, indentation is crucial for defining the scope of loops and functions.

2. **Range Inclusivity**:
  The `range(start, end)` function in Python will include `start` but exclude `end`. If you need to include `end` in the sum, you should use `range(start, end + 1)`.

3. **Docstring**:
  It might be helpful to add a docstring to explain what the function does.

### Fixed Code

Here is the corrected version of the method:

```python
def sum_range(start, end):
    """
    Returns the sum of all integers from start to end (inclusive).

    Parameters:
    start (int): The starting integer of the range.
```

2. The range should include the `end` value if you want to sum up to and including `end`.

Here is the corrected version of the method:

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

Explanation of changes:
1. Indented the `total` variable and the `return` statement correctly.
2. Changed `range(start, end)` to `range(start, end + 1)` to include the `end` value in the range.

This will correctly sum all integers from `start` to `end` inclusive.

GPT-4o-mini:
The provided method has a bug related to the range of numbers it sums. The `range` function in Python generates numbers from `start` up to, but not including, `end`. If you want to include `end` in the sum, you'll need to adjust the range accordingly.

Additionally, the indentation is incorrect in the code, which would lead to a syntax error. Here is the corrected version of the method:

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

Now, the `sum_range` function will correctly compute the sum of all integers from `start` to `end`, inclusive.

```
    end (int): The ending integer of the range.

    Returns:
    int: The sum of all integers in the range.
    """
    total = 0
    for i in range(start, end + 1):  # Include end in the range
        total += i
    return total
```

### Explanation of Fixes

1. **Indentation**:
Proper indentation is added to ensure the code is syntactically correct.

2. **Range Inclusivity**:
Changed `range(start, end)` to `range(start, end + 1)` to include the `end` value in the sum.

3. **Docstring**:
Added a docstring to describe the function's purpose, parameters, and return value.

With these fixes, the function should work correctly and provide the sum of integers from `start` to `end` (inclusive).

GPT-4o-mini:
Let's analyze the provided method `sum_range` step by step:

### Step-by-step Analysis

1. **Function Definition**: The function `sum_range` takes two parameters: `start` and `end`.

2. **Initialization**: The variable `total` is initialized to 0. This variable is intended to accumulate the sum of the numbers in the specified range.

3. **For Loop**: The loop `for i in range(start, end)` iterates over a sequence of numbers starting from `start` up to, but not including, `end`.

4. **Accumulation**: Inside the loop, `total += i` adds the current value of `i` to `total`.

5. **Return Statement**: Finally, the function returns the total sum accumulated in the variable `total`.

### Identifying Issues

The function works correctly for most cases, but there are a few things to consider:

1. **Behavior with Negative Ranges**: The function may not behave as expected if the `start` is greater than `end`, as the `range` function will produce an empty sequence in that case. This means the function would return `0` instead of the expected behavior (which could be to return the sum of an empty range or raise an error).

2. **Indentation**: The code provided has incorrect indentation for the lines inside the function. Python relies on indentation to define the scope of loops and functions.

### Suggested Improvements

1. **Handle Invalid Ranges**: We should handle the case where `start` is greater than `end`. This can be done by raising a ValueError or returning a specific value.

2. **Correct Indentation**: Ensure that the code is properly indented so that it follows Python syntax rules.

### Fixed Code

Here is the corrected version of the function:

```python
def sum_range(start, end):
    if start > end:
        raise ValueError("Start must be less than or equal to end.")

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

### Explanation of Fixes

- **Error Handling**: The line `if start > end: raise ValueError("Start must be less than or equal to end.")` checks if the starting point is greater than the ending point and raises an appropriate error if so.
- **Indentation**: Indentation is corrected for readability and to ensure the proper execution of the code.

### Additional Considerations

If you want to include the `end` value in the sum, you can change the `range` function to `range(start, end + 1)`.

Here’s how that would look:

```python
def sum_range(start, end):
    if start > end:
        raise ValueError("Start must be less than or equal to end.")

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

This version of the function includes `end` in the summation as well.

Analysis:

Zero-Shot caught deeper structural issues in the code, offering thoughtful corrections, whereas Few-Shot focused narrowly and skipped some design flaws. Although Few-Shot posted a slightly better BLEU score (about 0.32), Zero-Shot's broader analysis made its output more valuable.

Task 3: Bug Classification (C++)

Role-Playing (Strategy 6)

~~~
Codestral:
Certainly! Let's review the provided C++ function for any issues,
especially focusing on the potential bugs.

```cpp
int * getArray ( int size ) {
    int arr [ size ]; // Warning: local array
    return arr ; // Bug: returning pointer to local variable
}
```

### Analysis

1. **Local Array Declaration**:
   ```cpp
   int arr [ size ];
   ```
   This declares an array `arr` of size `size` on the stack. The size
of `arr` is determined by the parameter `size` passed to the
function.

2. **Returning a Pointer to a Local Variable**:
   ```cpp
   return arr;
   ```
   This line returns a pointer to the local array `arr`. The problem
with this is that `arr` is a local variable that exists only within the
scope of the `getArray` function. Once the function returns, `arr`
goes out of scope, and any pointer to it becomes invalid.

### Type of Bug

The issue here is a **dangling pointer bug**, specifically a
**returning pointer to local variable bug**. This type of bug
occurs when a function returns a pointer to a local variable, which
is automatically destroyed when the function exits. Any attempt to
use this pointer after the function returns will lead to undefined
behavior.

### Suggestion

To fix this issue, you should allocate memory dynamically using
`new` or `malloc`, and then return the pointer. Additionally, you
must ensure that the caller of this function is responsible for
deallocating the memory using `delete` or `free` to avoid memory
leaks.

Here is a revised version of the function:

```cpp
int * getArray(int size) {
    int *arr = new int[size]; // Dynamically allocate memory
    return arr; // Return the pointer to the dynamically allocated
~~~

Zero-Shot (Strategy 1)

~~~
Codestral:
The provided C++ code has a bug and a warning. Let's break it
down:

1. **Warning: Local array**
   ```cpp
   int arr[size];
   ```
   This line declares a local array `arr` of size `size`. The issue here
is that `arr` is allocated on the stack and will be deallocated when
the function returns. Therefore, any pointer or reference to this
array becomes invalid after the function exits.

2. **Bug: Returning pointer to local variable**
   ```cpp
   return arr;
   ```
   This line returns a pointer to the local array `arr`. As mentioned
above, `arr` is allocated on the stack and will be destroyed when
the function `getArray` returns. Thus, the pointer returned by this
function will be pointing to an invalid memory location, leading to
undefined behavior if dereferenced.

### Correcting the Code

To fix this, you should allocate memory dynamically using `new`
or use a `std::vector` for automatic memory management. Here are
two correct approaches:

#### Using `new` for dynamic allocation:
```cpp
int * getArray(int size) {
    int *arr = new int[size]; // Dynamically allocate memory
    return arr; // Return pointer to dynamically allocated memory
}
```

#### Using `std::vector` for automatic memory management:
```cpp
#include <vector>

std::vector<int> getArray(int size) {
    std::vector<int> arr(size); // Automatically managed vector
    return arr; // Return vector (copy semantics or move semantics)
}
```

### Summary

The original code has a bug because it returns a pointer to a local
variable, which is invalid outside the scope of the function. The
corrected versions use either dynamic memory allocation or
automatic memory management provided by `std::vector` to
~~~

```
memory
}

// Example usage in the caller function
int main() {
    int size = 10;
    int *array = getArray(size);

    // Use the array...

    // Don't forget to delete the allocated memory to avoid memory
leaks
    delete[] array;
    return 0;
}
```

In this corrected version, the memory for the array is allocated on the heap using `new`, which ensures that the memory persists even after the function returns. The caller is responsible for deallocating this memory using `delete[]` to prevent memory leaks.

GPT-4o-mini:
The function you've provided has a critical bug related to returning a pointer to a local variable.

### Bug Type: Returning Pointer to Local Variable

#### Explanation:
1. **Local Array**: The line `int arr[size];` creates a local array `arr` whose lifetime is limited to the scope of the function `getArray`. Once the function exits, the memory for `arr` is deallocated.

2. **Returning Pointer**: The line `return arr;` returns a pointer to this local array. After the function exits, the pointer returned will point to memory that is no longer valid. In other words, the data that `arr` points to may be overwritten or become inaccessible, leading to undefined behavior if the caller tries to access this memory.

### Recommendation for Fix:
To fix this issue, you could dynamically allocate memory for the array using `new`, or better yet, use a standard container like `std::vector<int>` to handle memory management automatically. Here's how you might modify the function using `new`:

```cpp
int* getArray(int size) {
    int* arr = new int[size]; // Dynamically allocate memory
    return arr; // Return pointer to dynamically allocated memory
}

// Remember to free the allocated memory later
```

Or, using `std::vector<int>`:

```cpp
#include <vector>
```

ensure the array's lifetime extends beyond the function's scope.

GPT-4o-mini:
The C++ code you provided has a critical bug related to returning a pointer to a local variable. Here's a breakdown of the issues:

1. **Local Array Declaration**: The line `int arr[size];` declares a local array on the stack. This array is only valid within the scope of the `getArray` function. Once the function exits, the memory allocated for this array is reclaimed, and the array goes out of scope.

2. **Returning Pointer to Local Variable**: The function returns a pointer to the local array with `return arr;`. This is problematic because after the function returns, the memory that `arr` points to is no longer valid. Accessing this memory location (the pointer returned) will lead to undefined behavior, as the memory might have been overwritten or may no longer exist.

To fix this bug, you can allocate the array dynamically using `new`, which allocates memory on the heap, or use a more modern approach like `std::vector`. Here are two possible fixes:

### Fix 1: Using Dynamic Memory Allocation
```cpp
int* getArray(int size) {
    int* arr = new int[size]; // Allocate memory dynamically
    return arr; // Return pointer to the allocated memory
}
```
Note: If you use `new`, remember to `delete[]` the allocated memory after you're done using it to avoid memory leaks.

### Fix 2: Use `std::vector`
```cpp
#include <vector>

std::vector<int> getArray(int size) {
    return std::vector<int>(size); // Return a vector, which manages
its own memory
}
```
Using `std::vector` is generally preferred in modern C++ as it automatically manages memory and provides better safety and convenience.

~~~
std::vector<int> getArray(int size) {
   return std::vector<int>(size); // Return a vector with the
specified size
}
```

Using `std::vector` is generally preferred in modern C++ as it
manages memory automatically and reduces the risk of memory
leaks.
~~~

Analysis:

When addressing the dynamic array bug, Zero-Shot separated warnings from bugs and gave two solid fixes, making the logic easier to follow. Chain-of-Thought touched on similar ideas but wasn't organized as cleanly. Zero-Shot's stronger structure lined up with its slightly better BLEU performance.

Task 4: Generating Email Validators (Python + Regex)

Few-Shot (Strategy 2)

~~~
Codestral:
To complete the Python function `is_valid_email` using regular
expressions (regex) to check if an email is valid, you can follow
these steps:

1. Import the `re` module.
2. Define a regex pattern to match a valid email format.
3. Use the `re.match` function to check if the email matches the
pattern.

Here's the complete function:

```python
import re

def is_valid_email(email):
   # Define the regex pattern for a valid email
   pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

   # Use re.match to check if the email matches the pattern
   if re.match(pattern, email):
      return 'valid'
   else:
      return 'invalid'

# Example usage:
print(is_valid_email("valid@gmail.com"))  # Output: valid
print(is_valid_email("fake@fake.com"))   # Output: valid
print(is_valid_email("invalid-email")) # Output: invalid
```

### Explanation of the regex pattern:
- `^[a-zA-Z0-9_.+-]+`: Ensures the email starts with one or more
alphanumeric characters, dots, underscores, plus signs, or hyphens.
- `@[a-zA-Z0-9-]+`: Ensures the email contains an '@' symbol
followed by one or more alphanumeric characters or hyphens.
- `\.[a-zA-Z0-9-.]+$`: Ensures the email ends with a dot followed
by one or more alphanumeric characters, dots, or hyphens.

This pattern should cover most standard email formats. However,
email validation can be quite complex, and this pattern may not
~~~

Zero-Shot (Strategy 1)

~~~
Codestral:
Certainly! To check the validity of an email address using regular
expressions (regex) in Python, you can use the `re` module. Below
is a function that defines a regex pattern for validating email
addresses:

```python
import re

def is_valid_email(email):
   # Define the regex pattern for validating an email
   email_pattern =
r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

   # Use the re.match function to check if the email matches the
pattern
   if re.match(email_pattern, email):
      return True
   else:
      return False

# Example usage:
print(is_valid_email("example@example.com"))  # Should return
True
print(is_valid_email("invalid.email@com"))    # Should return
False
```

### Explanation of the Regex Pattern:
- `^[a-zA-Z0-9._%+-]+`: This part ensures that the email starts
with one or more alphanumeric characters, dots, underscores,
percent signs, plus signs, or hyphens.
- `@[a-zA-Z0-9.-]+`: This part ensures that there is an '@' symbol
followed by one or more alphanumeric characters, dots, or
hyphens.
- `\.[a-zA-Z]{2,}$`: This part ensures that there is a dot followed
by at least two alphabetic characters, which represents the domain
suffix (like ".com", ".org", etc.).

This pattern should cover most common valid email formats, but
email validation can be complex due to the variety of valid email
~~~

cover all edge cases. For more robust email validation, you might want to use a more comprehensive library or service.

GPT-4o-mini:
You can use the `re` module in Python to create a regular expression that checks for valid email formats. Here's a completed version of the `is_valid_email` function using regex:

```python
import re

def is_valid_email(email):
    # Define a regex pattern for validating an email
    pattern =
r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    # Use re.match to check if the email matches the pattern
    if re.match(pattern, email):
        return "valid"
    else:
        return "invalid"

# Example usage:
print(is_valid_email("valid@gmail.com"))  # Output: valid
print(is_valid_email("fake@fake.com"))    # Output: invalid
```

### Explanation:
- The regex pattern used here breaks down as follows:
  - `^[a-zA-Z0-9._%+-]+` : The email username can contain letters (both uppercase and lowercase), digits, and some special characters (`._%+-`). The `+` indicates that there must be at least one character present.
  - `@` : The `@` symbol is required to separate the username from the domain.
  - `[a-zA-Z0-9.-]+` : The domain name can contain letters, digits, dots, and hyphens.
  - `\.[a-zA-Z]{2,}$` : The domain must end with a dot followed by at least two letters (this represents the top-level domain, like .com, .org, etc.).
- `re.match` checks if the entire string matches our regex pattern from the beginning.

You can test the function with different email inputs to see if they are correctly classified as valid or invalid.

formats. For more robust validation, you might want to use a library like `validators` which provides a more comprehensive email validation function.

GPT-4o-mini:
You can use the `re` module in Python to create a function that checks the validity of an email address using regular expressions. Here's how you can implement the `is_valid_email` function:

```python
import re

def is_valid_email(email):
    # Define the regex pattern for a valid email
    pattern =
r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    # Use re.match to check if the email matches the pattern
    if re.match(pattern, email):
        return True
    else:
        return False

# Example usage
print(is_valid_email("test@example.com"))  # True
print(is_valid_email("invalid-email@"))    # False
```

### Explanation of the Regex Pattern:
- `^` asserts the start of the string.
- `[a-zA-Z0-9._%+-]+` matches one or more characters that are either letters, digits, or any of the specified special characters (`._%+-`).
- `@` matches the at symbol.
- `[a-zA-Z0-9.-]+` matches one or more characters that are either letters, digits, or any of the specified special characters (`.-`).
- `\.` matches the dot (.) character.
- `[a-zA-Z]{2,}` matches two or more letters, which represents the top-level domain (like `.com`, `.org`, etc.).
- `$` asserts the end of the string.

Feel free to test the function with different email addresses to see how it performs.

Analysis:

Both models handled email validation reasonably well, but Zero-Shot explained the regex breakdown more carefully. Chain-of-Thought was faster but glossed over key pattern matching logic. Even though the BLEU difference was small (about 0.26 vs 0.23), Zero-Shot's output felt clearer overall.

Task 5: Generating Flask APIs (Python)

Role-Playing (Strategy 6)

Codestral:
Sure! Below is the completed Flask API endpoint that greets the user with a JSON response:

```python

Zero-Shot (Strategy 1)

Codestral:
To create a Flask endpoint that returns a JSON greeting when given a username in the URL, you need to modify the `greet` function to return a JSON response. Here's the complete implementation:
```

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # Create the greeting message
    greeting = f"Hello, {username}!"
    # Return the greeting as a JSON response
    return jsonify(message=greeting)

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

### Explanation:
1. **Importing Required Modules**: We import `Flask` and `jsonify` from `flask`.
2. **Creating the Flask App**: We create an instance of the Flask class.
3. **Defining the Route**: We define a route `/greet/<username>` where `<username>` is a variable part of the URL.
4. **Greeting Function**: The `greet` function takes `username` as an argument, creates a greeting message, and returns it as a JSON response using `jsonify`.
5. **Running the App**: We run the app with `debug=True` for easier development and debugging.

You can run this Flask application, and when you access `http://127.0.0.1:5000/greet/yourname`, you should get a JSON response like:

```json
{
    "message": "Hello, yourname!"
}
```

GPT-4o-mini:
Certainly! Below is a complete implementation of the Flask API endpoint that greets the user with a JSON response.

```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
1. **Importing Required Modules**: We import `Flask` and

```python
from flask import Flask, jsonify

app = Flask(__name__)

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

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

### Explanation:
1. **Importing Necessary Modules**:
   - `Flask` is imported to create the Flask application.
   - `jsonify` is imported to convert Python dictionaries to JSON responses.

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

3. **Defining the Route**:
   - `@app.route('/greet/<username>')` defines a route that accepts a username in the URL.
   - The `greet` function is called when this route is accessed.

4. **Creating the Greeting Message**:
   - Inside the `greet` function, a greeting message is created using an f-string.

5. **Returning the JSON Response**:
   - `jsonify(message=greeting)` converts the greeting message into a JSON response.

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

With this implementation, when you access the endpoint `/greet/<username>`, it will return a JSON object with a greeting message. For example, accessing `http://127.0.0.1:5000/greet/John` will return:

```json
{
    "message": "Hello, John!"
}
```

GPT-4o-mini:
You can create a Flask endpoint that returns a JSON greeting using the given structure. Here's the complete code with the implementation of the `greet` function:

<table>
<tr>
<td>`jsonify` from the `flask` package.
2. **Creating the Flask App**: We create an instance of the Flask application.
3. **Defining the Route**: We define a route `/greet/<username>`, where `<username>` is a dynamic part of the URL that will be passed to the function.
4. **Greet Function**: Inside the `greet` function, we create a greeting message using the provided username and then return it as a JSON response using `jsonify`.
5. **Running the App**: We check if the script is run directly and start the Flask application in debug mode.

Usage
Once this code is running, you can test the endpoint by navigating to `http://127.0.0.1:5000/greet/YourName` in your web browser or using a tool like Postman or curl. You should receive a JSON response like this:

```json
{
    "greeting": "Hello, YourName!"
}
```</td>
<td>```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    # Return a JSON greeting
    return jsonify({"greeting": f"Hello, {username}!"})

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

Explanation:
1. **Importing Modules**: We're importing `Flask` and `jsonify` from the `flask` package.
2. **Creating the Flask App**: We instantiate a Flask application.
3. **Defining the Route**: The route `'/greet/<username>'` captures a username from the URL.
4. **Defining the `greet` Function**: This function takes the `username` parameter and returns a JSON response with a greeting message using `jsonify`.
5. **Running the App**: The `app.run(debug=True)` line starts the Flask app with debug mode enabled, which is useful for development.

Usage:
- Start the Flask application.
- Navigate to `http://127.0.0.1:5000/greet/YourUsername` in your web browser or use a tool like `curl` or Postman.
- You should see a JSON response like:
```json
{
  "greeting": "Hello, YourUsername!"
}
```</td>
</tr>
</table>

Analysis:

Zero-Shot's walkthrough for setting up the application skipped important steps, while Prompt Chaining gave a more detailed explanation of dependencies and flow. Prompt Chaining felt more natural and was easier to follow, matching its stronger BLEU score.

Task 6: SQL Schema Design (SQL)

<table>
<tr>
<td>Chain-of-Thought (Strategy 3)</td>
<td>Zero-Shot (Strategy 1)</td>
</tr>
<tr>
<td>Codestral:
Designing a schema for an application where users review books involves several steps. We need to consider the relationships between the entities, appropriate data types, primary keys, foreign keys, and constraints to ensure data integrity and consistency. Here's a step-by-step thought process to design the schema:

1. Identify the Entities and Their Attributes
First, we identify the entities (users, books, reviews) and their respective attributes.

- **Users**: `id`, `name`
- **Books**: `id`, `title`</td>
<td>Codestral:
Sure, below is a schema for a book review app in SQL format. The schema includes appropriate constraints for the `users`, `books`, and `reviews` tables.

```sql
-- Create the users table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

-- Create the books table</td>
</tr>
</table>

- **Reviews**: `id`, `user_id`, `book_id`, `rating`

### 2. Define Primary Keys
Primary keys uniquely identify each record in a table.

- **Users**: `id`
- **Books**: `id`
- **Reviews**: `id`

### 3. Define Foreign Keys
Foreign keys create relationships between tables.

- **Reviews**: `user_id` references `users(id)`
- **Reviews**: `book_id` references `books(id)`

### 4. Consider Constraints
Constraints ensure data integrity.

- **Users**:
  - `id` should be unique and not null.
  - `name` should not be null.
- **Books**:
  - `id` should be unique and not null.
  - `title` should not be null.
- **Reviews**:
  - `id` should be unique and not null.
  - `user_id` should reference an existing `id` in the `users` table.
  - `book_id` should reference an existing `id` in the `books` table.
  - `rating` should be a valid value (e.g., between 1 and 5).

### 5. Design the SQL Schema
Putting it all together, here's the SQL schema:

```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),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

### Explanation of the Schema:

1. **Users Table**:
   - `id`: Unique identifier for each user, auto-incremented.
   - `name`: Name of the user, cannot be null.

2. **Books Table**:

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

-- Create the reviews table
CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
    review_text TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```

### Explanation:

1. **users Table**:
   - `id`: A unique identifier for each user, set as the primary key.
   - `name`: The name of the user, which cannot be null.

2. **books Table**:
   - `id`: A unique identifier for each book, set as the primary key.
   - `title`: The title of the book, which cannot be null.

3. **reviews Table**:
   - `id`: A unique identifier for each review, set as the primary key.
   - `user_id`: A foreign key that references the `id` in the `users` table, ensuring that each review is associated with a valid user.
   - `book_id`: A foreign key that references the `id` in the `books` table, ensuring that each review is associated with a valid book.
   - `rating`: An integer representing the rating given to the book, constrained to be between 1 and 5.
   - `review_text`: An optional text field for additional review comments.
   - `created_at`: A timestamp that records when the review was created, defaulting to the current timestamp.
   - `FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE`: Ensures referential integrity and cascades delete operations.
   - `FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE`: Ensures referential integrity and cascades delete operations.

This schema should provide the necessary structure and constraints for a basic book review application.

GPT-4o-mini:
Certainly! Below is a SQL schema for a book review app that includes tables for users, books, and reviews. The schema includes appropriate keys and constraints to maintain data integrity.

```sql
-- Create the users table
```

  - `id`: Unique identifier for each book, auto-incremented.
  - `title`: Title of the book, cannot be null.

3. **Reviews Table**:
  - `id`: Unique identifier for each review, auto-incremented.
  - `user_id`: Foreign key referencing the `id` in the `users` table, ensures each review is associated with a valid user.
  - `book_id`: Foreign key referencing the `id` in the `books` table, ensures each review is associated with a valid book.
  - `rating`: Rating given by the user, must be between 1 and 5.

### Considerations for Constraints:
- **Primary Key Constraints**: Ensure each row in the table is unique.
- **Foreign Key Constraints**: Ensure referential integrity between tables.
- **Check Constraints**: Ensure the rating value is within a valid range.

This schema ensures that users can review books, and each review is associated with a valid user and book, maintaining data integrity and consistency.

GPT-4o-mini:
Designing a schema for an app where users can review books involves a few key steps. Here's a structured thought process to create the SQL schema, along with appropriate keys and constraints.

### Step 1: Identify the Entities

In this case, we have three main entities:
1. **Users**: Individuals who can write reviews.
2. **Books**: The books that are being reviewed.
3. **Reviews**: The reviews written by users for specific books.

### Step 2: Define the Attributes for Each Entity
Next, we need to define the attributes for each entity.

1. **Users Table**:
  - `id`: Unique identifier for each user (Primary Key).
  - `name`: Name of the user (String).

2. **Books Table**:
  - `id`: Unique identifier for each book (Primary Key).
  - `title`: Title of the book (String).

3. **Reviews Table**:
  - `id`: Unique identifier for each review (Primary Key).
  - `user_id`: Foreign key referencing the `users` table (Integer).
  - `book_id`: Foreign key referencing the `books` table (Integer).
  - `rating`: Numeric score for the book (Integer, with constraints).

### Step 3: Define Relationships
We need to establish the relationships between the entities:
- A user can write many reviews, but each review is written by one user (1-to-many).
- A book can have many reviews, but each review is for one book (1-to-many).

### Step 4: Define Constraints

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,            -- Unique identifier for
each user
    name VARCHAR(100) NOT NULL        -- The name of the user
);

-- Create the books table
CREATE TABLE books (
    id SERIAL PRIMARY KEY,            -- Unique identifier for
each book
    title VARCHAR(255) NOT NULL       -- The title of the book
);

-- Create the reviews table
CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,            -- Unique identifier for
each review
    user_id INT NOT NULL,             -- Foreign key referencing the
users table
    book_id INT NOT NULL,             -- Foreign key referencing
the books table
    rating INT CHECK (rating >= 1 AND rating <= 5), -- Rating
must be between 1 and 5
    review_text TEXT,                 -- Optional text for the review
    created_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP, -- Timestamp for when the review was
created
    FOREIGN KEY (user_id) REFERENCES users(id) ON
DELETE CASCADE, -- Foreign key constraint with cascading
delete
    FOREIGN KEY (book_id) REFERENCES books(id) ON
DELETE CASCADE, -- Foreign key constraint with cascading
delete
    UNIQUE (user_id, book_id)         -- Ensure a user can only
review a book once
);
```

### Explanation of the Schema:

1. **Users Table**:
  - `id`: An auto-incrementing primary key.
  - `name`: A string that cannot be null.

2. **Books Table**:
  - `id`: An auto-incrementing primary key.
  - `title`: A string that cannot be null.

3. **Reviews Table**:
  - `id`: An auto-incrementing primary key.
  - `user_id`: A foreign key that references the `id` in the `users` table. It cannot be null.
  - `book_id`: A foreign key that references the `id` in the `books` table. It cannot be null.
  - `rating`: An integer that must be between 1 and 5 (inclusive).
  - `review_text`: Optional text for the review.
  - `created_at`: A timestamp that defaults to the current time when the review is created.
  - Foreign key constraints are added to ensure that if a user or a book is deleted, the corresponding reviews are also deleted (cascade delete).

Constraints ensure data integrity. Here are the constraints we will apply:
- **Primary Keys**: Ensure each record is unique.
- **Foreign Keys**: Ensure that references to other tables are valid.
- **Unique Constraints**: To prevent duplicate entries where necessary (e.g., a user should not be able to review the same book multiple times).
- **Check Constraints**: For attributes like `rating` to ensure valid values (e.g., between 1 and 5).

### Step 5: Write the SQL Schema

Here's how the SQL schema would look based on the above considerations:

```sql
CREATE TABLE users (
   id SERIAL PRIMARY KEY,  -- Unique identifier for users
   name VARCHAR(255) NOT NULL -- Name of the user
);

CREATE TABLE books (
   id SERIAL PRIMARY KEY,  -- Unique identifier for books
   title VARCHAR(255) NOT NULL -- Title of the book
);

CREATE TABLE reviews (
   id SERIAL PRIMARY KEY, -- Unique identifier for reviews
   user_id INT NOT NULL,  -- Foreign key for users
   book_id INT NOT NULL,  -- Foreign key for books
   rating INT CHECK (rating >= 1 AND rating <= 5), -- Rating must be between 1 and 5
   UNIQUE (user_id, book_id), -- Prevents a user from reviewing the same book multiple times
   FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,  -- Ensures referential integrity
   FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE -- Ensures referential integrity
);
```

### Step 6: Explanation of the SQL Schema
- **Users Table**: Contains a unique identifier and user name.
- **Books Table**: Contains a unique identifier and book title.
- **Reviews Table**: Links users and books through foreign keys, ensures that each user can only review a book once using a unique constraint, and checks that ratings are valid.
- **ON DELETE CASCADE**: This option on foreign keys ensures that if a user or book is deleted, their associated reviews are also removed, maintaining data integrity.

### Step 7: Consider Future Scalability
While the current schema meets the basic requirements, consider future needs:
- You might want to add timestamps to the reviews for tracking when they were created or updated.
- Consider adding more attributes to books (like author, publication date) or users (like email, profile picture).

#### Example of Adding Timestamps:

   - A unique constraint on `(user_id, book_id)` ensures that each user can only submit one review per book.

This schema provides a solid foundation for a book review application while ensuring data integrity and appropriate relationships between users, books, and reviews.

```sql
ALTER TABLE reviews ADD COLUMN created_at
TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
ALTER TABLE reviews ADD COLUMN updated_at
TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP;
```

### Conclusion
This step-by-step thought process allows for a clear and structured design of an SQL schema for a book review application, ensuring that all entities, relationships, and constraints are appropriately defined for data integrity and future scalability.

Analysis:

The SQL schema breakdown from Zero-Shot stayed direct, focusing on table relationships and constraints without drifting off-topic. Role-Playing added extra commentary but sometimes wandered away from the main points. BLEU favored Zero-Shot, showing its structure was more consistent.

Task 7: Null Dereference Detection (Java)

Chain-of-Thought (Strategy 3)

Codestral:
The provided code defines a method `getLength` in Java that returns the length of a given string. The potential issue with this method is that it does not handle the case where the input string `s` might be `null`. If `s` is `null`, calling `s.length()` will result in a `NullPointerException`, which would cause a crash.

### Thought Process:
1. **Identify the potential issue**: The method does not check whether the input string `s` is `null`.
2. **Determine the consequence**: If `s` is `null`, invoking `s.length()` will throw a `NullPointerException`.
3. **Plan the solution**: Add a check to ensure `s` is not `null` before trying to call `s.length()`.
4. **Implement the fix**: Return a default value (e.g., `-1` or `0`) if `s` is `null`.

### Fixed Code:
Here is the revised code with a null check:
```java
public int getLength(String s) {
    if (s == null) {
        return -1; // or any default value you deem appropriate
    }
    return s.length();
}
```

### Explanation:
1. **Null Check**: The method now includes a check to see if `s` is `null`. If `s` is `null`, the method returns `-1` (or any other default value that makes sense for your application).
2. **Safe Operation**: If `s` is not `null`, the method proceeds to call `s.length()` safely.

By adding this null check, the method is now robust against

Zero-Shot (Strategy 1)

Codestral:
If the `getLength` method is called with a `null` input, it will throw a `NullPointerException` because the `length()` method cannot be called on a `null` object. To prevent this risk, you should add a check to ensure that the input string is not `null` before attempting to call the `length()` method. Here is the revised method with the necessary check:

```java
public int getLength(String s) {
    if (s == null) {
        throw new IllegalArgumentException("Input string cannot be null");
    }
    return s.length();
}
```

In this revised version, the method checks if the input `s` is `null`. If it is, an `IllegalArgumentException` is thrown with an appropriate message. This way, the method does not attempt to call the `length()` method on a `null` object, thereby preventing a `NullPointerException`.

Alternatively, if you want to handle the `null` case by returning a specific value (e.g., `0`), you can modify the method like this:

```java
public int getLength(String s) {
    if (s == null) {
        return 0;
    }
    return s.length();
}
```

`NullPointerException` and should not crash when `s` is `null`.

GPT-4o-mini:
The method `getLength` is designed to return the length of a string passed to it. However, there is a potential for a `NullPointerException` if the input string `s` is `null`. In Java, calling a method on a `null` object reference results in this exception. Therefore, if the method is invoked with a `null` argument, it will crash the program.

### Thought Process for Fixing the Code

1. **Identify the Problem**: The primary issue is that if `s` is `null`, invoking `s.length()` will throw a `NullPointerException`.

2. **Determine Desired Behavior**: We need to decide how to handle a `null` input. Common approaches include:
  - Returning a special value (like `0`) to indicate that the string is `null`.
  - Throwing an exception to signal that `null` is not a valid input.

3. **Implement a Solution**: Based on our decision, let's assume we choose to return `0` if `s` is `null`. This approach is generally user-friendly, as it allows the caller to handle the result without worrying about exceptions.

4. **Code Implementation**: We can add a simple check for `null` before attempting to access the `length()` method.

Here is the updated code:

```java
public int getLength(String s) {
    if (s == null) {
        return 0; // Return 0 if the string is null
    }
    return s.length(); // Otherwise return the length of the string
}
```

### Summary of Changes
- Added a check for `null` at the beginning of the method.
- If `s` is `null`, the method returns `0`.
- If `s` is not `null`, it safely calls `s.length()`.

This solution ensures that the method will not crash due to `null` inputs and provides a clear and predictable return value.

This approach returns `0` if the input string is `null`, which can be a reasonable behavior depending on the context of your application.

GPT-4o-mini:
If the method `getLength` is called with a `null` input, it will throw a `NullPointerException` when it tries to access the `length()` method on the `null` object. To prevent this risk, you can modify the method to handle the `null` case gracefully. One common approach is to return `0` when the input string is `null`, or you could throw an `IllegalArgumentException` to indicate that the input is not valid.

Here are two possible implementations:

### Option 1: Return 0 for null input
```java
public int getLength(String s) {
    return (s == null) ? 0 : s.length();
}
```

### Option 2: Throw an exception for null input
```java
public int getLength(String s) {
    if (s == null) {
        throw new IllegalArgumentException("Input string cannot be null");
    }
    return s.length();
}
```

Choose the approach that best fits your application's requirements. If returning `0` is acceptable, go with the first option; if you want to enforce that a valid string is always required, use the second option.

Analysis:

I think Zero-Shot gave a strong, straightforward explanation for handling null checks, directly highlighting defensive coding practices. Few-Shot was concise but left important safeguards less emphasized though. The BLEU score difference wasn’t huge, but Zero-Shot still edged out in clarity.

Task 8: CSV Parser Variants (Python)

Few-Shot (Strategy 2)	Zero-Shot (Strategy 1)
Codestral: To handle quoted fields in a CSV line, you need to account for cases where fields may contain commas and be enclosed in quotes.	Codestral: To handle commas inside quotes correctly, we need to implement a more sophisticated CSV parser that can distinguish between

Here is an updated version of the `parse_csv_line` function that properly handles quoted fields:

```python
import csv
import io

def parse_csv_line(line):
    reader = csv.reader(io.StringIO(line))
    return next(reader)

# Example usage
input_line = '"a,b",c'
parsed_line = parse_csv_line(input_line)
print(parsed_line)  # Output: ['a,b', 'c']
```

### Explanation
1. **csv.reader**: This is a CSV reader that can handle quoted fields properly. It parses the CSV line and handles commas inside quoted strings correctly.
2. **io.StringIO**: This creates a file-like object from the input string. The `csv.reader` can read from this file-like object.
3. **next(reader)**: This retrieves the first (and only) row from the CSV reader.

This approach ensures that the function correctly interprets quoted fields and handles commas within those quoted fields as part of the field content.

### Example Usage
```python
input_line = '"a,b",c'
parsed_line = parse_csv_line(input_line)
print(parsed_line)  # Output: ['a,b', 'c']
```

This will correctly parse the input line `'"a,b",c'` into `['a,b', 'c']`.

GPT-4o-mini:
To update the given CSV parser so that it can handle quoted fields, you can use the `csv` module from Python's standard library. This module is specifically designed to handle CSV data, including cases with quoted fields. Here's how you can modify your function:

```python
import csv
from io import StringIO

def parse_csv_line(line):
    # Use StringIO to create a file-like object from the string
    f = StringIO(line)
    # Use the csv.reader to parse the line
    reader = csv.reader(f)
    # Read the first (and only) row from the reader
    parsed_line = next(reader)
    return parsed_line

# Example usage:
input_line = '"a,b",c'
```

quoted and unquoted fields. Here's a basic implementation in Python:

```python
def parse_csv_line(line):
    fields = []
    field = ''
    in_quotes = False
    escape = False

    for char in line:
        if char == '"':
            if escape:
                field += char
                escape = False
            else:
                in_quotes = not in_quotes
        elif char == ',':
            if in_quotes:
                field += char
            else:
                fields.append(field.strip())
                field = ''
        elif char == '\\' and in_quotes:
            escape = True
            field += char
        else:
            field += char
            escape = False

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

    return fields

# Example usage:
line = 'name,"John, Doe",age,30'
print(parse_csv_line(line))
```

### Explanation:
1. **Initialization**: We initialize an empty list `fields` to store the parsed fields, an empty string `field` to accumulate characters for the current field, and boolean flags `in_quotes` and `escape` to handle quoted fields and escape characters respectively.

2. **Character Loop**: We loop through each character in the line:
  - If the character is a double quote (`"`):
    - If it's escaped, we add it to the current field and clear the escape flag.
    - Otherwise, we toggle the `in_quotes` flag.
  - If the character is a comma (`,`):
    - If we are inside quotes, we add the comma to the current field.
    - Otherwise, we finish the current field, strip any leading/trailing whitespace, add it to the list of fields, and reset the `field` string.
  - If the character is a backslash (`\`) and we are inside quotes, we set the escape flag.
  - Otherwise, we add the character to the current field and clear the escape flag.

result = parse_csv_line(input_line)
print(result) # Output: ['a,b', 'c']

### Explanation:
1. **StringIO**: This is used to convert the string into a file-like object, which allows the `csv.reader` to read from it.
2. **csv.reader**: This function is used to read the CSV data correctly, handling quoted fields properly.
3. **next(reader)**: This retrieves the first row from the reader, which contains the parsed fields.

This implementation will correctly parse lines with quoted fields, commas inside quotes, and other CSV formatting rules.

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

### Example Usage:
The example provided (`line = 'name,"John, Doe",age,30'`) should output:
```python
['name', 'John, Doe', 'age', '30']
```

This implementation should handle commas inside quotes correctly by respecting the CSV formatting rules.

GPT-4o-mini:
To fix the CSV parser so that it correctly handles commas inside quoted fields, you'll need to implement a way to recognize when you're inside quotes and avoid splitting on commas in those cases. Below is a simple implementation that does this:

```python
def parse_csv_line(line):
    result = []
    current_field = ''
    in_quotes = False

    for char in line:
        if char == '"':
            in_quotes = not in_quotes  # Toggle the in_quotes flag
        elif char == ',' and not in_quotes:
            result.append(current_field)  # Add the current field to the
result
            current_field = ''  # Reset the current field
            continue  # Skip adding the comma to the field
        else:
            current_field += char  # Add the character to the current
field

    # Append the last field after the loop
    result.append(current_field)
    return result

# Example usage:
csv_line = 'value1,"value, with, commas",value3'
parsed_line = parse_csv_line(csv_line)
print(parsed_line)  # Output: ['value1', 'value, with, commas',
'value3']
```

### Explanation:
1. **Initialization**: We start with an empty list `result` to hold the parsed fields, an empty string `current_field` to build each field, and a boolean `in_quotes` to track whether we are currently within quotes.

2. **Iterating through characters**: We loop through each character in the input line.
  - If we encounter a double quote (`"`), we toggle the `in_quotes` flag.
  - If we encounter a comma (`,`), we check if we are not inside quotes. If we are outside quotes, we append the current field to the

`result` list and reset `current_field`. If we are inside quotes, we simply add the comma to `current_field`.
   - For any other character, we add it to `current_field`.

3. **Final field**: After the loop, we append any remaining characters in `current_field` to `result`.

This implementation should work correctly for most typical CSV formats that include quoted fields.

Analysis:

When building a manual CSV parser, Zero-Shot carefully explained character handling step-by-step whereas Few-Shot used the Python csv module instead, which was efficient but surprisingly skipped the deeper logic in comparison. Zero-Shot's output showed stronger understanding and matched with a higher BLEU of around 0.24.

Task 9: Data Class to API Conversion (Kotlin)

Role-Playing (Strategy 6)

Codestral:
Certainly! Let's transform the given `Product` data class into an API that can handle GET and POST requests using Ktor.

First, ensure you have the necessary dependencies in your `build.gradle.kts` file:
```kotlin
dependencies {
    implementation("io.ktor:ktor-server-core:2.1.0")
    implementation("io.ktor:ktor-server-netty:2.1.0")
    implementation("io.ktor:ktor-server-content-negotiation:2.1.0")
    implementation("io.ktor:ktor-serialization-kotlinx-json:2.1.0")
    implementation("io.ktor:ktor-server-auth:2.1.0")

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

Next, set up the API endpoints in your Ktor application. Here's an example of how you can do it:

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

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

val productList = mutableListOf(
    Product(1, "Product A", 19.99),
```

Zero-Shot (Strategy 1)

Codestral:
Sure, I'll help you set up a simple Ktor REST API with GET and POST routes using the provided Kotlin data class. Below is a step-by-step guide to achieve this:

1. **Add Dependencies**: First, you need to add the necessary Ktor dependencies to your `build.gradle.kts` file.

```kotlin
dependencies {
    implementation("io.ktor:ktor-server-core:2.1.0")
    implementation("io.ktor:ktor-server-netty:2.1.0")
    implementation("io.ktor:ktor-server-content-negotiation:2.1.0")
    implementation("io.ktor:ktor-serialization-kotlinx-json:2.1.0")
    implementation("io.ktor:ktor-server-status-pages:2.1.0")
    implementation("io.ktor:ktor-server-tests:2.1.0")
}
```

2. **Create the Product Data Class**: Use the provided data class.

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

3. **Set Up the Ktor Application**: Create a Ktor application with the necessary routes.

```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.embeddedServer
import io.ktor.server.netty.Netty

fun main() {
    Product(2, "Product B", 29.99)
)

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json(Json {
                prettyPrint = true
                isLenient = true
            })
        }
        install(StatusPages) {
            exception<Throwable> { cause ->
                call.respond(HttpStatusCode.InternalServerError,
cause.localizedMessage)
            }
        }
        routing {
            get("/products") {
                call.respond(productList)
            }
            post("/products") {
                val product = call.receive<Product>()
                productList.add(product)
                call.respond(HttpStatusCode.Created, product)
            }
        }
    }.start(wait = true)
}

```

### Explanation:
1. **Dependencies**: Ensure you have the necessary dependencies for Ktor and serialization.
2. **Product Data Class**: The `Product` data class is annotated with `@Serializable` to enable serialization.
3. **Product List**: A mutable list `productList` is initialized with some sample products.
4. **Main Function**:
  - `embeddedServer`: Starts an embedded Netty server on port 8080.
  - `install(ContentNegotiation)`: Configures content negotiation to use JSON.
  - `install(StatusPages)`: Configures status pages to handle exceptions.
  - `routing`: Defines the routing for the API endpoints.
    - `get("/products")`: Returns the list of products.
    - `post("/products")`: Receives a new product, adds it to the list, and responds with the created product.

### Running the Server:
To run the server, simply execute the `main` function. The server will start, and you can use tools like `curl` or Postman to interact with the endpoints:

- **GET /products**: Fetch all products.
- **POST /products**: Create a new product.

Example `curl` commands:
```sh
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }
        install(StatusPages) {
            exception<Throwable> { cause ->
                call.respond(HttpStatusCode.InternalServerError, "500:
$cause")
            }
        }
        routing {
            // GET route to retrieve all products
            get("/products") {
                val products = listOf(
                    Product(1, "Product A", 10.0),
                    Product(2, "Product B", 20.0)
                )
                call.respond(products)
            }

            // POST route to create a new product
            post("/products") {
                val product = call.receive<Product>()
                // Here you would typically save the product to a
database
                call.respond(HttpStatusCode.Created, product)
            }
        }
    }.start(wait = true)
}
```

4. **Run the Application**: Run the `main` function to start the Ktor server. You should now have a simple REST API with:

- **GET /products**: Retrieves a list of products.
- **POST /products**: Creates a new product.

This setup includes basic error handling and JSON serialization using Ktor's Content Negotiation feature. You can expand this by adding a database to persist products, more routes, and additional error handling as needed.

GPT-4o-mini:
To set up a simple Ktor REST API with GET and POST routes using the provided Kotlin data class `Product`, you can follow the steps outlined below. In this example, I'll show you how to create the Ktor application with the necessary dependencies, configure routing, and handle both GET and POST requests.

### Step 1: Set Up Your Project

1. **Create a new Ktor project**. You can use IntelliJ IDEA or any other Kotlin-compatible IDE. If you're using IntelliJ, you can create a new Ktor project from the project wizard.

2. **Add Dependencies**: Make sure to include the required Ktor dependencies in your `build.gradle.kts` file (or `pom.xml` if you are using Maven). Here is an example of how to set up your `build.gradle.kts`:

~~~
# GET all products
curl -X GET http://localhost:8080/products

# POST a new product
curl -X POST http://localhost:8080/products \
    -H "Content-Type: application/json" \
    -d '{"id": 3, "name": "Product C", "price": 39.99}'
```

This setup should give you a functional Ktor-based API for
handling `Product` objects.

GPT-4o-mini:
To create a simple API using Ktor that can handle `Product`
objects with GET and POST endpoints, follow these steps:

1. **Set up your Ktor project**: Make sure you have the
necessary dependencies in your `build.gradle.kts` or `pom.xml`
file. For a Gradle setup, you might include:

  ```kotlin
  dependencies {
     implementation("io.ktor:ktor-server-core:2.1.0") // Use the
latest version
     implementation("io.ktor:ktor-server-netty:2.1.0")
     implementation("io.ktor:ktor-server-json:2.1.0")
     implementation("io.ktor:ktor-serialization-kotlinx-json:2.1.0")

implementation("io.ktor:ktor-server-content-negotiation:2.1.0")
     testImplementation("io.ktor:ktor-server-tests:2.1.0")
  }
  ```

2. **Create the `Product` data class**:

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

3. **Set up Ktor application with GET and POST endpoints**:

Here's how you can implement a simple Ktor application that
provides GET and POST endpoints for `Product` objects:

```kotlin
import io.ktor.application.*
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.*
import io.ktor.features.ContentNegotiation
import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json

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

// In-memory storage for simplicity
~~~

~~~
```kotlin
plugins {
  application
  kotlin("jvm") version "1.8.0"
  id("io.ktor.plugin") version "2.2.2"
}

repositories {
  mavenCentral()
}

dependencies {
  implementation("io.ktor:ktor-server-core:2.2.2")
  implementation("io.ktor:ktor-server-netty:2.2.2")
  implementation("io.ktor:ktor-serialization:2.2.2")
  implementation("io.ktor:ktor-server-content-negotiation:2.2.2")
  implementation("io.ktor:ktor-serialization-kotlinx-json:2.2.2")
  testImplementation("io.ktor:ktor-server-tests:2.2.2")
  testImplementation("org.jetbrains.kotlin:kotlin-test:1.8.0")
}

application {
  mainClass.set("com.example.ApplicationKt") // Adjust
according to your package structure
}
```

### Step 2: Create the Ktor Application

Now, create a new Kotlin file (e.g., `Application.kt`) and set up the
Ktor application with the necessary routes.

```kotlin
package com.example

import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.serialization.kotlinx.json.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

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

fun main() {
  embeddedServer(Netty, port = 8080, module =
Application::module).start(wait = true)
}

fun Application.module() {
  install(ContentNegotiation) {
    json()
  }

  val products = mutableListOf<Product>()

  routing {
    // GET endpoint to retrieve all products
~~~

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

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

        routing {
            // GET endpoint to retrieve all products
            get("/products") {
                call.respond(products)
            }

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

### Explanation:

1. **Dependencies**: The code uses Ktor for building the server, along with JSON serialization for handling `Product` objects.

2. **Product Data Class**: The `Product` class is marked with `@Serializable`, which allows it to be serialized and deserialized when handling requests and responses.

3. **In-memory storage**: A mutable list `products` is used to store products in memory. In a production scenario, you would likely use a database.

4. **Routing**:
  - **GET `/products`**: This endpoint returns the list of all products in JSON format.
  - **POST `/products`**: This endpoint accepts a `Product` object in the request body, adds it to the in-memory list, and responds with a `201 Created` status along with the created product.

### Running the Server:
- Start the server by running the `main` function.
- You can test the API using tools like Postman or cURL.

### Example Requests:
- **GET Products**:
  ```
  GET http://localhost:8080/products
  ```

- **POST Product**:
  ```
  POST http://localhost:8080/products
  Content-Type: application/json

---

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

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

### Step 3: Run Your Application

1. **Run your application** by executing the `main` function. The Ktor server will start, and you will be able to access it at `http://localhost:8080`.

### Step 4: Test Your API

You can test your API using tools like Postman, cURL, or any HTTP client:

- **GET Request**: To get all products, send a GET request to `http://localhost:8080/products`.

- **POST Request**: To add a new product, send a POST request to `http://localhost:8080/products` with a JSON body. For example:

```json
{
    "id": 1,
    "name": "Sample Product",
    "price": 19.99
}
```

### Example cURL Commands

- **GET all products**:
```bash
curl -X GET http://localhost:8080/products
```

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

That's it! You now have a simple Ktor REST API with GET and POST routes for managing `Product` objects.

{
"id": 1,
"name": "Sample Product",
"price": 29.99
}
```
This setup provides a basic framework to get you started with a Ktor application that handles `Product` objects. You can expand it by adding features like error handling, validation, or persistence with a database.
```

Analysis:

Zero-Shot's Ktor setup guide was extremely basic, leaving out context around dependencies and routes. Prompt Chaining gave a full setup walkthrough, making the structure easier to grasp. The better BLEU score for Prompt Chaining (around 0.26) was definitely the more detailed explanation.

Task 10: Function Summarization (Python)

Few-Shot (Strategy 2)	Zero-Shot (Strategy 1)

Codestral:
Certainly! The function `reverse_words` takes a sentence as input and returns the sentence with the order of words reversed.
 	
Here's the summary:
```python
def reverse_words(sentence):
    return ' '.join(sentence.split()[::-1])
```
→ Reverses the order of words in the input sentence.
 	
GPT-4o-mini:
Reverses the order of words in the input sentence.
	Codestral:
	This Python function takes a sentence as input and returns the sentence with the words in reverse order, maintaining the word sequence but reversing the order of individual words.
	
	GPT-4o-mini:
	This Python function takes a sentence as input and returns a new string with the order of the words reversed. It splits the sentence into words, reverses the list of words, and then joins them back together with spaces.

Analysis:

When describing the function, Zero-Shot included purpose, structure, and key considerations, while Few-Shot provided a quick but thinner explanation. I noticed the BLEU favored Zero-Shot slightly, and its fuller description made it the stronger output overall.

Task 11: Prompt from Code Comments (Python)

Prompt Chaining (Strategy 5)	Zero-Shot (Strategy 1)
Codestral: To reverse-engineer a prompt that would lead a model to write the given function, you need to specify the requirements clearly. Here is a specific prompt that would guide the model to write the `is_prime` function: --- **Prompt:** "Write a Python function named `is_prime` that takes a single integer parameter `n` and returns `True` if the number is prime, and `False` otherwise. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. Your function should:	Codestral: To create a good prompt for generating the correct function, you should provide clear instructions and context. Here is an example of how you could structure such a prompt: --- **Prompt:** You are required to write a Python function that checks if a given number is prime. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. The function should have the following signature: ```python

1. Return `False` immediately if `n` is less than or equal to 1.
2. Iterate from 2 to the square root of `n` (inclusive) to check divisibility.
3. Return `False` if `n` is divisible by any number in this range.
4. Return `True` if no divisors are found in the range.

Include appropriate comments and ensure the function is well-structured."

This prompt provides clear instructions on the requirements, the conditions to check, and the expected behavior of the function, which should help in generating the correct code.

GPT-4o-mini:
Here's a prompt that would lead a model to write the same function based on the provided comment:

Prompt:

Write a Python function named `is_prime` that takes a single integer parameter `n`. The function should determine if `n` is a prime number.

1. A prime number is defined as a number greater than 1 that has no positive divisors other than 1 and itself.
2. If `n` is less than or equal to 1, the function should return `False`.
3. If `n` is greater than 1, the function should check for factors from 2 up to the square root of `n`. If any of these numbers divide `n` evenly (i.e., `n % i == 0`), the function should return `False`.
4. If no factors are found, the function should return `True`.

Make sure to handle the indentation correctly in your Python code.

```
def is_prime(n: int) -> bool:
```

Here are the requirements for the function:
1. The function should return `False` if the input number `n` is less than or equal to 1.
2. The function should check for factors from 2 up to the square root of `n` (inclusive).
3. If `n` is divisible by any of these factors, it should return `False`.
4. If no factors are found, the function should return `True`.

Ensure that the function is well-commented and follows Python best practices.

This prompt is detailed enough to guide the model in generating the correct function while ensuring clarity and correctness. Here is the expected correct function based on the prompt:

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

This function adheres to the specified requirements and provides the correct implementation.

GPT-4o-mini:
To generate a function that correctly checks if a number is prime, you might use a prompt like the following:

"Write a Python function that checks if a number is prime. The function should take a single integer input `n` and return `True` if `n` is a prime number and `False` otherwise. Remember that a prime number is a number greater than 1 that has no positive divisors other than 1 and itself. Ensure your code is properly indented and handles edge cases like negative numbers and 0."

This prompt clearly specifies the requirements and includes details about the behavior of prime numbers to guide the model in generating the correct function.

Analysis:

The Few-Shot strategy seemed to generate cleaner prompts for the prime-checking task, avoiding the verbosity that showed up in Zero-Shot. The BLEU score for Few-Shot (around 0.29) reflects the smoother, more natural prompt design.

Task 12: Fixing Factorial Bug (Python)

Chain-of-Thought (Strategy 3)

Codestral:
The provided code is intended to calculate the factorial of a number \(n \), but there are a few issues that need to be addressed.

### Issues and Edge Cases

1. **Loop Range**: The loop `for i in range(1, n):` should include \(n \) to properly compute the factorial. Currently, it stops at \(n-1 \), which means it doesn't multiply by \(n \).
2. **Return Statement**: The `return result` statement is not properly indented and outside the loop, which will cause a syntax error.
3. **Input Validation**: There is no validation for the input \(n \). For example, if \(n \) is negative or not an integer, the function should handle these cases.
4. **Edge Case for Zero**: The factorial of 0 (0!) should be handled specifically, as it equals 1.

### Fixes

1. **Loop Range**: Change the loop to `for i in range(1, n + 1):` to include \(n \).
2. **Indentation**: Correct the indentation of the return statement.
3. **Input Validation**: Add checks for non-negative integers.
4. **Edge Case for Zero**: Handle the case where \(n \) is 0.

Here is the corrected code:

```python
def factorial(n):
    if not isinstance(n, int) or n < 0:
        raise ValueError("Input must be a non-negative integer.")

    if n == 0:
        return 1

    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Example usage:
print(factorial(5))  # Output: 120
```

### Explanation of Fixes

1. **Loop Range**: The loop `for i in range(1, n + 1):` ensures that the loop runs from 1 to \(n \), inclusive, thereby correctly computing the factorial.
2. **Return Statement**: The `return result` statement is now properly indented inside the function.
3. **Input Validation**: The function checks if \(n \) is a non-negative integer. If not, it raises a `ValueError`.
4. **Edge Case for Zero**: The function explicitly checks if \(n \) is 0 and returns 1 immediately, since 0! is 1.

Zero-Shot (Strategy 1)

Codestral:
The issue with the provided factorial function is that it does not handle the case when the input `n` is 0. Factorial of 0 is defined as 1, but the current implementation returns 1 only for values greater than 1. When `n` is 0, the `range(1, n)` generates an empty sequence, and the loop doesn't execute, resulting in the function returning 1 incorrectly.

To fix this, you need to add a check for `n` being 0 and return 1 immediately if that is the case. Additionally, the current loop should iterate from 1 to `n` inclusive to correctly calculate the factorial.

Here's the corrected version of the function:

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

### Explanation:
1. **Check for `n` being 0:**
   ```python
   if n == 0:
       return 1
   ```
   This line ensures that if `n` is 0, the function immediately returns 1, which is the correct value for 0!.

2. **Correct Range for Loop:**
   ```python
   for i in range(1, n + 1):
   ```
   By using `range(1, n + 1)`, the loop now correctly iterates from 1 to `n` inclusive, ensuring that all values from 1 to `n` are multiplied together to get the correct factorial value.

### Example Usage:
```python
print(factorial(0))  # Output: 1
print(factorial(5))  # Output: 120
```

This corrected version correctly handles the case when `n` is 0 and computes the factorial for any non-negative integer `n`.

GPT-4o-mini:
The issue with the provided factorial function is that it does not handle the case where the input `n` is 0. According to the definition of factorial, the factorial of 0 (denoted as `0!`) is defined to be 1.

With these changes, the function should work correctly for all valid inputs and handle edge cases appropriately.

GPT-4o-mini:
The provided `factorial` function contains a logic error that causes it to incorrectly calculate the factorial of a number. Let's analyze the code:

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

### Loop Logic

1. The function initializes `result` to 1.
2. It iterates through a range from 1 to `n-1` (because the range is exclusive of the stop value `n`).
3. In each iteration, it multiplies `result` by `i`.
4. Finally, it returns `result`.

### Edge Case

The calculation of the factorial is incorrect because:
- The loop only runs from 1 to `n-1`, meaning it doesn't include `n` in the multiplication.
- Additionally, if `n` is 0 or 1, the function will return 1, which is correct for `0!` and `1!`, but it will fail to handle the case of `n = 0` properly when following the standard mathematical definition of factorial.

### What Breaks

1. For `n = 0`, the factorial should return `1` (0! = 1), but since the loop does not execute, the function currently returns `1`, which is correct.
2. For `n = 1`, the loop does not execute and returns `1`, which is correct since `1! = 1`.
3. For `n = 2`, it computes `1!` (which is `1`), but it should compute `2!` (which is `2`).
4. For `n = 3`, it computes `1 * 2`, which results in `2`, but it should compute `3!` (which is `6`).

### Fix

To fix the code, we need to adjust the range in the loop to include `n`, and ensure that we handle the case when `n` is 0 correctly. Here's the corrected version:

```python
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative 
numbers.")
    result = 1
    for i in range(1, n + 1):  # Change n to n + 1 to include n
        result *= i
    return result
```

The current implementation of the function will return 1 for `n = 0`, but this is due to the way the loop is structured rather than an explicit handling of the case. However, it's good practice to explicitly handle the case where `n` is 0 to improve code readability and correctness.

Here’s an updated version of the factorial function that explicitly checks for the case when `n` is 0:

```python
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative 
numbers.")
    elif n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):  # Include n in the calculation
            result *= i
        return result
```

### Explanation of Changes:

1. **Zero Case Handling**:
   - We added an explicit check for when `n` equals 0. If `n` is 0, the function returns 1 immediately, aligning with the mathematical definition of factorial.

2. **Negative Input Handling**:
   - We included a check for negative input values. Factorials are not defined for negative numbers, so the function raises a `ValueError` if `n` is negative.

3. **Loop Adjustment**:
   - The loop now iterates up to and including `n` (i.e., `range(1, n + 1)`). This is necessary because the factorial of a number `n` is the product of all positive integers from 1 to `n`. The original implementation would have skipped `n` entirely, which would lead to incorrect results for positive integers.

### Why This Version Works for Zero:
- By explicitly checking if `n` is 0 and returning 1, we ensure that the function behaves correctly for this edge case. The mathematical definition states that `0! = 1`, so our function reflects this definition directly in its logic. This makes the function more robust and clear, ensuring that users of the function can rely on it to provide correct results across the entire range of valid inputs.

~~~
```
~~~

### Reason for Fix

1. **Loop Adjustment**: The loop now runs from 1 to `n`, inclusive, allowing proper computation of the factorial.
2. **Negative Input Handling**: The added check raises an exception for negative numbers since the factorial is not defined for them. This adds robustness to the function.

Now the function will correctly compute the factorial for all non-negative integers.

Analysis:

Zero-Shot fixed the factorial loop cleanly without overcomplicating it, keeping the correction focused. Chain-of-Thought added extra input validation, but sometimes it distracted from the main point. The slightly higher BLEU for Zero-Shot confirmed the tighter focus helped.

Task 13: Linked List Node Deletion (C)

Role-Playing (Strategy 6)

Codestral:
Certainly! Below is a C implementation of the `deleteNode` function that handles the deletion of a node by value, including edge cases such as deleting the head node.

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

// Define the structure for a node
struct Node {
    int data;
    struct Node *next;
};

// Function to delete a node by value
void deleteNode(struct Node **head, int key) {
    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;
        free(temp);
        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 the list
    if (temp == NULL) return;

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

Zero-Shot (Strategy 1)

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

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

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

void deleteNode(struct Node **head, int key) {
    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; // Changed 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
}
```

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

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

// Function to push a new node to the front of the list
void push(struct Node **head, int new_data) {
    struct Node *new_node = (struct Node *)malloc(sizeof(struct
Node));
    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

int main() {
    struct Node *head = NULL;

    // Create a linked list: 7 -> 1 -> 3 -> 2
    push(&head, 2);
    push(&head, 3);
    push(&head, 1);
    push(&head, 7);

    printf("Linked list before deletion: ");
    printList(head);

    // Delete node with value 1
    deleteNode(&head, 1);
    printf("Linked list after deleting 1: ");
    printList(head);

    // Delete head node (node with value 7)
    deleteNode(&head, 7);
    printf("Linked list after deleting 7: ");
    printList(head);

    // Delete a node that does not exist
    deleteNode(&head, 5);
    printf("Linked list after trying to delete 5: ");
    printList(head);

    return 0;
}
```

### Explanation:
1. **Node Structure**: The `Node` structure represents a node in the linked list.
2. **deleteNode Function**:
   - `temp` points to the current node being checked.
   - `prev` points to the node before `temp`.
   - The function first checks if the head node needs to be deleted.
   - It then traverses the list to find the node with the specified key.

```
void printList(struct Node *node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    // Allocate 3 nodes in the heap
    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;

    second->data = 2;
    second->next = third;

    third->data = 3;
    third->next = NULL;

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

    deleteNode(&head, 2);

    printf("List after deleting 2: ");
    printList(head);

    deleteNode(&head, 1);

    printf("List after deleting 1: ");
    printList(head);

    deleteNode(&head, 3);

    printf("List after deleting 3: ");
    printList(head);

    return 0;
}
```

### Explanation:
1. **Check if the head node itself holds the key:**
  - If the head node's data matches the key, update the head pointer to the next node and free the old head node.

2. **Search for the key in the linked list:**
  - Traverse the list while keeping track of the previous node.
  - If the key is found, unlink the node by updating the previous node's `next` pointer to skip over the current node.
  - Free the memory of the node to be deleted.

   - If the key is found, it updates the `next` pointer of the previous node to skip the current node.
   - Finally, it frees the memory of the node to be deleted.
3. **printList Function**: Utility function to print the linked list.
4. **push Function**: Utility function to add a new node at the beginning of the list.
5. **main Function**: Demonstrates the deletion functionality by creating a linked list, deleting nodes, and printing the list after each deletion.

This implementation handles all the specified edge cases, including deleting the head node and handling attempts to delete a non-existent node.

GPT-4o-mini:
To implement a function that deletes a node by its value from a singly linked list in C, we need to handle several edge cases, including:

1. Deleting the head of the list.
2. Deleting a node that is in the middle or end of the list.
3. Deleting a node that does not exist in the list.
4. Handling an empty list.

Below is the complete implementation of the `deleteNode` function along with a simple linked list setup for demonstration.

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

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

// Function to delete a node by value (key)
void deleteNode(struct Node** head, int key) {
    // Store the head node
    struct Node* temp = *head;
    struct Node* prev = NULL;

    // If the list is empty
    if (temp == NULL) {
        printf("List is empty. Cannot delete %d.\n", key);
        return;
    }

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

    // Search for the key to be deleted, maintain a pointer to the previous node
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
```

3. **Handle the case where the key is not present in the list:**
   - If the loop completes without finding the key, do nothing.

### Additional Functions:
- `printList`: A utility function to print the linked list.
- `main`: Demonstrates the usage of `deleteNode` by creating a simple linked list and deleting nodes with specific values.

This implementation ensures that the node with the specified key is deleted from the linked list correctly.

GPT-4o-mini:
To implement the logic for deleting a node with a given value from a linked list in C, we need to consider several cases:

1. The list is empty (head is NULL).
2. The node to be deleted is the head node.
3. The node to be deleted is in the middle or at the end of the list.

Here’s how you can implement the `deleteNode` function:

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

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

void deleteNode(struct Node **head, int key) {
    // Case 1: The list is empty
    if (*head == NULL) {
        return;
    }

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

    // Case 2: The node to be deleted is the head node
    if (temp != NULL && temp->data == key) {
        *head = temp->next; // Change head
        free(temp);         // Free the old head
        return;
    }

    // Case 3: The node to be deleted is not the head
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

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

    // Unlink the node from the linked list
    prev->next = temp->next;
    free(temp); // Free the memory allocated for the node
}
```

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

    // Unlink the node from the linked list
    prev->next = temp->next;
    free(temp); // Free memory
}

// Function to push a new node at the beginning of the linked list
void push(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct
Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    *head_ref = new_node;
}

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

// Main function to demonstrate the deleteNode function
int main() {
    struct Node* head = NULL;

    // Create a linked list: 1->2->3->4->5
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    printf("Initial Linked List: ");
    printList(head);

    deleteNode(&head, 1); // Deleting head
    printf("Linked List after deleting 1: ");
    printList(head);

    deleteNode(&head, 3); // Deleting a middle node
    printf("Linked List after deleting 3: ");
    printList(head);

    deleteNode(&head, 5); // Deleting last node
    printf("Linked List after deleting 5: ");
    printList(head);

    deleteNode(&head, 10); // Attempting to delete a non-existent
node

    return 0;
}
```

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

// Helper function to print the linked list
void printList(struct Node *node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

// Example usage
int main() {
    struct Node* head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(3);
    head->next->next->next = newNode(4);

    printf("Linked list before deletion:\n");
    printList(head);

    deleteNode(&head, 3); // Delete node with value 3

    printf("Linked list after deletion:\n");
    printList(head);

    // Free remaining nodes
    deleteNode(&head, 1);
    deleteNode(&head, 2);
    deleteNode(&head, 4);

    return 0;
}
```

### Explanation:
1. **Check if the list is empty:** If the head is NULL, we simply return, as there is nothing to delete.
2. **Delete the head node:** If the head node's data matches the key, we update the head to the next node and free the old head.
3. **Traverse the list:** We traverse the list while keeping track of the previous node (`prev`). If we find the node with the matching key, we unlink it from the list by setting `prev->next` to `temp->next`.
4. **Free the node:** If the node is found and unlinked, we free its memory.

### Note:
Always ensure to free the memory allocated for nodes to prevent memory leaks. The helper functions `newNode` and `printList` are provided to facilitate the creation of nodes and displaying the list for testing purposes.

```
### Explanation:
1. Struct Definition: We define the `Node` structure to represent each element in the linked list.
2. Delete Function: The `deleteNode` function takes a double pointer to the head of the list and the `key` to be deleted. It handles:
- The case where the list is empty.
- The case where the head node matches the key.
- Traversing the list to find the node with the matching key while keeping track of the previous node.
- Unlinking the node and freeing its memory.
3. Helper Functions: `push` is used to add nodes to the front of the list, and `printList` is used to display the list.
4. Main Function: It demonstrates the creation of a linked list, deletion of nodes, and displays the list after each deletion.

This implementation captures the necessary edge cases and maintains the integrity of the linked list.
```

Analysis:

Zero-Shot's explanation for deleting nodes in a linked list was short and clear, sticking to the core logic which is good for digestability. Self-Consistency expanded the discussion with helper functions but didn't improve readability. BLEU scores were close, but Zero-Shot felt easier to follow.

Task 14: Recursive Function Completion (Python)

Chain-of-Thought (Strategy 3)	Zero-Shot (Strategy 1)
Codestral: Sure, let's break down the logic of the Fibonacci sequence and fill in the base cases and recursive call step by step. ### Fibonacci Sequence Overview The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. The sequence goes like this: 0, 1, 1, 2, 3, 5, 8, 13, ... Mathematically: - \\(F(0) = 0 \\) - \\(F(1) = 1 \\) - \\(F(n) = F(n-1) + F(n-2) \\) for \\(n \geq 2 \\) ### Steps to Implement Fibonacci Recursively 1. **Base Cases**: - If \\(n = 0 \\), return 0. - If \\(n = 1 \\), return 1. 2. **Recursive Call**: - For \\(n \geq 2 \\), return the sum of the Fibonacci numbers of \\(n-1 \\) and \\(n-2 \\). ### Implementation Let's put this all together in the `fibonacci` function:	Codestral: Sure, here is a basic implementation of the Fibonacci function using recursion: This function works by defining two base cases for \\(n = 0 \\) and \\(n = 1 \\), and then recursively calling itself to compute the Fibonacci number for \\(n \\) by summing the results of the two preceding numbers in the sequence. GPT-4o-mini: You can complete the `fibonacci` function by adding the base cases and the recursive call. Here's a simple implementation:

Zero-Shot (Strategy 1)

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

# Example usage:
print(fibonacci(10))  # This will print the 10th Fibonacci number
```

```python
def fibonacci(n):
    # Base case: If n is 0, return 0
    if n == 0:
        return 0

    # Base case: If n is 1, return 1
    if n == 1:
        return 1

    # Recursive call: Return the sum of the Fibonacci numbers of n-1 and n-2
    return fibonacci(n - 1) + fibonacci(n - 2)
```

### Explanation

1. **Base Cases**:
   - `if n == 0: return 0`: This handles the smallest Fibonacci number.
   - `if n == 1: return 1`: This handles the second smallest Fibonacci number.

2. **Recursive Call**:
   - `return fibonacci(n - 1) + fibonacci(n - 2)`: For any \(n \geq 2 \), the function calls itself with \(n-1 \) and \(n-2 \) and returns their sum.

### Example Usage

```python
print(fibonacci(0))  # Output: 0
print(fibonacci(1))  # Output: 1
print(fibonacci(5))  # Output: 5
print(fibonacci(10)) # Output: 55
```

The recursive approach is straightforward but can be inefficient for large values of \(n \) due to repeated calculations. For better performance, consider using memoization or an iterative approach.

GPT-4o-mini:
To implement the Fibonacci function using recursion, we need to identify the base cases and the recursive call. The Fibonacci sequence is defined as follows:

- **Base cases**:
  - `fibonacci(0) = 0`
  - `fibonacci(1) = 1`

- **Recursive case**:
  - For `n > 1`, the Fibonacci number can be found by summing the two preceding Fibonacci numbers:
    - `fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)`

Now, let's fill in the `fibonacci` function step by step with these considerations:

```python
def fibonacci(n):
```

```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)

# Example usage:
print(fibonacci(5))  # Output: 5
print(fibonacci(10)) # Output: 55
```

In this implementation:
- The base cases return 0 for `fibonacci(0)` and 1 for `fibonacci(1)`.
- For values of `n` greater than 1, the function calls itself recursively to sum the two preceding Fibonacci numbers.

~~~~
    # Base cases
    if n == 0:
        return 0
    elif n == 1:
        return 1

    # Recursive call
    return fibonacci(n-1) + fibonacci(n-2)
```

### Explanation:
1. **Base Cases**:
   - When `n` is `0`, the function returns `0` since the 0th Fibonacci number is `0`.
   - When `n` is `1`, the function returns `1` since the 1st Fibonacci number is `1`.

2. **Recursive Call**:
   - For any number greater than `1`, the function calls itself twice: once with `n-1` and once with `n-2`, and adds the results of these two calls together.

This implementation will correctly compute the Fibonacci number for any non-negative integer `n`. However, it is worth noting that this naive recursive approach can be inefficient for large `n` due to repeated calculations. Using memoization or an iterative approach can help improve performance.
~~~~

Analysis:

Zero-Shot produced a direct recursive solution for Fibonacci generation that stayed focused on implementing the logic cleanly, without drifting into unnecessary detail. Chain-of-Thought, on the other hand, spent more time explaining the mathematical background, which made the response more educational but slightly harder to navigate if someone was mainly looking for the code solution. BLEU slightly favored Zero-Shot, reflecting its more straightforward and task-focused delivery.

Task 15: Constructor Completion (Python)

Few-Shot (Strategy 2)

Codestral:

~~~~
Certainly! You can modify the constructor for the `Person` class to take `name`, `age`, and an optional `email` parameter. Here's how you can do it:

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

# Example usage:
person1 = Person("Alice", 30)
person2 = Person("Bob", 25, "bob@example.com")

print(person1.name, person1.age, person1.email)  # Output: Alice 30 None
print(person2.name, person2.age, person2.email)  # Output: Bob 25 bob@example.com
```
~~~~

Zero-Shot (Strategy 1)

Codestral:

~~~~
To complete the constructor for the `Person` class, you need to add parameters for `name`, `age`, and an optional `email`. The optional `email` can be set to `None` by default if it is not provided. Here is the completed constructor:

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

In this constructor:
- `name` is a required parameter.
- `age` is a required parameter.
- `email` is an optional parameter with a default value of `None`.

This allows you to create instances of the `Person` class with either two or three arguments:
~~~~

In this implementation:
- The `name` and `age` parameters are required.
- The `email` parameter is optional and defaults to `None` if not provided.

GPT-4o-mini:
Here's a constructor for the `Person` class that includes `name`, `age`, and an optional `email` parameter:

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

In this implementation:
- `name` and `age` are required parameters.
- `email` is an optional parameter that defaults to `None` if not provided.

```python
person1 = Person("John Doe", 30)
person2 = Person("Jane Doe", 25, "jane.doe@example.com")

print(person1.name)  # Output: John Doe
print(person1.age)    # Output: 30
print(person1.email)  # Output: None

print(person2.name)    # Output: Jane Doe
print(person2.age)      # Output: 25
print(person2.email)    # Output: jane.doe@example.com
```

GPT-4o-mini:
You can complete the constructor for the `Person` class by adding parameters for `name`, `age`, and an optional `email`. Here's how you can do it:

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

# Example of creating an instance of Person
person1 = Person("Alice", 30, "alice@example.com")
person2 = Person("Bob", 25)  # email is optional

print(person1.name, person1.age, person1.email)  # Output: Alice 30 alice@example.com
print(person2.name, person2.age, person2.email)  # Output: Bob 25 None
```

In this implementation:
- `name` and `age` are required parameters.
- `email` is an optional parameter with a default value of `None`. If no email is provided, it will simply be set to `None`.

Analysis:

In the task of designing the Person class, Zero-Shot effectively covered the required and optional fields while also providing useful example usages that clarified the intent behind each parameter. Few-Shot's output was more compact and functional but lost some helpful explanatory context that could aid a new developer in understanding the purpose of each attribute. Zero-Shot's slightly stronger BLEU score aligned well with the broader clarity and usability of its explanation.

Task 16: Binary Search Completion (Java)

Chain-of-Thought (Strategy 3)

Codestral:
The provided Java binary search implementation is incomplete because it lacks the logic to compare the middle element of the array with the target and adjust the bounds (`left` and `right`) accordingly. Here's a step-by-step explanation of where things could go wrong and how to fill in the missing logic:

### Potential Issues:
1. **No Comparison**: Without comparing the middle element to the target, the algorithm cannot determine whether to search the

Zero-Shot (Strategy 1)

Codestral:
To complete the binary search method, you need to implement the logic to compare the middle element of the array with the target value and adjust the bounds accordingly. Here is how you can do it:

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

left half, the right half, or if the target has been found.
2. **No Bound Adjustment**: Without adjusting the bounds (`left` or `right`), the loop will either run indefinitely or fail to narrow down the search space properly.
3. **Missing Return Statement**: If the target is found, the current implementation will not return the index of the target. Instead, it will return `-1` after the loop ends.

### Complete Implementation:
Here's the completed binary search implementation with explanations:

```java
public int binarySearch(int[] arr, int target) {
    int left = 0;
    int 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
        }

        // 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;
        }
    }

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

### Explanation:
1. **Comparison**:
  - The `if (arr[mid] == target)` statement checks if the middle element is the target.
  - If it is, the function returns the index `mid`.

2. **Bound Adjustment**:
  - If the target is greater than `arr[mid]`, the left bound (`left`) is adjusted to `mid + 1` to search the right half.
  - If the target is less than `arr[mid]`, the right bound (`right`) is adjusted to `mid - 1` to search the left half.

3. **Loop Condition**:
  - The `while (left <= right)` condition ensures that the loop continues until the bounds cross each other, indicating that the target is not in the array.

4. **Return `-1`**:
  - If the loop exits without finding the target, the function returns `-1` to indicate that the target is not present in the array.

By filling in these details, the binary search algorithm is complete

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

        if (arr[mid] == target) {
            return mid;  // Target found
        } else if (arr[mid] < target) {
            left = mid + 1;  // Search in the right half
        } else {
            right = mid - 1;  // Search in the left half
        }
    }

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

Explanation:
1. **Initialization**: `left` is set to the start of the array, and `right` is set to the end of the array.
2. **While Loop**: The loop continues as long as `left` is less than or equal to `right`.
3. **Mid Calculation**: Calculate the middle index `mid`. We use `left + (right - left) / 2` to avoid potential integer overflow.
4. **Comparison**:
  - If `arr[mid]` is equal to `target`, return `mid` as the index of the target value.
  - If `arr[mid]` is less than `target`, adjust the `left` bound to `mid + 1` to search in the right half.
  - If `arr[mid]` is greater than `target`, adjust the `right` bound to `mid - 1` to search in the left half.
5. **Return -1**: If the target value is not found after the loop, return -1.

This method ensures that the binary search algorithm is correctly implemented to find the target value in the sorted array.

GPT-4o-mini:
To complete the binary search method, we need to compare the middle element of the array with the target value and adjust the bounds accordingly. Here's the completed method:

```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; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

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

and functional for searching a sorted array for a target value.

GPT-4o-mini:
The provided Java binary search implementation is incomplete because it lacks the essential logic for comparing the target value with the middle element of the array and adjusting the search bounds accordingly.

### Key Missing Logic
1. **Comparison with the Middle Element**: The algorithm needs to compare the `target` value with the element at the `mid` index of the array.

2. **Adjusting Bounds**:
  - If the `target` is equal to the middle element, the search is successful, and you should return the `mid` index.
  - If the `target` is less than the middle element, it means the target can only be in the left half of the array. Therefore, adjust the `right` boundary to `mid - 1`.
  - If the `target` is greater than the middle element, adjust the `left` boundary to `mid + 1`, indicating the target can only be in the right half.

### Completed Implementation
Here is the corrected and complete implementation of the binary search algorithm:

```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
        } 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 Logic
- **Initialization**: We start with two pointers, `left` and `right`, which represent the current bounds of the search space.
- **Looping**: The `while` loop continues until `left` exceeds `right`, indicating that the search space is exhausted.
- **Finding Midpoint**: The midpoint `mid` is calculated as the average of `left` and `right`.
- **Condition Checking**:
  - If `arr[mid]` matches `target`, the search is successful, and we return `mid`.
  - If `arr[mid]` is less than `target`, we discard the left half by moving `left` to `mid + 1`.
  - If `arr[mid]` is greater than `target`, we discard the right half by

```

### Explanation:
1. We initialize two pointers, `left` and `right`, which represent the current bounds of the search.
2. We enter a loop that continues as long as `left` is less than or equal to `right`.
3. We calculate the middle index `mid`.
4. We compare the middle element `arr[mid]` with the `target`:
  - If they are equal, we return `mid` (the index of the target).
  - If the middle element is less than the target, we adjust the `left` pointer to `mid + 1`, effectively narrowing the search to the right half of the array.
  - If the middle element is greater than the target, we adjust the `right` pointer to `mid - 1`, narrowing the search to the left half of the array.
5. If the loop ends without finding the target, we return `-1` to indicate that the target is not present in the array.
```

moving `right` to `mid - 1`.
- **Return Value**: If the loop completes without finding the target, return `-1` to indicate that the target is not in the array.

### Conclusion
If the comparison and bounds adjustment logic were omitted, the function would either return an incorrect result or enter an infinite loop, as it would not effectively narrow down the search space. This is crucial for the efficiency and correctness of the binary search algorithm.

Analysis:

When describing binary search, Self-Consistency maintained better structural organization throughout the explanation, while Zero-Shot tended to be more verbose and meandering, occasionally diluting the key ideas. Self-Consistency presented a clear, step-by-step breakdown that made following the search logic much easier, especially for readers less familiar with the algorithm. BLEU reflected this difference, with Self-Consistency earning a noticeably stronger score that matched its tighter structure.

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

Role-Playing (Strategy 6)

Codestral:
The inconsistency lies in the function name `isOdd` and the logic inside the function. The function is named `isOdd` but it actually returns `true` if `x` is even due to the condition `x % 2 == 0`. Therefore, the function should be renamed to accurately reflect its behavior.

The corrected function name should be `isEven`:

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

This makes the function name and its behavior consistent: it returns `true` if `x` is even.

GPT-4o-mini:
The inconsistency in the function is that it is named `isOdd`, suggesting that it should return `true` if the input `x` is odd. However, the logic inside the function actually checks if `x` is even by returning `true` for the condition `x % 2 == 0`.

To fix this inconsistency, the function name should be changed to `isEven`, or the logic should be modified to correctly reflect the odd-checking behavior. Here's how it could be corrected:

1. Change the name to `isEven`:
   ```cpp
   bool isEven(int x) {
       return x % 2 == 0; // Now the name matches the behavior
   }
   ```

2. Or keep the name `isOdd` and change the logic:
   ~~~cpp

Self-Consistency (Strategy 4)

Codestral:
Certainly! The function name `isOdd` suggests that it should return `true` if the input `x` is odd, but the current logic in the function returns `true` if `x` is even. To make the function consistent with its name, we need to change the logic to check if `x` is odd.

Here are a few versions we can consider:

1. **Direct Fix:**
   ```python
   def isOdd(x):
       return x % 2 != 0
   ~~~

2. **Using a Function for Clarity:**
   ```python
   def isOdd(x):
       return not isEven(x)
   ```

3. **Using a Helper Function:**
   ```python
   def isEven(x):
       return x % 2 == 0

   def isOdd(x):
       return not isEven(x)
   ```

4. **Explicitly Checking for Odd:**
   ```python
   def isOdd(x):
       return x % 2 == 1
   ```

5. **Using a Conditional Statement:**

```cpp
  bool isOdd(int x) {
    return x % 2 != 0; // Now the logic matches the name
  }
```

Either adjustment would resolve the inconsistency.

```python
  def isOdd(x):
    if x % 2 == 0:
      return False
    else:
      return True
```

Among these, let's choose the most straightforward and clear version. The **Direct Fix** version is simple and directly addresses the requirement:

```python
def isOdd(x):
  return x % 2 != 0
```

This version clearly states that the function returns `true` if `x` is not divisible by 2, which means `x` is odd. This makes the function consistent with its name.

GPT-4o-mini:
The function name `isOdd` suggests that it should return `true` if the number is odd and `false` if it is not. However, the logic inside the function actually checks if the number is even, which is inconsistent with the name. To make it consistent with its name, we need to change the logic. Here are a few versions to consider:

### Version 1: Correcting Logic
```cpp
bool isOdd(int x) {
  return x % 2 != 0; // Returns true if x is odd
}
```

### Version 2: Renaming the Function
If we want to preserve the original logic but correct the naming, we could rename the function to `isEven`:
```cpp
bool isEven(int x) {
  return x % 2 == 0; // Returns true if x is even
}
```

### Version 3: Returning Boolean Value Directly
Another version that keeps the name and optimizes the return statement:
```cpp
bool isOdd(int x) {
  return (x & 1) == 1; // Using bitwise operation to check for
oddness
}
```

### Conclusion
The most straightforward and clear solution is **Version 1**, which maintains the function name `isOdd` and corrects the logic to accurately reflect its purpose:

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

This version clearly communicates that the function will return `true` for odd numbers and `false` for even numbers, aligning the function's name with its behavior.

## Analysis:

Prompt Chaining distinguished itself  by tying multiple approaches to the problem together in a coherent, progressive way that felt natural to follow. Zero-Shot, while covering the basic idea, tended to repeat similar suggestions without evolving them, making the output feel a bit stagnant. Although the BLEU scores between them were fairly close, Prompt Chaining ultimately offered a clearer and more thoughtfully organized response.

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

Prompt Chaining (Strategy 5)

Codestral:
The JavaScript function `isEven` is intended to determine if a number `n` is even. However, the current implementation returns `1` or `0` instead of `true` or `false`. Here's a step-by-step breakdown:

1. **Current Function Issue**:
    - The expression `n % 2` returns `0` if `n` is even and `1` if `n` is odd.
    - The function should return `true` for even numbers and `false` for odd numbers, but it currently returns `0` and `1`.

2. **Fixing the Function**:
    - To convert the result to a boolean, we need to compare the result of `n % 2` to `0`.

Here is the corrected version of the function:

```
```javascript
function isEven(n) {
    return n % 2 === 0; // Returns true if n is even, false otherwise
}
```
```

### Explanation:
- `n % 2` calculates the remainder when `n` is divided by `2`.
- `n % 2 === 0` checks if the remainder is `0`, which means `n` is even.
- The expression `n % 2 === 0` evaluates to `true` if `n` is even and `false` if `n` is odd.

This ensures that the function returns a proper boolean value.

GPT-4o-mini:
The issue with the provided JavaScript function `isEven` is that it returns the result of the modulus operation (`n % 2`), which yields `0` for even numbers and `1` for odd numbers. However, the function is expected to return a boolean value (`true` for even numbers and `false` for odd numbers).

Zero-Shot (Strategy 1)

Codestral:
To fix the JavaScript function so that it correctly returns `true` or `false` when checking if a number is even, you can use a comparison to check if the remainder when dividing by 2 is zero. Here's the corrected function:

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

This function will return `true` if `n` is even and `false` otherwise. Here's a breakdown of how it works:

- `n % 2` computes the remainder when `n` is divided by 2.
- `n % 2 === 0` checks if the remainder is zero, which indicates that `n` is even.
- The `===` operator ensures that the comparison is both in value and type.

Here are a few examples of how the function works:

```
```javascript
console.log(isEven(4));  // true
console.log(isEven(7));  // false
console.log(isEven(0));  // true
console.log(isEven(-2)); // true
console.log(isEven(-3)); // false
```
```

This should correctly determine if a number is even or odd.

GPT-4o-mini:
To fix the JavaScript function so that it correctly returns `true` for even numbers and `false` for odd numbers, you need to modify the return statement. Instead of returning `n % 2`, you can check if the remainder is equal to `0`. Here's the corrected function:

```
```javascript
```

To fix this, we need to modify the return statement to explicitly return `true` when `n` is even (when `n % 2` equals `0`) and `false` when `n` is odd.
 	
Here’s the corrected version of the function:
```javascript
function isEven(n) {
   return n % 2 === 0; // Returns true for even numbers, false for odd numbers
}
```
 	
### Explanation: 	
- The expression `n % 2 === 0` checks if the remainder of `n` divided by `2` is `0`.
- If it is `0`, the number is even and the function returns `true`.
- If it is not `0`, the number is odd and the function returns `false`.
 	
This ensures that the function now correctly returns a boolean value as intended.
	function isEven(n) {
	   return n % 2 === 0; // Returns true if n is even, false if n is odd
	}
	```
	Now, this function will return `true` when `n` is an even number and `false` when it is an odd number.
	```

Analysis:

For fixing the isEven function, Zero-Shot walked through the correction process methodically, explaining each step and why the previous version failed, which made the reasoning much easier to grasp. Few-Shot moved quicker to the correct solution but glossed over the thought process behind the fix, which could leave gaps for someone unfamiliar with the error. Zero-Shot’s slightly stronger BLEU score aligned with its more detailed and pedagogical flow.

Task 19: Summary Decomposition (C++)

Chain-of-Thought (Strategy 3)	Zero-Shot (Strategy 1)

Codestral:
Sure, let's break down the function `process` step by step to understand its logic and behavior.
 	
### Function Definition: 	
```python
int process(int x) {
```
 	
- **Purpose**: The function is named `process` and takes a single integer parameter `x`.
- **Return Type**: The function returns an integer value.
 	
### Step-by-Step Breakdown: 	
 	
1. **Input Validation**:
```python
   if (x < 0) return -1;
```
   - **Condition Check**: The function first checks if the input `x` is less than 0.
   - **Return Value**: If `x` is negative, the function immediately returns `-1`.
   - **Why**: This is likely to handle invalid input cases where negative values are not allowed. Returning `-1` can serve as an error code or indicator.
	Codestral:
	The function `process` performs the following tasks:
	
	1. **Checks**:
	  - It checks if the input integer `x` is less than 0.
	
	2. **Calculates**:
	  - If `x` is not less than 0, it calculates the square of `x`.
	
	3. **Returns**:
	  - If `x` is less than 0, it returns `-1`.
	  - Otherwise, it returns the square of `x`.
	
	In summary, the function `process` takes an integer input `x`, checks if it is negative, and returns `-1` if it is. If `x` is non-negative, it returns the square of `x`.
	
	GPT-4o-mini:
	The function `process` takes an integer input `x` and performs the following tasks:
	
	1. **Validation Check**: It checks if the input `x` is negative. If `x` is less than 0, the function returns -1 as an indication of invalid input.
	
	2. **Calculation**: If the input is non-negative (0 or greater), the

2. **Square Calculation**:
   ```python
   return x * x;
   ```
   - **Operation**: If the input `x` is not less than 0 (i.e., it is 0 or positive), the function calculates the square of `x`.
   - **Return Value**: The result of `x * x` is returned.
   - **Why**: The function is likely intended to calculate the square of the input `x`, which is a common mathematical operation.

### Summary of Logical Steps:
1. **Input Validation**: Check if `x` is less than 0.
  - If true, return `-1` (indicating an error or invalid input).
2. **Square Calculation**: If `x` is 0 or positive, calculate the square of `x`.
  - Return the result of the square calculation.

### Example Scenarios:
- **If x = 5**: The function calculates `5 * 5`, which equals `25`, and returns `25`.
- **If x = -3**: The function checks `x < 0`, which is true, so it returns `-1`.
- **If x = 0**: The function calculates `0 * 0`, which equals `0`, and returns `0`.

This breakdown helps understand the function's behavior and why it handles inputs in the specified manner.

GPT-4o-mini:
Let's break down the provided function `process` into logical steps to understand what it does and why:

1. **Function Definition**:
  - The function is named `process` and takes a single parameter `x` of type `int`. This indicates that it is designed to handle integer inputs.

2. **Input Validation**:
  - The first operation inside the function is a conditional check: `if ( x < 0)`.
  - This checks whether the input value `x` is less than zero. The purpose of this check is to validate the input and ensure that it is non-negative since the subsequent operation (squaring) is typically defined for non-negative numbers in many contexts.

3. **Return Value for Invalid Input**:
  - If the condition `x < 0` evaluates to true (meaning `x` is negative), the function executes the statement `return -1;`.
  - This serves as an early exit from the function, indicating that the input was invalid. Returning `-1` is a way to signal an error or invalid state, as squaring a negative number might not be the intended operation.

4. **Squaring the Input**:
  - If the input `x` is non-negative (this means the condition `x < 0` is false), the function proceeds to the next line: `return x * x;`.
  - This line calculates the square of the input `x`. The operation `x * x` computes the product of `x` with itself, effectively calculating the square.

function calculates the square of `x` (i.e., `x * x`).

3. **Return Value**: The function returns the calculated square of `x` if the input is valid; otherwise, it returns -1 for invalid input.

In summary, it validates the input, computes the square of non-negative integers, and returns either the square or -1 for negative inputs.

5. **Returning the Result**:
   - The result of the squaring operation is then returned as the output of the function. This is the final value that the function will produce for valid non-negative inputs.

### Summary of Steps:
1. **Function Definition**: Define `process` with parameter `x`.
2. **Input Validation**: Check if `x` is less than 0.
3. **Invalid Input Handling**: If `x` is negative, return `-1`.
4. **Square Calculation**: If `x` is non-negative, calculate `x * x`.
5. **Return Result**: Return the calculated square of `x`.

### Conclusion:
The function effectively handles input validation for non-negative integers, returns a specific error code for invalid inputs, and computes the square of valid inputs, thus ensuring that only appropriate values are processed for squaring.

Analysis:

Prompt Chaining quickly summarized the intended behavior of the function, getting straight to the point without lingering too long on side notes, which made the response efficient and easy to follow. In comparison, Zero-Shot elaborated more, but often added unnecessary explanation that didn't contribute much additional value to understanding the task. BLEU scores and readability both leaned toward Prompt Chaining's simpler, more purpose-driven style.

Task 20: Purpose Inference → Completion (Python)

Few-Shot (Strategy 2)

Codestral:
To complete the `calculate_average` function, you need to iterate through the list of scores, sum them up, and then divide by the number of scores to get the average. Here's how you can do it:

```python
def calculate_average(scores):
    if not scores:  # Check if the list is empty to avoid division by zero
        return 0

    total = 0
    for score in scores:
        total += score

    average = total / len(scores)
    return average
```

This function first checks if the `scores` list is empty to avoid division by zero. If the list is not empty, it iterates through the list, summing up the scores, and then computes the average by dividing the total by the length of the list.

Here is an example of how you can use this function:

```python
scores = [85, 90, 78, 92, 88]
average = calculate_average(scores)
```

Role-Playing (Strategy 6)

Codestral:
To complete the `calculate_average` function, you need to iterate through the `scores` list, sum up all the values, and then divide by the number of elements in the list to get the average. Here's how you can do it:

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

# Example usage:
scores = [85, 90, 78, 92, 88]
print(calculate_average(scores))  # Output should be the average of the scores list
```

Explanation:
1. Initialize `total` to 0.
2. Iterate through each `score` in the `scores` list and add it to `total`.
3. Calculate the average by dividing `total` by the number of elements in the `scores` list (`len(scores)`).
4. Return the calculated average.

You can also add a check to handle the case where the list might

```
print(f"The average score is: {average}")
```

This will output:
```
The average score is: 86.4
```

GPT-4o-mini:
Certainly! To complete the `calculate_average` function that averages a list of scores, you can calculate the sum of the scores and then divide by the number of scores. Here's how you can implement it:

```python
def calculate_average(scores):
    total = sum(scores)  # Calculate the total sum of the scores
    count = len(scores)  # Get the number of scores
    # Return the average, handling the case where count is zero
    return total / count if count > 0 else 0
```

### Explanation:
- `sum(scores)`: This function calculates the total sum of all the values in the `scores` list.
- `len(scores)`: This function returns the number of elements in the `scores` list.
- Finally, we divide `total` by `count` to get the average. We also handle the case where `count` is zero to avoid division by zero by returning 0 in that case.

be empty to avoid division by zero:

```python
def calculate_average(scores):
    if not scores:
        return 0  # or raise an exception if you prefer

    total = 0
    for score in scores:
        total += score
    average = total / len(scores)
    return average

# Example usage:
scores = [85, 90, 78, 92, 88]
print(calculate_average(scores))  # Output should be the average of the scores list
```

This way, if the `scores` list is empty, the function will return 0 (or you can raise an exception if you prefer to handle this case differently).

GPT-4o-mini:
To complete the function that calculates the average of a list of scores, you need to sum up all the scores and then divide by the number of scores. Here's the completed function:

```python
def calculate_average(scores):
    if not scores:  # Check if the list is empty
        return 0  # Return 0 or you could raise an exception based on your needs
    total = sum(scores)  # Sum up all the scores
    average = total / len(scores)  # Calculate the average
    return average  # Return the average
```

In this implementation:
- We first check if the `scores` list is empty. If it is, we return `0` to avoid division by zero.
- We use the built-in `sum()` function to calculate the total of the scores.
- We then divide the total by the number of scores using `len(scores)` to compute the average.
- Finally, we return the calculated average.

Analysis:

When averaging numbers in a list, Zero-Shot took the time to break down the process explicitly, detailing how summation and counting lead to the average calculation in a way that would be particularly helpful for beginners. Role-Playing offered a tighter, more advanced solution that used Python's built-in functions efficiently, but it was less explanatory for those still learning. BLEU scores were fairly close between the two, but Zero-Shot's clear and gradual walkthrough made its output stand out for teaching purposes.

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

Chain-of-Thought (Strategy 3)	Role-Playing (Strategy 6)
Codestral: There are several logical issues and stylistic problems in the	Codestral: Certainly! Let's go through the provided Python utility and identify

provided script. Here are some of the issues:

1. **Indentation Errors**: The `read_csv` function has incorrect indentation.
2. **Unmatched Quotation Marks**: The script uses incorrect quotation marks (`’` instead of `'`).
3. **Unnecessary List Comprehension in `read_csv`**: The list comprehension can be simplified.
4. **Hardcoded Filepath**: The filepath is hardcoded which is not flexible.
5. **Index Handling in `summarize_column`**: The column index is hardcoded.
6. **Error Handling**: There is no error handling for file reading or data conversion.
7. **Magic Numbers**: The column index should ideally be passed as an argument.

Here is a revised version of the script that addresses these issues:

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

def read_csv(filepath):
    try:
        with open(filepath, 'r') as f:
            return list(csv.reader(f))
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
        return []
    except Exception as e:
        print(f"Error reading file: {e}")
        return []

def summarize_column(data, index):
    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 IndexError:
        print(f"Error: Column index {index} out of range.")
        return None, None
    except ZeroDivisionError:
        print("Error: Division by zero encountered.")
        return None, None

def main(filepath, column_index):
    data = read_csv(filepath)
    if not data:
        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__':
```

potential bugs and design flaws.

### Potential Issues:

1. **Syntax Errors**:
   - The code contains non-ASCII quotes (`’`) instead of standard ASCII single quotes (`'`). This will cause a syntax error.
   - Similarly, the `main()` function call in the `if __name__ == '__main__':` block should use standard ASCII single quotes.

2. **File Path Hardcoding**:
   - The `filepath` is hardcoded as `'data.csv'`. This makes the script less flexible and harder to test with different files.

3. **Error Handling**:
   - There is no error handling for file operations or CSV parsing. If the file doesn't exist or is malformed, the script will fail silently.

4. **Column Index Assumption**:
   - The `summarize_column` function assumes that the column index to summarize is always `1`. This is not flexible and does not handle cases where the relevant column might not be the second one.

5. **Magic Numbers**:
   - The column index `1` is a magic number. It should be passed as an argument or configured in a more flexible way.

6. **Code Formatting**:
   - The indentation is inconsistent and the code is not PEP 8 compliant. This makes the code harder to read and maintain.

### Cleaner Version:

Here is a revised version of the script that addresses these issues:

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

def read_csv(filepath):
    try:
        with open(filepath, 'r') as f:
            return [row for row in csv.reader(f)]
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
        sys.exit(1)
    except Exception as e:
        print(f"Error: {e}")
        sys.exit(1)

def summarize_column(data, index):
    try:
        values = [float(row[index]) for row in data[1:]]  # skip header
        total = sum(values)
        avg = total / len(values)
        return total, avg
    except (ValueError, IndexError) as e:
        print(f"Error: {e}")
        sys.exit(1)
    filepath = 'data.csv'
    column_index = 1  # Example column index, can be changed as needed
    main(filepath, column_index)
```

### Changes Made:
1. **Corrected Indentation**: Fixed indentation issues in the `read_csv` function.
2. **Corrected Quotation Marks**: Changed incorrect quotation marks to standard double quotes.
3. **Simplified `read_csv`**: Used `list(csv.reader(f))` for better readability.
4. **Flexible Filepath and Column Index**: Made `filepath` and `column_index` configurable via function arguments.
5. **Added Error Handling**: Added error handling for file reading, data conversion, and potential runtime errors.
6. **Magic Numbers**: Removed hardcoded values and made the script more flexible.

This revised script should be more robust and easier to maintain.

GPT-4o-mini:
The original script has several logical and formatting issues that can cause it to fail or behave unexpectedly. Here are the main issues:

1. **Indentation Errors**: Python relies on indentation to define blocks of code. The `with` statement and other parts of the code are not properly indented.

2. **Quotation Marks**: The script uses non-standard quotation marks (`’` instead of `'`). This can lead to syntax errors in Python.

3. **Index Out of Range**: The `summarize_column` function does not handle the case where the provided index might be out of range, which would raise an `IndexError`.

4. **Empty Data Handling**: If the CSV file is empty or only contains a header, the `summarize_column` function will attempt to compute the average from an empty list, resulting in a `ZeroDivisionError`.

5. **Hardcoded File Path**: The file path is hardcoded, which limits the reusability of the script. It would be better to allow the user to specify the file path.

6. **No Error Handling**: The script does not handle potential errors, such as file not found or invalid data in the CSV.

7. **Lack of Documentation**: There are no comments or docstrings explaining the purpose of functions and parameters.

Here is a revised version of the script addressing these issues:

```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."""

def main(filepath, column_index):
    data = read_csv(filepath)
    total, avg = summarize_column(data, column_index)
    print(f"Total: {total}")
    print(f"Average: {avg}")

if __name__ == '__main__':
    if len(sys.argv) != 3:
        print("Usage: python utils.py <filepath> <column_index>")
        sys.exit(1)

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

    main(filepath, column_index)
```

### Changes Made:

1. **Fixed Syntax Errors**:
  - Replaced non-ASCII quotes with standard single quotes.

2. **Added Error Handling**:
  - Added try-except blocks to handle file not found errors, general exceptions during file reading, and value/column index errors during summarization.

3. **Made File Path and Column Index Configurable**:
  - Added command-line arguments to specify the filepath and column index.

4. **Improved Code Formatting**:
  - Ensured consistent indentation and use of PEP 8 guidelines.

5. **Added Usage Instructions**:
  - Provided usage instructions when the script is run with incorrect arguments.

This revised version should be more robust, flexible, and easier to maintain.

GPT-4o-mini:
Thank you for sharing your Python utility. I'll conduct a code review and highlight some potential bugs and design flaws, along with suggestions for improvement.

### Issues Identified

1. **Indentation Error**: The `with` statement in `read_csv` is not properly indented, which will lead to an `IndentationError`.

2. **Quotes**: You're using non-standard quote characters (`’`) instead of standard single quotes (`'`) or double quotes (`"`). This will cause syntax errors.

3. **Error Handling**: There’s no error handling for file

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

def summarize_column(data, index):
    """Summarizes the specified column (index) in the data.

    Returns the total and average of the numeric values in the column.
    """
    if len(data) <= 1:  # Check if there's no data apart from the header
        return 0, 0

    values = []
    for row in data[1:]:  # Skip header
        try:
            values.append(float(row[index]))
        except (ValueError, IndexError):
            print(f"Warning: Invalid data encountered at row {data.index(row) + 1}. Skipping.")

    if not values:  # Check if the values list is empty
        return 0, 0

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

def main(filepath):
    """Main function to read data and summarize a specific column."""
    data = read_csv(filepath)
    total, avg = summarize_column(data, 1)  # Assuming we want to summarize the second column

    print("Total:", total)
    print("Average:", avg)

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print("Usage: python utils.py <path_to_csv_file>")
    else:
        main(sys.argv[1])  # Pass the file path from command line arguments
```
~~~

### Changes Made:
- Fixed indentation and quotation marks.
- Added error handling for index out of range and ValueError when converting to float.
- Handled cases for empty CSV files or invalid data.
- Made the file path a command-line argument for better flexibility.
- Added comments and docstrings for clarity.

operations or when converting strings to floats. If the file does not exist or contains invalid data, the program will crash.

4. **Hardcoded File Path**: The file path is hardcoded in the `main` function. It would be better to allow the user to specify the file path as an argument.

5. **Magic Numbers**: The index `1` in `summarize_column` is hardcoded, which could lead to confusion. It would be more readable if it were defined as a constant or passed as a parameter.

6. **Empty Data Handling**: The function `summarize_column` assumes that there will always be data after the header, which may not be the case. It should check for empty data to avoid a `ZeroDivisionError`.

### Cleaner Version

Here’s a revised version of your utility that addresses the identified issues:

```python
import csv
import sys

def read_csv(filepath):
    """Read CSV file and return 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}' was not found.")
        sys.exit(1)
    except Exception as e:
        print(f"Error reading the file: {e}")
        sys.exit(1)

def summarize_column(data, index):
    """Summarize the specified column from the data."""
    values = []
    for row in data[1:]:  # skip header
        try:
            values.append(float(row[index]))
        except ValueError:
            print(f"Warning: Unable to convert '{row[index]}' to float. Skipping this value.")

    if not values:
        print("Warning: No valid numeric data found.")
        return 0, 0

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

def main(filepath, index):
    """Main function to read CSV and summarize a column."""
    data = read_csv(filepath)
    total, avg = summarize_column(data, index)
    print(f"Total: {total}")
    print(f"Average: {avg}")
```

```
if __name__ == '__main__':
    if len(sys.argv) != 3:
        print("Usage: python utils.py <csv_file_path> <column_index>")
        sys.exit(1)

    file_path = sys.argv[1]
    column_index = int(sys.argv[2])  # Convert to integer
    main(file_path, column_index)
```

### Key Improvements Made

- **Fixed Indentation and Quotes**: Corrected indentation and replaced non-standard quotes with standard quotes.
- **Error Handling**: Added try-except blocks to handle file reading errors and value conversion issues gracefully.
- **Parameterization**: The file path and column index are now passed as command-line arguments, allowing for greater flexibility.
- **Empty Data Handling**: Added checks for empty lists and warnings for invalid values to ensure robustness.
- **Documentation**: Added docstrings to functions for better code documentation.

This cleaner version should be more robust and user-friendly while maintaining clarity.

Analysis:

Zero-Shot successfully identified several major logic errors in the code, but it missed some edge cases related to error handling and robustness that Role-Playing caught with more thorough fixes. The Role-Playing output was not only more comprehensive but also cleaner and better structured, addressing both core functionality and polish. This was reflected in its noticeably higher BLEU score compared to Zero-Shot, which confirms for me the quality gap between the two strategies.

Task 22: Code Completion and Robustness Enhancement (Python)

Few-Shot (Strategy 2)

Codestral:
To complete the `file_processor.py` script, you need to implement the `clean_line` function to remove punctuation and convert the text to lowercase. Then, the script will be able to process the text file, clean the lines, count the words, and print the results.

Here's the 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)
    cleaned_line = line.translate(translator).lower()
    return cleaned_line
```

Prompt Chaining (Strategy 5)

Codestral:
The script is intended to process a text file by cleaning each line (removing punctuation and converting to lowercase) and then counting the frequency of each word. The current script is incomplete and contains some issues, such as improper string handling, missing functionality in `clean_line`, and incorrect string literals. Let's improve the script step by step:

1. **Fixing the String Literals**: The script uses curly quotes (`’`) instead of straight quotes (`'`).

2. **Implementing `clean_line`**: We need to remove punctuation and convert the line to lowercase.

3. **Ensuring Proper Word Splitting**: We need to ensure that words are split correctly and handled properly.

Here is the improved script:

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

```
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(filepath)`**: This function reads all lines from the file specified by `filepath` and returns them as a list.

2. **`clean_line(line)`**: This function removes punctuation from the input line and converts it to lowercase. It uses `str.maketrans` and `str.translate` to achieve this.

3. **`count_words(lines)`**: This function iterates over each line, cleans it using `clean_line`, splits the cleaned line into words, and updates the word counts in a dictionary.

4. **`main()`**: This function defines the filepath, loads the lines from the file, counts the words, and prints the word counts in alphabetical order.

### Usage:

1. Save the script to a file named `file_processor.py`.
2. Create an input file named `input.txt` with the text you want to process.
3. Run the script using `python file_processor.py`.

This script will read the file, clean the text, count the words, and print the results.

GPT-4o-mini:
Sure! Let's complete the file-processing function by implementing the `clean_line` function to remove punctuation and convert text to lowercase. Here’s the corrected and completed version of your script:

```python
# file_processor.py - Script for processing text files
import string

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

```
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)
        words = clean.split()
        for word in words:
            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 of Changes:

1. **String Literals**:
  - Changed `’` to `'` for all string literals.

2. **`clean_line` Function**:
  - Used `str.maketrans` and `str.translate` to remove punctuation.
  - Converted the line to lowercase using `str.lower()`.

3. **Word Splitting**:
  - Ensured that `clean.split()` correctly splits the line into words.

4. **Main Function**:
  - Ensured the file path is specified correctly.
  - Sorted the word counts before printing.

This improved script should correctly clean each line, count the words, and print the word counts without errors.

GPT-4o-mini:
The provided script aims to process a text file by loading its content, cleaning each line of punctuation and converting it to lowercase, and then counting the occurrences of each word in the text file. However, the script contains several issues and is incomplete. Let's address those issues and improve the script.

### Issues Identified
1. **Indentation**: The `load_file` function's code is not properly indented, which will cause a syntax error.

```
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 of Changes:

1. **`clean_line` Function**:
  - This function now uses `str.translate` and `string.punctuation` to remove all punctuation from the input line and converts it to lowercase using `lower()`.
  - A translation table is created with `str.maketrans` to facilitate the removal of punctuation.

2. **File Handling**:
  - The file handling code is kept intact, and it will read all lines from the specified file.

3. **Word Counting**:
  - The `count_words` function now processes each cleaned line and counts occurrences of each word using a dictionary.

4. **Main Function**:
  - The main function reads from 'input.txt' and prints out the word counts sorted in alphabetical order.

### Usage:
- Save the script to a file named `file_processor.py`.
- Create a text file named `input.txt` in the same directory with the text you want to analyze.
- Run the script using Python to see the cleaned word counts printed to the console.

2. **String handling**: The `clean_line` function is not implemented, so we need to add logic to remove punctuation and convert text to lowercase.
3. **Quotation marks**: The script uses incorrect quotation marks (e.g., `’` instead of `'`).
4. **File path**: The input file path is surrounded by spaces, which may cause an error when trying to read the file.
5. **Word counting logic**: The current logic does not handle empty lines or lines that could potentially only contain whitespace.

### Improved Script

```python
# file_processor.py - Improved script for processing text files
import string

def load_file(filepath):
    """Load the file and return its lines."""
    with open(filepath, 'r') as f:
        return f.readlines()

def clean_line(line):
    """Remove punctuation and make the line lowercase."""
    # Remove punctuation and convert to lowercase
    return line.translate(str.maketrans('', '',
string.punctuation)).lower().strip()

def count_words(lines):
    """Count the occurrences of each word in the provided lines."""
    word_counts = {}
    for line in lines:
        clean = clean_line(line)
        # Only split if the line is not empty
        if clean:
            for word in clean.split():
                word_counts[word] = word_counts.get(word, 0) + 1
    return word_counts

def main():
    filepath = 'input.txt'  # Ensure there are no extra spaces
    lines = load_file(filepath)
    counts = count_words(lines)
    for word, count in sorted(counts.items()):
        print(f"{word}: {count}")

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

### Improvements Made
1. **Corrected Indentation**: Fixed the indentation of the `load_file` function.
2. **Implemented `clean_line`**: Added functionality to remove punctuation and convert the line to lowercase.
3. **Proper Quotation Marks**: Replaced smart quotes with regular quotes for string literals.
4. **Whitespace Handling**: Added a check in the `count_words` function to skip empty lines after cleaning.
5. **General Cleanup**: Removed unnecessary spaces from the file path string and ensured overall readability.

	This improved script should now function correctly, processing the text file as intended and accurately counting word occurrences without errors.

Analysis:

When it came to cleaning and splitting text files, Prompt Chaining offered a much more complete and thoughtful correction, addressing issues like quotation marks, word splitting, and file path handling that Zero-Shot overlooked. Zero-Shot provided a basic fix that would work for simple cases but left several important edge conditions unresolved. Prompt Chaining's stronger BLEU score matched the broader improvements it made to both functionality and clarity.