

GenAI for Software Development: **Assignment 3**

AJ Arnold

Max White

ajarnold@wm.edu

mjwhite05@wm.edu

-
- Notes in the README
 - GitHub Repo: <https://github.com/ajarnold920/Prompt-Engineering-In-Context-Learning>
-

1. **Code Summarization (Java)** Summarize the functionality of the following method:

```
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;  
}
```

a. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```
**Summary of Functionality:**
```

```
The `countWordFrequency` method takes a list of  
strings (`words`) and returns a map that contains  
each unique word as a key and the number of times  
that word appears in the list as its value. In  
other words, it counts the frequency of each word  
in the input list.
```

2. Comments:

- This provides a good summary, but it is all on one line and not the most readable

ii. Prompt Strategy 2: Role Playing

1. Output:

```
Assistant: Certainly! Here is a summary of the  
provided Java method:
```

```
**Summary:**
```

```
The method `countWordFrequency` takes a list of  
strings (`words`) and returns a map where each  
unique string is a key, and its value is the number  
of times that string appears in the list. It counts  
the frequency of each word in the input list.
```

```
**How it works:**
```

- It creates an empty `HashMap` to store word counts.
- For each word in the list, it increments its count in the map.
- Finally, it returns the map with word frequencies.

2. Comments:

- a. When role playing, chatGPT goes more detail on the method implementation

b. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output:

This method counts the frequency of each word in a given list of words and returns a map where the keys are the unique words and the values are their corresponding frequencies.

Here's a step-by-step breakdown:

1. It takes a list of strings (`words`) as input.
2. It initializes an empty HashMap (`freqMap`) to store the word frequencies.
3. It iterates over each word in the input list.
4. For each word, it checks if the word is already present in the `freqMap`. If it is, it increments the count by 1. If not, it adds the word to the map with a count of 1.
5. Finally, it returns the `freqMap` which contains the frequency of each word in the input list. For example, if the input list is `["apple", "banana", "apple", "orange", "banana", "banana"]`, the method would return a map like `{apple=2, banana=3, orange=1}`.

2. Comments:

- a. LLaMa is much more detailed in its summary for the same prompt, going through the implementation of the method

ii. Prompt Strategy 2: Role Playing

1. Output:

Assistant: This Java method, `countWordFrequency`, takes a list of strings as input and returns a map where the keys are the unique words in the list and the values are their respective frequencies. In other words, it counts the occurrence of each word in the list and returns the result as a map.

2. Comments:

- a. LLaMa becomes less detailed as a coding assistant

Model Analysis:

- The two models had opposite reactions to the change in prompts, ChatGPT became more detailed, while LLaMa became more general. This shows the importance of knowing the model you are dealing with when prompting
-

2. **Bug Fixing (Python – Off-by-One)** Identify and fix the off-by-one error in this function:

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

c. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero-shot

1. Output:

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

2. Comments:

- a. Provided the correct implementation of the fixed function.

ii. Prompt Strategy 2: Prompt Chaining

1. Output

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

2. Comments:

- a. The response was the same as through zero-shot.

d. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output:

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

2. Comments:

- a. Same implementation as previous model.
- ii. Prompt Strategy 2: Prompt Chaining
 1. Output:

```
Alternatively, you can use the formula for the sum of an arithmetic series
...
def sum_range(start, end):
    return (end - start + 1) * (start + end) // 2
...
```

2. Comments:

- a. The model once again gave the same output as the previous 3 prompt runs. However, it did suggest this change, which has a time complexity of $O(1)$, making it more efficient for larger ranges.

Model Analysis:

- The models provided the same answers through each's prompting strategies. The Llama model, through prompt chaining, did however provide a more efficient way to calculate the solution through another algorithm.

3. Bug Classification (C++) Classify the type of bug in the following C++ function:

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

e. Model: **OpenAI GPT - 4.1**

- i. Prompt Strategy 1: Chain of Thought
 1. Output:

Certainly! Let's analyze the code step by step and classify the bug.

The Code

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

---
```

1. ****Understanding the Code****

- ****Function Purpose:****

``getArray`` is supposed to return a pointer to an array of integers of size ``size``.

- ****Local Variable:****

Inside the function, ``arr`` is a local array allocated on the stack.

2. ****Line-by-Line Analysis****

Line 1: ``int arr[size];``

- ****What it does:****

Declares a local array of integers.

- ****Where is it allocated?****

On the stack (automatic storage duration).

***Note:** In standard C++, variable-length arrays (VLAs) like ``int arr[size];`` are not part of the language; they are a GCC extension. Standard C++ requires the array size to be a compile-time constant.*

Line 2: ``return arr;``

- ****What it does:****

Returns a pointer to the local array.

- ****What's the issue?****

Once the function exits, the local array (``arr``) goes out of scope and ****ceases to exist****. The pointer returned now points to ****invalid (dangling) memory****.

3. ****Type of Bug****

****Classification:****

- ****Type:**** **Dangling pointer** (also called returning address of local variable)

- ****Category:**** **Undefined behavior**

- **Why:** The function returns a pointer to a stack-allocated local variable that is destroyed when the function returns. Using this pointer later leads to undefined behavior.

4. Step-by-Step Breakdown

- Function is called:**
`getArray(10)` is called.
- Local array is created:**
`arr` is created on the stack, valid only inside the function.
- Pointer to local array is returned:**
The address of `arr` is returned.
- Function returns:**
Stack frame is destroyed, `arr` no longer exists.
- Pointer used by caller:**
Any use of the returned pointer is undefined behavior because it refers to memory that is no longer valid.

5. Summary Table

| Line | Issue |
|------------------|--|
| Type of Bug | |
| ----- | ----- |
| ----- | ----- |
| `int arr[size];` | Local array on stack (lifetime limited) Implementation detail |
| `return arr;` | Returns pointer to destroyed local variable Dangling pointer bug |

6. How to Fix

- **Allocate on the heap:**
Use `new` or `std::vector<int>`.
`cpp`
int* getArray(int size) {
 return new int[size]; // caller must delete[]
}

```

` ``
- **Or use `std::vector` for safety:**
` ``cpp
    std::vector<int> getArray(int size) {
        return std::vector<int>(size);
    }
` ``

---

## **Conclusion**

**The bug is:**
- **Returning a pointer to a local
(stack-allocated) variable, resulting in a dangling
pointer.**
- **This is undefined behavior and should be
avoided.**

**Category:**
- *Dangling pointer* / *Use-after-free* /
*Returning address of local variable*

```

2. Comments:

- a. This prompting strategy leads to a very wordy response, which is good if you need a lot of guidance, bad for a quick answer.

ii. Prompt Strategy 2: Self-Consistency

1. Output:

```

First chain:
    Certainly! Here's the classification process,
    repeated three times as you requested, followed by
    the aggregate.

---

```

```

**First Classification:**
- **Type:** Lifetime bug (Dangling pointer)
- **Description:** Returning a pointer to a local
(stack-allocated) array, which goes out of scope
when the function returns. Accessing this pointer
later leads to undefined behavior.

---

```

```

**Second Classification:**
- **Type:** Lifetime bug (Dangling pointer)

```

```
- **Description:** The function returns a pointer to a local variable on the stack. The memory is invalid once the function returns, leading to undefined behavior.
```

```
---
```

```
**Third Classification:**
```

```
- **Type:** Lifetime bug (Dangling pointer)  
- **Description:** Returning a pointer to a stack-allocated array. The array ceases to exist after the function returns. Using the returned pointer is undefined behavior.
```

```
---
```

```
## **Aggregate Classification**
```

```
- **Most Common Type:** Lifetime bug (Dangling pointer)  
- **Summary:**  
  The code returns a pointer to a local stack-allocated array. Once the function returns, the array's memory is invalid, and using the returned pointer results in undefined behavior. This is a classic case of a dangling pointer due to returning a pointer to a local variable.
```

```
---
```

```
**Recommendation:**
```

```
To fix this, allocate the array dynamically (e.g., with `new int[size]` or better, use `std::vector<int>`) and ensure the caller is responsible for managing the memory.
```

2. Comments:

- a. Much more concise, still gives solutions to the bug

f. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Chain of Thought

1. Output:

```
Let's break down the bug in this C++ code step by step:
```

```
**Step 1: Local array declaration**
```

```
`int arr[size];`
```


Here, an array `arr` is declared with a size determined by the `size` parameter. This is a local array, meaning it is allocated on the stack and has automatic storage duration. This is not a problem in itself, but it sets the stage for the bug.

****Step 2: Returning a pointer to the local array****

```
`return arr;`
```

The function returns a pointer to the first element of the local array `arr`. This is where the bug occurs.

****The bug: Returning a pointer to a local variable****

The problem is that the local array `arr` is allocated on the stack, and its lifetime ends when the function returns. When the function returns, the memory occupied by `arr` is released, and the pointer returned by the function becomes invalid.

In C++, when a function returns, its local variables are destroyed, and their memory is reclaimed. This means that the pointer returned by the function points to memory that is no longer valid. Attempting to access this memory through the returned pointer will result in undefined behavior.

****Classification:****

This bug is a classic example of a ****dangling pointer**** or ****use-after-free**** bug. A dangling pointer is a pointer that points to memory that has already been freed or is no longer valid. In this case, the pointer is returned by the function, but the memory it points to is no longer valid after the function returns.

To fix this bug, you can consider the following alternatives:

1. Dynamic memory allocation: Use `new` to allocate the array on the heap, and return a pointer to the allocated memory. Don't forget to `delete` the array when it's no longer needed.

2. Container classes: Use a container class like ``std::vector`` to manage the array, and return the container by value or move semantics.

3. Pass by reference: Modify the function to take a reference to an array or a container as an output parameter, and fill it inside the function.

2. Comments:

- a. The LLaMA output is shorter and ends with how to fix the bug rather than classification, which is interesting because the prompt asks for classification.

ii. Prompt Strategy 2: Self-Consistency

1. Output:

First chain:

Here are three classifications for the bug in the given C++ code:

****Classification 1:****

* Category: Memory Management

* Type: Dangling Pointer

* Severity: High

****Classification 2:****

* Category: Scope and Lifetime

* Type: Returning Pointer to Local Variable

* Severity: Critical

****Classification 3:****

* Category: Memory Safety

* Type: Use-After-Free

* Severity: High

Aggregate Classification:

* Category: Memory Management/Memory Safety

* Type: Dangling Pointer/Returning Pointer to Local Variable

* Severity: Critical/High

The bug in the code is that it returns a pointer to a local array, which goes out of scope as soon as the function returns. This leads to undefined behavior, as the returned pointer points to memory that is no longer valid. The aggregate classification reflects the consensus among the three individual classifications, highlighting the critical severity of this bug.

2. Comments:

- a. More variety over the different runs, but still selects correct one in the aggregate, doesn't attempt to solve problem

Model Analysis:

- Both models often tried to fix the bug even though the prompts only mentioned classification, which is likely encouraged due to helping people with problems they didn't know AI could solve.

4. **Generating Email Validators (Python + Regex)** Complete the function using regex to validate basic email addresses:

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

g. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```
```python  
import re

def is_valid_email(email):
 return re.match(r"^[\\w\\.\\-]+@[\\w\\.\\-]+\\.\\w+$", email) is not None
```
```

2. Comments:

- a. Concise and to the point.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```
```python  
import re

def is_valid_email(email):
 # This regex checks for a typical email address format
 pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$'
 return re.match(pattern, email) is not None
```
```

2. Comments:

- a. Since the AI was prompted to be a helpful coding assistant, the code is clearer / more human readable. It also included comments for clarity.

h. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```

...
import re

def is_valid_email(email):
    pattern = r"^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-\.]+$"
    if re.match(pattern, email):
        return True
    return False
...

```

2. Comments:

- a. This code isn't as concise compared to the previous outputs from ChatGPT.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```

...
import re

def is_valid_email(email):
    """
    Returns True if the email address is valid, False otherwise.
    """
    email_regex = r"^[^@]+@[^@]+\.[^@]+"
    if re.match(email_regex, email):
        return True
    return False
...

```

2. Comments:

- a. Once again prompted to be a helpful coding assistant, the AI provides comments for clarification. The implementation is different from zero-shot.

Model Analysis:

- ChatGPT's answers seem to be more concise compared to Llama's.

5. **Generating Flask APIs (Python)** Create a `/greet/<username>` endpoint that returns a JSON greeting:

```

from flask import Flask, jsonify

app = Flask(__name__)

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

```

i. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero Shot

1. Output:

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

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
 return jsonify({'message': f'Hello, {username}!'})
```
```

2. Comments:

- a. Returned code with python descriptor on top

ii. Prompt Strategy 2: Role Playing

1. Output:

Assistant: Certainly! Here's how you can implement the /greet/<username> endpoint to return a JSON greeting:

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

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
 return jsonify({'greeting': f'Hello, {username}!'})
```

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

****Notes:****

- Make sure to use standard single or double quotes (not typographic quotes) around the route.
- Remove any extra spaces in <username> in the route string.
- The jsonify function will produce a JSON response like: {"greeting": "Hello, Alice!"}

2. Comments:

- a. Contains a main method and more of an explanation

j. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
Here is the fixed code:

'''
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    return jsonify({'message': f'Hello, {username}!'})
'''
```

2. Comments:

- a. Still had a text intro despite the prompt asking only for code.

ii. Prompt Strategy 2: Role Playing

1. Output:

```
Assistant: Here's the completed code:
'''
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/greet/<username>')
def greet(username):
    greeting = {'message': f'Hello, {username}!'}
    return jsonify(greeting)

if __name__ == '__main__':
    app.run()
'''
Here's what's changed:

* I removed the extra space in the route definition
`@app.route('/greet/<username>')`.
* In the `greet` function, I created a dictionary
`greeting` with a single key-value pair, where the
key is `message` and the value is a string that
includes the `username` parameter using an
f-string.
* I returned the `greeting` dictionary using the
`jsonify` function, which will convert the
dictionary to a JSON response.
```

Now, when you send a GET request to
`/greet/<username>`, you'll receive a JSON response
like this:

```
```json
{
 "message": "Hello, <username>!"
}
```

Replace `<username>` with the actual username you  
want to greet!

2. Comments:
  - a. Added an explanation and also edited other parts of the method

Model Analysis:

- LLMs seem unable to help themselves if they see a problem they can fix, or an explanation they can include, even if the prompt doesn't include it.

---

6. **SQL Schema Design (SQL)** Write the schema for a review app with users, books, and reviews:

```
-- TODO: Design schema with appropriate keys and constraints
-- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id,
rating)
```

- k. Model: **OpenAI GPT - 4.1**
  - i. Prompt Strategy 1: Zero-Shot
    1. Output:

```

```sql
-- 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 INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    book_id INTEGER NOT NULL REFERENCES books(id) ON DELETE CASCADE,
    rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
    UNIQUE (user_id, book_id) -- Each user reviews a book only once
);
```

```

2. Comments:

- a. Does not contain a column that will store a text-based review.

ii. Prompt Strategy 2: Role-Playing

1. Output:



```

```sql
-- 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 INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
    review_text TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, book_id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```

```

## 2. Comments:

- a. Compared to the zero shot prompting, this approach yielded a more thorough schema, seen through its use of the foreign keys. This makes sense, since user ids and book ids will be contained in another parent table.

## I. Model: **Llama 3 70b Instruct**

- i. Prompt Strategy 1: Zero Shot
  1. Output:

```

Here is a possible schema for a review app with users, books, and reviews:
...
CREATE TABLE users (
 id SERIAL PRIMARY KEY,
 name VARCHAR(50) NOT NULL
);

CREATE TABLE books (
 id SERIAL PRIMARY KEY,
 title VARCHAR(200) 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 BETWEEN 1 AND 5),
 review_text TEXT,
 created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
 FOREIGN KEY (user_id) REFERENCES users(id),
 FOREIGN KEY (book_id) REFERENCES books(id)
);
...

```

2. Comments:

- a. Surprisingly more thorough than the Role-Playing version of the prompt.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```

Assistant: Here is the schema for a review app with users, books, and reviews:
...
CREATE TABLE users (
 id INT PRIMARY KEY,
 name VARCHAR(255) NOT NULL
);

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

CREATE TABLE reviews (
 id INT PRIMARY KEY,
 user_id INT NOT NULL,
 book_id INT NOT NULL,
 rating DECIMAL(3,2) CHECK (rating >= 0 AND rating <= 5),
 FOREIGN KEY (user_id) REFERENCES users(id),
 FOREIGN KEY (book_id) REFERENCES books(id)
);
...

```

2. Comments:

- a. Does not contain a column that will store a text-based review.

Model Analysis:

- These results are interesting. Each prompt strategy / model combination came up with a different schema for the “reviews” table. Importantly, Llama w/ role playing and ChatGPT with zero-shot did not include a text-based column that would allow for a detailed review to be made.

---

7. Null Dereference Detection (Java) Identify any null dereference risk:

```
public int getLength(String s) {
 return s.length(); // What if s is null?
}
```

m. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
Yes, **there is a null dereference risk** in the
code you provided.
```

```
Here's the code:
```

```
```java  
public int getLength(String s) {  
    return s.length(); // What if s is null?  
}  
```
```

```
Explanation:
```

```
- If `s` is `null` and you call `s.length()`, Java
will throw a `NullPointerException`.
```

```
- There is no null check before dereferencing `s`.
```

```
How to fix:
```

```
Add a null check to handle the case where `s` is
`null`. For example:
```

```
```java  
public int getLength(String s) {  
    if (s == null) {  
        return 0; // or throw an exception, or  
        handle as needed  
    }  
    return s.length();  
}
```

```

    }
    return s.length();
}
```

```

**\*\*Summary:\*\***

The original code has a null dereference risk because it assumes `s` is never null. Always check for null before dereferencing an object in Java

## 2. Comments:

- a. The model identifies and fixes the bug, while explaining what to do in the future

## ii. Prompt Strategy 2: Chain of Thought

### 1. Output:

Let's analyze your code for **\*\*null dereference risk\*\***, step by step.

### Code

```

```java
public int getLength(String s) {
    return s.length(); // What if s is null?
}
```

```

---

## Step-by-Step Analysis

### 1. **\*\*Parameter Declaration\*\***

- The method takes a single parameter: `String s`.

### 2. **\*\*Method Body\*\***

- It immediately returns `s.length();`

### 3. **\*\*What does `s.length()` do?\*\***

- If `s` is a non-null String, `length()` returns the number of characters in that String.

- If `s` is **\*\*null\*\***, calling any method (including `length()`) on it will throw a `NullPointerException`.

---

### ### 4. **\*\*Null Dereference Risk\*\***

- **\*\*Dereferencing\*\***: ``s.length()`` is a dereference of ``s``.

- **\*\*Risk\*\***: If the caller of ``getLength`` passes ``null`` for ``s``, the method will throw a ``NullPointerException`` at runtime.

---

#### ## Example

```
```java
getLength("Hello"); // returns 5, works fine
getLength(null);    // throws NullPointerException
```
```

---

#### ## Best Practice

**\*\*To avoid the risk\*\***, you should check for ``null`` before dereferencing:

```
```java
public int getLength(String s) {
    if (s == null) {
        return 0; // or throw an exception, or
        handle as appropriate
    }
    return s.length();
}
```
```

---

#### ## **\*\*Summary Table\*\***

| Step                           | What happens                            |
|--------------------------------|-----------------------------------------|
| Null Risk?                     |                                         |
| -----                          | -----                                   |
| -----                          |                                         |
| Method takes <code>`s`</code>  | Could be null or not                    |
|                                |                                         |
| Call <code>`s.length()`</code> | Dereference ( <code>`s.length`</code> ) |
| <b>**Risk if s is null**</b>   |                                         |

| Return result | Return int (length) |
|---------------|---------------------|
| N/A           |                     |

---

## \*\*Conclusion\*\*

**\*\*Yes, there is a null dereference risk.\*\***  
 If `s` is `null`, the code will throw a  
 `NullPointerException` at `s.length()`.  
**\*\*Always check for null before dereferencing  
 objects.\*\***

## 2. Comments:

- a. Output is very long to explain something very simple

## n. Model: **Llama 3 70b Instruct**

### i. Prompt Strategy 1: Zero Shot

#### 1. Output:

There is a null dereference risk in this code. If the input `s` is `null`, calling `s.length()` will throw a `NullPointerException` because you're trying to access the `length` method on a null object.

To mitigate this risk, you can add a null check before calling `length()`:

```
```
```

```
public int getLength(String s) {
    return s != null ? s.length() : 0; // or some
    other default value
}
```

```
```
```

Alternatively, you can throw a  
 `NullPointerException` explicitly if `s` is null,  
 depending on your requirements:

```
```
```

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

```
```
```

Either way, it's essential to handle the possibility of `s` being null to avoid a  
 `NullPointerException` at runtime.

## 2. Comments:

- a. LLaMA has a shorter explanation but provides two possible solutions
- ii. Prompt Strategy 2: Chain of Thought

1. Output:

Let's break down the code step by step to identify the null deference risk:

```
1. `public int getLength(String s) {`:
 * The method takes a `String` parameter `s`.
 * The method returns an `int` value.
2. `return s.length ();`:
 * The method tries to call the `length()`
method on the `s` object.
 * This is where the null deference risk comes
in:
 + If `s` is `null`, calling `length()`
on it will throw a `NullPointerException` (NPE).
```

The risk is that if the caller of this method passes `null` as an argument, the method will attempt to call `length()` on a null reference, resulting in an NPE.

To mitigate this risk, you can add a null check before calling `length()`:

```
```\npublic int getLength(String s) {\n    if (s == null) {\n        return 0; // or throw an exception,\ndepending on your requirements\n    }\n    return s.length();\n}\n```\n
```

By doing so, you ensure that the method will not throw an NPE if `s` is `null`.

2. Comments:

- a. Doesn't spend much time identifying the bug, just fixing it

Model Analysis:

- Both models are proficient at identifying the bug and providing possible solutions.
-

8. **CSV Parser Variants (Python)** Improve the parser to support quoted fields:

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

o. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Few-Shot

1. Output:

```
```python  
def parse_csv_line(line):
 fields = []
 field = ''
 in_quotes = False
 i = 0
 while i < len(line):
 char = line[i]
 if char == '"':
 if in_quotes and i + 1 < len(line) and line[i + 1] == '"':
 # Escaped quote inside quoted field
 field += '"'
 i += 1
 else:
 # Toggle in_quotes state
 in_quotes = not in_quotes
 elif char == ',' and not in_quotes:
 fields.append(field)
 field = ''
 else:
 field += char
 i += 1
 fields.append(field)
 return fields
```
```

2. Comments:

- a. Chose to use an altered version of the manual implementation as opposed to the shorter version which uses the csv module.

ii. Prompt Strategy 2: Chain-of-Thoughts

1. Output:


```

'''python
def parse_csv_line(line):
    fields = []
    field = []
    in_quotes = False
    i = 0
    while i < len(line):
        char = line[i]
        if char == '"':
            if in_quotes:
                # Check for escaped quote
                if i + 1 < len(line) and line[i+1] == '"':
                    field.append('"')
                    i += 1 # Skip the next quote
                else:
                    in_quotes = False # End of quoted field
            else:
                in_quotes = True # Begin quoted field
        elif char == ',' and not in_quotes:
            # End of field
            fields.append(''.join(field))
            field = []
        else:
            field.append(char)
        i += 1
    # Append the last field
    fields.append(''.join(field))
    return fields
'''

```

2. Comments:

- a. Wrote in comments for clarity and used a list for storage (as opposed to a string).

p. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Few-Shot

1. Output:

```

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

    for char in line:
        if char == '"':
            if in_quotes and len(field) > 0 and field[-1] == '"':
                field = field[:-1]
            else:
                in_quotes = not in_quotes
        elif char == ',' and not in_quotes:
            fields.append(field)
            field = ''
        else:
            field += char

    fields.append(field)
    return fields
'''

```

2. Comments:

- a. No significant changes from other implementations.

ii. Prompt Strategy 2: Chain-of-Thoughts

1. Output:

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

 for char in line:
 if char == '"':
 in_quote = not in_quote
 elif char == ',' and not in_quote:
 fields.append(''.join(current_field))
 current_field = []
 else:
 current_field.append(char)

 if current_field:
 fields.append(''.join(current_field))

 return fields
```
```

2. Comments:

- a. Uses a list as opposed to storing things in the string.

Model Analysis:

- With the few-shot prompts, both models surprisingly created longer, manual implementations - even when one of the examples given to the model was simple and used the CSV module. Llama also preferred using a while-loop implementation as opposed to ChatGPT using a for-loop. Also, with both few-shot prompts, both models switched to using a list for storage (as opposed to zero-shot's string usage).

9. **Data Class to API Conversion (Kotlin)** Convert the data class to a REST API using Ktor:

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

// TODO: Create GET and POST endpoints using Ktor
```

q. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
// 2. In-memory "database"
val products = mutableListOf<Product>()

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(io.ktor.features.ContentNegotiation) {
            io.ktor.serialization.json()
        }
        routing {
            // GET /products
            get("/products") {
                call.respond(products)
            }

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

2. Comments:

- a. Code interspersed with explanations for the user

ii. Prompt Strategy 2: Self Consistency

1. Output:

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

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

    val products = ConcurrentHashMap<Int, Product>()

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

        get("/products/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            val product = products[id]
            if (product != null) {
                call.respond(product)
            } else {
                call.respond(HttpStatusCode.NotFound, "Product not found")
            }
        }

        post("/products") {
            val product = call.receive<Product>()
            if (product.price < 0.0) {
                call.respond(HttpStatusCode.BadRequest, "Price must be non-negative")
                return@post
            }
            if (products.containsKey(product.id)) {
                call.respond(HttpStatusCode.BadRequest, "Product with this ID already exists")
                return@post
            }
            products[product.id] = product
            call.respond(HttpStatusCode.Created, product)
        }
    }
}
...
```

2. Comments:

- a. Uses a module rather than including a main function

r. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            gson()
        }
        routing {
            get("/products") {
                val products = listOf(
                    Product(1, "Product 1", 10.99),
                    Product(2, "Product 2", 9.99),
                    Product(3, "Product 3", 12.99)
                )
                call.respond(products)
            }
            post("/products") {
                val product = call.receive<Product>()
                // Save the product to a database or perform some other action
                call.respond(product)
            }
        }
    }.start(wait = true)
}
```

2. Comments:

- a. Larger blocks of code between explanations

ii. Prompt Strategy 2: Self Consistency

1. Output:

```

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

interface ProductService {
    fun getAllProducts(): List<Product>
    fun createProduct(product: Product)
}

class ProductServiceImpl : ProductService {
    val products = mutableListOf<Product>()

    override fun getAllProducts(): List<Product> = products

    override fun createProduct(product: Product) {
        products.add(product)
    }
}

fun Application.configureRouting() {
    val productService = ProductServiceImpl()

    install(ContentNegotiation) {
        json()
    }

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

        post("/products") {
            val product = call.receive<Product>()
            productService.createProduct(product)
            call.respond(HttpStatusCode.Created)
        }
    }
}

```

2. Comments:
 - a. No more main method when calling the aggregate

Model Analysis:

- When using self consistency, the output did not include anything about adding dependencies

10. **Function Summarization (Python)** Write a brief summary of the function:

```

def reverse_words(sentence):
    return ' '.join(sentence.split()[::-1])

```

- s. Model: **OpenAI GPT - 4.1**
 - i. Prompt Strategy 1: Chain-of-Thoughts
 1. Output:

```

### Step-by-step Explanation:

1. **Input:**
   The function takes a single argument called `sentence`, which is expected to be a string.

2. **Splitting the Sentence:**
   `sentence.split()` splits the input string into a list of words, using whitespace as the delimiter.
   For example, `"Hello world"` becomes `['Hello', 'world']`.

3. **Reversing the List:**
   `[::-1]` reverses the order of the list created by `split()`.
   So, `['Hello', 'world']` becomes `['world', 'Hello']`.

4. **Joining the Words:**
   `' '.join(...)` joins the elements of the reversed list back into a single string, with a space between each word.
   The result is `"world Hello"`.

5. **Return:**
   The function returns the new string with the word order reversed.

---

```

2. Comments:

- a. For this task, breaking it down step by step would only be necessary if you want the full understanding of each segment of Python code. For a brief summary, this might be too much.

ii. Prompt Strategy 2: Role-Playing

1. Output:

- a. The function `reverse_words` takes a sentence as input and returns a new string where the order of the words is reversed. It does this by splitting the sentence into words, reversing the list of words, and then joining them back into a string with spaces.

2. Comments:

- a. Simple and effective. Much more concise and to the point than the previous prompting (obviously).

t. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Chain-of-Thoughts

1. Output:

a.

1. `Sentence.split()`: The function starts by splitting the input sentence into a list of individual words using the `split()` method. This method splits a string into a list of substrings separated by whitespace characters (spaces, tabs, etc.). For example, if the input sentence is "Hello World", this step would produce the list `["Hello", "World"]`.
2. `[::-1]`: The list of words is then sliced using the syntax `[::-1]`. This syntax tells Python to start from the end of the list and move backwards to the

beginning, stepping backwards by 1 element each time. This effectively reverses the order of the list. In our example, this would produce the list

```
`["World", "Hello"]`
```

3. ``.join(...)``: The reversed list of words is then joined back into a single string using the ``.join()`` method. The string ``. `` (a single space) is used as the separator between words. This produces the final output sentence with the words in reverse order. In our example, this would produce the string ``. "World Hello" ``.

2. Comments:

- a. Same as with ChatGPT, this breaks down the Python syntax well assuming you want a very detailed response.

ii. Prompt Strategy 2: Role-Playing

1. Output:

- a. The ``.reverse_words`` function takes a sentence as input and returns the sentence with the order of its words reversed.

2. Comments:

- a. Simple and effective, similar to ChatGPT's response. Llama also accomplished the task with much fewer words.

Model Analysis:

- As a summary task, the problem presented to both models is relatively straightforward. Llama seemed to produce more concise answers compared to ChatGPT's more descriptive breakdowns.

11. **Prompt from Code Comments (Python)** Write a prompt that could generate the code:

```
# 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
```

u. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
Sure! Here's a prompt you could give to a large
language model to produce the code you provided:
```

```
---
```

```
**Prompt:**
```

```
Write a Python function called `is_prime` that takes an integer `n` and returns `True` if `n` is a prime number and `False` otherwise. A prime number is greater than 1 and has no divisors other than 1 and itself. Optimize the function by only checking for divisibility up to the square root of `n`.
```

```
---
```

```
This prompt provides clear instructions for the model to generate the code you shared.
```

2. Comments:

- a. Provide a prompt that will generate code to strictly match the original code

ii. Prompt Strategy 2: Chain of Thought

1. Output:

```
Certainly! Here's a step-by-step breakdown of how you could prompt a large language model to generate the g

---

**Prompt:**

Write a Python function to determine if a given number is prime. Follow these steps:

1. Define a function called is_prime that takes a single integer parameter n.
2. If n is less than or equal to 1, return False (since numbers less than or equal to 1 are not prime).
3. Use a for loop to iterate over all integers i from 2 up to and including the square root of n (rounded d
4. Inside the loop, check if n is divisible by i (i.e., n % i == 0). If it is, return False (since n is not
5. If the loop completes without finding any divisors, return True (since n is prime).

Format your code clearly with proper indentation and comments.

---

This prompt should guide the language model to produce the code you provided.
```

2. Comments:

- a. Provides a step-by-step prompt, which is a different interpretation than we intended, but still interesting.

v. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero Shot

1. Output:

```
Here is a prompt that could generate the given code:

"Write a Python function named 'is_prime' that takes an integer 'n' as input and returns 'True' if 'n' is a prime number, and 'False' otherwise. The function should use a efficient a
Alternatively, you could also use a more concise prompt:

"Python function to check if a number is prime"

Or, if you want to provide more guidance on the algorithm to use:

"Write a Python function to check if a number is prime using a trial division method with an optimization to only check up to the square root of the number."
```

2. Comments:

- a. Provides multiple options based on how exact you want to match the code
- ii. Prompt Strategy 2: Chain of Thought

1. Output:

```
Here's a prompt that could be fed to a large language model to produce the given code:

**Prompt:** "Write a Python function named 'is_prime' that takes an integer 'n' as input and returns 'True' if 'n' is a prime number, and 'False' otherwise. Implement the function using a simple trial division method."

Let's break it down step by step:

1. **Function signature:** "Write a Python function named 'is_prime'" - This tells the model to generate a Python function with the specified name.
2. **Function parameters:** "that takes an integer 'n' as input" - This specifies the input parameter type and name.
3. **Function purpose:** "and returns 'True' if 'n' is a prime number, and 'False' otherwise" - This describes the function's behavior and output.
4. **Implementation hint:** "Implement the function using a simple trial division method" - This gives the model a hint about the algorithm to use, which is a common approach for checking primality.

By providing this prompt, the language model should generate a Python function that meets the specified requirements, which is the given code.
```

2. Comments:

- a. LLaMa understands that the prompt should be the same but the user needs it explained step by step

Model Analysis:

- Both models had similar responses but only LLaMa understood the intention behind asking for a step-by-step explanation.

12. Fixing Factorial Bug (Python) Fix the bug when input is 0:

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

- w. Model: **OpenAI GPT - 4.1**

- i. Prompt Strategy 1: Zero-Shot

1. Output:

```
**Optional:** If you want to handle negative inputs (which are not allowed for factorial), you can add a check:
```python
def factorial(n):
 if n < 0:
 raise ValueError("Factorial is not defined for negative numbers")
 result = 1
 for i in range(1, n+1):
 result *= i
 return result
```
```

2. Comments:

- a. Gave the solution found below, as well as this “optional” version that handles negative inputs.

- ii. Prompt Strategy 2: Prompt Chaining

1. Output:

```

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

```

2. Comments:

- a. The straightforward implementation.

x. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output

```

```
def factorial(n):
 if n == 0:
 return 1
 result = 1
 for i in range(1, n + 1): # Also, the range should be up to n + 1, not n
 result *= i
 return result
```

```

2. Comments:

- a. This output got indentation wrong (under the for loop and with the first indentation), but was correct other than those mistakes.

ii. Prompt Strategy 2: Prompt Chaining

1. Output:

```

def factorial(n):
    """
    Calculate the factorial of a given integer n.

    Args:
        n (int): The input integer.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is a non-positive integer (except 0).
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    if n == 0:
        return 1
    result = 1
    for i in range(1, n+1):
        result *= i
    return result

```

2. Comments:

- a. This implementation went all-out, accounting for multiple other potential issues.

Model Analysis:

- There seemed to be no correlation between the models and the types of prompts used. For ChatGPT's prompt chaining, it gave a simple, straightforward approach. For Llama's prompt chaining, it gave a more complex approach that covered more cases. The zero-shots saw the same phenomena, with the models reversed in that case.

13. **Linked List Node Deletion (C)** Implement node deletion by value:

```

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

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

```

- y. Model: **OpenAI GPT - 4.1**
 - i. Prompt Strategy 1: Zero Shot

1. Output:

```
Certainly! Here's how you can implement deletion of the first occurrence of a node with a given value (`key`) from a singly linked list in C.

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

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

void deleteNode(struct Node** head, int key) {
 struct Node* temp = *head;
 struct Node* prev = NULL;

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

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

 // If key wasn't present in linked list
 if (temp == NULL)
 return;

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

```

## 2. Comments:

- a. Provides a long explanation that cannot be fit here
- ii. Prompt Strategy 2: Few Shot
  - 1. Output:

```

First chain:
Understood! I have reviewed both functions and their logic:

- **deleteAllByValue**: Deletes all nodes in a singly linked list whose `data` matches `value`.
- **deleteByValue**: Deletes only the first occurrence of a node with `data == value`.

If you have any questions or need further explanation or modifications, let me know!

Second chain:
Certainly! Here is a complete implementation of `deleteNode` that deletes the first occurrence of `key` from the linked list, similar to your `deleteByValue` example:
'''c
struct Node {
 int data;
 struct Node* next;
};

void deleteNode(struct Node** head, int key) {
 struct Node *temp = *head, *prev = NULL;

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

 // Search for the key to be deleted, keep track of the previous node
 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);
}
'''

How it works:
- Checks if the head node itself holds the key and deletes it if so.
- Otherwise, traverses the list to find the key, keeping track of the previous node.
- If found, unlinks and deletes the node.

Let me know if you need the version for deleting all occurrences!

```

## 2. Comments:

- a. The model used the assumptions of the last implementation it saw

## z. Model: **Llama 3 70b Instruct**

### i. Prompt Strategy 1: Zero Shot

## 1. Output:

```

Here is the implementation of node deletion by value:
'''c
void deleteNode(struct Node** head, int key) {
 // Check if the list is empty
 if (*head == NULL) return;

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

 // Find the node to be deleted
 struct Node* current = *head;
 while (current->next != NULL) {
 if (current->next->data == key) {
 struct Node* temp = current->next;
 current->next = current->next->next;
 free(temp);
 return;
 }
 current = current->next;
 }
}
'''

Here's an explanation of the implementation:

1. First, we check if the list is empty. If it is, we simply return.
2. We then check if the node to be deleted is the head node. If it is, we update the head pointer to point to the next node, free the current head node, and return.
3. If the node to be deleted is not the head node, we iterate through the list until we find the node with the matching value.
4. When we find the node, we update the `next` pointer of the previous node to skip over the node to be deleted, and free the node to be deleted.
5. Finally, we return.

Note that this implementation assumes that the list is singly linked, and that the `next` pointer of each node points to the next node in the list.

```

## 2. Comments:

- a. Includes an assumption of the implementation that can help with debugging
- ii. Prompt Strategy 2: Few Shot
  1. Output:

```

First chain:
Understood! I have reviewed both functions:
- 'deleteAllByValue' deletes all nodes in a linked list that match a given value.
- 'deleteByValue' deletes only the first occurrence of a node with the given value.

Both functions handle the head node as a special case and use a combination of 'prev' and 'temp' pointers to traverse and delete nodes. The functions use 'free()' to deallocate memory of the deleted nodes.

I'm ready to answer any questions or perform any task related to these functions!

Second chain:
Here is the implementation of the 'deleteNode' method to delete nodes by value:

'''c
void deleteNode(struct Node** head, int key) {
 struct Node *temp = *head, *prev = NULL;

 // If head node holds the key, change head
 if (temp != NULL && temp->data == key) {
 *head = temp->next;
 free(temp);
 return;
 }

 // Search for the key
 while (temp != NULL && temp->data != key) {
 prev = temp;
 temp = temp->next;
 }

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

 // Node temp->next is the node to be deleted
 // Unlink the node from linked list
 prev->next = temp->next;

 free(temp); // Free memory
}
...
'''

This method deletes the first occurrence of the node with the given 'key' in the linked list. It handles the head node as a special case and uses a combination of 'prev' and 'temp' pointers to traverse and delete the

```

2. Comments:
  - a. The LLaMa model uses a similar approach, likely do to the context given

#### Model Analysis:

- Few shot prompting can constrain the types of implementations the model will output, which can allow for users to be very deliberate in the type of code they produce.

#### 14. Recursive Function Completion (Python) Complete the recursive function for Fibonacci:

```

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

```

- aa. Model: **OpenAI GPT - 4.1**
  - i. Prompt Strategy 1: Zero-Shot
    1. Output:

```

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

```

2. Comments:

- a. Correct implementation; simple and to the point.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```

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

```

2. Comments:

- a. Same exact implementation as Zero-Shot.

bb. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```

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

```

2. Comments:

- a. Same as ChatGPT output for zero-shot.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```

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

```

2. Comments:
  - a. Same as the previous solutions.

Model Analysis:

- The model-prompt combinations differ slightly in how they choose to explain the solution, but each provides the same implementation. This is unsurprising, since this problem is talked about online frequently due to its common appearance as an introduction to recursive functions.

#### 15. Constructor Completion (Python) Complete the class constructor:

```

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

```

- cc. Model: **OpenAI GPT - 4.1**
  - i. Prompt Strategy 1: Zero Shot
    1. Output:

```

Certainly! Here is a completed version of the constructor that takes `name`, `age`, and an optional `email` parameter:

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

Usage example:

```python
p1 = Person("Alice", 30)
p2 = Person("Bob", 25, "bob@example.com")
```

- `email` is optional; if not provided, it will be `None`.

```

2. Comments:
  - a. Gives a good example of how the constructor could be used
- ii. Prompt Strategy 2: Self Consistency



## 1. Output:

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

Classification 2
```python
class Person:
    def __init__(self, name:str, age:int, email:str=None):
        self.name = name
        self.age = age
        self.email = email
...

## **Classification 3**
```python
class Person:
 def __init__(self, name, age, email=''):
 self.name = name
 self.age = age
 self.email = email
...

Aggregate Constructor

Taking the aggregate of these three, we see:

- All versions have `name`, `age`, and an optional `email`.
- Some use type hints, some don't.
- Default values for `email` are either `None` or `''`.

Combining best practices, the aggregate version will:
- Use type hints (as in #2).
- Make `email` optional and default to `None` (as in #1 and #2, which is generally preferred over empty string).

Final Aggregate Version
```python
class Person:
    def __init__(self, name: str, age: int, email: str = None):
        self.name = name
        self.age = age
        self.email = email
...

**Summary:**
This constructor takes `name` and `age` as required arguments, and `email` as an optional argument defaulting to `None`, with type hints for clarity.
```

2. Comments:

- a. Using self consistency on a simple problem just muddies the waters

dd. Model: Llama 3 70b Instruct

i. Prompt Strategy 1: Zero Shot

1. Output:

```
Here is the completed class constructor:
...
class Person:
    def __init__(self, name, age, email=None):
        self.name = name
        self.age = age
        self.email = email
...

In this constructor, `name` and `age` are required parameters, and `email` is an optional parameter with a default value of `None`. This means that when creating a new `Person` object, you must provide a `name` and an `age`.
```

2. Comments:

- a. No example, but uses the same exact implementation
- ii. Prompt Strategy 2: Self Consistency

1. Output:

```

First chain:
Here are three different classifications for the 'Person' class constructor:

**Classification 1**
...
class Person:
    def __init__(self, name: str, age: int, email: str = None):
        self.name = name
        self.age = age
        self.email = email
...

**Classification 2**
...
class Person:
    def __init__(self, name: str, age: int, email: str = "not_provided"):
        self.name = name
        self.age = age
        self.email = email
...

**Classification 3**
...
class Person:
    def __init__(self, name: str, age: int, email: str = None):
        self.full_name = name
        self.age_in_years = age
        self.contact_email = email
...

Now, let's take the aggregate of these classifications:

* All three classifications agree on the 'name' and 'age' parameters, with 'name' being a string and 'age' being an integer.
* Two out of three classifications use 'email' as an optional parameter with a default value of 'None'. The third classification uses 'email' with a default value of "not_provided".
* The attribute names used in the classifications are different, but they all convey the same information.

Taking these similarities and differences into account, here's a revised constructor that combines the best of all three:
...
class Person:
    def __init__(self, name: str, age: int, email: str = None):
        self.full_name = name
        self.age = age
        self.email = email
...

This constructor uses 'full_name' to emphasize that the 'name' parameter should be a full name, and keeps the 'age' attribute simple. The 'email' attribute is optional, with a default value of 'None' to indicate that it

```

2. Comments:

a. Similar issue with self consistency

Model Analysis:

- For simple problems, self consistency may not be the best choice, because it will rewrite simple implementations that are already good.

16. Binary Search Completion (Java) Complete the binary search implementation:

```

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;
}

```

ee. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Few-Shot

1. Output:

```

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

 if (arr[mid] == target) {
 return mid; // Target found
 } else if (arr[mid] < target) {
 left = mid + 1; // Search in the right half
 } else {
 right = mid - 1; // Search in the left half
 }
 }
 return -1; // Target not found
}
```

```

2. Comments:

- a. Many comments on the code provided compared to the other prompts.

ii. Prompt Strategy 2: Self-Consistency

1. Output:

```

```java
public int binarySearch(int[] arr, int target) {
 if (arr == null || arr.length == 0) return -1; // Robustness
 int left = 0, right = arr.length - 1;
 while (left <= right) {
 int mid = left + ((right - left) >> 1); // Avoid overflow
 if (arr[mid] == target) {
 return mid;
 } else if (arr[mid] < target) {
 left = mid + 1;
 } else {
 right = mid - 1;
 }
 }
 return -1;
}
```

```

2. Comments:

- a. Handled overflow differently than other responses.

ff. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Few-Shot

1. Output:

```

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

 if (arr[mid] == target) {
 return mid; // Target found
 } else if (arr[mid] < target) {
 left = mid + 1; // Search in the right half
 } else {
 right = mid - 1; // Search in the left half
 }
 }
 return -1; // Target not found
}
```

```

2. Comments:

- a. The normal response. Nothing of importance to note.

ii. Prompt Strategy 2: Self-Consistency

1. Output:

```

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

```

2. Comments:

- a. Same as the previous Llama prompt.

Model Analysis:

- The model-prompt combinations provided relatively similar solutions. The ChatGPT's self-consistency solution handled overflow differently from all other combinations (which were similar to themselves). Other than this, there is nothing major to note. There were also small differences in where the models chose to comment.
-

17. **Self-Consistency Bug Fixing (C++)** Resolve inconsistency between function name and logic:

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

gg. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero Shot

1. Output:

Certainly! The inconsistency is that the function is named `isOdd`, but the logic checks if the number is **even** (`x % 2 == 0`). To resolve this, you should either:

1. **Change the function name to match the logic:**

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

or

2. **Change the logic to match the name:**

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

Choose one of the above based on what you want the function to do.

2. Comments:

a. Provides multiple options to resolve the issue

ii. Prompt Strategy 2: Role Playing

1. Output:

Assistant: You've correctly spotted the inconsistency: The function is named `isOdd`, but its logic checks if `x` is even.

To fix this, either:

1. Fix the function name to reflect the logic:

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

2. Fix the logic to match the original name:

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

Pick one approach depending on what you want (`isOdd` or `isEven`).

For `isOdd`, this is the correct implementation:

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

Now, it returns `true` when `x` is odd, matching the function name.

2. Comments:

a. Provides more explanation when roleplaying

hh. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero Shot

1. Output:

```

The function name 'isOdd' suggests that it should return 'true' if the input 'x' is odd, but the logic 'x % 2 == 0' returns 'true' if 'x' is even. To resolve this inconsistency, you can simply change the function name.

**Option 1: Change function name to 'isEven'**
```c
bool isEven(int x) {
 return x % 2 == 0; // Correct logic for even numbers
}
...
Option 2: Flip the logic to 'x % 2 != 0'
```c
bool isOdd(int x) {
    return x % 2 != 0; // Correct logic for odd numbers
}
...
Either of these changes will resolve the inconsistency between the function name and the logic.

```

2. Comments:

- a. Provides an explanation of alongside the two implementations

ii. Prompt Strategy 2: Role Playing

1. Output:

```

Assistant: The method 'isOdd' is supposed to return 'true' if the number is odd, but the logic inside the method is checking if the number is even. Here's the corrected code:

```c++
bool isOdd(int x) {
 return x % 2 != 0; // Return true if x is odd
}
...

Alternatively, if you want to keep the original logic but make the method name consistent, you could rename the method to 'isEven':

```c++
bool isEven(int x) {
    return x % 2 == 0; // Return true if x is even
}
...

```

2. Comments:

- a. The LLaMa model explains less as am coding assistant

Model Analysis:

- Role playing can be a good way to get explanations and better implementations from a large language model.

18. Prompt Chaining: Bug Identification → Fix (JavaScript) Identify and fix the bug:

```

function isEven(n) {
    return n % 2; // Returns 1 or 0, not true/false
}

```

ii. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Prompt-Chaining

1. Output:

```

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

```

## 2. Comments:

- a. Straightforward response.

## ii. Prompt Strategy 2: Zero-Shot

### 1. Output:

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

2. Comments:

a. Same as the previous Chat-GPT prompt.

jj. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Prompt-Chaining

1. Output:

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

2. Comments:

a. Same as both ChatGPT responses.

ii. Prompt Strategy 2: Zero-Shot

1. Output:

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

2. Comments:

a. Added comments for clarity, but they are not necessary.

Model Analysis:

- Since this solution is an easy one to implement, all model-prompt combinations came to the same conclusion when fixing the bug. Nothing really to note here. Raw output was different in how each explained things, but the outputs were the exact same.

---

19. **Summary Decomposition (C++)** Decompose the high-level comment/summary into logical steps:

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

kk. Model: **OpenAI GPT - 4.1**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```

Decomposed into logical steps:

1. **Validate the input:**
 Check if the input `x` is negative.

2. **Handle invalid input:**
 If `x` is negative, return `-1` to indicate an error or invalid input.

3. **Calculate the square:**
 If `x` is non-negative, compute the square of `x` (`x * x`).

4. **Return the result:**
 Return the calculated square as the final result.

```

2. Comments:

- a. Correct implementation, nothing important to note. Accurately describes the comment, and decomposes it correctly.

ii. Prompt Strategy 2: Role-Playing

1. Output:

```

Assistant: Certainly! Here's the comment decomposed into logical steps:

1. **Validates the input:** Checks if the input x is negative.
2. **Handles invalid input:** If x is negative, returns -1 to indicate invalid input.
3. **Calculates the square:** If x is non-negative, calculates x squared.
4. **Returns the result:** Returns the squared value as the result.

```

2. Comments:

- a. Same result as previous ChatGPT prompt with slight deviations.

II. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Zero-Shot

1. Output:

```

Step 1: Validate the input

* Check if the input `x` is less than 0.
* If `x` is less than 0, return an error code `-1` immediately.

Step 2: Calculate the square

* If `x` is not less than 0, proceed to calculate the square of `x`.
* The square of `x` is calculated as `x * x`.

Step 3: Return the result

* Return the calculated square of `x` as the result of the function.

```

2. Comments:

- a. Correct results. Less steps than ChatGPT's prompts.

ii. Prompt Strategy 2: Role-Playing



### 1. Output:

```
Step 1: Validate the input

* Check if the input `x` is less than 0.
* If `x` is less than 0, return -1 (indicating an invalid input).

Step 2: Calculate the square

* If `x` is valid (i.e., not less than 0), calculate the square of `x` by multiplying it with itself (`x * x`).

Step 3: Return the result

* Return the calculated square of `x` as the result of the `process` function.
```

### 2. Comments:

- a. Same results as the other Llama prompt, just worded slightly different.

### Model Analysis:

- Each model-prompt combination provided an accurate description of the comment at the top of the code snippet. They varied slightly in terms of wording, and ChatGPT preferred a 4-point response as opposed to Llama's 3-point response, but the results ultimately achieved the same things.

---

## 20. Purpose Inference → Completion (Python) Complete the function based on intent:

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

mm. Model: **OpenAI GPT - 4.1**

### i. Prompt Strategy 1: Role-Playing

#### 1. Output:

```
```python
def calculate_average(scores):
    total = 0
    for score in scores:
        total += score
    if len(scores) == 0:
        return 0 # or raise an Exception if preferred
    return total / len(scores)
```
```

#### 2. Comments:

- a. This model contains the straightforward approach, but there is a slight error in the layout. Instead of checking if scores are empty first, it checks after scores have been iterated through. This doesn't really make sense.

### ii. Prompt Strategy 2: Chain-of-Thoughts

1. Output:

```
```python
def calculate_average(scores):
    total = 0
    if len(scores) == 0:
        return 0 # or could raise ValueError("Empty list")
    for score in scores:
        total += score
    average = total / len(scores)
    return average
```

Alternate implementation using built-in functions:

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

2. Comments:

- a. This prompt corrected the issue found in the previous prompt. It also provided an alternative implementation that uses built-in functions.

nn. Model: **Llama 3 70b Instruct**

i. Prompt Strategy 1: Role-Playing

1. Output:

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

2. Comments:

- a. The standard / basic implementation. Nothing to comment.

ii. Prompt Strategy 2: Chain-of-Thoughts

1. Output:

```
```
def calculate_average(scores):
    if not scores:
        return 0 # or some other default value
    total = 0
    for score in scores:
        total += score
    average = total / len(scores)
    return average
```
```

2. Comments:

- a. The basic implementation, with the check first to make sure scores is not empty (it does this in a different way using “if not scores”)

Model Analysis:

- All of the models implemented essentially the same answer, with one implementation being slightly incorrect. Other than that, there were slight syntax deviations in how each model-prompt combination decided to write the solution. The outlier here would definitely be ChatGPT’s Chain-of-thoughts implementation, which provided an alternative solution which used built-in functions.
-