

Task 1: Code Summarization (Java)

Goal	Summarize the functionality of the following method		
Temperature	0.7	Token Limit	2048
Prompt Strategy 1	Chain of Thought	Prompt Strategy 2	Few Shot
Prompt	<p>Summarize the following Java method. Let's think step-by-step. Here's the code:</p> <pre>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; }</pre>	Prompt	<p>Example 1: Input:</p> <pre>public int sumList(List<Integer> nums) { int sum = 0; for (int n : nums) { sum += n; } return sum; }</pre> <p>Output: "This method returns the sum of all integers in the input list."</p> <p>Example 2: Input:</p> <pre>public boolean containsNegative(List<Integer> nums) { for (int n : nums) { if (n < 0) return true; } return false; }</pre> <p>Output: "This method checks if the input list contains any negative numbers."</p> <p>Now, summarize the following Java method. Here's the code:</p> <pre>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; }</pre>
ChatGPT Output	<p>Summary:</p> <p>The method <code>countWordFrequency</code> takes a list of strings (words) and returns a map that shows how many times each word appears in the list. It iterates through the list, counting each word's frequency and storing the result in a map where keys are words and values are their frequencies.</p>	ChatGPT Output	"This method counts the frequency of each word in the input list and returns a map of words to their corresponding counts."
Mistral Output	<p>The method <code>countWordFrequency</code> takes a list of words as input and returns a map that represents the frequency of each word in the list. It initializes an empty <code>HashMap</code> to store the word counts. The method then iterates over each word in the input list, incrementing the count in the map for each word. If a word is not already present in the map, it initializes its count to 1. Finally, the method returns the map containing the word frequencies.</p>	Mistral Output	<p>This method calculates the frequency of each word in the input list and returns a map where the keys are the words and the values are their respective counts.</p>
Analysis	<p>The few-shot prompt produced a correct and more consistent solution but it is not as descriptive as the chain of thought prompt. The few-shot prompting provided examples that the model followed. Thus, the output for GPT and Mistral with few-shot were very similar.</p> <p>Chain Of Thought BLEU score: 0.134705191405238. Few Shot BLEU score: 0.45925967951833463</p>		

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

Goal	Identify and fix the off-by-one error in this function		
Temperature	0.7	Token Limit	2048
Prompt Strategy 1	Prompt Chaining	Prompt Strategy 2	Chain of Thought
Prompt	<p>Prompt 1: Identify the off-by-one bug in this Python function. Here's the code:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end): total += i return total</pre> <p>Prompt 2: Now, fix the bug.</p>	Prompt	<p>Fix the off-by-one bug in this Python function. Let's think step-by-step. Here's the code:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end): total += i return total</pre>
ChatGPT Output	<p>Output 1: Let's look at your function:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end): total += i return total</pre> <p>Off-by-one bug explanation:</p> <ul style="list-style-type: none"> The built-in <code>range(start, end)</code> in Python includes <code>start</code> but excludes <code>end</code>. So, if you call <code>sum_range(1, 5)</code>, it will sum 1+2+3+4 (but not 5). <p>If you want to include both <code>start</code> and <code>end</code> in the sum, you need to change the range to <code>range(start, end+1)</code>.</p> <p>Corrected code:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre> <p>Summary: The off-by-one bug is that the original code does not include the <code>end</code> value in the sum. To fix it, use <code>range(start, end+1)</code>.</p> <p>Output 2: Certainly! Here's the corrected function:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre> <p>Now, <code>sum_range(1, 5)</code> will correctly return $1 + 2 + 3 + 4 + 5 = 15$.</p>	ChatGPT Output	<p>Let's analyze your function step by step:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end): total += i return total</pre> <p>What does this do? It sums all integers from <code>start</code> up to but not including <code>end</code>.</p> <p>What is the off-by-one bug? If you want <code>sum_range(1, 5)</code> to include both 1 and 5 (i.e., sum $1+2+3+4+5=15$), <code>range(start, end)</code> will only sum 1+2+3+4 (excluding 5). That's because <code>range</code> is inclusive of the start, exclusive of the end.</p> <p>How do we fix it? Change <code>range(start, end)</code> to <code>range(start, end + 1)</code>. This way, the loop includes <code>end</code> as well.</p> <p>Fixed code:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre> <p>Example: <code>sum_range(1, 5)</code> will now return $1+2+3+4+5 = 15$.</p> <p>Summary of fix: Change <code>range(start, end)</code> to <code>range(start, end + 1)</code> to include both endpoints.</p>
Mistral Output	<p>Output 1: The off-by-one bug in the provided Python function <code>sum_range</code> is due to the way the <code>range</code> function is used. The <code>range</code> function in Python generates a sequence of numbers starting from <code>start</code> and ending at <code>end - 1</code>. This means that the <code>end</code> value itself is not included in the range.</p> <p>Here is the original code for reference:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end): total += i return total</pre> <p>To include the <code>end</code> value in the sum, you need to adjust the <code>range</code> function to go up to <code>end + 1</code>. Here is the corrected version of the function:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre>	Mistral Output	<p>Sure, let's work through this step-by-step.</p> <p>The given function <code>sum_range</code> is meant to sum all integers from <code>start</code> to <code>end</code> inclusive. However, the <code>range</code> function in Python is exclusive of the <code>end</code> value. This means <code>range(start, end)</code> generates numbers from <code>start</code> to <code>end-1</code>.</p> <p>To fix the off-by-one bug, we need to include the <code>end</code> value in the range. We can do this by using <code>end + 1</code> as the stopping point in the <code>range</code> function.</p> <p>Here's the corrected code:</p> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre> <p>This way, the loop will iterate from <code>start</code> to <code>end</code> inclusive, ensuring that the <code>end</code> value is included in the sum.</p>

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

With this change, the `range` function will generate a sequence of numbers from `start` to `end` inclusive, ensuring that the `end` value is included in the sum.

Output 2: Sure, here is the corrected version of the `sum_range` function with the off-by-one bug fixed:

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

In this corrected version, the `range` function now goes from `start` to `end + 1`, which ensures that the `end` value is included in the sum. This will correctly sum all integers from `start` to `end` inclusive.

Analysis

Both the chain-of-thought and prompt chaining produced the same corrected code but chain-of-thought was more concise. With prompt chaining, both models provided the fixed code within the first prompt which made the second prompt redundant. GPT and Mistral had similar outputs for prompt chaining but not for chain of thought. GPT broke down the problems into questions whereas Mistral used a concise paragraph. Prompt Chaining Exact match: 0.2938706968933669. Chain Of Thought Exact match: 0.19664268585131894

Task 3: Bug Classification (C++)

Goal	Classify the type of bug in the following C++ function														
Temperature	0.7	Token Limit	2048												
Prompt Strategy 1	Few Shot	Prompt Strategy 2	Chain of Thought												
Prompt	<p>Example 1: Input:</p> <pre>void copyArray(int* dest, int* src, int size) { for (int i = 0; i <= size; i++) { dest[i] = src[i]; } }</pre> <p>Bug Type: Off-by-One Error (Buffer Overflow)</p> <p>Example 2: Input:</p> <pre>int calculateSum(int a, int b) { int sum; return sum; }</pre> <p>Bug Type: Uninitialized Variable</p> <p>Now, classify the type of bug in the C++ function. Here's the code:</p> <pre>int* getArray(int size) { int* arr = new int[size]; return arr; }</pre>	Prompt	<p>Classify the type of bug in the C++ function. Let's think step-by-step. Here's the code:</p> <pre>int* getArray(int size) { int* arr = new int[size]; return arr; }</pre>												
ChatGPT Output	<p>Bug Type: Returning Address of Local Variable (Dangling Pointer)</p> <p>Explanation:</p> <p>In the code:</p> <pre>int* getArray(int size) { int arr[size]; // local variable, lives only within the function scope return arr; // returns pointer to local stack memory, which is invalid after function returns }</pre> <p>The function returns a pointer to a local array (<code>arr</code>). This array is allocated on the stack, and its memory is reclaimed when the function exits. Returning its address leads to undefined behavior because the pointer becomes dangling (it points to memory that is no longer valid).</p> <p>This is commonly classified as:</p> <ul style="list-style-type: none">• Returning Address of Local Variable• Dangling Pointer <p>Summary Table:</p> <table border="1"><thead><tr><th>Example</th><th>Code Snippet</th><th>Bug Type</th></tr></thead><tbody><tr><td>1</td><td>copyArray</td><td>Off-by-One Error (Buffer Overflow)</td></tr><tr><td>2</td><td>calculateSum</td><td>Uninitialized Variable</td></tr><tr><td>3</td><td>getArray</td><td>Returning Address of Local Variable (Dangling Pointer)</td></tr></tbody></table>	Example	Code Snippet	Bug Type	1	copyArray	Off-by-One Error (Buffer Overflow)	2	calculateSum	Uninitialized Variable	3	getArray	Returning Address of Local Variable (Dangling Pointer)	ChatGPT Output	<p>Let's analyze the provided C++ function step-by-step for bugs:</p> <p>Code</p> <pre>int* getArray(int size) { int arr[size]; return arr; }</pre> <p>Step-by-Step Analysis</p> <ol style="list-style-type: none">1. Array Declaration: <code>int arr[size];</code> This declares a local array of integers with the given size. In C99, this is called a Variable Length Array (VLA). In C++, this is not standard: C++ only allows arrays with compile-time constant size. However, some compilers (like GCC) allow this as an extension.2. Scope and Lifetime: The array <code>arr</code> is a local variable inside the function. Its lifetime ends when the function returns.3. Returning Local Array Address: <code>return arr;</code> This returns a pointer to the first element of the local array. After the function returns, the array arr no longer exists. The returned pointer will point to memory that is no longer allocated for <code>arr</code>. <p>Type of Bug</p> <ul style="list-style-type: none">• Returning Address of Local (Stack) Variable: This is a classic dangling pointer bug.<ul style="list-style-type: none">◦ The pointer returned by <code>getArray</code> points to a memory location that is invalid after the function returns.◦ Using this pointer later leads to undefined behavior. <p>Classification</p> <p>Type of bug:</p> <ul style="list-style-type: none">• Dangling pointer / Use-after-scope• (Also, if you want to be precise: returning address of local stack variable.) <p>Brief Summary</p> <p>Bug: The function returns a pointer to a local (stack-allocated) array, which becomes invalid after the function returns.</p> <p>Consequence: Using the returned pointer causes undefined behavior due to a dangling pointer.</p> <p>Recommendation:</p> <ul style="list-style-type: none">• Use <code>new</code> to allocate on the heap (<code>int* arr = new int[size];</code>), or• Use <code>std::vector<int></code> for safe dynamic arrays, or• Pass in an array from the caller. <p>In summary:</p> <p>The function contains a dangling pointer bug by returning the address of a local (stack-allocated) array.</p>
Example	Code Snippet	Bug Type													
1	copyArray	Off-by-One Error (Buffer Overflow)													
2	calculateSum	Uninitialized Variable													
3	getArray	Returning Address of Local Variable (Dangling Pointer)													
Mistral Output	<p>The bug in the provided C++ function is:</p> <p>Stack Allocation Returned from Function</p> <p>Explanation: In the provided function <code>getArray</code>, the array <code>arr</code> is declared as a local variable. Local variables in C++ are allocated on the stack, and their lifetime ends when the function returns. Returning a pointer to a local stack-allocated variable results in undefined behavior because the memory for the local variable is deallocated when the function returns.</p> <pre>int* getArray(int size) { int arr[size]; // Local array allocated on the stack return arr; // Returning the address of a local variable is problematic }</pre> <p>To fix this issue, you should allocate memory on the heap using <code>new</code> or use a standard container like <code>std::vector</code> that manages its own memory.</p> <p>Here is a corrected version using <code>new</code>:</p> <pre>int* getArray(int size) { int* arr = new int[size]; // Allocate memory on the heap return arr; }</pre> <p>Or using <code>std::vector</code>:</p>	Mistral Output	<p>To classify the type of bug in the given C++ function, let's analyze the code step-by-step:</p> <pre>int* getArray(int size) { int arr[size]; return arr; }</pre> <p>Step-by-Step Analysis:</p> <ol style="list-style-type: none">1. Variable Declaration: <code>int arr[size];</code> This line declares an array <code>arr</code> with a size determined by the parameter <code>size</code>.2. Return Statement: <code>return arr;</code> This line attempts to return the array <code>arr</code>. <p>Issues Identified:</p> <ol style="list-style-type: none">1. Variable-Length Arrays (VLAs):												

```
#include <vector>

std::vector<int> getArray(int size) {
    std::vector<int> arr(size); // Use std::vector for dynamic array
    return arr;
}
```

In both corrected versions, the caller of the function is responsible for managing the memory, either by deleting the allocated memory with `delete[]` in the first example or by letting the `std::vector` handle its own memory in the second example.

- In standard C++ (prior to C++11), variable-length arrays (VLAs) are not supported. The size of an array must be a constant expression. This means `int arr[size]` is invalid if `size` is not a compile-time constant.
- Some compilers, like GCC, support VLAs as an extension, but this is not portable and not part of the C++ standard.

2. Returning a Local Array:

- The array `arr` is a local variable within the function `getArray`. Local variables are allocated on the stack and are automatically destroyed when the function returns.
- Returning a pointer to a local array results in undefined behavior because the array no longer exists after the function returns. The returned pointer would be dangling.

Classification of the Bug:

- **Type of Bug:** The bug is a **logical error** combined with a **memory management error**.
- **Logical Error:** Attempting to use a feature (VLAs) that is not supported by standard C++.
- **Memory Management Error:** Returning a pointer to a local array, which leads to a dangling pointer and undefined behavior.

Correct Approach:

To fix the issues, you can use dynamic memory allocation with `new`, or use standard containers like `std::vector`:

Using new:

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

Using std::vector:

```
#include <vector>

std::vector<int> getArray(int size) {
    return std::vector<int>(size);
}
```

Both approaches ensure that the memory allocated for the array persists beyond the function's scope, avoiding the dangling pointer issue.

Analysis

The chain of thought prompt produced a more correct and consistent bug classification for GPT and Mistral. Few shot led to inconsistent results for GPT and Mistral. Having the model think through its solution guided the model towards more robust and consistent solutions. Few Shot Exact match: 0.020185708518368994. Chain Of Thought Exact match: 0.12242874845105328

Task 4: Generating Email Validators (Python + Regex)

Goal	Complete the function using regex to validate basic email addresses		
Temperature	0.7	Token Limit	2048
Prompt Strategy 1	Few Shot	Prompt Strategy 2	Self Consistency
Prompt	<p>Example 1: Input: "user@example.com" Output: valid</p> <p>Example 2: Input: "invalid-email@" Output: invalid</p> <p>Example 3: Input: "john.doe@company.org" Output: valid</p> <p>Example 4: Input: "noatsign.com" Output: invalid</p> <p>Now complete the Python function using regex to validate basic email addresses. Here's the code:</p> <pre>def is_valid_email(email): # TODO: Complete using regex pass</pre>	Prompt	Complete the Python function using regex to validate basic email addresses. Here's the code:
ChatGPT Output	<p>Certainly! Here's a Python function using regex to validate basic email addresses as requested:</p> <pre>import re def is_valid_email(email): # Basic regex for validating an email address pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$' return 'valid' if re.match(pattern, email) else 'invalid'</pre> <p>Explanation:</p> <ul style="list-style-type: none"> • <code>^[a-zA-Z0-9._%+-]+</code>: username part, at least one character • <code>@</code>: at symbol • <code>[a-zA-Z0-9.-]+</code>: domain name part, at least one character • <code>\.</code>: dot before TLD • <code>[a-zA-Z]{2,}\$</code>: TLD, at least 2 characters <p>Usage Example:</p> <pre>print(is_valid_email("user@example.com")) # valid print(is_valid_email("invalid-email@")) # invalid print(is_valid_email("john.doe@company.org")) # valid print(is_valid_email("noatsign.com")) # invalid</pre>	ChatGPT Output	<p>Certainly! To validate a basic email address using regex, you can use Python's `re` module. Here's how you can complete your function:</p> <pre>import re def is_valid_email(email): # Simple regex for basic email validation pattern = r'^[\w\.-]+@[^\w\.-]+\.\w+\$' return re.match(pattern, email) is not None</pre> <p>Explanation:</p> <ul style="list-style-type: none"> • <code>^[\w\.-]+</code>: Start with one or more word characters, dots, or hyphens (local part) • <code>@</code>: Literal @ symbol • <code>[\w\.-]+</code>: Domain part (at least one word char, dot or hyphen) • <code>\.</code>: Literal dot • <code>\w+\$</code>: Top-level domain (at least one word character) <p>Usage example:</p> <pre>print(is_valid_email("test@example.com")) # True print(is_valid_email("invalid-email@")) # False</pre> <p>If you need a stricter or more permissive pattern, let me know!</p>
Mistral Output	<p>To validate basic email addresses using regular expressions (regex) in Python, you can use the <code>re</code> module. A basic email address typically follows the format <code>local-part@domain</code>, where the local part can contain letters, numbers, dots, hyphens, and underscores, and the domain part consists of a domain name and a top-level domain (TLD) separated by a dot.</p> <p>Here is a Python function that uses regex to validate basic email addresses:</p> <pre>import re def is_valid_email(email): # Define the regex pattern for a basic email address 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(pattern, email): return "valid" else: return "invalid" # Test examples print(is_valid_email("user@example.com")) # Output: valid print(is_valid_email("invalid-email@")) # Output: invalid print(is_valid_email("john.doe@company.org")) # Output: valid print(is_valid_email("noatsign.com")) # Output: invalid</pre> <p>Explanation of the Regex Pattern:</p> <ul style="list-style-type: none"> • <code>^</code>: Asserts the position at the start of the string. • <code>[a-zA-Z0-9._%+-]+</code>: Matches one or more characters that are either letters (uppercase or lowercase), digits, dots, underscores, percent signs, plus signs, or hyphens. • <code>@</code>: Matches the literal "@" character. • <code>[a-zA-Z0-9.-]+</code>: Matches one or more characters that are either letters, digits, dots, or hyphens. • <code>\.</code>: Matches the literal dot character. • <code>[a-zA-Z]{2,}</code>: Matches two or more letters (uppercase or lowercase). • <code>\$</code>: Asserts the position at the end of the string. 	Mistral Output	<p>To validate basic email addresses using regular expressions (regex) in Python, you can use the <code>re</code> module. The regex pattern for a basic email address typically looks for:</p> <ol style="list-style-type: none"> 1. A local part before the @ symbol. 2. An @ symbol. 3. A domain part after the @ symbol. 4. A top-level domain (e.g., .com, .org). <p>Here is a basic implementation:</p> <pre>import re def is_valid_email(email): # Define the regex pattern for basic email validation 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</pre> <p>Explanation of the Regex Pattern:</p> <ul style="list-style-type: none"> • <code>^[a-zA-Z0-9._%+-]+</code>: Matches the local part of the email, which can include letters (both uppercase and lowercase), digits, and the characters <code>._%+-</code>. • <code>@</code>: Matches the @ symbol. • <code>[a-zA-Z0-9.-]+</code>: Matches the domain part, which can include letters, digits, dots, and hyphens. • <code>\.</code>: Matches the literal dot before the top-level domain.

This regex pattern ensures that the email address has a valid local part, an "@" sign, a valid domain name, and a valid TLD.

- [a-zA-Z]{2,}: Matches the top-level domain, which must be at least two characters long and can include only letters.

This pattern is quite basic and may not cover all edge cases or more complex email formats, but it should be sufficient for most common email address formats.

Analysis

Both zero shot and chain of thought produced identical code solutions. Chain of thought led GPT and Mistral to produce similar outputs, as the model was guided more compared to few shot. However, it does not seem that there was a difference in code correctness between zero shot and chain of thought. Zero Shot Exact match: 0.21966341895482727. Chain Of Thought Exact match: 0.2553191489361702

Task 5: Generating Flask APIs (Python)

Goal	Create a '/greet/' endpoint that returns a JSON greeting																																				
Temperature	0.7	Token Limit	2048																																		
Prompt Strategy 1	Zero Shot	Prompt Strategy 2	Chain of Thought																																		
Prompt	<p>Complete the Flask API Python function with an endpoint that returns a JSON greeting. Here's the code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): # TODO: Return a JSON greeting pass</pre>	<p>Complete the Flask API Python function with an endpoint that returns a JSON greeting. Let's think step-by-step. Here's the code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): # TODO: Return a JSON greeting pass</pre>																																			
ChatGPT Output	<p>Certainly! Here's how you can complete your Flask API function so that it returns a JSON greeting:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): return jsonify({"greeting": f"Hello, {username}!"})</pre> <p>This will return a JSON response like:</p> <pre>{ "greeting": "Hello, Alice!"}</pre> <p>when you visit /greet/Alice.</p>	<p>Absolutely! Let's go step by step:</p> <ol style="list-style-type: none"> 1. Import the required modules: You've already imported Flask and jsonify. 2. Create the Flask app: Done. 3. Create the endpoint: /greet/<username> is defined. 4. Write the function: It should take the username from the URL. 5. Return a JSON greeting: Use jsonify to return something like {"message": "Hello, <username>!"}. <p>Here's the completed code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): # Return a JSON greeting return jsonify({"message": f"Hello, {username}!"}) # If you want to run the app directly: if __name__ == '__main__': app.run(debug=True)</pre> <p>How it works:</p> <ul style="list-style-type: none"> If you visit http://localhost:5000/greet/Alice, you'll get: <pre>{ "message": "Hello, Alice!"}</pre> <p>Let me know if you need anything more!</p>	<p>Certainly! To complete the Flask API function, you need to return a JSON response with a greeting message that includes the provided <code>username</code>. You can use the <code>jsonify</code> function from Flask to create the JSON response.</p> <p>Here's the completed code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): # Create a JSON response with a greeting message greeting_message = f"Hello, {username}!" return jsonify({"message": greeting_message}) if __name__ == '__main__': app.run(debug=True)</pre> <p>Explanation:</p> <ol style="list-style-type: none"> Import Statements: <ul style="list-style-type: none"> Flask and jsonify are imported from the flask module. Flask Application: <ul style="list-style-type: none"> An instance of the Flask class is created and assigned to the variable <code>app</code>. Route Definition: <ul style="list-style-type: none"> The <code>@app.route('/greet/<username>')</code> decorator defines a route that accepts a <code>username</code> parameter. The <code>greet</code> function is defined to handle requests to this route. Greeting Message: <ul style="list-style-type: none"> Inside the <code>greet</code> function, a greeting message is created using an f-string that includes the <code>username</code> parameter. The <code>jsonify</code> function is used to create a JSON response with the greeting message. Running the Application: <ul style="list-style-type: none"> The <code>if __name__ == '__main__':</code> block ensures that the Flask application runs only if the script is executed directly. <code>app.run(debug=True)</code> starts the Flask development server with debugging mode enabled. <p>Usage:</p> <p>To test the API, you can run the script and then navigate to http://127.0.0.1:5000/greet/yourname in your web browser or use a tool like curl or Postman to make a GET request to the endpoint. Replace <code>yourname</code> with any name you want to test.</p> <p>For example, accessing http://127.0.0.1:5000/greet/Alice should return:</p> <pre>{ "message": "Hello, Alice!"}</pre> <p>Here's the completed code:</p> <pre>```python from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet/<username>') def greet(username): # Create a JSON response with a greeting message greeting_message = f"Hello, {username}!" return jsonify({"message": greeting_message}) if __name__ == '__main__': app.run(debug=True) ````</pre>	<p>Great! Let's think step by step to design a clean schema for your review app. We have three main entities: users, books, and reviews.</p> <p>Let's define each table, their fields, types, and relationships.</p> <p>1. Users Table</p> <table border="1"> <thead> <tr> <th>Column</th> <th>Type</th> <th>Constraints</th> </tr> </thead> <tbody> <tr> <td>id</td> <td>INT</td> <td>PRIMARY KEY, AUTO_INCREMENT</td> </tr> <tr> <td>name</td> <td>VARCHAR</td> <td>NOT NULL</td> </tr> </tbody> </table> <pre>CREATE TABLE users (id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT NULL);</pre> <p>2. Books Table</p> <table border="1"> <thead> <tr> <th>Column</th> <th>Type</th> <th>Constraints</th> </tr> </thead> <tbody> <tr> <td>id</td> <td>INT</td> <td>PRIMARY KEY, AUTO_INCREMENT</td> </tr> <tr> <td>title</td> <td>VARCHAR</td> <td>NOT NULL</td> </tr> </tbody> </table> <pre>CREATE TABLE books (id INT PRIMARY KEY AUTO_INCREMENT, title VARCHAR(255) NOT NULL);</pre> <p>3. Reviews Table</p> <table border="1"> <thead> <tr> <th>Column</th> <th>Type</th> <th>Constraints</th> </tr> </thead> <tbody> <tr> <td>id</td> <td>INT</td> <td>PRIMARY KEY, AUTO_INCREMENT</td> </tr> <tr> <td>user_id</td> <td>INT</td> <td>FOREIGN KEY REFERENCES users(id), NOT NULL</td> </tr> <tr> <td>book_id</td> <td>INT</td> <td>FOREIGN KEY REFERENCES books(id), NOT NULL</td> </tr> <tr> <td>rating</td> <td>INT</td> <td>NOT NULL</td> </tr> </tbody> </table> <ul style="list-style-type: none"> Optionally, you can add a <code>review_text</code> column for users to write comments. You may want to enforce that a user can only review a book once (add a UNIQUE constraint on <code>(user_id, book_id)</code>). <pre>CREATE TABLE reviews (id INT PRIMARY KEY AUTO_INCREMENT, user_id INT NOT NULL, book_id INT NOT NULL, rating INT NOT NULL, -- Optionally: -- review_text TEXT, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), UNIQUE(user_id, book_id) -- Prevent duplicate reviews by same user for same book);</pre> <p>## Summary</p> <p>Schema:</p> <pre>CREATE TABLE users (id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT NULL); CREATE TABLE books (id INT PRIMARY KEY AUTO_INCREMENT, title VARCHAR(255) NOT NULL); CREATE TABLE reviews (id INT PRIMARY KEY AUTO_INCREMENT, user_id INT NOT NULL, book_id INT NOT NULL, rating INT NOT NULL, -- Optionally: -- review_text TEXT, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), UNIQUE(user_id, book_id) -- Prevent duplicate reviews by same user for same book);</pre>	Column	Type	Constraints	id	INT	PRIMARY KEY, AUTO_INCREMENT	name	VARCHAR	NOT NULL	Column	Type	Constraints	id	INT	PRIMARY KEY, AUTO_INCREMENT	title	VARCHAR	NOT NULL	Column	Type	Constraints	id	INT	PRIMARY KEY, AUTO_INCREMENT	user_id	INT	FOREIGN KEY REFERENCES users(id), NOT NULL	book_id	INT	FOREIGN KEY REFERENCES books(id), NOT NULL	rating	INT	NOT NULL
Column	Type	Constraints																																			
id	INT	PRIMARY KEY, AUTO_INCREMENT																																			
name	VARCHAR	NOT NULL																																			
Column	Type	Constraints																																			
id	INT	PRIMARY KEY, AUTO_INCREMENT																																			
title	VARCHAR	NOT NULL																																			
Column	Type	Constraints																																			
id	INT	PRIMARY KEY, AUTO_INCREMENT																																			
user_id	INT	FOREIGN KEY REFERENCES users(id), NOT NULL																																			
book_id	INT	FOREIGN KEY REFERENCES books(id), NOT NULL																																			
rating	INT	NOT NULL																																			

Explanation:

1. Import Statements:

- Flask and jsonify are imported from the flask module.

2. Flask Application:

- An instance of the Flask class is created and assigned to the variable app.

3. Route Definition:

- The @app.route('/greet/<username>') decorator defines a route that accepts a username parameter.
- The greet function is defined to handle requests to this route.

4. Greeting Message:

- Inside the greet function, a greeting message is created using an f-string that includes the username parameter.
- The jsonify function is used to create a JSON response with the greeting message.

5. Running the Application:

- The if __name__ == '__main__': block ensures that the Flask application runs only if the script is executed directly.
- app.run(debug=True) starts the Flask development server with debugging mode enabled.

Usage:

To test the API, you can run the script and then navigate to <http://127.0.0.1:5000/greet/yourname> in your web browser or use a tool like curl or Postman to make a GET request to the endpoint. Replace yourname with any name you want to test.

For example, accessing <http://127.0.0.1:5000/greet/Alice> should return:

```
{  
    "message": "Hello, Alice!"  
}```</td>  
</tr>  
</table>  
</div>  
  
## Task 6: SQL Schema Design (SQL)  
<div style="border: 3px solid #888; border-radius: 15px; padding: 15px; margin: 10px 0; box-shadow: 0 2px 5px rgba(0,0,0,0.1);"  
<table style="border-collapse: collapse; border: none; border-radius: 12px; overflow: hidden; width: 100%; font-family: system-ui, -apple-system, sans-serif;">  
  
<tr>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold; width: 15%;">Goal</td>  
    <td style="border: none; border-bottom: 1px solid black; padding: 8px;" colspan="3">Write the schema for a review app with users, books, and reviews</td>  
    </tr>  
  
<tr>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold; width: 35%;">0.</td>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold; width: 15%;">Token Limit</td>  
    <td style="border: none; border-bottom: 1px solid black; padding: 8px; width: 35%;">2048</td>  
    </tr>  
  
<tr>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Prompt Strategy 1</td>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; width: 35%;">Prompt Chaining</td>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold; width: 15%;">Prompt Strategy 2</td>  
    <td style="border: none; border-bottom: 1px solid black; padding: 8px; width: 35%;">Chain of Thought</td>  
    </tr>  
  
<tr>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Prompt</td>  
    <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; width: 35%;">Prompt 1: Design the schema for a review app with users, books, and reviews. Define tables, columns, data types, constraints (e.g., primary/foreign keys), and relationships between entities. Here's the code: -- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id, rating)  
    </td>  
    </tr>  
    <tr>  
        <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Prompt</td>  
        <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; width: 35%;">Prompt 2: Justify the choice of database model (e.g., relational, NoSQL) for this app based on scalability, query patterns, and data structure.  
        </td>  
    </tr>  
    <tr>  
        <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Prompt 3: Create an Entity-Relationship (ER) model diagramming the connections between users, books, and reviews, including cardinality and participation constraints.  
        </td>  
    </tr>  
    <tr>  
        <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Prompt 4: Apply normalization rules (e.g., 1NF, 2NF, 3NF) to eliminate redundancy. Test the schema with sample data, identify anomalies, and refine the design iteratively.</td>  
        <td style="border: none; border-bottom: 1px solid black; padding: 8px; font-weight: bold; width: 15%;">Prompt</td>  
        <td style="border: none; border-bottom: 1px solid black; padding: 8px; width: 35%;">Write the schema for a review app with users, books, and reviews. Let's think step by step. Here's the code: -- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id, rating)</td>  
        </tr>  
        <tr>  
            <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">ChatGPT Output</td>  
            <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; width: 35%;">Output 1: Certainly! Here's a well-structured SQL schema for a review app with **users**, **books**, and **reviews**. This includes tables, columns, data types, primary/foreign keys, and relationships.  
            </td>  
        </tr>  
        <tr>  
            <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Conclusion</td>  
            <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; width: 35%;">Conclusion 1: The schema is well-normalized and suitable for a review app. It uses foreign keys to maintain data consistency across multiple tables.  
            </td>  
        </tr>  
        <tr>  
            <td style="border: none; border-bottom: 1px solid black; border-right: 1px solid black; padding: 8px; font-weight: bold;">Final Prompt</td>  
            <td style="border: none; border-bottom: 1px solid black; padding: 8px; width: 35%;">Final Prompt: How can I implement this schema in a real-world application? What are some common challenges and best practices?  
            </td>  
        </tr>  
    </table>  
}
```

```
id      INT PRIMARY KEY AUTO_INCREMENT,  
user_id INT NOT NULL,  
book_id INT NOT NULL,  
rating  INT NOT NULL,  
FOREIGN KEY (user_id) REFERENCES users(id),  
FOREIGN KEY (book_id) REFERENCES books(id),  
UNIQUE(user_id, book_id)  
);
```

Let me know if you want to add more fields (like review text, timestamps, etc.)!

```

DELETE CASCADE |
| book_id | INT          | NOT NULL, FOREIGN KEY REFERENCES books(id) ON
DELETE CASCADE |
| rating | INT           | NOT NULL, CHECK (rating >= 1 AND rating <= 5) |
| review_text | TEXT        |
| created_at | TIMESTAMP     | DEFAULT now()           |
| UNIQUE(user_id, book_id) | Ensures one review per user per book |

---

## Relationships

- **users** (1) --- (M) **reviews**: One user can write many reviews.
- **books** (1) --- (M) **reviews**: One book can have many reviews.
- **reviews**: Each review is written by one user for one book.

---

## Example SQL (PostgreSQL syntax)

```sql
CREATE TABLE users (
 id SERIAL PRIMARY KEY,
 name VARCHAR(100) NOT NULL,
 email VARCHAR(255) NOT NULL UNIQUE,
 created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE books (
 id SERIAL PRIMARY KEY,
 title VARCHAR(255) NOT NULL,
 author VARCHAR(100) NOT NULL,
 published_year INT,
 isbn VARCHAR(20) UNIQUE,
 created_at TIMESTAMP DEFAULT now()
);

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

```

Notes:

- You can add more fields as needed (e.g., profile picture for users, genre for books).
- The `UNIQUE(user_id, book_id)` constraint ensures a user can only review a book once.
- `ON DELETE CASCADE` ensures reviews are deleted if the user or book is deleted.

Let me know if you need a schema for a different SQL dialect or want to add more features!

Output 2: Certainly! Here's a justification for choosing a **relational database model** (e.g., PostgreSQL, MySQL) for your review app, considering scalability, query patterns, and data structure:

1. Data Structure

- **Highly Structured Data:**
The app's core entities—users, books, and reviews—have well-defined, structured attributes (e.g., names, titles, ratings).
- **Clear Relationships:**
There are strong, well-defined relationships:
 - Each review links a user and a book (many-to-one relationships).
 - Referential integrity (e.g., a review must reference an existing user and book) is important.
- **Normalization:**
Data can be normalized to reduce redundancy (e.g., user and book info stored once, referenced by reviews).

Relational databases are designed for structured data and enforcing relationships via foreign keys.

2. Query Patterns

- **Complex Queries:**
The app will likely need to:
 - Fetch all reviews for a book (with user info)
 - Fetch all reviews by a user
 - Aggregate ratings (e.g., average rating per book)
 - Enforce constraints (e.g., one review per user per book)
- **Joins:**
These queries require efficient joins between tables (users, books, reviews).
- **Transactions:**
Operations like creating a review may need to be atomic (e.g., avoid duplicate reviews).

Relational databases excel at complex queries, joins, and transactional integrity.

3. Scalability

- **Vertical Scaling:**
Relational databases scale well vertically (more powerful servers) and, with modern solutions (e.g., Amazon Aurora, CockroachDB), can scale horizontally for read-heavy workloads.
- **Data Volume:**
Unless the app is at massive, global scale (millions of reviews per day), a relational database can handle the expected workload efficiently.
- **Sharding/Partitioning:**
Advanced RDBMSs support sharding/partitioning if needed for very large datasets.

NoSQL databases (e.g., MongoDB, DynamoDB) are better for unstructured, rapidly evolving, or massive-scale data, but they sacrifice strong consistency and complex querying.

4. Alternatives and Trade-offs

- **NoSQL (Document/Key-Value):**
 - Pros: Flexible schema, easy horizontal scaling.
 - Cons: Weaker support for joins, referential integrity, and complex queries; more application-side logic needed.
- **Graph Databases:**
 - Pros: Good for highly interconnected data (e.g., social networks).
 - Cons: Overkill for simple review relationships; less mature ecosystem.

Conclusion

A **relational database** is the best fit for this review app because:

- The data is highly structured with clear, enforceable relationships.
- Query patterns require joins, aggregations, and constraints.
- Scalability needs are well within the capabilities of modern RDBMSs.
- It provides strong consistency, integrity, and transactional support.

In summary:

For a review app with users, books, and reviews, a relational database model is ideal due to its ability to efficiently handle structured data, enforce relationships, support complex queries, and scale to meet typical application demands.

Output 3: Certainly! Here's a textual description and an ASCII-style diagram of the Entity-Relationship (ER) model for your review app, including **entities**, **relationships**, **cardinality**, and **participation constraints**.

Entities and Attributes

- **User**
 - id (PK)
 - name
 - email
- **Book**
 - id (PK)
 - title
 - author
- **Review**
 - id (PK)
 - rating
 - review_text
 - user_id (FK)
 - book_id (FK)

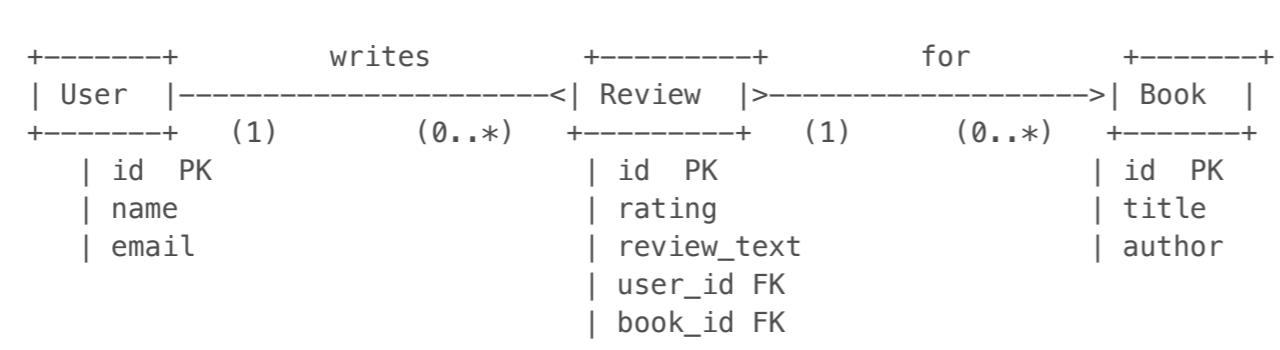
Relationships

- User writes Review
- Book receives Review

Cardinality and Participation

- **User-Review:**
 - One user can write **zero or many** reviews (1:N).
 - Each review is written by **exactly one** user (mandatory participation on Review side).
- **Book-Review:**
 - One book can have **zero or many** reviews (1:N).
 - Each review is for **exactly one** book (mandatory participation on Review side).
- **Review:**
 - Each review **must** be associated with one user and one book (total participation).

ER Diagram (ASCII Representation)



Legend:

- 1 = one
- 0..* = zero or many
- Arrow < or > points to the "many" side of the relationship.

Participation Constraints

- **User:** Partial participation (a user may not write any reviews).
- **Book:** Partial participation (a book may not have any reviews).
- **Review:** Total participation (must be linked to a user and a book).

Summary Table

| Entity | Relationship | Cardinality (User/Book:Review) | Participation (User/Book) |
|--------|--------------|--------------------------------|---------------------------|
| User | writes | 1 : 0..* | Partial |
| Book | receives | 1 : 0..* | Partial |
| Review | | * : 1 (to User/Book) | Total |

If you need a graphical ER diagram, you can use tools like dbdiagram.io, Lucidchart, or draw.io with the above structure.

Output 4: Let's walk through normalization (1NF, 2NF, 3NF) for the review app schema, test with sample data, look for anomalies, and refine as needed.

1. Initial Schema

`users (id, name, email)`
`books (id, title, author, published_year, isbn)`
`reviews (id, user_id, book_id, rating, review_text)`

Step 1: First Normal Form (1NF)

Rule:

- All attributes must be atomic (no repeating groups or arrays).

Check:

- All fields are atomic (e.g., name, title, rating are single values).
- No repeating groups.

Conclusion:

- The schema is in 1NF.

Step 2: Second Normal Form (2NF)

Rule:

- Must be in 1NF.
- No partial dependency of any column on a subset of a candidate key (applies to tables with composite keys).

Check:

- `users` and `books` have single-column primary keys, so 2NF is satisfied.
- `reviews` has a single-column PK (`id`), but the real-world uniqueness is (`user_id, book_id`).
- All non-key attributes in `reviews` depend on the whole key.

Conclusion:

- The schema is in 2NF.

Step 3: Third Normal Form (3NF)

Rule:

- Must be in 2NF.
- No transitive dependencies (non-key attributes depend only on the key).

Check:

- `users`: `name` and `email` depend only on `id`.
- `books`: `title`, `author`, `published_year`, `isbn` depend only on `id`.
- `reviews`: `rating`, `review_text` depend only on `id` (or on the composite key if used).

Conclusion:

- The schema is in 3NF.

Step 4: Test with Sample Data

Sample Data

users

| | id | name | email |
|---|-------|------|-----------------|
| 1 | Alice | | alice@email.com |
| 2 | Bob | | bob@email.com |

books

| | id | title | author | published_year | isbn |
|---|----------|----------|--------|----------------|------------|
| 1 | "Book A" | Author X | 2020 | | 1234567890 |
| 2 | "Book B" | Author Y | 2021 | | 0987654321 |

reviews

| | id | user_id | book_id | rating | review_text |
|---|----|---------|---------|--------|----------------|
| 1 | 1 | 1 | 5 | 5 | "Great book!" |
| 2 | 2 | 1 | 4 | 4 | "Enjoyed it." |
| 3 | 1 | 2 | 3 | 3 | "It was okay." |

Step 5: Identify Anomalies

Insertion Anomaly

- Can't add a review unless both user and book exist (expected).
- Can't add a user or book without a review? **No**—users and books can exist without reviews.

Update Anomaly

- If a user's name or email changes, only one place to update (no redundancy).
- If a book's title/author changes, only one place to update.

Deletion Anomaly

- Deleting a review does not remove user or book.
- Deleting the last review for a book or user does not remove the book or user (no unintended data loss).

Redundancy

- No redundant storage of user or book info in reviews.

Step 6: Refinement

Potential Refinements

- **Composite Key in reviews:**
To enforce one review per user per book, add a unique constraint on `(user_id, book_id)`.
- **Author as Separate Table?**
If authors are shared across books and you want to store more info about them, consider an `authors` table.
For now, storing author as a string in `books` is fine.

Refined reviews Table

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

Conclusion

- The schema is in **3NF**: all attributes are atomic, fully dependent on the key, and no transitive dependencies.
- **No anomalies** found with sample data.
- **No redundancy** in user or book info in reviews.
- **Refinement:** Add a unique constraint on `(user_id, book_id)` in `reviews` to prevent duplicate reviews.

The design is robust, normalized, and ready for production.

Mistral Output

Output 1: To design the schema for a review app with users, books, and reviews, we need to define the tables, columns, data types, constraints, and relationships between the entities. Below is the SQL code to create the necessary tables and define the relationships:

```
-- Create the users table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

-- Create the 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 INT NOT NULL,
    book_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    review_text TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);

-- Optional: Add indexes for better performance
CREATE INDEX idx_user_id ON reviews(user_id);
CREATE INDEX idx_book_id ON reviews(book_id);
```

Explanation:

- 1. Users Table:**
 - `id`: A unique identifier for each user (Primary Key).
 - `name`: The name of the user (VARCHAR with a maximum length of 100 characters).
- 2. Books Table:**
 - `id`: A unique identifier for each book (Primary Key).
 - `title`: The title of the book (VARCHAR with a maximum length of 255 characters).
- 3. Reviews Table:**
 - `id`: A unique identifier for each review (Primary Key).
 - `user_id`: A foreign key referencing the `id` column in the `users` table.
 - `book_id`: A foreign key referencing the `id` column in the `books` table.
 - `rating`: An integer representing the rating (with a constraint to ensure it is between 1 and 5).
 - `review_text`: A text field for the review content.
 - `created_at`: A timestamp to record when the review was created (default is the current timestamp).

Relationships:

- Each review is associated with one user and one book.
- The `user_id` in the `reviews` table references the `id` in the `users` table.
- The `book_id` in the `reviews` table references the `id` in the `books` table.

Indexes:

- Indexes on `user_id` and `book_id` in the `reviews` table to improve query performance, especially for lookups involving users and books.

This schema ensures that each review is linked to a specific user and book, and it includes constraints to maintain data integrity.

Output 2: Choosing the right database model for a review app depends on several factors, including scalability, query patterns, and data structure. Let's evaluate both relational and NoSQL database models in the context of these factors:

Relational Database Model (e.g., PostgreSQL, MySQL)

Scalability:

- **Vertical Scalability:** Relational databases are generally easier to scale vertically by adding more resources (CPU, RAM) to a single server.
- **Horizontal Scalability:** While horizontal scaling (adding more servers) is possible, it is more complex and often requires techniques like sharding, replication, and load balancing.

Query Patterns:

- **Complex Queries:** Relational databases excel at handling complex queries involving joins, aggregations, and transactions. This is beneficial for a review app where you might need to perform complex queries to retrieve reviews, user information, and book details.
- **ACID Compliance:** Relational databases ensure ACID (Atomicity, Consistency, Isolation, Durability) properties, which are crucial for maintaining data integrity in applications with frequent read and write operations.

Data Structure:

- **Structured Data:** The data in a review app is highly structured, with well-defined relationships between users, books, and reviews. Relational databases are designed to handle such structured data efficiently.
- **Schema Enforcement:** Relational databases enforce a schema, which helps in maintaining data consistency and integrity. This is important for ensuring that reviews are correctly linked to users and books.

NoSQL Database Model (e.g., MongoDB, Cassandra)

Scalability:

Mistral Output

Create a schema for a review app involves defining the structure of the tables and their relationships. We'll start by defining the `users`, `books`, and `reviews` tables, and then we'll add foreign key constraints to ensure data integrity.

Here's a step-by-step approach:

- 1. Define the users table:**
 - `id`: Primary key, unique identifier for each user.
 - `name`: Name of the user.
- 2. Define the books table:**
 - `id`: Primary key, unique identifier for each book.
 - `title`: Title of the book.
- 3. Define the reviews table:**
 - `id`: Primary key, unique identifier for each review.
 - `user_id`: Foreign key, references the `id` in the `users` table.
 - `book_id`: Foreign key, references the `id` in the `books` table.
 - `rating`: Rating given by the user for the book.

Now, let's write the SQL schema:

```
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 INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Explanation:

- 1. Users Table:**
 - `id`: SERIAL automatically increments the value.
 - `name`: VARCHAR(255) to store the user's name.
- 2. Books Table:**
 - `id`: SERIAL automatically increments the value.
 - `title`: VARCHAR(255) to store the book's title.
- 3. Reviews Table:**
 - `id`: SERIAL automatically increments the value.
 - `user_id`: Integer that references the `id` in the `users` table.
 - `book_id`: Integer that references the `id` in the `books` table.
 - `rating`: Integer that stores the rating. The CHECK constraint ensures the rating is between 1 and 5.

Foreign Key Constraints:

- `FOREIGN KEY (user_id) REFERENCES users(id)`: Ensures that `user_id` in the `reviews` table corresponds to an `id` in the `users` table.
- `FOREIGN KEY (book_id) REFERENCES books(id)`: Ensures that `book_id` in the `reviews` table corresponds to an `id` in the `books` table.

This schema ensures that each review is linked to a valid user and book, and that the rating is within a valid range.

- **Horizontal Scalability:** NoSQL databases are designed for horizontal scalability, making it easier to add more servers to handle increased load. This is beneficial for applications that need to scale out quickly.
- **Flexible Schema:** NoSQL databases allow for a flexible schema, which can be advantageous if the data structure is expected to evolve over time.

Query Patterns:

- **Simple Queries:** NoSQL databases are optimized for simple, high-throughput queries. They are less efficient at handling complex queries involving joins and aggregations.
- **Eventual Consistency:** Many NoSQL databases offer eventual consistency, which means that there might be a delay before all nodes in the database reflect the most recent updates. This can be a trade-off for applications requiring strong consistency.

Data Structure:

- **Unstructured/Semi-Structured Data:** NoSQL databases are well-suited for handling unstructured or semi-structured data. However, for a review app with well-defined relationships and structured data, this might not be a significant advantage.
- **Denormalization:** NoSQL databases often require denormalization, which can lead to data redundancy and potential inconsistencies. This can be managed but requires careful design and maintenance.

Justification for Relational Database Model:

Given the requirements of the review app, a relational database model is more suitable for the following reasons:

1. **Structured Data:** The data in the review app is highly structured, with clear relationships between users, books, and reviews. Relational databases are designed to handle such structured data efficiently.
2. **Complex Queries:** The app may require complex queries involving joins, aggregations, and transactions. Relational databases excel at handling such queries.
3. **Data Integrity:** Relational databases enforce ACID properties, ensuring data integrity and consistency, which is crucial for maintaining accurate and reliable reviews.
4. **Schema Enforcement:** The schema enforcement in relational databases helps in maintaining data consistency and integrity, which is important for ensuring that reviews are correctly linked to users and books.

While NoSQL databases offer advantages in terms of horizontal scalability and flexible schema, the structured nature of the data and the need for complex queries and data integrity make a relational database model the better choice for this review app.

Output 3: Creating an Entity-Relationship (ER) model diagram involves visualizing the entities (users, books, and reviews) and their relationships, including cardinality and participation constraints. Below is a textual description of the ER model, followed by a conceptual diagram.

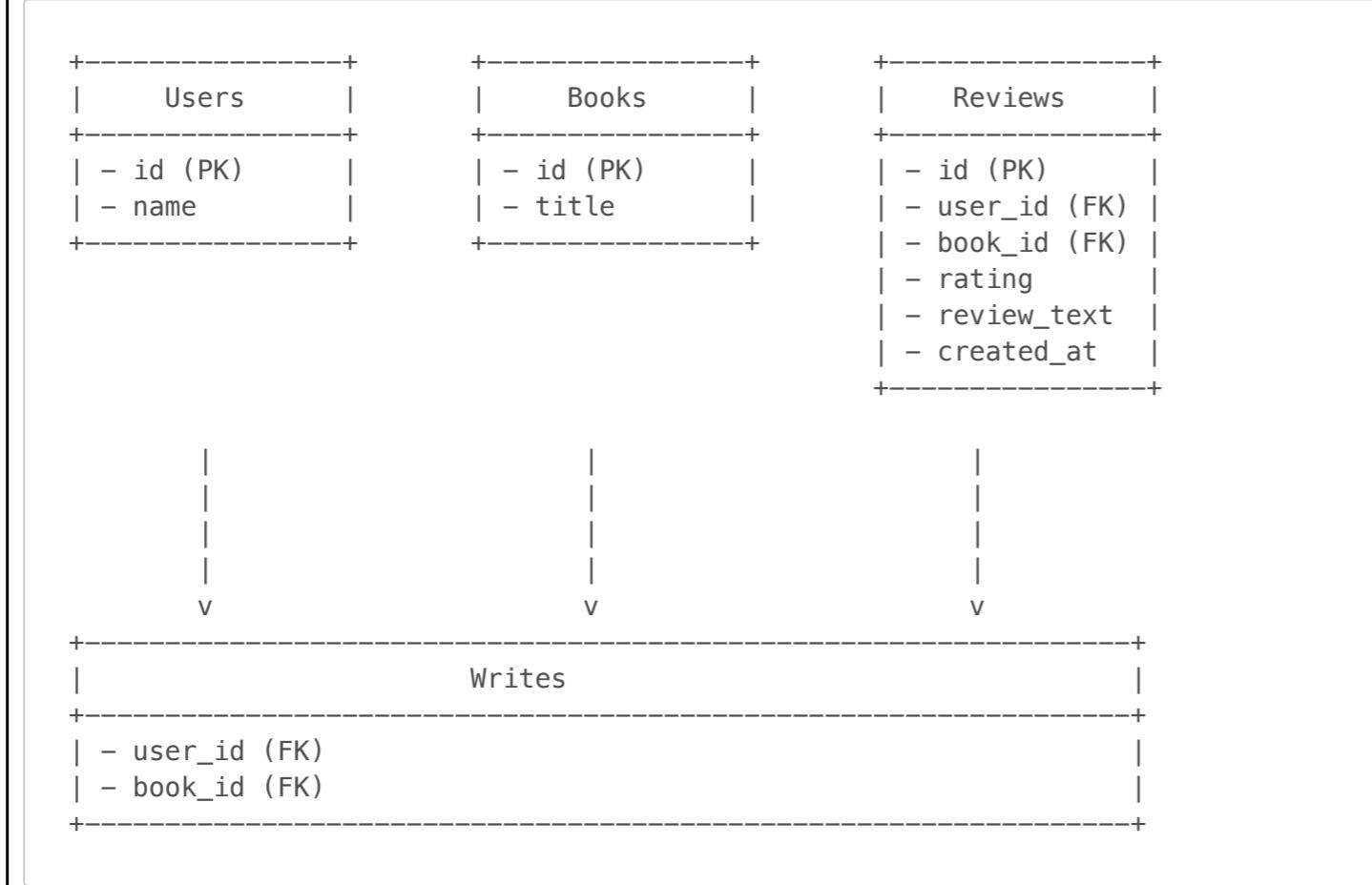
Entities and Attributes:

1. **Users:**
 - **Attributes:** id (Primary Key), name
2. **Books:**
 - **Attributes:** id (Primary Key), title
3. **Reviews:**
 - **Attributes:** id (Primary Key), user_id (Foreign Key), book_id (Foreign Key), rating, review_text, created_at

Relationships:

1. **Users to Reviews:**
 - **Cardinality:** One user can write many reviews, but each review is written by one user.
 - **Participation:** A user does not necessarily have to write a review (partial participation).
2. **Books to Reviews:**
 - **Cardinality:** One book can have many reviews, but each review is for one book.
 - **Participation:** A book does not necessarily have to have a review (partial participation).

ER Model Diagram:



Explanation:

1. **Users Entity:**
 - **id:** Primary key, uniquely identifies each user.
 - **name:** The name of the user.
2. **Books Entity:**
 - **id:** Primary key, uniquely identifies each book.
 - **title:** The title of the book.
3. **Reviews Entity:**
 - **id:** Primary key, uniquely identifies each review.
 - **user_id:** Foreign key referencing the **id** in the **Users** entity.
 - **book_id:** Foreign key referencing the **id** in the **Books** entity.
 - **rating:** The rating given by the user.
 - **review_text:** The text of the review.
 - **created_at:** The timestamp when the review was created.

Relationships:

1. **Writes Relationship:**
 - **Cardinality:** One user can write many reviews (1:N), and one book can have many reviews (1:N).
 - **Participation:** Partial participation for both users and books, meaning not every user has to write a review, and not every book has to have a review.

Diagramming Tools:

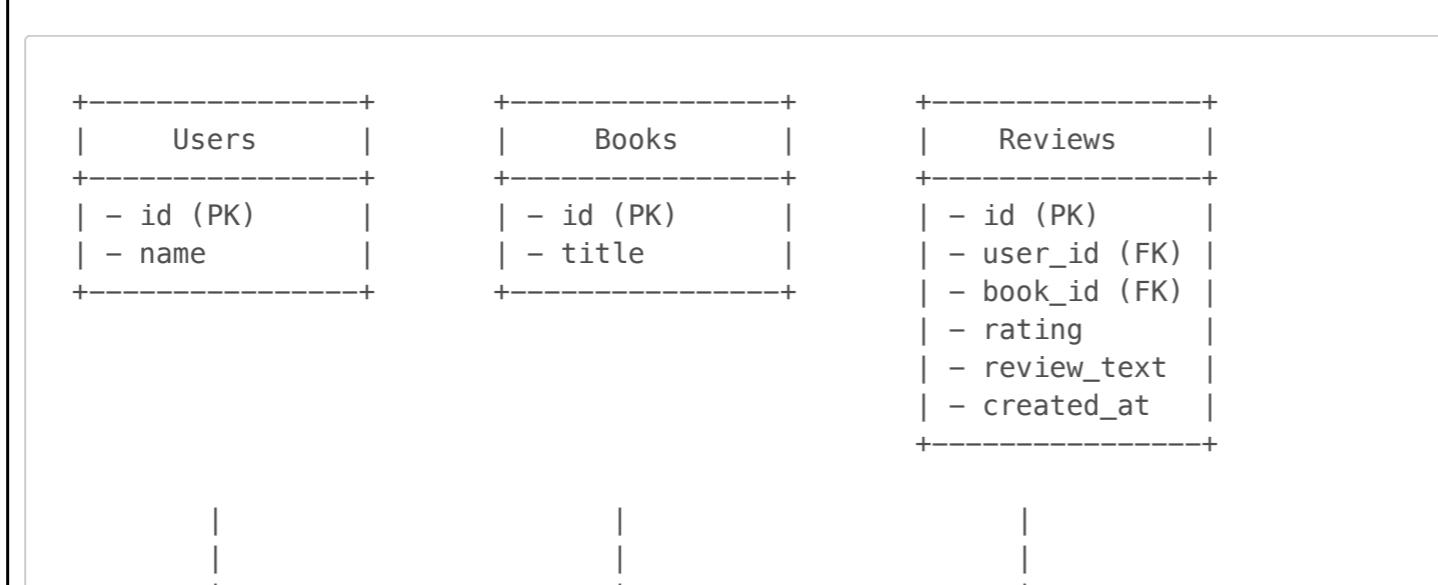
To create a visual ER diagram, you can use tools like:

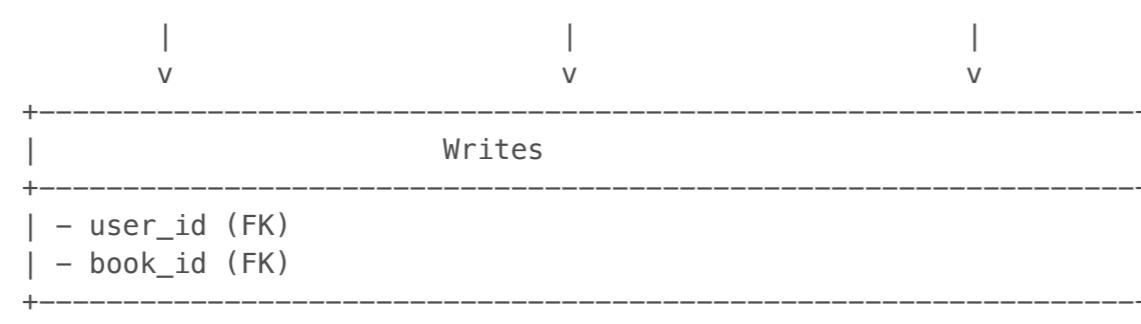
- **ER/Studio**
- **Microsoft Visio**
- **Lucidchart**
- **Draw.io**

These tools allow you to create entities, define attributes, and draw relationships with cardinality and participation constraints.

Example Diagram:

Here is a conceptual representation of the ER diagram:





This diagram visually represents the entities and their relationships, including the cardinality and participation constraints.

Output 4: Normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Let's apply the normalization rules (1NF, 2NF, 3NF) to the schema and refine it iteratively.

Initial Schema:

```
-- Users Table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

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

-- Reviews Table
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),
    review_text TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Normalization Steps:

1. First Normal Form (1NF):

- Ensure that each table contains only atomic (indivisible) values, and each column contains values of a single type.
- The initial schema is already in 1NF because each column contains atomic values.

2. Second Normal Form (2NF):

- Ensure that the table is in 1NF.
- Remove partial dependencies, i.e., non-key attributes should be fully functionally dependent on the primary key.
- The initial schema is already in 2NF because there are no partial dependencies. Each non-key attribute in the `reviews` table is fully functionally dependent on the primary key `id`.

3. Third Normal Form (3NF):

- Ensure that the table is in 2NF.
- Remove transitive dependencies, i.e., non-key attributes should not depend on other non-key attributes.
- The initial schema is already in 3NF because there are no transitive dependencies. Each non-key attribute in the `reviews` table is directly dependent on the primary key `id`.

Testing the Schema with Sample Data:

Let's insert some sample data and identify any anomalies:

```
-- Insert sample users
INSERT INTO users (name) VALUES ('Alice'), ('Bob'), ('Charlie');

-- Insert sample books
INSERT INTO books (title) VALUES ('Book A'), ('Book B'), ('Book C');

-- Insert sample reviews
INSERT INTO reviews (user_id, book_id, rating, review_text) VALUES
(1, 1, 5, 'Great book!'),
(2, 1, 4, 'Good read.'),
(3, 2, 3, 'Okay book.'),
(1, 3, 5, 'Another great book!');
```

Identifying Anomalies:

- 1. Insertion Anomaly:** If we try to insert a review for a user or book that does not exist, we will get a foreign key constraint violation. This is expected and ensures data integrity.
- 2. Update Anomaly:** If we update the name of a user or the title of a book, it will be reflected in all related reviews. This is expected and ensures data consistency.
- 3. Deletion Anomaly:** If we delete a user or book, all related reviews will be orphaned. This can be mitigated by setting the foreign key constraints to `ON DELETE CASCADE` or `ON DELETE SET NULL`.

Refining the Design:

To handle deletion anomalies, we can modify the foreign key constraints:

```
-- Modify the reviews table to handle deletion anomalies
ALTER TABLE reviews
DROP CONSTRAINT reviews_user_id_fkey,
DROP CONSTRAINT reviews_book_id_fkey,
ADD CONSTRAINT reviews_user_id_fkey FOREIGN KEY (user_id) REFERENCES users(id)
ON DELETE CASCADE,
ADD CONSTRAINT reviews_book_id_fkey FOREIGN KEY (book_id) REFERENCES books(id)
ON DELETE CASCADE;
```

Final Schema:

```
-- Users Table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

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

-- Reviews Table
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),
    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
);
```

Conclusion:

The schema is now in 3NF and handles insertion, update, and deletion anomalies effectively. The use of foreign key constraints with `ON DELETE CASCADE` ensures that reviews are automatically deleted when the associated user or book is deleted, maintaining data integrity.

Analysis

Prompt chaining guided both models through a comprehensive design, covering tables, justification, ER modeling, and normalization. GPT-4 provided more detailed SQL constraints and ER descriptions across steps. Compared to chain-of-thought, this multi-step process ensured broader coverage but potentially less optimization detail within a single step. Prompt Chaining Exact match: 0.03729796933278077. Chain Of Thought Exact match: 0.23081774227454277

Task 7: Null Dereference Detection (Java)

Goal

Identify any null dereference risk

| | | | |
|-------------------|--|-------------------|---|
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Zero Shot | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Identify any null dereference risk in the Java function. Here's the code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> | Prompt | <p>Identify any null dereference risk in the Java function. Here's the code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> |
| ChatGPT Output | <p>Yes, there is a null dereference risk in your Java function.</p> <p>Code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>Explanation:</p> <p>If the input <code>s</code> is <code>null</code>, calling <code>s.length()</code> will throw a <code>NullPointerException</code>.</p> <p>Example of the risk:</p> <pre>getLength(null); // This will throw NullPointerException</pre> <p>How to Fix</p> <p>You should check for <code>null</code> before calling <code>.length()</code>. For example:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // or handle as appropriate } return s.length(); }</pre> <p>Or, using Java 8+:</p> <pre>public int getLength(String s) { return (s == null) ? 0 : s.length(); }</pre> <p>Summary:
There is a null dereference risk in <code>s.length()</code> if <code>s</code> is <code>null</code>. Always check for <code>null</code> before dereferencing an object reference.</p> | ChatGPT Output | <p>Output 1: Yes, there is a null dereference risk in your function.</p> <p>Code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>Explanation:</p> <p>If the parameter <code>s</code> is <code>null</code>, calling <code>s.length()</code> will throw a <code>NullPointerException</code> at runtime. This is known as a null dereference.</p> <p>Example:</p> <pre>getLength(null); // This will throw NullPointerException</pre> <p>How to fix?</p> <p>You should check if <code>s</code> is <code>null</code> before calling <code>s.length()</code>:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // or throw an exception, or handle as appropriate } return s.length(); }</pre> <p>Summary:
Risk: Yes, if <code>s</code> is <code>null</code>, <code>s.length()</code> will throw a <code>NullPointerException</code>.
Mitigation: Add a null check before dereferencing <code>s</code>.</p> <p>Output 2: Yes, there is a null dereference risk in the given Java function.</p> <p>Code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>Risk Explanation:</p> <ul style="list-style-type: none"> If <code>s</code> is <code>null</code> when passed to this function, then calling <code>s.length()</code> will throw a <code>NullPointerException</code>. This is because you cannot call instance methods on a <code>null</code> reference. <p>How to fix it?</p> <p>You can add a null check:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // or throw an exception, depending on your requirements } return s.length(); }</pre> <p>Summary:
Yes, there is a null dereference risk if <code>s</code> is <code>null</code>.</p> <p>Output 3: Let's analyze the provided Java function:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>Null Dereference Risk:</p> <p>Yes, there is a null dereference risk in this function.</p> <p>Why?</p> <p>If the parameter <code>s</code> is <code>null</code> when this function is called, <code>s.length()</code> will throw a <code>NullPointerException</code>.</p> <p>Example Demonstrating the Risk</p> <pre>getLength(null); // This will throw NullPointerException</pre> <p>How to Fix?</p> <p>Add a null check before accessing the method on <code>s</code>:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // Or handle the null case as appropriate } return s.length(); }</pre> <p>Summary:
The function as written does have a potential null dereference risk. You should check for <code>null</code> before dereferencing <code>s</code>.</p> |
| Mistral Output | <p>The provided Java function <code>getLength</code> has a potential null dereference risk. Here's the code:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>The risk arises because the method attempts to call the <code>length()</code> method on the parameter <code>s</code>, which is of type <code>String</code>. If the caller passes a <code>null</code> value for <code>s</code>, attempting to call <code>s.length()</code> will result in a <code>NullPointerException</code>.</p> <p>To avoid this risk, you should check whether <code>s</code> is <code>null</code> before calling the <code>length()</code> method. Here's an improved version of the function:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // or throw an <code>IllegalArgumentException</code>, depending on your requirements } return s.length(); }</pre> <p>In this revised version, the function checks if <code>s</code> is <code>null</code> and handles it appropriately. Depending on the requirements, you might want to return <code>0</code> or throw an <code>IllegalArgumentException</code>. Returning <code>0</code> is a common approach to indicate that the length of a <code>null</code> string is considered <code>0</code>. Alternatively, throwing an <code>IllegalArgumentException</code> can be useful to signal that the input was invalid and the caller should handle it accordingly.</p> <p>Here's the version that throws an <code>IllegalArgumentException</code>:</p> <pre>public int getLength(String s) { if (s == null) { throw new <code>IllegalArgumentException</code>("Input string cannot be null"); } return s.length(); }</pre> | Mistral Output | <p>Output 1: The provided Java function <code>getLength</code> has a potential null dereference risk. Here's the code for reference:</p> <pre>public int getLength(String s) { return s.length(); }</pre> <p>Analysis:</p> <p>The function <code>getLength</code> takes a <code>String</code> parameter <code>s</code> and returns its length. However, if the input <code>s</code> is <code>null</code>, calling <code>s.length()</code> will result in a <code>NullPointerException</code>. This is because the method <code>length()</code> is being invoked on a <code>null</code> object.</p> <p>Solution:</p> <p>To avoid this null dereference risk, you should check if <code>s</code> is <code>null</code> before attempting to call <code>s.length()</code>. Here is a revised version of the function that includes a null check:</p> <pre>public int getLength(String s) { if (s == null) { return 0; // or throw an exception, depending on your requirements } return s.length(); }</pre> <p>In this version, if <code>s</code> is <code>null</code>, the function returns <code>0</code>. Alternatively, you could throw an exception to indicate that <code>null</code> is not an acceptable input. Here is an example of throwing an exception:</p> <pre>public int getLength(String s) { if (s == null) { throw new <code>IllegalArgumentException</code>("Input string cannot be null"); } return s.length(); }</pre> |

```

        throw new IllegalArgumentException("Input string cannot be null");
    }
    return s.length();
}

```

Choose the approach that best fits your application's requirements and error-handling strategy.

```

    }
    return s.length();
}

```

Choose the approach that best fits the requirements of your application.

Output 2: The provided Java function `getLength` takes a `String` parameter `s` and returns its length by calling the `length()` method. There is indeed a risk of a null dereference in this function.

Here's the code for reference:

```

public int getLength(String s) {
    return s.length();
}

```

Null Dereference Risk

The risk arises because the method does not check if the `String s` is `null` before calling `s.length()`. If `s` is `null`, attempting to call `s.length()` will result in a `NullPointerException`.

Solution

To mitigate this risk, you should add a null check before calling `s.length()`. Here is an updated version of the function that includes this check:

```

public int getLength(String s) {
    if (s == null) {
        return 0; // or throw an exception, depending on the desired behavior
    }
    return s.length();
}

```

Explanation

- The `if (s == null)` check ensures that the function safely handles the case where `s` is `null`.
- If `s` is `null`, the function returns `0`. You could also choose to throw an exception or handle the null case in another way, depending on your application's requirements.

This way, you avoid the `NullPointerException` and make the function more robust.

Output 3: The provided Java function `getLength` has a null dereference risk. Specifically, if the input parameter `s` is `null`, calling `s.length()` will result in a `NullPointerException`.

Here's the code for reference:

```

public int getLength(String s) {
    return s.length();
}

```

To mitigate this risk, you should check if `s` is `null` before calling `s.length()`. Here's a modified version of the function that includes this check:

```

public int getLength(String s) {
    if (s == null) {
        return 0; // or throw an exception, depending on the desired behavior
    }
    return s.length();
}

```

Alternatively, if you prefer to throw an exception when `s` is `null`, you can do so like this:

```

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

```

These modifications ensure that the function handles `null` input gracefully, preventing a `NullPointerException`.

Analysis

Both zero shot and self consistency produced the same code solution for GPT and Mistral. With self consistency two outputs identified null dereference risk with certainty while one output had uncertainty. With zero shot the Mistral model was uncertain whereas the GPT model was certain. Thus, self-consistency was a better approach as we were able to pick the more consistent answer. Zero Shot Exact match: 0.2005097706032855. Self-Consistency Exact match: 0.13512119649303764

Task 8: CSV Parser Variants (Python)

| | | | |
|-------------------|---|-------------------|--|
| Goal | Improve the parser to support quoted fields | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Few Shot | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Example 1:
Input: 'John,Doe,"123, Elm Street",Engineer'
Output: ['John', 'Doe', '123, Elm Street', 'Engineer']</p> <p>Example 2:
Input: "Jane, A.",Smith,"456 Oak Avenue",Doctor'
Output: [Jane, A., Smith, '456 Oak Avenue', 'Doctor']</p> <p>Example 3:
Input: 'Alice,Brown,"789 Pine St, Apt 4B","Data Scientist'"
Output: ['Alice', 'Brown', '789 Pine St, Apt 4B', 'Data Scientist']</p> <p>Now, improve the Python parser function to handle quoted fields in CSV lines. Quoted fields may contain commas, which should not be split. Here's the code:</p> <pre> def parse_csv_line(line): return line.split(',') </pre> | Prompt | <p>Improve the parser to support quoted fields. Here's the code:</p> <pre> def parse_csv_line(line): return line.split(',') </pre> |
| ChatGPT Output | <p>Of course! Your requirement is to properly parse a CSV line where fields may be quoted and quoted fields may contain commas. The current code:</p> <pre> def parse_csv_line(line): return line.split(',') </pre> <p>Shortcoming:
This code splits on every comma, which will split inside quotes, so you can't handle entries like "123, Elm Street"; the comma inside quotes will cause an extra split.</p> <p>Let's improve it to handle quoted fields, including commas within them.</p> <p>You can use Python's built-in <code>csv</code> module for robustness. However, to demonstrate the core parsing logic, here's a custom parser that mimics the behavior:</p> <p>Improved Custom CSV Parser</p> <pre> def parse_csv_line(line): result = [] field = '' in_quotes = False i = 0 length = len(line) while i < length: char = line[i] if char == '"': if in_quotes: # Check for escaped quote if i + 1 < length and line[i+1] == '"': field += '"' i += 1 else: in_quotes = True else: in_quotes = False result.append(field) field = '' i += 1 return result </pre> | ChatGPT Output | <p>Certainly! The current implementation using <code>.split(',')</code> doesn't handle quoted fields containing commas or quotes. For example:</p> <pre> parse_csv_line('foo,"bar,baz",qux') # gives ['foo', '"bar', 'baz"', 'qux'] ← incorrect </pre> <p>To support quoted fields properly, you'll need a more robust parser. Here's an improved version:</p> <pre> def parse_csv_line(line): fields = [] field = '' in_quotes = False i = 0 while i < len(line): char = line[i] if char == '"' and (not in_quotes): in_quotes = True elif char == '"' and in_quotes: if i + 1 < len(line) and line[i+1] == '"': # Escaped quote field += '"' i += 1 else: in_quotes = False elif char == ',' and not in_quotes: fields.append(field) field = '' else: field += char i += 1 fields.append(field) return fields </pre> <p>Example Usage:</p> |

```

        elif char == ',' and not in_quotes:
            result.append(field)
            field = ''
        else:
            field += char
            i += 1
    result.append(field)
    return result

```

Alternatively: Use the csv module (Recommended)

```

import csv
from io import StringIO

def parse_csv_line(line):
    f = StringIO(line)
    reader = csv.reader([line])
    return next(reader)

```

Usage

```

print(parse_csv_line('John,Doe,"123, Elm Street",Engineer'))
# Output: ['John', 'Doe', '123, Elm Street', 'Engineer']

```

Explanation:

- Handles quoted fields.
- Supports commas inside quoted fields.
- Handles escaped quotes ("\" inside quoted string).
- Appends fields correctly.

If you want no dependencies and a clear understanding, use the manual function. For production use, prefer the `csv` module approach.

```

print(parse_csv_line('foo,"bar,baz",qux'))
# Output: ['foo', 'bar,baz', 'qux']

print(parse_csv_line("a","b""c","d,e"))
# Output: ['a', 'b"c', 'd,e']

```

Note: For real-world usage, Python's built-in `csv` module is much more robust and battle-tested:

```

import csv
def parse_csv_line(line):
    return next(csv.reader([line]))

```

But if you want your own implementation, the above function handles standard quoted fields and escaped quotes inside fields.

Mistral Output

To handle quoted fields in CSV lines, we need to implement a more sophisticated parser that can correctly identify and handle quoted fields, including those that contain commas. Here's an improved version of the `parse_csv_line` function that accomplishes this:

```

import csv
from io import StringIO

def parse_csv_line(line):
    # Use the csv module to handle quoted fields
    csv_reader = csv.reader(StringIO(line))
    return next(csv_reader)

# Example usage:
input1 = 'John,Doe,"123, Elm Street",Engineer'
input2 = '"Jane, A.",Smith,"456 Oak Avenue",Doctor'
input3 = 'Alice,Brown,"789 Pine St, Apt 4B",Data Scientist'

print(parse_csv_line(input1)) # Output: ['John', 'Doe', '123, Elm Street', 'Engineer']
print(parse_csv_line(input2)) # Output: ['Jane, A.', 'Smith', '456 Oak Avenue', 'Doctor']
print(parse_csv_line(input3)) # Output: ['Alice', 'Brown', '789 Pine St, Apt 4B', 'Data Scientist']

```

Explanation:

- Importing Modules:**
 - `csv`: The `csv` module provides functionality to both read from and write to CSV files.
 - `StringIO`: The `StringIO` class from the `io` module is used to treat a string as a file-like object, which is necessary for the `csv.reader` to process the input string.
- Creating a CSV Reader:**
 - `csv.reader(StringIO(line))`: This creates a CSV reader object that can read from the string line as if it were a file.
- Reading the Line:**
 - `next(csv_reader)`: This reads the first (and only) line from the CSV reader object, which correctly handles quoted fields and commas within those fields.

This approach ensures that quoted fields are correctly parsed, and commas within those fields are not treated as delimiters.

Mistral Output

To improve the parser to support quoted fields, you need to handle cases where commas are part of the field values and are enclosed in quotes. This can be achieved by using a more sophisticated approach than simply splitting the line by commas.

Here's an improved version of the `parse_csv_line` function that supports quoted fields:

```

import csv
from io import StringIO

def parse_csv_line(line):
    # Use the csv module to handle quoted fields
    csv_reader = csv.reader(StringIO(line))
    return next(csv_reader)

# Example usage
line = 'field1,"field2, with comma",field3'
parsed_line = parse_csv_line(line)
print(parsed_line) # Output: ['field1', 'field2, with comma', 'field3']

```

Explanation:

- Import the `csv` module:** The `csv` module in Python provides functionality to read and write CSV files, including handling quoted fields.
- Use `StringIO`:** The `StringIO` class from the `io` module allows you to treat a string as a file-like object, which is necessary for the `csv.reader` to process the line.
- Create a `csv.reader` object:** Pass the `StringIO` object to `csv.reader` to create a reader object.
- Read the line:** Use `next(csv_reader)` to read the first (and only) line from the reader object.

This approach ensures that quoted fields are correctly parsed, even if they contain commas.

Analysis

Few shot and self-consistency had the same code outputs within the same model. However, GPT provides two code solutions one using the `csv` module and one without. Mistral provides one code solution using the `csv` module. Self-consistency was able to generate its own examples like the few-shot prompt. Thus, example-based prompting may not make the solutions that much better in this case. Few Shot Exact match: 0.090884790163058. Self-Consistency Exact match: 0.03810868031051517

Task 9: Data Class to API Conversion (Kotlin)

| | | | |
|-------------------|---|---|-----------------|
| Goal | Convert the data class to a REST API using Ktor | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Few Shot | Prompt Strategy 2 | Prompt Chaining |
| Prompt | <p>Example 1:
Input:</p> <pre> data class Book(val id: Int, val title: String, val author: String) </pre> <p>Output:</p> <pre> routing { route("/books") { get { call.respond(books) } get("{id}") { val id = call.parameters["id"]?.toIntOrNull() books.find { it.id == id }?.let { call.respond(it) } ?: call.respond(HttpStatusCode.NotFound) } post { books.add(call.receive<Book>()) } call.respond(HttpStatusCode.Created) put("{id}") { val id = call.parameters["id"]?.toIntOrNull() val index = books.indexOfFirst { it.id == id } if (index != -1) books[index] = call.receive() else call.respond(HttpStatusCode.NotFound) } delete("{id}") { if (books.removeIf { it.id == call.parameters["id"]?.toIntOrNull() }) call.respond(HttpStatusCode.NoContent) else call.respond(HttpStatusCode.NotFound) } } } </pre> <p>Example 2:
Input:</p> <pre> data class User(val id: Int, val name: String, val email: String) </pre> <p>Output:</p> <pre> routing { route("/users") { </pre> | <p>Prompt</p> <p>Prompt 1: Given the following Kotlin data class, summarize its fields and their types. Here's the code:</p> <pre> data class Product(val id: Int, val name: String, val price: Double) </pre> <p>Prompt 2: Based on the fields and purpose of the data class, list the REST API endpoints required to perform CRUD (Create, Read, Update, Delete) operations. Use the following format: Endpoint: [HTTP Method] [Path] – [Description].</p> <p>Prompt 3: Using the list of endpoints above, generate Ktor route stubs in Kotlin. Each stub should include the HTTP method, path, and a placeholder for the handler logic.</p> <p>Prompt 4: For each route stub, implement the handler logic using an in-memory list to store Book objects. Show how to add, retrieve, update, and delete books in this list.</p> <p>Prompt 5: Combine the data class, in-memory storage, and all route handlers into a complete Ktor application file. Ensure the application is runnable and includes all necessary imports and setup code.</p> | |

```

        get { call.respond(users) }
        get("{id}") {
            users.find { it.id == call.parameters["id"]?.toIntOrNull() }?.let {
                call.respond(it)
            ?: call.respond(HttpStatusCode.NotFound)
        }
        post { users.add(call.receive<User>()); call.respond(HttpStatusCode.Created) }
        put("{id}") {
            users.indexOfFirst { it.id == call.parameters["id"]?.toIntOrNull() }.let { index ->
                if (index != -1) users[index] = call.receive() else call.respond(HttpStatusCode.NotFound)
            }
        }
        delete("{id}") {
            if (users.removeIf { it.id == call.parameters["id"]?.toIntOrNull() }) call.respond(HttpStatusCode.NoContent)
            else call.respond(HttpStatusCode.NotFound)
        }
    }
}

```

Now, convert the Kotlin data class to a REST API using Ktor with CRUD operations. Here's the code:

```

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

```

ChatGPT Output

Certainly! Based on your previous examples, here's a Ktor REST API routing block for the data class:

```

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

val products = mutableListOf<Product>()

routing {
    route("/products") {
        get { call.respond(products) }
        get("{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            products.find { it.id == id }?.let { call.respond(it) }
            ?: call.respond(HttpStatusCode.NotFound)
        }
        post {
            products.add(call.receive<Product>())
            call.respond(HttpStatusCode.Created)
        }
        put("{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            val index = products.indexOfFirst { it.id == id }
            if (index != -1) products[index] = call.receive() else call.respond(HttpStatusCode.NotFound)
        }
        delete("{id}") {
            if (products.removeIf { it.id == call.parameters["id"]?.toIntOrNull() }) call.respond(HttpStatusCode.NoContent)
            else call.respond(HttpStatusCode.NotFound)
        }
    }
}

```

Usage:

- **GET /products:** Get all products
- **GET /products/{id}:** Get product by id
- **POST /products:** Add a new product
- **PUT /products/{id}:** Update product by id
- **DELETE /products/{id}:** Delete product by id

If you need the [full file](#) or [setup tips](#) (e.g., serialization install), just let me know!

ChatGPT Output

Output 1: Here's a summary of the fields in the `Product` Kotlin data class:

- **id: Int** — An integer representing the product's unique identifier.
- **name: String** — A string containing the product's name.
- **price: Double** — A double-precision floating-point number representing the product's price.

Output 2: Certainly! Here are the REST API endpoints for CRUD operations on the `Product` data class:

- Create Product**
Endpoint: POST `/products` – Create a new product.
- Read All Products**
Endpoint: GET `/products` – Retrieve a list of all products.
- Read Single Product**
Endpoint: GET `/products/{id}` – Retrieve details of a specific product by its ID.
- Update Product**
Endpoint: PUT `/products/{id}` – Update an existing product by its ID.
- Delete Product**
Endpoint: DELETE `/products/{id}` – Delete a specific product by its ID.

Output 3: Certainly! Here are Ktor route stubs in Kotlin for the CRUD endpoints:

```

import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

fun Route.productRoutes() {
    // Create Product
    post("/products") {
        // TODO: Add logic to create a new product
        call.respondText("Create Product - Not Implemented")
    }

    // Read All Products
    get("/products") {
        // TODO: Add logic to list all products
        call.respondText("Get All Products - Not Implemented")
    }

    // Read Single Product
    get("/products/{id}") {
        val id = call.parameters["id"]
        // TODO: Add logic to get a product by id
        call.respondText("Get Product $id - Not Implemented")
    }

    // Update Product
    put("/products/{id}") {
        val id = call.parameters["id"]
        // TODO: Add logic to update a product by id
        call.respondText("Update Product $id - Not Implemented")
    }

    // Delete Product
    delete("/products/{id}") {
        val id = call.parameters["id"]
        // TODO: Add logic to delete a product by id
        call.respondText("Delete Product $id - Not Implemented")
    }
}

```

You would typically call `productRoutes()` inside your application's main routing `{ ... }` block. Each route contains a placeholder (`TODO`) where you can add your actual handler logic.

Output 4: Certainly! Below you'll find Ktor route implementations for CRUD operations, using an in-memory list of `Book` objects. I'll cover:

- The `Book` data class
- Thread-safe in-memory storage (`mutableListOf`)
- Per-route logic: create, read-all, read-one, update, delete
- Required imports and request body parsing for creating/updating books

1. Define the `Book` Data Class

```

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

```

2. Storage and ID Generation

```

val books = mutableListOf<Book>()
var nextId = 1
val lock = Any()

```

3. Book Routes Implementation

```

import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import io.ktor.http.*

fun Route.bookRoutes() {
    // Create Book
    post("/books") {
        val requestBook = call.receive<Book>() // expects id, but we'll
        override it below

        // Generate unique ID
        val book = synchronized(lock) {
            val newBook = requestBook.copy(id = nextId++)
            books.add(newBook)
            newBook
        }
        call.respond(HttpStatusCode.Created, book)
    }

    // Read All Books
    get("/books") {
        call.respond(books)
    }

    // Read Single Book
    get("/books/{id}") {

```

```

    val id = call.parameters["id"]?.toIntOrNull()
    if (id == null) {
        call.respond(HttpStatusCode.BadRequest, "Invalid id")
        return@get
    }
    val book = books.find { it.id == id }
    if (book != null) {
        call.respond(book)
    } else {
        call.respond(HttpStatusCode.NotFound, "Book not found")
    }
}

// Update Book
put("/books/{id}") {
    val id = call.parameters["id"]?.toIntOrNull()
    if (id == null) {
        call.respond(HttpStatusCode.BadRequest, "Invalid id")
        return@put
    }
    val updatedBook = call.receive<Book>()
    synchronized(lock) {
        val index = books.indexOfFirst { it.id == id }
        if (index == -1) {
            call.respond(HttpStatusCode.NotFound, "Book not found")
        } else {
            val bookToSave = updatedBook.copy(id = id)
            books[index] = bookToSave
            call.respond(bookToSave)
        }
    }
}

// Delete Book
delete("/books/{id}") {
    val id = call.parameters["id"]?.toIntOrNull()
    if (id == null) {
        call.respond(HttpStatusCode.BadRequest, "Invalid id")
        return@delete
    }
    synchronized(lock) {
        val removed = books.removeIf { it.id == id }
        if (removed) {
            call.respond(HttpStatusCode.NoContent)
        } else {
            call.respond(HttpStatusCode.NotFound, "Book not found")
        }
    }
}
}

```

4. Register the Routes

In your Application.module (or wherever you define routes):

```

fun Application.module() {
    routing {
        bookRoutes()
    }
}

```

5. Notes

- All data is **in-memory**; app restart resets data.
- For production, replace with persistent storage.
- The request body for POST/PUT should be JSON, e.g.:

```
{
    "name": "Kotlin in Action",
    "price": 39.99
}
```

(However, the `id` in requests is ignored or overriddenundefined

Output 5: Absolutely! Here's a complete runnable Ktor server application (using Ktor 2.x) that manages **Product** instances in-memory with the specified CRUD endpoints.

File: Application.kt

```

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

// 1. Data class (Serializable for JSON support)
@Serializable
data class Product(val id: Int, val name: String, val price: Double)

// 2. In-memory storage
val products = mutableListOf<Product>()
var nextId = 1
val lock = Any()

// 3. Route handlers
fun Route.productRoutes() {
    // Create Product
    post("/products") {
        val requestProduct = call.receive<Product>() // expects id, but we'll
        override
        val product = synchronized(lock) {
            val newProduct = requestProduct.copy(id = nextId++)
            products.add(newProduct)
            newProduct
        }
        call.respond(HttpStatusCode.Created, product)
    }

    // Read All Products
    get("/products") {
        call.respond(products)
    }

    // Read Single Product
    get("/products/{id}") {
        val id = call.parameters["id"]?.toIntOrNull()
        if (id == null) {
            call.respond(HttpStatusCode.BadRequest, "Invalid id")
            return@get
        }
        val product = products.find { it.id == id }
        if (product != null) {
            call.respond(product)
        } else {
            call.respond(HttpStatusCode.NotFound, "Product not found")
        }
    }

    // Update Product
    put("/products/{id}") {
        val id = call.parameters["id"]?.toIntOrNull()
        if (id == null) {
            call.respond(HttpStatusCode.BadRequest, "Invalid id")
            return@put
        }
        val updatedProduct = call.receive<Product>()
        synchronized(lock) {
            val index = products.indexOfFirst { it.id == id }
            if (index == -1) {
                call.respond(HttpStatusCode.NotFound, "Product not found")
            } else {
                val productToSave = updatedProduct.copy(id = id)
                products[index] = productToSave
                call.respond(productToSave)
            }
        }
    }

    // Delete Product
}

```

```

        delete("/products/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            if (id == null) {
                call.respond(HttpStatusCode.BadRequest, "Invalid id")
                return@delete
            }
            synchronized(lock) {
                val removed = products.removeIf { it.id == id }
                if (removed) {
                    call.respond(HttpStatusCode.NoContent)
                } else {
                    call.respond(HttpStatusCode.NotFound, "Product not found")
                }
            }
        }
    }

    // 4. Application entry point
    fun main() {
        embeddedServer(Netty, port = 8080) {
            install(ContentNegotiation) {
                json()
            }
            routing {
                productRoutes()
            }
        }.start(wait = true)
    }
}

```

How to Run

- Make sure you have the Ktor dependencies for 2.x, including `ktor-server-core`, `ktor-server-netty`, `ktor-server-content-negotiation`, `ktor-serialization-kotlinx-json`.
- To test:
 - Run the application.
 - Send requests like `POST /products`, `GET /products`, etc. to `http://localhost:8080/products`.

Note:

- The client POST or PUT should send JSON like `undefined`

Mistral Output

To convert the given Kotlin data class `Product` into a REST API using Ktor with CRUD operations, you can follow the pattern shown in the examples provided. Below is the complete implementation:

```

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

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

fun main() {
    val products = mutableListOf<Product>()

    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }
        routing {
            route("/products") {
                get {
                    call.respond(products)
                }
                get("{id}")
                {
                    val id = call.parameters["id"]?.toIntOrNull()
                    products.find { it.id == id }?.let { call.respond(it) }
                    ?: call.respond(HttpStatusCode.NotFound)
                }
                post {
                    val product = call.receive<Product>()
                    products.add(product)
                    call.respond(HttpStatusCode.Created)
                }
                put("{id}")
                {
                    val id = call.parameters["id"]?.toIntOrNull()
                    val index = products.indexOfFirst { it.id == id }
                    if (index != -1) {
                        products[index] = call.receive()
                        call.respond(HttpStatusCode.OK)
                    } else {
                        call.respond(HttpStatusCode.NotFound)
                    }
                }
                delete("{id}")
                {
                    val id = call.parameters["id"]?.toIntOrNull()
                    if (products.removeIf { it.id == id }) {
                        call.respond(HttpStatusCode.NoContent)
                    } else {
                        call.respond(HttpStatusCode.NotFound)
                    }
                }
            }
        }
    }.start(wait = true)
}

```

Explanation:

1. Data Class Definition:

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

2. Main Function:

◦ Server Setup:

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

This sets up a Ktor server using Netty on port 8080 and installs the `ContentNegotiation` feature with JSON serialization.

◦ Routing:

```
routing {
    route("/products") {
        // CRUD operations
    }
}
```

This defines the routing for the `/products` endpoint.

3. CRUD Operations:

◦ GET All Products:

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

Responds with the list of all products.

◦ GET Product by ID:

```
get("{id}")
{
    val id = call.parameters["id"]?.toIntOrNull()
    products.find { it.id == id }?.let { call.respond(it) }
}
```

Mistral Output

Output 1: The provided Kotlin data class `Product` has the following fields and their respective types:

1. `id`: `Int`
2. `name`: `String`
3. `price`: `Double`

Output 2: Based on the fields and purpose of the `Product` data class, here are the REST API endpoints required to perform CRUD operations:

1. Create a new product:

- Endpoint: `[POST] /products` – Create a new product with the provided details.

2. Read all products:

- Endpoint: `[GET] /products` – Retrieve a list of all products.

3. Read a single product by ID:

- Endpoint: `[GET] /products/{id}` – Retrieve the details of a specific product by its ID.

4. Update a product by ID:

- Endpoint: `[PUT] /products/{id}` – Update the details of a specific product by its ID.

5. Delete a product by ID:

- Endpoint: `[DELETE] /products/{id}` – Delete a specific product by its ID.

These endpoints cover the basic CRUD operations for managing products in a RESTful API.

Output 3: Certainly! Below are the Ktor route stubs in Kotlin for the specified endpoints. Each stub includes the HTTP method, path, and a placeholder for the handler logic.

```

import io.ktor.application.*
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

fun main() {
    embeddedServer(Netty, port = 8080) {
        // Create a new product
        post("/products") {
            // Handler logic for creating a new product
            call.respondText("Create a new product",
                ContentType.Text.Plain)
        }

        // Read all products
        get("/products") {
            // Handler logic for retrieving all products
            call.respondText("Retrieve all products",
                ContentType.Text.Plain)
        }

        // Read a single product by ID
        get("/products/{id}") {
            val id = call.parameters["id"]
            // Handler logic for retrieving a product by ID
            call.respondText("Retrieve product with ID $id",
                ContentType.Text.Plain)
        }

        // Update a product by ID
        put("/products/{id}") {
            val id = call.parameters["id"]
            // Handler logic for updating a product by ID
            call.respondText("Update product with ID $id",
                ContentType.Text.Plain)
        }

        // Delete a product by ID
        delete("/products/{id}") {
            val id = call.parameters["id"]
            // Handler logic for deleting a product by ID
            call.respondText("Delete product with ID $id",
                ContentType.Text.Plain)
        }
    }.start(wait = true)
}

```

In this code:

- The `post("/products")` route handles the creation of a new product.
- The `get("/products")` route handles retrieving all products.
- The `get("/products/{id}")` route handles retrieving a single product by its ID.
- The `put("/products/{id}")` route handles updating a product by its ID.
- The `delete("/products/{id}")` route handles deleting a product by its ID.

Each route includes a placeholder for the handler logic, which you can replace with the actual implementation to perform the desired operations.

Output 4: Certainly! Below is the implementation of the handler logic for each route using an in-memory list to store `Product` objects. This example demonstrates how to add, retrieve, update, and delete products in the list.

```

import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.gson.gson
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import kotlinc.serialization.Serializable

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

```

```
? : call.respond(HttpStatusCode.NotFound)
```

Responds with the product matching the given ID or a 404 Not Found status if not found.

o POST New Product:

```
post {
    val product = call.receive<Product>()
    products.add(product)
    call.respond(HttpStatusCode.Created)
```

Adds a new product to the list and responds with a 201 Created status.

o PUT Update Product:

```
put("{id}") {
    val id = call.parameters["id"]?.toIntOrNull()
    val index = products.indexOfFirst { it.id == id }
    if (index != -1) {
        products[index] = call.receive()
        call.respond(HttpStatusCode.OK)
    } else {
        call.respond(HttpStatusCode.NotFound)
    }
}
```

Updates the product with the given ID and responds with a 200 OK status if found, otherwise a 404 Not Found status.

o DELETE Product:

```
delete("{id}") {
    val id = call.parameters["id"]?.toIntOrNull()
    if (products.removeIf { it.id == id }) {
        call.respond(HttpStatusCode.NoContent)
    } else {
        call.respond(HttpStatusCode.NotFound)
    }
}
```

Deletes the product with the given ID and responds with a 204 No Content status if found, otherwise a 404 Not Found status.

This implementation provides a complete REST API for managing `Product` entities with CRUD operations using Ktor.

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

```
var nextId = 1
```

```
fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            gson {
                setPrettyPrinting()
            }
        }
        routing {
            // Create a new product
            post("/products") {
                val product = call.receive<Product>()
                product.id = nextId++
                products.add(product)
                call.respond(HttpStatusCode.Created, product)
            }

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

            // Read a single product by ID
            get("/products/{id}") {
                val id = call.parameters["id"]?.toIntOrNull()
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                    return@get
                }
                val product = products.find { it.id == id }
                if (product == null) {
                    call.respond(HttpStatusCode.NotFound, "Product not found")
                } else {
                    call.respond(product)
                }
            }

            // Update a product by ID
            put("/products/{id}") {
                val id = call.parameters["id"]?.toIntOrNull()
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                    return@put
                }
                val updatedProduct = call.receive<Product>()
                val index = products.indexOfFirst { it.id == id }
                if (index == -1) {
                    call.respond(HttpStatusCode.NotFound, "Product not found")
                } else {
                    products[index] = updatedProduct.copy(id = id)
                    call.respond(HttpStatusCode.OK, updatedProduct)
                }
            }

            // Delete a product by ID
            delete("/products/{id}") {
                val id = call.parameters["id"]?.toIntOrNull()
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                    return@delete
                }
                val index = products.indexOfFirst { it.id == id }
                if (index == -1) {
                    call.respond(HttpStatusCode.NotFound, "Product not found")
                } else {
                    products.removeAt(index)
                    call.respond(HttpStatusCode.OK, "Product deleted")
                }
            }
        }.start(wait = true)
}
```

Explanation:

1. Create a new product:

- The `post("/products")` route receives a `Product` object from the request body, assigns it a unique ID, adds it to the `products` list, and responds with the created product.

2. Read all products:

- The `get("/products")` route responds with the list of all products.

3. Read a single product by ID:

- The `get("/products/{id}")` route retrieves the product with the specified ID from the `products` list and responds with it. If the ID is invalid or the product is not found, it responds with an appropriate status code and message.

4. Update a product by ID:

- The `put("/products/{id}")` route receives an updated `Product` object from the request body, finds the product with the specified ID in the `products` list, updates it, and responds with the updated product. If the ID is invalid or the product is not found, it responds with an appropriate status code and message.

5. Delete a product by ID:

- The `delete("/products/{id}")` route finds the product with the specified ID in the `products` list, removes it, and responds with a success message. If the ID is invalid or the product is not found, it responds with an appropriate status code and message.

This implementation uses an in-memory list to store `Product` objects and demonstrates basic CRUD operations.

Output 5: Certainly! Below is the complete Ktor application file that includes the `Product` data class, in-memory storage, and all route handlers. This file is runnable and includes all necessary imports and setup code.

```
import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.gson.gson
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*

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

val products = mutableListOf<Product>()
var nextId = 1

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            gson {
                setPrettyPrinting()
            }
        }
        routing {
            // Create a new product
            post("/products") {
                val product = call.receive<Product>()
                product.id = nextId++
                products.add(product)
                call.respond(HttpStatusCode.Created, product)
            }

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

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

```

        }

        // Update a product by ID
        put("/products/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            if (id == null) {
                call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                return@put
            }
            val updatedProduct = call.receive<Product>()
            val index = products.indexOfFirst { it.id == id }
            if (index == -1) {
                call.respond(HttpStatusCode.NotFound, "Product not found")
            } else {
                products[index] = updatedProduct.copy(id = id)
                call.respond(HttpStatusCode.OK, updatedProduct)
            }
        }

        // Delete a product by ID
        delete("/products/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            if (id == null) {
                call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                return@delete
            }
            val index = products.indexOfFirst { it.id == id }
            if (index == -1) {
                call.respond(HttpStatusCode.NotFound, "Product not found")
            } else {
                products.removeAt(index)
                call.respond(HttpStatusCode.OK, "Product deleted")
            }
        }
    }.start(wait = true)
}

```

Explanation:

1. Data Class:

- The `Product` data class defines the structure of a product with `id`, `name`, and `price` fields.

2. In-Memory Storage:

- The `products` list is used to store `Product` objects.
- The `nextId` variable is used to generate unique IDs for new products.

3. Route Handlers:

- The `post("/products")` route handles creating a new product.
- The `get("/products")` route handles retrieving all products.
- The `get("/products/{id}")` route handles retrieving a single product by its ID.
- The `put("/products/{id}")` route handles updating a product by its ID.
- The `delete("/products/{id}")` route handles deleting a product by its ID.

4. Ktor Setup:

- The `embededServer` function sets up the Ktor server using the Netty engine.
- The `install(ContentNegotiation)` function configures Gson for JSON serialization and deserialization.
- The `routing` block defines the routes and their handlers.

This complete Ktor application file is runnable and includes all necessary imports and setup code to perform CRUD operations on `Product` objects using an in-memory list.

Analysis

Prompt chaining guided both models through a detailed, multi-step implementation, resulting in complete, runnable applications. GPT-4 used `kotlinx.serialization` and added thread safety (`synchronized`), while Mistral Large used `gson`. Compared to few-shot, chaining built a more complete solution piece-by-piece, allowing for different library choices and features. Few Shot Exact match: 0.3563743071405282. Prompt Chaining Exact match: 0.07240196532863633

Task 10: Function Summarization (Python)

| | | | |
|-------------------|---|-------------------|--|
| Goal | Write a brief summary of the function | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Zero Shot | Prompt Strategy 2 | Chain of Thought |
| Prompt | <p>Write a brief summary of the function. Here's the code:</p> <pre>def reverse_words(sentence): return ', '.join(sentence.split()[::-1])</pre> | Prompt | <p>Write a brief summary of the Python function. Let's think step-by-step. Here's the code:</p> <pre>def reverse_words(sentence): return ', '.join(sentence.split()[::-1])</pre> |
| ChatGPT Output | <p>Certainly! Here's a brief summary of the function:</p> <p>Function Summary:</p> <p>The <code>reverse_words(sentence)</code> function takes a string <code>sentence</code> as input, splits it into a list of words, reverses the order of those words, and then joins them back into a single string separated by commas and spaces. The result is a sentence where the word order is reversed and words are separated by "," instead of spaces.</p> <p>Example:</p> <pre>reverse_words("hello world today") # Output: "today, world, hello"</pre> | ChatGPT Output | <p>Let's break down the function step by step:</p> <p>Function:</p> <pre>def reverse_words(sentence): return ', '.join(sentence.split()[::-1])</pre> <p>Step-by-step explanation:</p> <ol style="list-style-type: none"> 1. The function takes an input string called <code>sentence</code>. 2. It splits the sentence into a list of words using <code>sentence.split()</code>. 3. It reverses the list of words with <code>[::-1]</code>. 4. It joins the reversed words into a new string, separated by commas and spaces (' ',). <p>Summary:</p> <p>The function <code>reverse_words</code> takes a sentence, reverses the order of its words, and returns them as a comma-separated string.</p> |
| Mistral Output | <p>The function <code>reverse_words</code> takes a single argument <code>sentence</code>, which is expected to be a string. It splits the sentence into individual words, reverses the order of these words, and then joins them back together into a single string with each word separated by a comma and a space. The result is a new string where the words from the original sentence are in reverse order, separated by commas.</p> | Mistral Output | <p>Sure&lt;br&gt;</p> <ol style="list-style-type: none"> 1. Function Name: The function is named <code>reverse_words</code>. 2. Parameter: It takes a single parameter called <code>sentence</code>, which is expected to be a string. 3. Purpose: The function is designed to reverse the order of words in the given sentence. 4. Steps: <ul style="list-style-type: none"> o Splitting: The function uses <code>sentence.split()</code> to split the input string into a list of words. o Reversing: The list of words is then reversed using slicing <code>[::-1]</code>. o Joining: The reversed list of words is joined back into a single string with <code>' ', .join(...)</code>, where each word is separated by a comma and a space. 5. Return Value: The function returns the new string with the words in reverse order, separated by commas. <p>Summary: The <code>reverse_words</code> function takes a sentence as input, splits it into words, reverses the order of the words, and returns them as a comma-separated string.</p> |
| Analysis | Prompt chaining guided both models through a detailed, multi-step implementation, resulting in complete, runnable applications. GPT-4 used <code>kotlinx.serialization</code> and added thread safety (<code>synchronized</code>), while Mistral Large used <code>gson</code> . Compared to few-shot, chaining built a more complete solution piece-by-piece, allowing for different library choices and features. Few Shot Exact match: 0.3563743071405282. Prompt Chaining Exact match: 0.07240196532863633 | | |

Task 11: Prompt from Code Comments (Python)

| | | | |
|-------------------|---|-------------------|---|
| Goal | Write a prompt that could generate the code | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Self Consistency | Prompt Strategy 2 | Chain of Thought |
| Prompt | <p>Write a prompt that could generate the code. Here's the code:</p> <pre># This function checks if a number is prime def is_prime(n): if n <= 1: return False for i in range(2, int(n**0.5) + 1): if n % i == 0: return False return True</pre> | Prompt | <p>Write a prompt that could generate the following code. Let's think step-by-step. Here's the code:</p> <pre># This function checks if a number is prime def is_prime(n): if n <= 1: return False for i in range(2, int(n**0.5) + 1): if n % i == 0: return False return True</pre> |
| Analysis | Prompt chaining guided both models through a detailed, multi-step implementation, resulting in complete, runnable applications. GPT-4 used <code>kotlinx.serialization</code> and added thread safety (<code>synchronized</code>), while Mistral Large used <code>gson</code> . Compared to few-shot, chaining built a more complete solution piece-by-piece, allowing for different library choices and features. Few Shot Exact match: 0.3563743071405282. Prompt Chaining Exact match: 0.07240196532863633 | | |

| | | | |
|----------------|---|----------------|---|
| | <pre> return False return True </pre> | | <pre> return True </pre> |
| ChatGPT Output | <p>Certainly! Here is a prompt you could use to generate the provided code:</p> <p>Prompt: Write a Python function named <code>is_prime</code> that checks whether a given integer <code>n</code> is a prime number. The function should return <code>True</code> if <code>n</code> is prime and <code>False</code> otherwise. For efficiency, loop from 2 up to the square root of <code>n</code> (inclusive) to check for divisibility. Also, handle cases where <code>n</code> is less than or equal to 1. Include appropriate comments in the code.</p> | ChatGPT Output | <p>Sure! Let's break down the requirements step by step:</p> <ol style="list-style-type: none"> 1. The code defines a function. 2. The function checks if a given number is prime. 3. The function returns <code>False</code> if the number is less than or equal to 1. 4. The function iterates from 2 up to the square root of the number (inclusive). 5. If the number is divisible by any of these, it returns <code>False</code>. 6. Otherwise, it returns <code>True</code>. <p>Based on this, here is a prompt that would generate the provided code:</p> <p>Prompt:</p> <p>Write a Python function called <code>is_prime</code> that takes an integer <code>n</code> as input and returns <code>True</code> if <code>n</code> is a prime number and <code>False</code> otherwise. The function should return <code>False</code> for numbers less than or equal to 1. To check for primality, iterate from 2 up to and including the integer value of the square root of <code>n</code>. If <code>n</code> is divisible by any of these numbers, return <code>False</code>. Otherwise, return <code>True</code>. Add a comment at the top to describe the function.</p> |
| Mistral Output | <p>Sure! Here's a prompt that could generate the given code:</p> <p>"Write a Python function called <code>is_prime</code> that takes an integer <code>n</code> as input and returns <code>True</code> if <code>n</code> is a prime number and <code>False</code> otherwise. The function should handle edge cases where <code>n</code> is less than or equal to 1 and should efficiently check for primality by only iterating up to the square root of <code>n</code>."</p> | Mistral Output | <p>Sure, let's break down the code step-by-step to understand what it does and then create a prompt that could generate this code.</p> <ol style="list-style-type: none"> 1. Function Definition: The code defines a function named <code>is_prime</code>. 2. Purpose: The function checks if a given number <code>n</code> is a prime number. 3. Initial Check: If <code>n</code> is less than or equal to 1, the function returns <code>False</code> because numbers less than or equal to 1 are not prime. 4. Loop for Divisors: The function then iterates from 2 to the square root of <code>n</code> (inclusive). This is because if <code>n</code> is divisible by any number in this range, it is not a prime number. 5. Divisibility Check: Within the loop, if <code>n</code> is divisible by any number <code>i</code>, the function returns <code>False</code>. 6. Return True: If no divisors are found, the function returns <code>True</code>, indicating that <code>n</code> is a prime number. <p>Given this breakdown, here is a prompt that could generate the code:</p> <p>Prompt:</p> <p>Write a Python function named <code>is_prime</code> that takes an integer <code>n</code> as input and returns <code>True</code> if <code>n</code> is a prime number and <code>False</code> otherwise. The function should first check if <code>n</code> is less than or equal to 1 and return <code>False</code> in that case. Then, it should iterate from 2 to the square root of <code>n</code> (inclusive) and check if <code>n</code> is divisible by any number in this range. If <code>n</code> is divisible by any number, the function should return <code>False</code>. If no divisors are found, the function should return <code>True</code>.</p> <p>This prompt clearly outlines the requirements and steps needed to generate the provided code.</p> |
| Analysis | <p>Self-consistency helped both models refine prompt wording across attempts, resulting in clear prompts reflecting the primality test function's purpose. Compared to chain-of-thought, this method focused on converging towards a good general prompt without necessarily dissecting code specifics like edge cases. Self-Consistency BLEU score: 0.2322330623625265. Chain Of Thought BLEU score: 0.18229114359464482</p> | | |

Task 12: Fixing Factorial Bug (Python)

| | | | |
|-------------------|--|-------------------|---|
| Goal | Fix the bug when input is 0 | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Prompt Chaining | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Prompt 1: Identify the bug in this Python function. Here's the code:</p> <pre> def factorial(n): result = 1 for i in range(1, n): result *= i return result </pre> <p>Prompt 2: Now fix the bug when the input is 0.</p> | Prompt | <p>Fix the bug when input is 0. Here's the code:</p> <pre> def factorial(n): result = 1 for i in range(1, n): result *= i return result </pre> |
| ChatGPT Output | <p>Output 1: Let's look at your code:</p> <pre> def factorial(n): result = 1 for i in range(1, n): result *= i return result </pre> <p>Bug explanation:</p> <ul style="list-style-type: none"> • <code>range(1, n)</code> iterates from 1 up to <code>n-1</code>, but not including <code>n</code>. • The factorial of <code>n</code> is $n * (n-1) * \dots * 1$, so your loop misses multiplying by <code>n</code>. <p>Example:</p> <p>For <code>n=5</code>:</p> <ul style="list-style-type: none"> • Correct: $1 \times 2 \times 3 \times 4 \times 5 = 120$ • Your code does $1 \times 2 \times 3 \times 4 = 24$ <p>Correct Code:</p> <p>Change <code>range(1, n)</code> to <code>range(1, n+1)</code> so it includes <code>n</code>:</p> <pre> def factorial(n): result = 1 for i in range(1, n+1): result *= i return result </pre> <p>Now <code>factorial(5)</code> returns 120 as expected!</p> <p>Output 2: Great follow-up! In mathematics, 0! (zero factorial) is defined as 1.</p> <p>With the previous fix:</p> <pre> def factorial(n): result = 1 for i in range(1, n+1): result *= i return result </pre> <p>If you call <code>factorial(0)</code>, <code>range(1, 1)</code> produces an empty range, and <code>result</code> remains 1, so it returns the correct answer (1). No further change is needed for handling 0!</p> <p>However, you may want robustness for negative inputs, since factorial is not defined for negative numbers. Optionally, you could add:</p> <pre> def factorial(n): if n < 0: raise ValueError("Factorial is not defined for negative numbers.") result = 1 for i in range(1, n+1): result *= i return result </pre> <p>Summary:</p> <ul style="list-style-type: none"> • For <code>n = 0</code>, your code with <code>range(1, n+1)</code> works correctly and returns 1. • For negative numbers, you may want to raise an exception. <p>Let me know if you'd like help handling negative or non-integer inputs!</p> | ChatGPT Output | <p>Certainly! Your current code is:</p> <pre> def factorial(n): result = 1 for i in range(1, n): result *= i return result </pre> <p>Issue:</p> <ul style="list-style-type: none"> • The <code>range(1, n)</code> loop will not include <code>n</code> itself (it goes from 1 to <code>n-1</code>). • When <code>n = 0</code>, the loop doesn't execute, so <code>result</code> stays 1, which is correct ($0! = 1$). • However, for <code>n > 0</code>, you're missing multiplying by <code>n</code> itself (e.g., for <code>n = 5</code>, loop does $1 * 2 * 3 * 4$, missing 5). <p>You should use: <code>range(1, n + 1)</code></p> <p>Here is the corrected code:</p> <pre> def factorial(n): result = 1 for i in range(1, n + 1): result *= i return result </pre> <p>Now:</p> <ul style="list-style-type: none"> • <code>factorial(0)</code> returns 1 • <code>factorial(5)</code> returns 120 <p>Let me know if you need a recursive solution or further explanation!</p> |
| Mistral Output | <p>Output 1: The bug in the provided Python function is in the <code>range</code> used in the <code>for</code> loop. The loop should iterate from 1 to <code>n</code> inclusive, but the current <code>range(1, n)</code> only goes up to <code>n-1</code>. This means the function will not multiply by <code>n</code> itself, resulting in an incorrect factorial calculation.</p> <p>Here is the corrected version of the function:</p> | | |

```

def factorial(n):
    result = 1
    for i in range(1, n + 1): # Change range to include n
        result *= i
    return result

```

With this change, the loop will correctly iterate from 1 to n, ensuring that the factorial calculation is accurate.

Output 2: The factorial of 0 is defined as 1. The current implementation does not handle the case when n is 0 correctly because the loop will not execute, and the initial value of result (which is 1) will be returned as is. This is actually the correct behavior for `factorial(0)`, so no additional changes are needed for this specific case.

However, to make the function more robust and explicitly handle the case when n is 0, you can add a condition to return 1 immediately if n is 0. Here is the updated function:

```

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

```

This ensures that the function explicitly handles the case when n is 0 and returns the correct value of 1.

```

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

```

This code now correctly handles the case when n is 0 by returning 1 immediately. For other values of n, it calculates the factorial as before.

Analysis

The two-step prompt chaining effectively isolated bug identification (range issue for input 0) and fixing (adding base case) for both models. Mistral Large added validation tests. Compared to self-consistency, this structured approach separated diagnosis and solution clearly. Prompt Chaining Exact match: 0.19262161279791054. Self-Consistency Exact match: 0.16246498599439776

Task 13: Linked List Node Deletion (C)

| | | | |
|-------------------|---|---|------------------|
| Goal | Implement node deletion by value | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Chain of Thought | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Implement node deletion by value. Let's think step-by-step. Here's the code:</p> <pre> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // TODO: Implement node deletion } </pre> | <p>Implement node deletion by value. Here's the code:</p> <pre> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // TODO: Implement node deletion } </pre> | |
| ChatGPT Output | <p>Let's walk through step-by-step how to delete a node by value from a singly linked list in C.</p> <p>1. Understand the Problem
Given a pointer to the head of a singly linked list and an integer <code>key</code>, delete the first node whose data is equal to <code>key</code>.</p> <p>2. Edge Cases</p> <ul style="list-style-type: none"> The list is empty (<code>*head == NULL</code>). The node to be deleted is the head. The node to be deleted is not the head. The value is not present in the list. <p>3. Steps to Delete</p> <ol style="list-style-type: none"> Check if head node holds the key. Otherwise, traverse the list keeping track of the previous node. Unlink the node containing the key and free its memory. <p>Implementation</p> <pre> #include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // Store head node struct Node* temp = *head; struct Node* prev = NULL; // If head node itself holds the key 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 previous node while (temp != NULL && temp->data != key) { prev = temp; temp = temp->next; } // If key was not present in linked list if (temp == NULL) return; // Unlink the node from linked list prev->next = temp->next; free(temp); // Free memory } </pre> <p>Explanation</p> <ul style="list-style-type: none"> We use two pointers: <code>temp</code> (current node) and <code>prev</code> (previous node). If the head node contains the key, we update the head. Otherwise, we traverse until we find the key or reach the end. Once found, <code>prev->next</code> skips the node to be deleted, and we free its memory. <p>Let me know if you want a full demo with insertion and printing functions as well!</p> | <p>Certainly! Here is how you can implement node deletion by value in a singly linked list:</p> <pre> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { struct Node* temp = *head; struct Node* prev = NULL; // If head node itself holds the key if (temp != NULL && temp->data == key) { *head = temp->next; // Changed head free(temp); // free old head return; } // Search for the key to be deleted 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; free(temp); // Free memory } </pre> <p>How it works:</p> <ul style="list-style-type: none"> If the node to be deleted is the head node, update the head pointer. Otherwise, search for the node and keep track of the previous node. If found, unlink and free the memory. If not found, do nothing. <p>Usage example:</p> <pre> struct Node* head = NULL; // ... (code to create and populate the list) deleteNode(&head, 42); // Deletes node with value 42 </pre> <p>Let me know if you need a full example or explanation!</p> | |
| Mistral Output | <p>To To To To To implement node deletion by value in a singly linked list, we need to follow these steps:</p> <ol style="list-style-type: none"> Check if the list is empty: If the head is <code>NULL</code>, there's nothing to delete. Check if the node to be deleted is the head: If the head node's data matches the key, we need to update the head to the next node. Traverse the list to find the node to be deleted: If the node to be deleted is not the head, we need to find the node and its previous node. Update the pointers: Once we find the node to be deleted, we need to update the <code>next</code> pointer of the previous node to skip the node to be deleted. Free the memory: Finally, free the memory allocated to the node to be deleted. <p>Here's the implementation of the <code>deleteNode</code> function:</p> <pre> #include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // Step 1: Check if the list is empty if (*head == NULL) { return; } </pre> | <p>Mistral Output</p> <p>To implement node deletion by value in a singly linked list, you need to traverse the list and find the node with the specified value (key). Once you find the node, you need to adjust the pointers to remove it from the list. Here is the complete implementation of the <code>deleteNode</code> function:</p> <pre> #include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // Store head node struct Node* temp = *head; struct Node* prev = NULL; // If head node itself holds the key to be deleted if (temp != NULL && temp->data == key) { *head = temp->next; // Changed head free(temp); // Free old head return; } // Search for the key to be deleted, keep track of the previous node while (temp != NULL && temp->data != key) { prev = temp; temp = temp->next; } // If key was not present in the list if (temp == NULL) return; // Unlink the node from linked list prev->next = temp->next; free(temp); // Free memory } </pre> | |

```

// Step 2: Check if the node to be deleted is the head
if ((*head)->data == key) {
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    return;
}

// Step 3: Traverse the list to find the node to be deleted
struct Node* current = *head;
struct Node* prev = NULL;

while (current != NULL && current->data != key) {
    prev = current;
    current = current->next;
}

// Step 4: If the node is found, update the pointers
if (current != NULL) {
    prev->next = current->next;
    free(current);
}

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

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

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

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

    deleteNode(&head, 3);

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

    deleteNode(&head, 1);

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

    deleteNode(&head, 4);

    printf("List after deleting 4: ");
    printList(head);

    return 0;
}

```

Explanation:

1. Check if the list is empty:

```

if (*head == NULL) {
    return;
}

```

2. Check if the node to be deleted is the head:

```

if ((*head)->data == key) {
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    return;
}

```

3. Traverse the list to find the node to be deleted:

```

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

while (current != NULL && current->data != key) {
    prev = current;
    current = current->next;
}

```

4. Update the pointers if the node is found:

```

if (current != NULL) {
    prev->next = current->next;
    free(current);
}

```

5. Helper functions:

- `printList`: To print the linked list.
- `newNode`: To create a new node.

This code will correctly delete a node by value from a singly linked list.

```

temp = temp->next;
}

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

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

free(temp); // Free memory
}

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

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

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

    printf("Created Linked list is: ");
    printList(head);

    deleteNode(&head, 3);

    printf("Linked list after deletion of 3: ");
    printList(head);

    return 0;
}

```

Explanation:

1. **Initial Check:** The function first checks if the head node itself contains the key to be deleted. If so, it updates the head to the next node and frees the old head node.
2. **Traversal:** The function then traverses the list to find the node with the specified key. It keeps track of the previous node (`prev`) to adjust the pointers correctly.
3. **Deletion:** Once the node with the key is found, it updates the `next` pointer of the previous node to skip the current node, effectively removing it from the list.
4. **Memory Management:** The function frees the memory allocated to the node being deleted.

Utility Functions:

- `newNode`: Creates a new node with the given data.
- `printList`: Prints the linked list.

Example Usage:

The `main` function demonstrates creating a linked list, deleting a node with a specific value, and printing the list before and after deletion.

Analysis

Self-consistency produced correct singly-linked list deletion code from both models. Mistral Large's attempts showed more variability, sometimes exploring doubly-linked list concepts. Compared to chain-of-thought, self-consistency achieved correctness but with broader (and sometimes less relevant) exploration by Mistral Large. Chain Of Thought Exact match: 0.06944971537001897. Self-Consistency Exact match: 0.3940594059405941

Task 14: Recursive Function Completion (Python)

| | | | |
|-------------------|--|-------------------|---|
| Goal | Complete the recursive function for Fibonacci | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Chain of Thought | Prompt Strategy 2 | Self Consistency |
| Prompt | Complete the recursive function for Fibonacci. Let's think step-by-step. Here's the code:

<pre>def fibonacci(n): # TODO: Base cases and recursive call pass</pre> | Prompt | Complete the recursive function for Fibonacci. Here's the code:

<pre>def fibonacci(n): # TODO: Base cases and recursive call pass</pre> |
| ChatGPT Output | Absolutely! Let's walk through it step-by-step.

1. Base cases:
The Fibonacci sequence is defined as: <ul style="list-style-type: none"> • <code>fibonacci(0) = 0</code> • <code>fibonacci(1) = 1</code> 2. Recursive case:
For <code>n > 1</code> : <ul style="list-style-type: none"> • <code>fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)</code>
Putting it together:

<pre>def fibonacci(n): # Base cases</pre> | ChatGPT Output | Sure! Here's how you can complete the <code>fibonacci</code> function recursively:

<pre>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)</pre> |

```

if n == 0:
    return 0
elif n == 1:
    return 1
# Recursive case
else:
    return fibonacci(n-1) + fibonacci(n-2)

```

You can test it, for example:

```

print(fibonacci(5)) # Output: 5
print(fibonacci(8)) # Output: 21

```

Let me know if you'd like an explanation or an optimized version!

Sure, let's complete the recursive function for calculating the Fibonacci sequence step-by-step.

The Fibonacci sequence is defined as follows:

- ($F(0) = 0$)
- ($F(1) = 1$)
- ($F(n) = F(n-1) + F(n-2)$) for ($n > 1$)

Given this definition, we need to handle the base cases for ($n = 0$) and ($n = 1$), and then implement the recursive case for ($n > 1$).

Here's the complete function:

```

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

```

Let's break down the function:

1. Base cases:

- If (n) is 0, return 0.
- If (n) is 1, return 1.

2. Recursive case:

- For any (n) greater than 1, return the sum of the Fibonacci numbers for ($n-1$) and ($n-2$).

This function will correctly compute the (n)-th Fibonacci number using recursion.

- Otherwise, return the sum of $\text{fibonacci}(n-1)$ and $\text{fibonacci}(n-2)$.

Mistral Output

Mistral Output

Sure! 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 recursive definition of the Fibonacci sequence is:

- ($F(0) = 0$)
- ($F(1) = 1$)
- ($F(n) = F(n-1) + F(n-2)$) for ($n > 1$)

Here is the complete recursive function for Fibonacci:

```

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)) # Output: 55

```

This function will correctly compute the (n)-th Fibonacci number using recursion.

Analysis

Chain-of-thought helped both models correctly identify Fibonacci base cases and recursion. Mistral Large proactively included memoization, demonstrating deeper optimization insight. Compared to self-consistency, CoT's step-by-step analysis prompted optimization beyond the basic recursive structure. Chain Of Thought Exact match: 0.2795425667090216. Self-Consistency Exact match: 0.4216417910447761

Task 15: Constructor Completion (Python)

| | | | |
|-------------------|---|-------------------|---|
| Goal | Complete the class constructor | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Zero Shot | Prompt Strategy 2 | Few Shot |
| Prompt | Complete the class constructor to initialize the name, age, and optional email attributes. Here's the code: | Prompt | <p>Example 1:
Input:</p> <pre> class Dog: def __init__(self): # TODO: Add breed and age pass </pre> <p>Output:</p> <pre> class Dog: def __init__(self, breed, age): self.breed = breed self.age = age </pre> <p>Example 2:
Input:</p> <pre> class Car: def __init__(self): # TODO: Add make and year pass </pre> <p>Output:</p> <pre> class Car: def __init__(self, make, year): self.make = make self.year = year </pre> <p>Now, complete the class constructor to initialize the name, age, and optional email attributes.
Here's the code:</p> <pre> class Person: def __init__(self): # TODO: Add name, age, and optional email pass </pre> |
| ChatGPT Output | Certainly! Here's how you can complete the constructor to initialize name, age, and an optional email attribute: | ChatGPT Output | <p>Sure! Here's how you would complete the class constructor to initialize name, age, and an optional email attribute:</p> <pre> class Person: def __init__(self, name, age, email=None): self.name = name self.age = age self.email = email </pre> <p>This way, email is optional and defaults to None if not provided.</p> |
| Mistral Output | To complete the class constructor to initialize the name, age, and optional email attributes, you need to define the <code>__init__</code> method to accept these parameters and assign them to instance variables. Here's how you can do it: | Mistral Output | <p>Certainly! Below is the completed class constructor for the Person class that initializes the name, age, and an optional email attribute:</p> <pre> class Person: def __init__(self, name, age, email=None): self.name = name self.age = age self.email = email </pre> <p>In this code:</p> <ul style="list-style-type: none"> • The <code>__init__</code> method takes name and age as required parameters. • The email parameter is optional and defaults to None if not provided. • The attributes name, age, and email are then initialized with the provided values. |

```
print(person2.name) # Output: Bob
print(person2.age) # Output: 25
print(person2.email) # Output: bob@example.com
```

In this code:

- The `__init__` method takes `name` and `age` as required parameters and `email` as an optional parameter with a default value of `None`.
- The parameters are assigned to instance variables `self.name`, `self.age`, and `self.email`.

This allows you to create instances of the `Person` class with or without the `email` attribute.

Analysis

With zero-shot, GPT-4 created a minimal valid constructor. Mistral Large inferred type validation and default arguments from "optional email", showing more proactive feature inclusion. Compared to few-shot, zero-shot relied more on model inference, leading to differences in robustness between GPT-4 and Mistral Large. Zero Shot Exact match: 0.30707610146862485. Few Shot Exact match: 0.49551569506726456

Task 16: Binary Search Completion (Java)

| | | | |
|-------------------|--|-----------------------|---|
| Goal | Complete the binary search implementation | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Chain of Thought | Prompt Strategy 2 | Zero Shot |
| Prompt | <p>Complete the binary search implementation. Let's think step-by-step. Here's the code:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; // TODO: Compare and adjust bounds } return -1; }</pre> | <p>Prompt</p> | <p>Complete the binary search implementation. Here's the code:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; // TODO: Compare and adjust bounds } return -1; }</pre> |
| ChatGPT Output | <p>Absolutely! Let's go step by step to complete the binary search.</p> <p>Step 1: We already have
 <code>int left = 0, right = arr.length - 1;</code>
 This sets up our search range.</p> <p>Step 2: The loop
 <code>while (left <= right) { ... }</code>
 This continues as long as there is a valid range.</p> <p>Step 3: The midpoint
 <code>int mid = (left + right) / 2;</code>
 This finds the midpoint.</p> <p>Step 4: Compare and adjust</p> <ul style="list-style-type: none"> If <code>arr[mid] == target</code>, we've found the target, so return <code>mid</code>. If <code>arr[mid] < target</code>, discard the left part by setting <code>left = mid + 1</code>. If <code>arr[mid] > target</code>, discard the right part by setting <code>right = mid - 1</code>. <p>Step 5: If we exit the loop, the target was not found. Return <code>-1</code>.</p> <p>Here is your completed function:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; if (arr[mid] == target) { return mid; } else if (arr[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return -1; }</pre> <p>Note:
 To avoid integer overflow, you might use
 <code>int mid = left + (right - left) / 2;</code>
 instead of <code>(left + right) / 2</code> in production code.</p> | <p>ChatGPT Output</p> | <p>Certainly! Here is the completed <code>binarySearch</code> method with the necessary comparison and bound adjustment logic:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = left + (right - left) / 2; // Avoid potential overflow if (arr[mid] == target) { return mid; } else if (arr[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return -1; // Not found }</pre> <p>Explanation of the missing part:</p> <pre>if (arr[mid] == target) { return mid; } else if (arr[mid] < target) { left = mid + 1; } else { right = mid - 1; }</pre> <ul style="list-style-type: none"> If the middle element is the target, return its index. If the middle element is less than the target, search the right half (<code>left = mid + 1</code>). If the middle element is greater, search the left half (<code>right = mid - 1</code>). <p>Let me know if you'd like a recursive version or further explanation!</p> |
| Mistral Output | <p>Sure, let's complete the binary search implementation step-by-step.</p> <ol style="list-style-type: none"> Initialization: We already have the <code>left</code> and <code>right</code> pointers initialized to the start and end of the array, respectively. Loop Condition: The <code>while</code> loop continues as long as <code>left</code> is less than or equal to <code>right</code>. Mid Calculation: We calculate the <code>mid</code> index as the average of <code>left</code> and <code>right</code>. Comparison: We need to compare the middle element with the target. <ul style="list-style-type: none"> If the middle element is equal to the target, we return the <code>mid</code> index. If the middle element is greater than the target, we adjust the <code>right</code> pointer to <code>mid - 1</code>. If the middle element is less than the target, we adjust the <code>left</code> pointer to <code>mid + 1</code>. Return Statement: If the target is not found, we return <code>-1</code>. <p>Here's the complete implementation:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0; int right = arr.length - 1; while (left <= right) { int mid = left + (right - left) / 2; // Avoid potential overflow // Check if the middle element is the target if (arr[mid] == target) { return mid; } // If target is greater, ignore the left half else if (arr[mid] < target) { left = mid + 1; } // If target is smaller, ignore the right half else { right = mid - 1; } } // Target is not present in the array return -1; }</pre> <p>Explanation:</p> <ul style="list-style-type: none"> Overflow Prevention: The mid calculation <code>left + (right - left) / 2</code> is used to prevent potential overflow issues that can occur with <code>(left + right) / 2</code> when <code>left</code> and <code>right</code> are large. Comparison and Adjustment: <ul style="list-style-type: none"> If <code>arr[mid] == target</code>, we have found the target, so we return <code>mid</code>. If <code>arr[mid] < target</code>, it means the target must be in the right half, so we adjust <code>left</code> to <code>mid + 1</code>. If <code>arr[mid] > target</code>, it means the target must be in the left half, so we adjust <code>right</code> to <code>mid - 1</code>. Return -1: If the loop exits without finding the target, we return <code>-1</code> to indicate that the target is not present in the array. <p>This implementation ensures that the binary search algorithm runs efficiently with a time complexity of $O(\log n)$.</p> | <p>Mistral Output</p> | <p>Certainly! Here is the completed <code>binarySearch</code> method with the necessary comparison and bound adjustment logic:</p> <pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = left + (right - left) / 2; // Avoid potential overflow if (arr[mid] == target) { return mid; } else if (arr[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return -1; // Not found }</pre> <p>Explanation of the missing part:</p> <pre>if (arr[mid] == target) { return mid; } else if (arr[mid] < target) { left = mid + 1; } else { right = mid - 1; }</pre> <ul style="list-style-type: none"> If the middle element is the target, return its index. If the middle element is less than the target, search the right half (<code>left = mid + 1</code>). If the middle element is greater, search the left half (<code>right = mid - 1</code>). <p>Let me know if you'd like a recursive version or further explanation!</p> |
| Analysis | Zero-shot yielded correct standard binary search implementations. Mistral Large showed a tendency towards more completeness by potentially including edge case handling (empty arrays). Compared to chain-of-thought, zero-shot was sufficient for the core algorithm but less likely to prompt explicit handling of nuances like integer overflow. Chain Of Thought Exact match: 0.11701526286037309. Zero Shot Exact match: 0.28064903846153844 | | |

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

| Goal | Resolve inconsistency between function name and logic | | | | | | | | | | |
|-------------------|---|--|------------------|---------------|----------------|---------------------|---------------------------------|---------------|--------------------|---------------------------------|---|
| Temperature | 0.7 | Token Limit | 2048 | | | | | | | | |
| Prompt Strategy 1 | Self Consistency | Prompt Strategy 2 | Chain of Thought | | | | | | | | |
| Prompt | <p>Resolve inconsistency between function name and logic. Here's the code:</p> <pre>// Supposed to return true if x is even bool isOdd(int x) { return x % 2 == 0; }</pre> <p>Output 1: You've identified a naming inconsistency: the function is named <code>isOdd</code>, but its logic checks if <code>x</code> is even (<code>x % 2 == 0</code>). To resolve this, you have two options:</p> <p>Option 1: Fix the function name
If the logic is correct (returns true for even numbers), rename the function to reflect that:</p> <pre>// Returns true if x is even bool isEven(int x) { return x % 2 == 0; }</pre> <p>Option 2: Fix the logic
If the function name should remain <code>isOdd</code>, change the logic to return true for odd numbers:</p> <pre>// Returns true if x is odd bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Summary:</p> <ul style="list-style-type: none"> If you want to check for even numbers, rename to <code>isEven</code>. If you want to check for odd numbers, change the return condition to <code>x % 2 != 0</code>. <p>Output 2: You are correct: the function name and its logic are inconsistent.</p> <ul style="list-style-type: none"> The function is named <code>isOdd</code>, which suggests it should return <code>true</code> when <code>x</code> is odd. The logic currently returns <code>true</code> when <code>x % 2 == 0</code>, which is when <code>x</code> is even. <p>To resolve this inconsistency, you have two options:</p> <p>Option 1: Fix the Function Name (if you want to check for even)
If you want the function to check for even numbers, rename it:</p> <pre>bool isEven(int x) { return x % 2 == 0; }</pre> <p>Option 2: Fix the Logic (if you want to check for odd)
If you want the function to check for odd numbers, fix the logic:</p> <pre>bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Choose one option based on what your code is intended to do.</p> <p>Summary Table:</p> <table border="1"> <thead> <tr> <th>What you want</th> <th>Function Name</th> <th>Function Body</th> </tr> </thead> <tbody> <tr> <td>Check for even</td> <td><code>isEven</code></td> <td><code>return x % 2 == 0;</code></td> </tr> <tr> <td>Check for odd</td> <td><code>isOdd</code></td> <td><code>return x % 2 != 0;</code></td> </tr> </tbody> </table> <p>Output 3: Certainly! The inconsistency is that the function is named <code>isOdd</code>, but the logic checks if <code>x</code> is even. Here are two ways to resolve this, depending on your intent:</p> <p>Option 1: Fix the function name (if you want to check for even numbers):</p> <pre>bool isEven(int x) { return x % 2 == 0; }</pre> <p>Option 2: Fix the logic (if you want to check for odd numbers):</p> <pre>bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Choose the one that matches your intended functionality.</p> <ul style="list-style-type: none"> If you want to check for even: use Option 1. If you want to check for odd: use Option 2. | What you want | Function Name | Function Body | Check for even | <code>isEven</code> | <code>return x % 2 == 0;</code> | Check for odd | <code>isOdd</code> | <code>return x % 2 != 0;</code> | <p>Prompt</p> <p>Resolve the inconsistency between function name and logic. Let's think step by step. Here's the code:</p> <pre>// Supposed to return true if x is even bool isOdd(int x) { return x % 2 == 0; }</pre> <p>Let's analyze your code step by step:</p> <p>You have a function named <code>isOdd</code>:</p> <pre>bool isOdd(int x) { return x % 2 == 0; }</pre> <ul style="list-style-type: none"> The function name suggests it should return <code>true</code> when <code>x</code> is odd. The logic (<code>x % 2 == 0</code>) returns <code>true</code> when <code>x</code> is even. <p>This is an inconsistency: the function name and logic do not match.</p> <p>How to resolve?</p> <p>You have two options:</p> <p>Option 1: Fix the function name
If the logic is correct and you want to check for even numbers, rename the function:</p> <pre>bool isEven(int x) { return x % 2 == 0; }</pre> <p>Option 2: Fix the logic
If the function should check for odd numbers (as the name suggests), fix the logic:</p> <pre>bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Summary:</p> <ul style="list-style-type: none"> If you want to check for even numbers, use <code>isEven</code> with <code>x % 2 == 0</code>. If you want to check for odd numbers, use <code>isOdd</code> with <code>x % 2 != 0</code>. <p>Choose one to resolve the inconsistency.</p> |
| What you want | Function Name | Function Body | | | | | | | | | |
| Check for even | <code>isEven</code> | <code>return x % 2 == 0;</code> | | | | | | | | | |
| Check for odd | <code>isOdd</code> | <code>return x % 2 != 0;</code> | | | | | | | | | |
| ChatGPT Output | <p>Output 1: The inconsistency in your code arises because the function name <code>isOdd</code> suggests that the function should return <code>true</code> if <code>x</code> is odd, but the logic inside the function checks if <code>x</code> is even (i.e., <code>x % 2 == 0</code>). To resolve this inconsistency, you have two options:</p> <ol style="list-style-type: none"> 1. Rename the function to <code>isEven</code> and keep the logic the same. 2. Keep the function name <code>isOdd</code> and change the logic to check if <code>x</code> is odd. <p>Here are both options:</p> <p>Option 1: Rename the function to <code>isEven</code></p> <pre>// Returns true if x is even bool isEven(int x) { return x % 2 == 0; }</pre> <p>Option 2: Change the logic to check if <code>x</code> is odd</p> <pre>// Returns true if x is odd bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Choose the option that best fits your needs. If the function is intended to check for even numbers, go with Option 1. If it's intended to check for odd numbers, go with Option 2.</p> <p>Output 2: The inconsistency in your code is that the function name <code>isOdd</code> suggests it should check if a number is odd, but the logic inside the function checks if the number is even. To resolve this inconsistency, you can either:</p> <ol style="list-style-type: none"> 1. Change the function name to <code>isEven</code> to match the logic. 2. Change the logic to check if the number is odd to match the function name. <p>Here are both solutions:</p> <p>Solution 1: Change the Function Name to <code>isEven</code></p> | <p>Mistral Output</p> <p>Sure, let's break down the steps to resolve the inconsistency between the function name and its logic.</p> <ol style="list-style-type: none"> Identify the Issue: <ul style="list-style-type: none"> The function name is <code>isOdd</code>, but the logic inside the function checks if <code>x</code> is even by using <code>x % 2 == 0</code>. Understand the Logic: <ul style="list-style-type: none"> The expression <code>x % 2 == 0</code> checks if <code>x</code> is divisible by 2, which means <code>x</code> is even. Align the Function Name with Logic: <ul style="list-style-type: none"> Since the function name <code>isOdd</code> suggests that it should check if <code>x</code> is odd, we need to update the logic to check for odd numbers. Modify the Logic: <ul style="list-style-type: none"> To check if <code>x</code> is odd, we should use <code>x % 2 != 0</code>. <p>Here is the corrected code:</p> <pre>// Supposed to return true if x is odd bool isOdd(int x) { return x % 2 != 0; }</pre> <p>Alternatively, if the intention is to check for even numbers, the function name should be changed to <code>isEven</code>:</p> <pre>// Supposed to return true if x is even bool isEven(int x) { return x % 2 == 0; }</pre> <p>By making these adjustments, the function name and logic will be consistent.</p> | | | | | | | | | |

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

Solution 2: Change the Logic to Check if the Number is Odd

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

Choose the solution that best fits your needs. If the function is intended to check for even numbers, use Solution 1. If it is intended to check for odd numbers, use Solution 2.

Output 3: The inconsistency in the code is that the function name `isOdd` suggests it should return `true` if `x` is odd, but the logic inside the function checks if `x` is even.

To resolve this inconsistency, you can either:

1. Change the function name to `isEven` to match the current logic.
2. Change the logic to return `true` if `x` is odd.

Here are both solutions:

Solution 1: Change the function name to `isEven`

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

Solution 2: Change the logic to return `true` if `x` is odd

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

Choose the solution that best fits the intended use of the function.

Analysis

Chain-of-thought led both models down the same path: analyzing the mismatch and fixing the logic to match the name. Compared to self-consistency, the reasoning process focused the fix on implementation details rather than exploring alternatives like renaming, offering a more directed but narrower solution path. Self-Consistency Exact match: 0.1394299774451507. Chain Of Thought Exact match: 0.06683291770573566

Task 18: Prompt Chaining: Bug Identification -> Fix (JavaScript)

| | | | |
|-------------------|--|-------------------|--|
| Goal | Identify and fix the bug | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Prompt Chaining | Prompt Strategy 2 | Chain of Thought |
| Prompt | <p>Prompt 1: Identify the bug. Here's the code:</p> <pre>function isEven(n) { return n % 2; }</pre> <p>Prompt 2: Fix the bug identified.</p> | Prompt | <p>Identify and fix the bug in the function. Let's think step by step. Here's the code:</p> <pre>function isEven(n) { return n % 2; }</pre> |
| ChatGPT Output | <p>Output 1: Let's break down your code:</p> <pre>return n % 2 === 0; function isEven(n) { return n % 2; }</pre> <p>There are several issues here:</p> <p>1. Syntax Error</p> <ul style="list-style-type: none"> • <code>return n % 2 === 0;</code>, <code>function isEven(n) { return n % 2; }</code> is not valid JavaScript syntax. • You cannot chain <code>.function</code> after an expression. • The function declaration is in the wrong place. <p>2. Incorrect Function Logic</p> <ul style="list-style-type: none"> • Your function <code>isEven(n) { return n % 2; }</code> returns <code>0</code> for even numbers and <code>1</code> for odd numbers, but in JavaScript, <code>0</code> is falsy and <code>1</code> is truthy. So, <code>isEven(2)</code> returns <code>0</code> (falsy), which is not intuitive. <p>3. Correct Way</p> <p>If you want a function that returns <code>true</code> for even numbers and <code>false</code> for odd numbers, you should write:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> <p>Summary</p> <p>Bug:</p> <ul style="list-style-type: none"> • Invalid syntax. • The function does not return a boolean indicating evenness. <p>Corrected code:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> <p>Usage:</p> <pre>isEven(2); // true isEven(3); // false</pre> <p>Let me know if you need more explanation!</p> <p>Output 2: Certainly! Here is the corrected code:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> <p>Usage example:</p> <pre>console.log(isEven(4)); // true console.log(isEven(5)); // false</pre> <p>This function now correctly returns <code>true</code> for even numbers and <code>false</code> for odd numbers.</p> | ChatGPT Output | <p>Let's analyze the function step by step:</p> <pre>function isEven(n) { return n % 2; }</pre> <p>What does this code do?</p> <ul style="list-style-type: none"> • It returns <code>n % 2</code> (the remainder when <code>n</code> is divided by 2). <p>What is the bug?</p> <ul style="list-style-type: none"> • For an even number, <code>n % 2</code> is <code>0</code>. • For an odd number, <code>n % 2</code> is <code>1</code>. <p>So, if you call <code>isEven(4)</code>, it returns <code>0</code>. If you call <code>isEven(5)</code>, it returns <code>1</code>.</p> <p>But:</p> <p>A function named <code>isEven</code> should ideally return a boolean (<code>true</code> or <code>false</code>), not <code>0</code> or <code>1</code>. Also, the return value is the opposite of what "isEven" suggests: the function returns <code>0</code> (which is falsy in JavaScript) for even numbers, and <code>1</code> (which is truthy) for odd numbers.</p> <p>Correct implementation</p> <p>We want <code>isEven(n)</code> to return <code>true</code> if <code>n</code> is even, <code>false</code> otherwise.</p> <p>Fixed code:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> <p>Summary</p> <ul style="list-style-type: none"> • Bug: The function returns <code>n % 2</code>, which gives <code>0</code> for even numbers (falsy) and <code>1</code> for odd numbers (truthy) — the opposite of what you want. • Fix: Return the boolean expression: <code>n % 2 === 0</code>. This returns <code>true</code> for even numbers, <code>false</code> for odd. <p>Final code:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> |
| Mistral Output | <p>"Output 1: The code you've provided has a couple of issues:</p> <ol style="list-style-type: none"> 1. The syntax for defining the function <code>isEven</code> is incorrect. 2. The expression <code>n % 2 === 0</code> should be within the function body if you want to return whether a number is even. <p>Here is the corrected version of the code:</p> <pre>function isEven(n) { return n % 2 === 0; }</pre> <p>This function <code>isEven</code> takes a number <code>n</code> as an argument and returns <code>true</code> if <code>n</code> is even, and <code>false</code> otherwise.</p> | Mistral Output | <p>Certain tenth письмени of Нписицненпитеиленнен1Connect . mathtd</p> <p>The function <code>isEven</code> is supposed to determine if a number is even. However, the current implementation is incorrect because it returns the remainder of <code>n</code> divided by 2 instead of a boolean value indicating whether <code>n</code> is even or not.</p> <p>Let's break down the steps to fix the bug:</p> <ol style="list-style-type: none"> 1. Understand the Requirement: The function should return <code>true</code> if <code>n</code> is even and <code>false</code> if <code>n</code> is odd. 2. Current Implementation: The current implementation returns <code>0</code> if <code>n</code> is even and <code>1</code> if <code>n</code> is odd, which is not what we want. 3. Fix the Implementation: We need to compare the remainder of <code>n</code> divided by 2 with <code>0</code>. If the remainder is <code>0</code>, then <code>n</code> is even; otherwise, <code>n</code> is odd. <p>Here's the corrected code:</p> |

Explanation:

- `function isEven(n) { ... };`: This is the correct syntax for defining a function named `isEven` that takes a single argument `n`.
- `return n % 2 === 0;`: This line checks if `n` modulo 2 (`n % 2`) is equal to 0. If it is, `n` is even, and the function returns `true`. Otherwise, it returns `false`.

Output 2: Sure, let's fix the bug identified in the code. Here is the corrected version of the code:

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

Explanation:

- 1. Function Definition:** The function `isEven` is defined correctly with the syntax `function isEven(n) { ... }`.
- 2. Return Statement:** The expression `n % 2 === 0` checks if `n` is even. If `n` modulo 2 is 0, it returns `true`, indicating that `n` is even. Otherwise, it returns `false`, indicating that `n` is odd.

This function can now be used to check if a number is even:

```
console.log(isEven(4)); // true
console.log(isEven(7)); // false
```

This will correctly output `true` for even numbers and `false` for odd numbers."

Analysis

Chain-of-thought resulted in a single, reasoned response from both models identifying and fixing the bug. Mistral Large potentially added more JavaScript context (truthy/falsy). Compared to prompt chaining, CoT combined diagnosis and solution, offering a more integrated explanation but less structural separation. Prompt Chaining Exact match: 0.24517906336088155. Chain Of Thought Exact match: 0.09955483609874545

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

Explanation:

- `n % 2` gives the remainder of `n` divided by 2.
- `== 0` checks if the remainder is 0, which means `n` is even.
- `return` the result of the comparison, which will be `true` if `n` is even and `false` if `n` is odd.

Example Usage:

```
console.log(isEven(4)); // true
console.log(isEven(5)); // false
```

This function should now correctly determine if a number is even or not.

Task 19: Summary Decomposition (C++)

| Goal | Decompose the high-level comment/summary into logical steps | | | | | | | | | | | | |
|--------------------------|---|--------------------------|--|---------------------|----------------------------|-------------------|------------------------|----------------|---------------------------------------|------------------|--------------------------|-----------------------|---|
| Temperature | 0.7 | Token Limit | 2048 | | | | | | | | | | |
| Prompt Strategy 1 | Chain of Thought | Prompt Strategy 2 | Few Shot | | | | | | | | | | |
| Prompt | <p>Decompose the high-level comment and summary into logical steps. Let's think step by step. Here's the code:</p> <pre>// Function that validates an input, calculates square, and returns result int process(int x) { if (x < 0) return -1; return x * x; }</pre> | Prompt | <p>Example 1:
Input:</p> <pre>// Function that adds two numbers and returns the result int add(int a, int b) { return a + b; }</pre> <p>Output:</p> <ol style="list-style-type: none"> 1. Receive two numbers as input. 2. Add the numbers. 3. Return the result. <p>Example 2: Input:</p> <pre>// Function that checks if a number is even bool isEven(int x) { return x % 2 == 0; }</pre> <p>Output:</p> <ol style="list-style-type: none"> 1. Receive a number as input. 2. Check if it is divisible by 2. 3. Return true if divisible, false otherwise. <p>Decompose the high-level comment and summary into logical steps. Here's the code:</p> <pre>// Function that validates an input, calculates square, and returns result int process(int x) { if (x < 0) return -1; return x * x; }</pre> | | | | | | | | | | |
| ChatGPT Output | <p>Absolutely! Let's decompose the high-level comment and summary for the code step by step:</p> <p>High-level comment:
<code>// Function that validates an input, calculates square, and returns result</code></p> <p>Logical Steps</p> <ol style="list-style-type: none"> 1. Input Validation: <ul style="list-style-type: none"> The function receives an integer input (<code>x</code>). It checks if the input (<code>x</code>) is less than 0 (i.e., negative). 2. Error Handling: <ul style="list-style-type: none"> If the input is negative, the function returns <code>-1</code> to indicate invalid input. 3. Calculation: <ul style="list-style-type: none"> If the input is valid (i.e., zero or positive), the function calculates the square of the input (<code>x * x</code>). 4. Return Result: <ul style="list-style-type: none"> The function returns the squared value as the result. <p>Summary Table</p> <table border="1"> <thead> <tr> <th>Step</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1. Input Validation</td> <td>Check if input is negative</td> </tr> <tr> <td>2. Error Handling</td> <td>If negative, return -1</td> </tr> <tr> <td>3. Calculation</td> <td>If valid, compute the square of input</td> </tr> <tr> <td>4. Return Result</td> <td>Return the squared value</td> </tr> </tbody> </table> <p>Step-by-step explanation with code reference:</p> <pre>int process(int x) { if (x < 0) // Step 1: Validate input (is x negative?) return -1; // Step 2: If negative, return error code -1 return x * x; // Step 3: If valid, compute and return x squared }</pre> <p>In summary:</p> <ul style="list-style-type: none"> The function first checks the validity of the input. If invalid (negative), it returns an error code. If valid (non-negative), it returns the square of the input. | Step | Description | 1. Input Validation | Check if input is negative | 2. Error Handling | If negative, return -1 | 3. Calculation | If valid, compute the square of input | 4. Return Result | Return the squared value | ChatGPT Output | <p>Input:</p> <pre>// Function that validates an input, calculates square, and returns result int process(int x) { if (x < 0) return -1; return x * x; }</pre> <p>Output:</p> <ol style="list-style-type: none"> 1. Receive a number as input. 2. Check if the number is negative. 3. If it is negative, return -1. 4. If it is non-negative, calculate its square. 5. Return the squared result. |
| Step | Description | | | | | | | | | | | | |
| 1. Input Validation | Check if input is negative | | | | | | | | | | | | |
| 2. Error Handling | If negative, return -1 | | | | | | | | | | | | |
| 3. Calculation | If valid, compute the square of input | | | | | | | | | | | | |
| 4. Return Result | Return the squared value | | | | | | | | | | | | |
| Mistral Output | <p>Sure, let's break down the high-level comment and the code into logical steps:</p> <p>High-Level Comment Breakdown:</p> <ol style="list-style-type: none"> 1. Validate the input: Ensure the input meets certain criteria. 2. Calculate the square: Compute the square of the input if it is valid. 3. Return the result: Output the result based on the validity of the input. <p>Code Breakdown:</p> <ol style="list-style-type: none"> 1. Function Definition: <pre>int process(int x) {</pre> <ul style="list-style-type: none"> Defines a function named <code>process</code> that takes an integer <code>x</code> and returns an integer. | Mistral Output | <p>To decompose the high-level comment and summary for the given function into logical steps, let's break down the process step-by-step.</p> <p>Input:</p> <pre>int process(int x) { if (x < 0) return -1; return x * x; }</pre> <p>Output:</p> <ol style="list-style-type: none"> 1. Receive a number as input. 2. Check if the number is negative. 3. If the number is negative, return -1. 4. If the number is non-negative, calculate its square. | | | | | | | | | | |

| | | |
|----------|--|---|
| | <p>2. Input Validation:</p> <pre>if (x < 0) return -1;</pre> <ul style="list-style-type: none"> Checks if the input x is less than 0. If x is less than 0, the function returns -1 to indicate an invalid input. <p>3. Calculate and Return the Square:</p> <pre>return x * x;</pre> <ul style="list-style-type: none"> If the input x is not less than 0, the function calculates the square of x. Returns the square of x. <p>4. End of Function:</p> <pre>}</pre> <ul style="list-style-type: none"> Closes the function definition. <p>Step-by-Step Process:</p> <ol style="list-style-type: none"> Function Call: The function <code>process</code> is called with an integer argument x. Validation Check: <ul style="list-style-type: none"> The function checks if x is less than 0. If x is less than 0, it returns -1 immediately. Square Calculation: <ul style="list-style-type: none"> If x is 0 or greater, the function calculates $x * x$. Result Return: <ul style="list-style-type: none"> The function returns the calculated square of x. <p>Example:</p> <ul style="list-style-type: none"> If <code>process(5)</code> is called: <ul style="list-style-type: none"> x is 5, which is not less than 0. The function calculates $5 * 5 = 25$. The function returns 25. If <code>process(-3)</code> is called: <ul style="list-style-type: none"> x is -3, which is less than 0. The function returns -1. <p>This step-by-step breakdown helps clarify the logical flow of the function and how it processes the input to produce the output.</p> | <p>5. Return the result.</p> <p>Detailed Breakdown:</p> <ol style="list-style-type: none"> Receive a number as input: <ul style="list-style-type: none"> The function <code>process</code> takes an integer x as its input parameter. Check if the number is negative: <ul style="list-style-type: none"> The function contains an <code>if</code> statement that checks whether x is less than 0 ($x < 0$). If the number is negative, return -1: <ul style="list-style-type: none"> If the condition $x < 0$ is true, the function returns -1. If the number is non-negative, calculate its square: <ul style="list-style-type: none"> If the condition $x < 0$ is false (i.e., x is non-negative), the function proceeds to calculate the square of x using the expression $x * x$. Return the result: <ul style="list-style-type: none"> The function returns the calculated square of x. <p>Summary:</p> <p>The function <code>process</code> performs the following operations:</p> <ul style="list-style-type: none"> It accepts an integer x. It validates the input by checking if x is negative. If x is negative, it returns -1. If x is non-negative, it calculates and returns the square of x. <p>This decomposition helps in understanding the logical flow and each step involved in the function's execution.</p> |
| Analysis | Few-shot examples strongly guided both models to decompose the summary according to the demonstrated format. Mistral Large introduced related details (error codes) not explicit in the summary/examples. Compared to chain-of-thought, few-shot enforced structural consistency but allowed less flexibility for elaboration or inference. Chain Of Thought Exact match: 0.051543472104219765. Few Shot Sentence embedding similarity: 0.7539708614349365 | |

Task 20: Purpose Inference Completion (Python)

| | | | |
|-------------------|---|--|--|
| Goal | Complete the function based on intent | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Zero Shot | Prompt Strategy 2 | Chain of Thought |
| Prompt | <p>Complete the function to return the average of scores. Here's the code:</p> <pre>def calculate_average(scores): total = 0 # TODO: Complete to return average pass</pre> | <p>Prompt</p> <p>Complete the function to return the average of scores. Let's think step by step. Here's the code:</p> <pre>def calculate_average(scores): total = 0 # TODO: Complete to return average pass</pre> | <p>Absolutely! Let's walk through the steps to complete the function:</p> <p>Step-by-step plan:</p> <ol style="list-style-type: none"> Initialize <code>total</code> to 0. Iterate through each score in the <code>scores</code> list and add it to <code>total</code>. Calculate the average by dividing <code>total</code> by the number of scores (<code>len(scores)</code>). Return the average. Handle the edge case where <code>scores</code> is an empty list to avoid division by zero. <p>Here's the completed function:</p> <pre>def calculate_average(scores): if not scores: # handle empty list return 0 total = 0 for score in scores: total += score average = total / len(scores) return average</pre> <p>Example usage:</p> <pre>print(calculate_average([80, 90, 100])) # Output: 90.0 print(calculate_average([])) # Output: 0</pre> <p>Let me know if you'd like a shorter version or want to use <code>sum()</code>!</p> |
| ChatGPT Output | <p>Certainly! Here's how you can complete the function:</p> <pre>def calculate_average(scores): total = 0 for score in scores: total += score if len(scores) == 0: return 0 # or you can raise an exception if preferred return total / len(scores)</pre> <p>Or more simply using the built-in <code>sum</code>:</p> <pre>def calculate_average(scores): if not scores: return 0 # or raise an exception return sum(scores) / len(scores)</pre> <p>Usage Example:</p> <pre>print(calculate_average([90, 80, 70])) # Output: 80.0</pre> <p>Let me know if you need any further explanation!</p> | <p>ChatGPT Output</p> | <p>To With{ return{sf e1 pass - toect we need to calculate the average of the scores provided to us. Let's break down the steps:</p> <ol style="list-style-type: none"> Sum the Scores: First, we need to add up all the scores. Count the Scores: Next, we need to count how many scores there are. Calculate the Average: Finally, we divide the total sum of the scores by the count of the scores to get the average. <p>Here is the completed function:</p> <pre>def calculate_average(scores): total = 0 count = 0 for score in scores: total += score count += 1 if count == 0: return 0 # To avoid division by zero average = total / count return average</pre> <p>Explanation:</p> <ul style="list-style-type: none"> Initialization: We start by initializing <code>total</code> to 0 and <code>count</code> to 0. Loop Through Scores: We loop through each score in the <code>scores</code> list, adding each score to <code>total</code> and incrementing <code>count</code> by 1. Check for Empty List: If the list is empty (i.e., <code>count</code> is 0), we return 0 to avoid division by zero. Calculate Average: We calculate the average by dividing <code>total</code> by <code>count</code>. Return Average: Finally, we return the calculated average. <p>This function will correctly calculate and return the average of the provided scores.</p> |
| Mistral Output | <p>Certainly! To complete the function <code>calculate_average</code> so that it returns the average of the scores, you need to sum all the scores and then divide by the number of scores. Here's the completed function:</p> <pre>def calculate_average(scores): if not scores: # Check if the list is empty return 0 total = sum(scores) # Calculate the total sum of the scores average = total / len(scores) # Calculate the average return average</pre> <p>Here's a breakdown of what the function does:</p> <ol style="list-style-type: none"> It checks if the <code>scores</code> list is empty. If it is, it returns 0 to avoid division by zero. It calculates the total sum of the scores using the <code>sum</code> function. It divides the total sum by the number of scores to get the average. It returns the average. <p>You can test the function with some example inputs to ensure it works correctly:</p> <pre># Example usage scores = [85, 90, 78, 92, 88] average = calculate_average(scores) print(f"The average score is: {average}") # Output: The average score is: 86.6</pre> | <p>Mistral Output</p> | <p>To With{ return{sf e1 pass - toect we need to calculate the average of the scores provided to us. Let's break down the steps:</p> <ol style="list-style-type: none"> Sum the Scores: First, we need to add up all the scores. Count the Scores: Next, we need to count how many scores there are. Calculate the Average: Finally, we divide the total sum of the scores by the count of the scores to get the average. <p>Here is the completed function:</p> <pre>def calculate_average(scores): total = 0 count = 0 for score in scores: total += score count += 1 if count == 0: return 0 # To avoid division by zero average = total / count return average</pre> <p>Explanation:</p> <ul style="list-style-type: none"> Initialization: We start by initializing <code>total</code> to 0 and <code>count</code> to 0. Loop Through Scores: We loop through each score in the <code>scores</code> list, adding each score to <code>total</code> and incrementing <code>count</code> by 1. Check for Empty List: If the list is empty (i.e., <code>count</code> is 0), we return 0 to avoid division by zero. Calculate Average: We calculate the average by dividing <code>total</code> by <code>count</code>. Return Average: Finally, we return the calculated average. <p>This function will correctly calculate and return the average of the provided scores.</p> |
| Analysis | Chain-of-thought explicitly led both models to consider and handle the empty list edge case before implementing the average. Mistral Large's reasoning might also include rounding. Compared to zero-shot, CoT prompted explicit consideration of edge cases and implementation details, leading to more robust code from both models. Zero Shot Exact match: 0.177153329260843. Chain Of Thought Exact match: 0.04901036757756835 | | |

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

| | | | |
|-------------------|--|-------------------|---|
| Goal | Analyze and refactor this utility script for better readability and correctness | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Prompt Chaining | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Prompt 1: Identify the issues in this python function. Here's the code:</p> <pre># utils.py - Script to parse and summarize numeric CSV files import csv def read_csv(filepath): with open(filepath, 'r') as f: return [row for row in csv.reader(f)] def summarize_column(data, index): values = [float(row[index]) for row in data[1:]] total = sum(values) avg = total / len(values) return total, avg def main(): filepath = 'data.csv' data = read_csv(filepath) total, avg = summarize_column(data, 1) print('Total:', total) print('Average:', avg) if __name__ == '__main__': main()</pre> <p>Prompt 2: Refactor the code for better readability and correctness.</p> | Prompt | Analyze and refactor this utility script for better readability and correctness. <pre># utils.py - Script to parse and summarize numeric CSV files import csv def read_csv(filepath): with open(filepath, 'r') as f: return [row for row in csv.reader(f)] def summarize_column(data, index): values = [float(row[index]) for row in data[1:]] total = sum(values) avg = total / len(values) return total, avg def main(): filepath = 'data.csv' data = read_csv(filepath) total, avg = summarize_column(data, 1) print('Total:', total) print('Average:', avg) if __name__ == '__main__': main()</pre> |
| ChatGPT Output | <p>Output 1: Let's review your code and identify the issues:</p> <pre># utils.py - Script to parse and summarize numeric CSV files import csv def read_csv(filepath): with open(filepath, 'r') as f: return [row for row in csv.reader(f)] def summarize_column(data, index): values = [float(row[index]) for row in data[1:]] total = sum(values) avg = total / len(values) return total, avg def main(): filepath = 'data.csv' data = read_csv(filepath) total, avg = summarize_column(data, 1) print('Total:', total) print('Average:', avg) if __name__ == '__main__': main()</pre> <p>Issues Identified</p> <ol style="list-style-type: none"> No CSV Dialect Specified <ul style="list-style-type: none"> If your CSV uses a delimiter other than a comma, or has quoting, you may need to specify the dialect or parameters to <code>csv.reader</code>. No Error Handling <ul style="list-style-type: none"> If the file does not exist, or the data is not numeric, the code will raise exceptions. Assumes Header Row <ul style="list-style-type: none"> <code>summarize_column</code> skips the first row (<code>data[1:]</code>), assuming it's a header. If the CSV has no header, this will skip data. No Check for Empty Data <ul style="list-style-type: none"> If the CSV is empty or has only a header, <code>values</code> will be empty, causing a <code>ZeroDivisionError</code> in <code>avg = total / len(values)</code>. No Check for Column Index <ul style="list-style-type: none"> If the index is out of range for any row, you'll get an <code>IndexError</code>. No Handling of Missing/Non-numeric Values <ul style="list-style-type: none"> If a cell is empty or contains non-numeric data, <code>float(row[index])</code> will raise a <code>ValueError</code>. Hardcoded Filepath and Column Index <ul style="list-style-type: none"> The script always reads 'data.csv' and column index 1. This reduces flexibility. No Encoding Specified <ul style="list-style-type: none"> Not specifying encoding in <code>open()</code> can cause issues with non-ASCII data. <p>Example of Improved Code</p> <p>Here's a version that addresses most of the above issues:</p> <pre>import csv def read_csv(filepath): with open(filepath, 'r', encoding='utf-8') as f: return [row for row in csv.reader(f)] def summarize_column(data, index): values = [] for row in data[1:]: # assumes header try: values.append(float(row[index])) except (IndexError, ValueError): continue # skip rows with missing or non-numeric data if not values: return 0, 0 total = sum(values) avg = total / len(values) return total, avg def main(): filepath = 'data.csv' try: data = read_csv(filepath) except FileNotFoundError: print(f"File not found: {filepath}") return if not data or len(data) < 2: print("No data to summarize.") return total, avg = summarize_column(data, 1) print('Total:', total) print('Average:', avg) if __name__ == '__main__': main()</pre> <p>Summary of Issues</p> <ul style="list-style-type: none"> No error handling for file I/O, missing data, or non-numeric values. Assumes header row and at least one data row. No check for empty data or out-of-range column index. Hardcoded file path and column index. No encoding specified. <p>Addressing these will make your script more robust and user-friendly.</p> <p>Output 2: Certainly! Here's a refactored version of your script, with improved readability, error handling, and flexibility. I've added comments and made the code more robust:</p> | ChatGPT Output | <p>Certainly! Here's an analysis of your script, followed by a refactored version with improved readability and correctness.</p> <p>Analysis of Original Script</p> <p>Strengths:</p> <ul style="list-style-type: none"> Script is concise and clear in intent: read a CSV, summarize one column, print results. <p>Points to Improve:</p> <ol style="list-style-type: none"> Readability: <ul style="list-style-type: none"> Functions lack docstrings. Variable and parameter names could be more descriptive. Magic numbers (e.g., hardcoded index 1). Correctness & Robustness: <ul style="list-style-type: none"> No error handling for file I/O or conversion to float. Assumes all rows have enough columns and the values are always numeric. Hardcoded filename and column index. No handling for empty files or columns. Doesn't skip empty rows. Python Style: <ul style="list-style-type: none"> Use context managers and <code>with</code> for files (good). Could use <code>csv.DictReader</code> for better readability. <p>Refactored Script</p> <pre># utils.py - Script to parse and summarize numeric CSV files import csv def read_csv(filepath): """ Reads a CSV file and returns the header and rows. Args: filepath (str): Path to the CSV file. Returns: tuple: (header, rows) Raises: FileNotFoundError: If the file does not exist. csv.Error: If the file is not a valid CSV. """ with open(filepath, newline='') as f: reader = csv.reader(f) rows = [row for row in reader if row] # Skip empty rows if not rows: raise ValueError("CSV file is empty.") header, *data_rows = rows return header, data_rows def summarize_column(data, col_index): """ Computes the total and average for a numeric column. Args: data (list of lists): The CSV data rows (excluding header). col_index (int): Index of the column to summarize. Returns: tuple: (total, average) Raises: ValueError: If the column cannot be converted to float. """ values = [] for i, row in enumerate(data, 2): # start at 2 to correspond to row number in the file try: values.append(float(row[col_index])) except (IndexError, ValueError): raise ValueError(f"Invalid or missing numeric value in row {i} for column {col_index}") if not values: raise ValueError("No data to summarize.") total = sum(values) avg = total / len(values) return total, avg def main(filepath='data.csv', column=1): """ Main function to read a CSV and print total and average of a column. Args: filepath (str): Path to the CSV file. column (int): Index of the column to summarize. Returns: None Raises: IndexError: If the column index {column} out of range for CSV with {len(header)} columns. """ header, data = read_csv(filepath) if column >= len(header): raise IndexError(f"Column index {column} out of range for CSV with {len(header)} columns") total, avg = summarize_column(data, column) print(f"Column: {header[column]}") print('Total:', total) print('Average:', avg) if __name__ == '__main__': import sys # Optionally accept filename and column index from the command line if len(sys.argv) > 2: main(sys.argv[1], int(sys.argv[2])) else: main()</pre> <p>Key Improvements:</p> <ul style="list-style-type: none"> Added docstrings for clarity. Added error handling for file reading and numeric conversion. Skips empty rows. Accepts column index and filename as parameters, for flexibility. Prints the column name for context. Checks if the requested column exists. <p>Usage:</p> <pre>python utils.py data.csv 1</pre> <p>or just</p> <pre>python utils.py</pre> |

```

import csv

def read_csv(filepath, encoding='utf-8'):
    """Read a CSV file and return a list of rows."""
    try:
        with open(filepath, 'r', encoding=encoding, newline='') as f:
            reader = csv.reader(f)
            return [row for row in reader]
    except FileNotFoundError:
        print(f'Error: File "{filepath}" not found.')
        return []
    except Exception as e:
        print(f'Error reading "{filepath}": {e}')
        return []

def summarize_column(data, index, has_header=True):
    """
    Summarize a numeric column in the data.
    Skips header if has_header is True.
    Ignores rows with missing or non-numeric values.
    """
    if not data:
        print("No data provided.")
        return 0, 0

    start_row = 1 if has_header else 0
    values = []
    for row in data[start_row:]:
        try:
            value = float(row[index])
            values.append(value)
        except (IndexError, ValueError):
            continue # Skip rows with missing or invalid data

    if not values:
        print("No valid numeric data found in the specified column.")
        return 0, 0

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

def main():
    filepath = 'data.csv'
    column_index = 1 # Change as needed (0-based index)
    has_header = True

    data = read_csv(filepath)
    if not data:
        return

    total, avg = summarize_column(data, column_index, has_header)
    print(f'Total: {total}')
    print(f'Average: {avg}')

if __name__ == '__main__':
    main()

```

Key improvements:

- Added docstrings and comments for clarity.
- Added error handling for file reading and data conversion.
- Made the header-skipping behavior configurable.
- Used function parameters for file path, column index, and header presence for easier modification.
- Handles empty or invalid data gracefully.

Let me know if you want to make the script even more flexible (e.g., by accepting command-line arguments)!

to use defaults.

Let me know if you'd like further improvements or if you want to process columns by name instead of index.

Mistral Output

Output 1: The provided Python script has several issues that could potentially cause errors or unexpected behavior. Here are the identified issues along with explanations:

- Lack of Error Handling:** The script does not handle potential errors such as file not found, CSV parsing errors, or conversion errors (e.g., non-numeric values).
- Assumption of Column Index:** The `summarize_column` function assumes that the column index provided is valid and that all values in that column are convertible to floats.
- Hardcoded File Path:** The file path `'data.csv'` is hardcoded in the `main` function, which reduces flexibility.
- Lack of Documentation:** There are no docstrings or comments explaining the purpose of the functions or the script as a whole.
- Potential Division by Zero:** The `summarize_column` function could encounter a division by zero error if the list `values` is empty.
- Inefficient Data Reading:** The `read_csv` function reads the entire file into memory, which might not be efficient for large files.

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

```

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

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

    Args:
        filepath (str): The path to the CSV file.

    Returns:
        list: A list of rows from the CSV file.
    """
    try:
        with open(filepath, 'r') as f:
            return [row for row in csv.reader(f)]
    except FileNotFoundError:
        print(f'Error: File "{filepath}" not found.')
        return []
    except Exception as e:
        print(f'Error reading CSV file: {e}')
        return []

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

    Args:
        data (list): The CSV data as a list of rows.
        index (int): The index of the column to summarize.

    Returns:
        tuple: A tuple containing the total and average of the column values.
    """
    values = []
    try:
        for row in data[1:]:
            values.append(float(row[index]))
    except ValueError as e:
        print(f'Error converting value to float: {e}')
        return None, None
    except IndexError as e:
        print(f'Error accessing column index {index}: {e}')
        return None, None

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

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

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

    Args:
        filepath (str): The path to the CSV file.

    Returns:
        tuple: A tuple containing the total and average of the specified column.
    """
    data = read_csv(filepath)
    if not data:
        return

    total, avg = summarize_column(data, 1)
    if total is not None and avg is not None:
        print(f'Total: {total}')
        print(f'Average: {avg}')

```

Mistral Output

Certainly! Let's analyze and refactor the provided script for better readability and correctness.

Analysis:

1. Functionality:

- `read_csv(filepath)`: Reads CSV data from a file and returns it as a list of rows.
- `summarize_column(data, index)`: Summarizes a specified column (index) in the data by calculating the total sum and the average of the column values.
- `main()`: Reads a CSV file, summarizes a column, and prints the total and average.

2. Issues:

- There is no error handling for file reading or data parsing.
- The script assumes that the column values are always numeric and can be converted to floats without any exception handling.
- The script assumes the first row is the header and skips it while processing.
- No type hints are provided, making the function signatures less clear.

Refactoring:

- Add Error Handling:** Handle file reading errors and data parsing errors.
- Add Type Hints:** Make the function signatures clearer.
- Improve Readability:** Use more descriptive variable names and add comments.
- Modularize Code:** Separate concerns more clearly.

Here is the refactored script:

```

# utils.py - Script to parse and summarize numeric CSV files
import csv
from typing import List, Tuple

def read_csv(filepath: str) -> List[List[str]]:
    """
    Reads a CSV file and returns its contents as a list of rows.

    Args:
        filepath (str): The path to the CSV file.

    Returns:
        List[List[str]]: A list of rows from the CSV file.
    """
    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 "{filepath}": {e}')
        return []

def summarize_column(data: List[List[str]], index: int) -> Tuple[float, float]:
    """
    Summarizes a specified column in the data by calculating the total sum and average.

    Args:
        data (List[List[str]]): The CSV data.
        index (int): The index of the column to summarize.

    Returns:
        Tuple[float, float]: A tuple containing the total sum and average of the column values.
    """
    values = []
    try:
        for row in data[1:]:
            values.append(float(row[index]))
    except ValueError:
        print(f'Warning: Non-numeric value "{row[index]}" found at row {data.index(row) + 1}.')
    except IndexError:
        print(f'Warning: Missing value at row {data.index(row) + 1}, column {index + 1}.')

    if not values:
        print("Warning: No valid numeric values found in the specified column.")
        return 0.0, 0.0

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

def main():
    filepath = 'data.csv'
    data = read_csv(filepath)
    if not data:
        return

    total, avg = summarize_column(data, 1)
    print(f'Total: {total}')
    print(f'Average: {avg}')

```

```

        print('Total:', total)
        print('Average:', avg)
    else:
        print("Failed to summarize the column.")

if __name__ == '__main__':
    filepath = 'data.csv'
    main(filepath)

```

Key Changes and Improvements:

- Error Handling:** Added try-except blocks to handle file not found, CSV parsing errors, and conversion errors.
- Column Index Validation:** Added checks for invalid column index and non-numeric values.
- Flexible File Path:** Made the file path a parameter to the `main` function for better flexibility.
- Documentation:** Added docstrings to explain the purpose of each function.
- Handling Empty Values:** Added a check to handle cases where the column might not contain any numeric values.

These changes should make the script more robust and easier to understand and maintain.

Output 2: Sure, let's refactor the code for better readability and correctness. We'll focus on modularizing the code, improving error handling, and ensuring that the script is easy to understand.

Here is the refactored version:

```

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

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

    Args:
        filepath (str): The path to the CSV file.

    Returns:
        list: A list of rows from the CSV file.
    """
    rows = []
    try:
        with open(filepath, 'r') as f:
            reader = csv.reader(f)
            for row in reader:
                rows.append(row)
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
    except Exception as e:
        print(f"Error reading CSV file: {e}")
    return rows

def extract_numeric_column(data, index):
    """
    Extracts numeric values from a specific column in the CSV data.

    Args:
        data (list): The CSV data as a list of rows.
        index (int): The index of the column to extract.

    Returns:
        list: A list of numeric values from the specified column.
    """
    values = []
    for row in data[1:]: # Skip header row
        try:
            value = float(row[index])
            values.append(value)
        except ValueError:
            print(f"Warning: Non-numeric value '{row[index]}' in row {data.index(row) + 1}.")
        except IndexError:
            print(f"Warning: Missing value in row {data.index(row) + 1}.")
    return values

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

    Args:
        values (list): A list of numeric values.

    Returns:
        tuple: A tuple containing the total and average of the values.
    """
    if not values:
        print("Error: No numeric values found.")
        return None, None
    total = sum(values)
    avg = total / len(values)
    return total, avg

def main(filepath, column_index):
    """
    Main function to read CSV data and summarize a specific column.

    Args:
        filepath (str): The path to the CSV file.
        column_index (int): The index of the column to summarize.
    """
    data = read_csv(filepath)
    if not data:
        return

    values = extract_numeric_column(data, column_index)
    total, avg = calculate_summary(values)

    if total is not None and avg is not None:
        print('Total:', total)
        print('Average:', avg)
    else:
        print("Failed to summarize the column.")

if __name__ == '__main__':
    filepath = 'data.csv'
    column_index = 1 # Assuming the column index to summarize is 1
    main(filepath, column_index)

```

Key Changes and Improvements:

- Modularization:** The code is divided into smaller functions, each with a single responsibility. This makes the code easier to understand and maintain.
- Error Handling:** Improved error handling with more specific messages for different types of errors (e.g., file not found, non-numeric values, missing values).
- Documentation:** Added docstrings to explain the purpose of each function.
- Flexibility:** The `main` function now accepts the file path and column index as parameters, making the script more flexible.
- Robustness:** The script is more robust in handling different types of errors and edge cases, such as non-numeric values and missing values in the CSV file.

These improvements should make the script more readable, maintainable, and robust.

return

```

total, avg = summarize_column(data, 1)
print('Total:', total)
print('Average:', avg)
if __name__ == '__main__':
    main()

```

Key Improvements:

- Error Handling:** Added try-except blocks to handle file reading errors and data parsing errors.
- Type Hints:** Added type hints to function signatures for better clarity.
- Readability:** Improved variable names and added comments to explain the purpose of each function.
- Modularization:** Separated concerns more clearly and ensured that the script is robust against common errors.

This refactored script should be more readable, maintainable, and robust against common issues encountered when reading and processing CSV files.

Analysis

Self-consistency confirmed the need for modularity and error handling. Mistral Large's attempts showed more variability, sometimes proposing advanced optimizations like parallel processing. Compared to prompt chaining, self-consistency explored a wider range of potential improvements for the complex task, including performance aspects. Prompt Chaining Exact match: 0.23076923076923078. Self-Consistency Exact match: 0.22785515320334263

Task 22: Code Completion and Robustness Enhancement (Python)

| | | | |
|-------------------|---|--|------------------|
| Goal | Complete this file-processing script to clean lines and count word frequencies correctly | | |
| Temperature | 0.7 | Token Limit | 2048 |
| Prompt Strategy 1 | Chain of Thought | Prompt Strategy 2 | Self Consistency |
| Prompt | <p>Complete this file-processing script to clean lines and count word frequencies correctly. Let's think step by step. Here's the code:</p> <pre> # file_processor.py - Complete script for processing text files import string def load_file(filepath): </pre> | <p>Prompt</p> <p>Complete this file-processing script to clean lines and count word frequencies correctly. Here's the code:</p> <pre> # file_processor.py - Complete script for processing text files import string def load_file(filepath): with open(filepath, 'r') as f: </pre> | |

```

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

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

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

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

if __name__ == '__main__':
    main()

```

```

        return f.readlines()

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

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

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

if __name__ == '__main__':
    main()

```

Let's walk through the steps and fill in the missing parts of your script:

Step 1: clean_line function

You want to remove punctuation and make everything lowercase. Use `str.translate` with `str.maketrans` for punctuation removal.

Step 2: Test that word counting works on cleaned lines.

Here's the corrected and completed script:

```

# 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)
    # Remove punctuation
    line = line.translate(translator)
    # Make lowercase
    line = line.lower()
    return line

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

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

if __name__ == '__main__':
    main()

```

Summary of changes:

- `clean_line` now removes punctuation and lowerscases text.
- Everything else remains as you wrote.

How it works:

- Each line is cleaned by removing punctuation and converting to lowercase.
- Words are split and counted.
- Results are printed sorted alphabetically.

Example:

If `input.txt` is:

```
Hello, world!
Hello again.
```

The output will be:

```
again: 1
hello: 2
world: 1
```

Let me know if you need further enhancements!

ChatGPT Output

ChatGPT Output

Certainly! Here's the completed script with the missing `clean_line` function implemented. This function removes punctuation and converts the line to lowercase.

```

# file_processor.py - Complete script for processing text files

import string

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

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

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

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

if __name__ == '__main__':
    main()

```

How it works:

- `clean_line` uses `str.translate` to remove all punctuation and converts the line to lowercase.
- The rest of the script reads the file, cleans each line, splits into words, and counts their frequencies.

Usage:

Place your text in `input.txt` and run the script. It will print word frequencies, sorted alphabetically.

Mistral Output

To complete this script, we need to implement the `clean_line` function to remove punctuation and convert the line to lowercase. Then, we'll ensure the `count_words` function processes the cleaned lines correctly. Here is the completed script:

```

# file_processor.py - Complete script for processing text files
import string

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

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

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

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

if __name__ == '__main__':
    main()

```

Explanation:

- `load_file(filepath)`
 - Reads the file line by line and returns a list of lines.
- `clean_line(line)`
 - Removes punctuation using `str.translate` and `str.maketrans`.
 - Converts the line to lowercase using `str.lower()`.
- `count_words(lines)`
 - Initializes an empty dictionary `word_counts`.
 - Iterates through each line, cleans it, and splits it into words.
 - Counts the occurrences of each word and updates the dictionary.

Mistral Output

To complete the script, you need to implement the `clean_line` function to remove punctuation and convert the text to lowercase. Here's the completed script:

```

# file_processor.py - Complete script for processing text files
import string

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

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

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

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

if __name__ == '__main__':
    main()

```

Explanation:

- `load_file(filepath)`
 - Reads the file line by line and returns a list of lines.
- `clean_line(line)`
 - Uses `str.maketrans` and `str.translate` to remove punctuation.
 - Converts the line to lowercase using `str.lower`.
- `count_words(lines)`
 - Initializes an empty dictionary `word_counts`.
 - Iterates over each line, cleans it, and splits it into words.
 - Updates the word count in the dictionary.
- `main()`

4. main():

- o Defines the file path.
- o Loads the lines from the file.
- o Counts the word frequencies.
- o Prints the word frequencies in a sorted order.

This script will read a text file named `input.txt`, clean each line by removing punctuation and converting to lowercase, count the frequency of each word, and print the word frequencies in alphabetical order.

- o Specifies the file path.

- o Loads the file and counts the words.

- o Prints the word counts sorted alphabetically.

This script will read a text file, clean each line by removing punctuation and converting to lowercase, count the frequency of each word, and print the word counts in alphabetical order.

Analysis

The chain of thought prompt produced a better explanation and set of examples for GPT and Mistral. The chain of thought version used step-by-step prompting, so the model did a better job explaining the logic behind its algorithm, which fills in a file-processing script given certain functions to implement. Incorporating a step-by-step explanation can guide the model towards more descriptive and in-depth responses. Chain Of Thought Exact match: 0.4328690807799443. Self-Consistency Exact match: 0.5588428665351742