Task 1: (Max tokens: 1024, Temperature: 0.7)

Zero-shot:

Prompt:

“Summarize the functionality of the following Java method, returning only a summary:

```
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 ;
}”
```

GPT 4.1 (Model 1):

```
This method takes a list of strings and returns a map showing how many times
each word appears in the list.
```

Codestral-2501 (Model 2):

```
The method `countWordFrequency` takes a list of strings (`words`) and returns a
map where the keys are the unique words from the list and the values are their
respective frequencies (counts) in the list.
```

Persona Prompting:

Prompt(s):

"You are a computer science professor. Please help me summarize the functionality of the following Java method in simple terms:

```
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 ;
}
```

Return only a summary."

GPT 4.1 (Model 1):

```
This method takes a list of words and returns a map showing how many times each
word appears in the list.
```

Codestral-2501 (Model 2):

This Java method, `countWordFrequency`, takes a list of words and returns a map that counts how many times each word appears in the list. It iterates through the list, updating the count for each word in a hash map. If a word is not already in the map, it initializes its count to 0 before incrementing.

Task 2: (Max tokens: 800, Temperature: 1.0)

Few-Shot:
Prompt:
“Identify and fix the bug in the following Python code without a follow-up summary:

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

Use the following code and output as an example:

```
python
def multiply_range(start, end) :
  total = 1
  for i in range(start, end) :
    total *= i
  return total
```

This method contains an off-by-one error, as the range() function only includes values from 'start' to 'end - 1'. To include the 'end' index in the for loop, the range arguments must be changed to (start, end + 1).”

GPT 4.1 (Model 1):

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

Codestral-2501 (Model 2):

To fix the bug in the sum_range function, you need to change the range() arguments to include the end index. Here is the corrected code:

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

Chain of Thought:

Prompt:

“Identify and fix the bug in the following Python code step-by-step:

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

GPT 4.1 (Model 1):

Step 1:

Examine the function definition and logic:

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

The loop uses range(start, end), which iterates from start up to but not including end.

Step 2:

Understand the intended behavior—probably that both start and end (inclusive) should be summed.

Example:

```
sum_range(1, 3)
```

Expected: 1 + 2 + 3 = 6

But actual: sums 1 + 2 = 3

Step 3:

To include end, range's stop argument should be end + 1.

Step 4:

Update the function:

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

Now sum_range(1, 3) returns 6, as expected.

Codestral-2501 (Model 2):

Let's fix the bug in the sum_range function step-by-step.

1. **Explanation of the bug:** The current implementation of sum_range uses the range(start, end) function which only includes values from start to end-1. This means the value end itself is not included in the summation.
2. **Objective:** Modify the function to include the end value in the total.
3. **Correction strategy:** Change the range(start, end) to range(start, end + 1) so that the for loop iterates from start to end inclusive.

Here's the corrected code:

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

This modification ensures that the end value is included in the summation.

Task 3: (Max tokens: 800, Temperature: 0.8)

Zero-Shot:

Prompt:

“Classify the bug in the following C++ code and add comments to clarify where the bug is happening. Do not return a summary:

```
c++
int* getArray(int size) {
        int arr [size];
        return arr;
}”
```

GPT 4.1 (Model 1):

```
int* getArray(int size) {
    int arr[size]; // BUG: 'arr' is a local variable with automatic storage duration (stack-allocated).
    return arr;    // BUG: Returning a pointer to local variable 'arr' which will be destroyed after the function returns.
}
// The function returns a pointer to a local stack variable, leading to undefined behavior.
```

Codestral-2501 (Model 2):

```
int* getArray(int size) {  // This function is intended to return a pointer to an array of integers of the given size.
    int arr[size];  // This declares an array of integers with a size specified by the input parameter.
    return arr;  // BUG: This line returns a pointer to a local variable. The array 'arr' goes out of scope when the function returns, making the returned pointer invalid.
}
```

In this code, the issue lies within the function getArray. The array arr declared within the function is a local variable, which means it only exists for the duration of the function call. Once the function returns, the memory allocated for arr is freed, and the pointer returned by the function becomes invalid. Therefore, using the returned pointer in any way results in undefined behavior.

Self-Consistency:

Prompt:

“Classify the bug in the following C++ code in one sentence:

```
c++
int* getArray(int size) {
        int arr [size];
        return arr;
}”
```

GPT 4.1 (Model 1):

Output 1:

This code returns a pointer to a local stack-allocated array, resulting in undefined behavior due to dangling pointer.

Output 2:

This code has a lifetime bug because it returns a pointer to a local variable that goes out of scope when the function ends, resulting in a dangling pointer.

Output 3:

This code has a lifetime error since it returns a pointer to a local stack-allocated array that is destroyed when the function exits, resulting in undefined behavior.

Codestral-2501 (Model 2):

Output 1:

The bug is that the function getArray returns a pointer to a local variable arr, which goes out of scope when the function returns, leading to undefined behavior.

Output 2:

The bug is that the function returns a pointer to a local array declared with variable-length array (VLA), which is not valid C++ as it doesn't support VLAs.

Output 3:

The bug is that the function getArray returns a pointer to a local array declared as a variable-length array (VLA), which is undefined behavior in C++.

<u>Task 4:</u> (Max tokens: 800, Temperature: 0.8)

Persona Prompting:

Prompt(s):

“You are a cybersecurity expert creating an email service. Complete the following Python function using regex to validate basic email addresses. Do not return an explanation:
python
def is_valid_email(email):
    #TODO: Complete using regex
    pass”

GPT 4.1 (Model 1):

```
import re

def is_valid_email(email):
    return re.fullmatch(r"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}", email)
is not None
```

Codestral-2501 (Model 2):

```
import re

def is_valid_email(email):
    # Define the regex pattern for a valid email address
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    # Use the re.match function to check if the email matches the pattern
    return re.match(pattern, email) is not None
```

Chain of Thought

Prompt:

“Complete the following Python function using regex to validate basic email addresses. Provide a concise explanation step-by-step:
python
def is_valid_email(email):
    #TODO: Complete using regex
    pass”

GPT 4.1 (Model 1):

Certainly! Here is a concise step-by-step solution with code and explanation:

```
import re

def is_valid_email(email):
    # 1. Define a regex pattern for a basic email:
    #    - Local part: one or more word chars, dots or hyphens
    #    - '@' symbol
    #    - Domain part: one or more word chars, dots or hyphens
    #    - '.' followed by at least two letters (TLD)
    pattern = r'^[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}$'

    # 2. Use re.match() to check if the pattern matches the whole string
    return re.match(pattern, email) is not None
```

Concise step-by-step explanation:

1. Import re: Required for regular expressions.
2. Define pattern: Matches basic email formats: local-part@domain.tld
3. Call re.match(): Checks for a match at the start of the string.
4. Return Boolean: True if the email matches the pattern, otherwise False.

Example:

```
print(is_valid_email('alice.smith@domain.com'))  # True
print(is_valid_email('invalid@address'))         # False
```

Codestral-2501 (Model 2):

Here's the completed function with concise step-by-step explanation:

```
import re

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

    # Use re.match to check if the email matches the pattern
    # re.match checks for a match only at the beginning of the string
    if re.match(pattern, email):
        return True
    else:
        return False

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

Step-by-Step Explanation:

1. Import the re module:

```
import re
```

This module provides the functionality to work with regular expressions.

2. Define the regex pattern:

```
pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
```

- `^[a-zA-Z0-9._%+-]+` asserts the username part at the start, allowing letters, digits, and special characters.
- `@[a-zA-Z0-9.-]+` matches the domain part after the '@' symbol, allowing letters, digits, dots, and hyphens.
- `\.[a-zA-Z]{2,}$` matches the top-level domain (TLD) at the end, ensuring it consists of at least two letters.

3. Use re.match to check if the email matches the pattern:

```
if re.match(pattern, email):
    return True
else:
    return False
```

- re.match(pattern, email) checks if the email string matches the pattern from the beginning to the end.
- If it matches, the function returns True; otherwise, it returns False.

This function will validate basic email addresses with a username, domain, and TLD.

Task 5: (Max tokens: 800, Temperature: 0.8)

Prompt Chaining

Prompt 1: “Create a ‘/greet/<username>‘ endpoint that returns a JSON greeting from the following Python method. Return only code:
from flask import Flask, jsonify
app = Flask(__name__)
@app.route(’/greet/<username>’)
def greet(username):
    # TODO: Return a JSON greeting
    pass”

Prompt 2: “Concisely explain the changes you made to the code above.”

GPT 4.1 (Model 1):

```
from flask import Flask, jsonify
app = Flask(__name__)

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

[PROMPT 2]

```
I replaced the placeholder comment and pass statement with a return statement that
sends a JSON response containing a greeting message using Flask's jsonify
function. The message includes the provided username.
```

Codestral-2501 (Model 2):

```
from flask import Flask, jsonify

app = Flask(__name__)

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

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

[PROMPT 2]

```
I modified the greet function to return a JSON response using Flask's jsonify
method. This method converts the dictionary passed to it into a JSON response. In
this case, I created a dictionary with a single key message containing the
greeting string.
```

Few-Shot:

Prompt: “Create a ‘/greet/‘ endpoint that returns a JSON greeting for the following Python method. The following two implementation descriptions have been provided as examples:
Example 1: I modified the greet function to return a JSON response using Flask's jsonify method. This method converts the dictionary passed to it into a JSON response. In this case, I created a dictionary with a single key message containing the greeting string.
Example 2: I replaced the placeholder comment and pass statement with a return statement that sends a JSON response containing a greeting message using Flask's jsonify function. The message includes the provided username. Return only code:

```
python
from flask import Flask, jsonify
app = Flask(__name__)
@app.route(’/greet/<username>’)
def greet(username):
        # TODO: Return a JSON greeting
        Pass”
```

GPT 4.1 (Model 1):

```
from flask import Flask, jsonify
app = Flask(__name__)

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

Codestral-2501 (Model 2):

```
from flask import Flask, jsonify
app = Flask(__name__)

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

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

Task 6: (Max tokens: 800, Temperature: 1.0)

Self-Consistency:

Prompt:

“Design a SQL schema for a review app with users, books, and reviews. Return only code:
sql
-- TODO: Design schema with appropriate keys and constraints
-- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id, rating)”

GPT 4.1 (Model 1):

Output 1:

```
-- 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 NOW(),
    CONSTRAINT fk_user
        FOREIGN KEY(user_id)
        REFERENCES users(id)
        ON DELETE CASCADE,
    CONSTRAINT fk_book
        FOREIGN KEY(book_id)
        REFERENCES books(id)
        ON DELETE CASCADE,
    CONSTRAINT unique_user_book_review
        UNIQUE(user_id, book_id)
);
```

Output 2:

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

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

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
    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
);
```

Output 3:

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

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

CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
    CONSTRAINT fk_book FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE
CASCADE,
    CONSTRAINT unique_user_book_review UNIQUE (user_id, book_id)
);
```

Codestral-2501 (Model 2):

Output 1:

```
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 CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Output 2:

```
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,
    book_id INT,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Output 3:

```
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 CHECK (rating >= 1 AND rating <= 5),

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE
);
```

Chain of Thought:

Prompt:

"Design a SQL schema for a review app with users, books, and reviews. Explain the process, without examples, step-by-step:
sql
-- TODO: Design schema with appropriate keys and constraints
-- Tables: users(id, name), books(id, title), reviews(id, user_id, book_id, rating)"

GPT 4.1 (Model 1):

Certainly! Here's a step-by-step process for designing a relational SQL schema for a review app with users, books, and reviews, focusing on keys and constraints:

---

1. Identify Entities and Attributes
Determine the main entities: users, books, reviews.
List their attributes, such as:
Users: id, name
Books: id, title
Reviews: id, user_id, book_id, rating

---

2. Determine Primary Keys
Assign a primary key for each table to uniquely identify each record.
Typically, id serves as the primary key in each table.
The primary key should be unique and not null.

---

3. Define Relationships and Foreign Keys
Establish how tables relate:
Each review is written by a user and relates to a book.
Use foreign keys in the reviews table:
user_id refers to the users table.
book_id refers to the books table.
Define foreign key constraints to ensure referential integrity.

---

4. Set Additional Constraints
Ensure required fields are marked as NOT NULL where appropriate (e.g., name, title, rating).
If you want to restrict each user to only one review per book, add a unique constraint on (user_id, book_id) in reviews.

---

5. Define Data Types
Choose appropriate SQL data types for each field, such as INTEGER for ids, VARCHAR for strings, etc.
For rating, select a numeric type and consider the value's range.

---

6. Consider Indexes and Cascade Options
Add indexes on frequently queried fields (e.g., user_id, book_id in reviews).
Decide on ON DELETE or ON UPDATE options for foreign keys (e.g., CASCADE or RESTRICT), based on how you want deletions/updates to behave.

---

7. Finalize and Review
Review for normalization (avoid redundancy).
Double-check for integrity constraints and that all relationships are enforced.

By following these steps, you ensure that your SQL schema is normalized, robust, and maintains data integrity across users, books, and reviews.

Codestral-2501 (Model 2):

To design a SQL schema for a review app with users, books, and reviews, you need to follow these steps:
Identify the Entities: Determine the main entities in your app. In this case, the entities are users, books, and reviews.
Define the Columns for Each Table:
For users: This table will store user information. You need at least an id and a name.
For books: This table will store book information. You need at least an id and a title.
For reviews: This table will store review information. It will need columns to reference the user_id (who wrote the review) and book_id (which book is being reviewed), along with a rating.
Set Primary Keys:
Each table should have a unique identifier (Primary Key) to uniquely identify each record. For users and books, you can use the id columns as primary keys. For reviews, you might consider composite primary keys involving user_id, book_id, and additional identifiers (e.g., a review timestamp or an auto-incrementing id).
Establish Foreign Keys:
The reviews table should have foreign keys that reference the primary keys of the users and books tables, ensuring data integrity.
Add Constraints (Optional but Recommended):
Add constraints like NOT NULL for mandatory fields.
Add constraints like UNIQUE for fields that should not have duplicate values.
Add constraints like CHECK to enforce specific constraints on data values.
Here is the step-by-step process to achieve this:
Step 1: Define the Tables and Columns
users table with id and name.
books table with id and title.
reviews table with id, user_id, book_id, and rating.
Step 2: Set Primary Keys
users(id) as the primary key.
books(id) as the primary key.
reviews(id) as the primary key.
Step 3: Establish Foreign Keys
reviews table should have user_id foreign key referencing users(id).
reviews table should have book_id foreign key referencing books(id).
Step 4: Add Constraints
Ensure id, user_id, and book_id are not nullable.
Ensure rating is within a valid range (e.g., 1 to 5).
Final SQL Schema Design

```
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 CHECK (rating >= 1 AND rating <= 5),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Explanation:

- users table: Contains id (primary key) and name (name of the user).
- books table: Contains id (primary key) and title (title of the book).
- reviews table: Contains id (primary key for unique review identification), user_id (foreign key referencing users(id)), book_id (foreign key referencing books(id)), and rating (review rating with a constraint to keep it within 1-5).

This design ensures that the relationships between users, books, and reviews are properly maintained and that data integrity is enforced.

Task 7: (Max tokens: 800, Temperature: 0.9)

Zero-Shot:

Prompt:

“Identify any null dereference risks in the following Java code without providing a fix:

public int getLength(String s) {

        return s.length();

}”

GPT 4.1 (Model 1):

Yes, there is a null dereference risk in the provided Java code. If the input parameter s is null when getLength is called, invoking s.length() will result in a NullPointerException.

Codestral-2501 (Model 2):

In the provided Java method getLength, there is a potential null dereference risk. The method takes a String parameter s and returns its length using s.length(). If the caller passes a null value for s, the s.length() invocation will throw a NullPointerException because you cannot call a method on a null object.

Here is the relevant code snippet that poses the risk:

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

If s is null when this method is called, it will result in a NullPointerException.
Example scenario causing NullPointerException:

```
public static void main(String[] args) {
    String test = null;
    int length = getLength(test);  // This will throw NullPointerException
}
```

Few-Shot:

Prompt:

“Given below are two examples of Java methods that pose a risk of null dereferencing:
java

```
private void exampleOne(String word) {
        System.out.println(m.length());
}
```

java

```
public float exampleTwo(float divideBy, String toDivide) {
        return toDivide.length() / divideBy;
}
```

Use the above two examples to identify any null dereferencing risks in the following Java method without returning code:
java

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

GPT 4.1 (Model 1):

The null dereferencing risk in the given method is that if the parameter s is null, calling s.length() will throw a NullPointerException.

Codestral-2501 (Model 2):

```
The method getLength(String s) carries a risk of null dereferencing if s is
null because s.length() will throw a NullPointerException.
```

Task 8: (Max tokens: 800, Temperature: 0.9)

Self-Consistency:

Prompt:

“Modify the following Python parser to support quoted fields, returning only the final code:
python
def parse_csv_line(line):
	return line.split(’,’)”

GPT 4.1 (Model 1):

Output 1:

```
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 string
                field += '"'
                i += 1
            else:
                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
```

Output 2:

```
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] == '"':
                field += '"'
                i += 1
            else:
                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
```

Output 3:

```
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
                field += '"'
                i += 1
            else:
                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
```

Codestral-2501 (Model 2):

Output 1:

Sure! To support quoted fields in a CSV line, we need to handle cases where fields might be enclosed in quotes and contain commas within them. Here's a modified version of the parse_csv_line function to handle this:

```
def parse_csv_line(line):
    import csv
    import io

    # Use StringIO to handle the line as a file-like object
    reader = csv.reader(io.StringIO(line))
    # Read the single line
    return next(reader)

# Example usage:
line = '"John Doe",john.doe@example.com,"New York, USA"'
parsed_line = parse_csv_line(line)
print(parsed_line)
```

In this version:

1. We use Python's built-in csv module, which has robust handling for CSV parsing, including support for quoted fields.
2. We use io.StringIO to treat the input line as a file-like object, which csv.reader can parse.
3. We read the single line using next(reader), which gives us the parsed fields as a list.

Output 2:

To modify the parse_csv_line function to support quoted fields, we need to handle cases where fields containing commas or quotes might be enclosed in double quotes. We can use Python's csv module to handle CSV parsing efficiently.

Here's the modified code:

```
import csv
import io

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

This function uses the csv.reader class from the csv module, which handles quoted fields correctly. The io.StringIO object is used to create a file-like object from the input string, allowing us to pass it to the csv.reader.

Here's an example of how to use this function:

```
line = '"name","age","address","notes"'
result = parse_csv_line(line)
print(result)  # Output: ['name', 'age', 'address', 'notes']
```

This approach ensures that quoted fields with commas or other special characters are correctly parsed into individual fields.

Output 3:

Sure, to modify the Python parser to support quoted fields, you need to handle cases where fields are enclosed in quotes and can contain commas within them.

Here is the modified code to handle quoted fields:

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

# Example usage
line = 'hello,"world,world",again'
print(parse_csv_line(line))
```

In this code:
We use the csv module, which is designed to handle CSV files correctly, including cases with quoted fields.
io.StringIO is used to create a file-like object from the line string, which is then passed to csv.reader.
next(reader) extracts the first (and only) row from the reader, which corresponds to the parsed CSV line.
This approach ensures that quoted fields are correctly parsed, and commas within quotes are treated as part of the field rather than as field separators.

Persona Prompting:

Prompt:

“You are a software engineer creating a Python method to parse a CSV file as efficiently as possible. Modify the following parser to support quoted fields, returning only the final code:
python
def parse_csv_line(line):
    return line.split(’,’)”

GPT 4.1 (Model 1):

```
import re

def parse_csv_line(line):
    # Handles quoted fields and commas inside quotes
    pattern = re.compile(r'''((?:[^,"']|"[^"]*"|'[^']*')+)''')
    # Split with regex, then strip any whitespace
    fields = re.findall(r'(?:^|,)(?:"([^"]*)"|([^",]*))', line)
    return [f[0] if f[0] != '' else f[1] for f in fields]
```

Example:

```
parse_csv_line('one,"two, too",three,"four ""quotes"""')
# Output: ['one', 'two, too', 'three', 'four "quotes"']
```

Note:

- This simple parser supports double-quoted fields, commas inside quotes, and escaped quotes via "".
- For more complex CSV formats, consider using the built-in csv module.

Codestral-2501 (Model 2):

To efficiently parse a CSV file that supports quoted fields, we can use the csv module from the Python standard library. This module handles quoted fields natively and provides a robust way to parse CSV data.
Here is the modified code:

```
import csv
```

```
def parse_csv_line(line):
    return list(csv.reader([line]))[0]
```

Explanation:

- The csv.reader function is used to create a CSV reader object which will iterate over lines in the given CSV file.
- By passing [line] as an argument, we instruct the reader to process a single line.
- list(csv.reader([line]))[0] converts the reader object to a list and extracts the first (and only) element, which is a list of fields.

This method is efficient and handles quoted fields correctly.

Task 9: (Max tokens: 800, Temperature: 0.9)

Zero-Shot:

Prompt:

“Convert the following Kotlin data class to a REST API using Ktor without explanation:
data class Product(val id: Int, val name: String, val price: Double)

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

GPT 4.1 (Model 1):

```
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.*
import kotlinx.serialization.Serializable

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

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

Codestral-2501 (Model 2):

```
import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.http.HttpStatusCode
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.serialization.*
import io.ktor.server.engine.embeddedServer
import io.ktor.server.netty.Netty

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

fun main() {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }
        var products = mutableListOf<Product>()

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

Prompt Chaining:

Prompt 1:

“Convert the following Kotlin data class to a REST API using Ktor without explanation and without import statements:
data class Product(val id: Int, val name: String, val price: Double)

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

GPT 4.1 (Model 1):

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

fun Application.module() {
    routing {
        get("/products") {
            call.respond(products)
        }
        post("/products") {
            val product = call.receive<Product>()
            products.add(product)
            call.respond(HttpStatusCode.Created, product)
        }
    }
}
```

Codestral-2501 (Model 2):

```
// Ktor-based REST API setup
val server = embeddedServer(Netty, port = 8080) {
    install(ContentNegotiation) {
        gson {
            setPrettyPrinting()
        }
    }

    routing {
        val products = mutableListOf(Product(1, "Item1", 10.99), Product(2, 
"Item2", 20.99))

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

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

Prompt 2:

"List the necessary import statements required for the previous code."

GPT 4.1 (Model 1):

```
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.*
```

Codestral-2501 (Model 2):

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

Task 10: (Max tokens: 400, Temperature: 1.0)

Self-Consistency:

Prompt:

"Briefly summarize the following Python function:

def reverse_words(sentence):

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

GPT 4.1 (Model 1):

Output 1:

```
The function reverse_words(sentence) takes a string, splits it into words,
reverses the order of the words, and then joins them back into a single string.
It returns the sentence with the word order reversed.
```

Output 2:

```
Summary:
The reverse_words function takes a sentence (a string), splits it into words,
reverses the order of the words, and then joins them back into a new sentence.
Example:
reverse_words("hello world") returns "world hello".
```

Output 3:

```
This function takes a sentence as input and returns a new string with the order
of the words reversed.
```

Codestral-2501 (Model 2):

Output 1:

```
The provided Python function reverse_words(sentence) takes a string sentence as
input and returns a new string where the order of the words is reversed. Here
is a brief summary of how it works:
   1. sentence.split(): Splits the input string sentence into a list of words
      based on whitespace.
   2. [::-1]: Reverses the order of the list of words.
```

3. ' '.join(...): Joins the reversed list of words back into a single string, with each word separated by a space.

For example, if the input is "hello world", the function would return "world hello".

Output 2:

The provided Python function reverse_words takes a sentence as input, reverses the order of the words in the sentence, and returns the reversed sentence.

Here’s a step-by-step breakdown of how it works:

1. Splitting the Sentence: The sentence.split() method splits the input string into a list of words based on whitespace.
2. Reversing the List: The [::-1] slice notation reverses the order of the list of words.
3. Joining the Words: The '.join()' method joins the reversed list of words back into a single string, with each word separated by a single space.

Example

If the input is:

reverse_words("hello world")

The output would be:

"world hello"

This function effectively reverses the order of the words in any given sentence.

Output 3:

The reverse_words function takes a sentence string as input and returns a new string with the order of the words reversed. Here's a step-by-step summary of what the function does:

1. Split the Sentence: The function first splits the input sentence into a list of words using the split() method. By default, this splits the string at whitespace.
2. Reverse the List: The [::-1] slicing notation is used to reverse the list of words.
3. Join the Words: The reversed list of words is then joined back into a single string with each word separated by a space using the ' '.join() method.
4. Return the Result: The resulting reversed string is returned.

For example, if you pass the string "hello world" to this function, it will return "world hello".

Few-Shot:

Prompt:

“Example 1: The descending_sort function takes a list of integers as input and sorts them in descending order.

Example 2: The remove_punctuation function takes a string as input and removes all punctuation.

Briefly summarize the following Python method:

def reverse_words(sentence):

```
    return ’ ’.join(sentence.split()[::-1])”
```

GPT 4.1 (Model 1):

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

Codestral-2501 (Model 2):

The reverse_words function takes a sentence string as input and returns the sentence with the words reversed while maintaining the order of the characters within each word. Here's a brief summary:

1. Input: A string representing a sentence.
2. Process:
    - Splits the input sentence into a list of words.
    - Reverses the order of the words in the list.
    - Joins the reversed list of words back into a single string with spaces in between.
3. Output: A string where the order of the words is reversed, but the characters within each word remain in their original order.

__Task 11:__ (Max tokens: 800, Temperature: 0.7)

__Zero-Shot:__

Prompt:

“Write a prompt that could generate the following Python 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
```
”

GPT 4.1 mini (Model 1):

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 check for numbers less than or equal to 1 and return False in that case. Use a loop to test divisibility from 2 up to the square root of n (inclusive). Include a comment at the beginning that explains the purpose of the function.

Codestral-2501 (Model 2):

Sure, here is a prompt that could generate the given Python code:

"Write a Python function named is_prime that takes a single integer n as input and returns True if n is a prime number and False otherwise. Ensure the function handles edge cases like numbers less than or equal to 1. Use an efficient method to check for primality, such as checking divisibility up to the square root of n."

Chain of Thought:

Prompt: “Write a step-by-step prompt that could generate the following Python 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```
~~~

Do not include hints or specific code in the prompt.”

GPT 4.1 mini (Model 1):

Sure! Here's a step-by-step prompt to generate that Python function:

1. Write a Python function that takes a single integer as input.
2. The function should determine whether the given number is a prime number.
3. If the number is less than or equal to 1, the function should return False.
4. For all integers starting from 2 up to the square root of the number (inclusive), check if the number is divisible by any of these.
5. If the number is divisible by any of these integers, return False.
6. If no divisors are found, return True to indicate the number is prime.

Codestral-2501 (Model 2):

Sure, here's a step-by-step prompt that could generate the given Python code:

1. Start by defining a function named is_prime that takes a single argument n.
2. Inside the function, add a condition to check if n is less than or equal to 1. If it is, return False because numbers less than or equal to 1 are not prime.
3. After the initial condition, use a for loop to iterate over a range of numbers starting from 2 up to the square root of n (inclusive).
4. Within the loop, add a condition to check if n is divisible by the current number i (i.e., n % i == 0). If it is, return False because n is not a prime number.
5. If the loop completes without finding any divisors, return True indicating that n is a prime number.

Task 12: (Max tokens: 800, Temperature: 1.0)

Zero-Shot:

Prompt:

“Fix the bug in the following Python method when input is 0:

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

GPT 4.1 mini (Model 1):

`The issue is that when n is 0, the function should return 1 (since 0! = 1), but also the loop currently runs from 1 to n-1 (since range(1, n) excludes n), so for n=1 it runs zero times, which is correct, but for other values it misses multiplying by n itself.`

`Also, the loop should go up to n inclusive, so the range should be range(1, n+1).`

`Here's the fixed code:`

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

`This works correctly for n=0 (returns 1), and for all positive integers.`

Codestral-2501 (Model 2):

The bug in the given factorial function is that it only multiplies numbers from 1 to *n*−1, which is incorrect for calculating the factorial of *n*. The correct range should be from 1 to *n* inclusive. Additionally, if the input *n* is 0, the function should return 1 because the factorial of 0 is defined to be 1.

Here is the corrected version of the function:

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

With this correction, the function will correctly compute the factorial for any non-negative integer *n*.

Self Consistency:

Prompt:

“Fix the bug in the following Python method when input is 0. Return only code:
```
def factorial(n):
    result = 1
    for i in range(1, n):
        result *= i
    return result
```
”

GPT 4.1 mini (Model 1):

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

Output 2:
```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Output 3:
```
def factorial(n):
  result = 1
  for i in range(1, n+1):
    result *= i
  return result
```

Codestral-2501 (Model 2):

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

Output 2:

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

Output 3:

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

<u>Task 13:</u> (Max tokens: 800, Temperature: 0.6)

<u>Zero-Shot:</u>

Prompt:

“Use the following C code to implement node deletion by value. Return only code:

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

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

GPT 4.1 mini (Model 1):

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

    // If head node itself holds the key to be deleted
    if (temp != NULL && temp->data == key) {
        *head = temp->next; // Changed head
        free(temp);         // free old head
        return;
    }
    // Search for the key to be deleted, keep track of the previous node
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

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

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

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

Codestral-2501 (Model 2):

Here is the implementation of the deleteNode function to delete a node by value in a singly linked list:

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

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

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

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

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

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

Chain of Thought:

Prompt:

“Use the following C code to implement node deletion by value. Do not add anything beyond the node deletion method. Explain your implementation step-by-step:
~~~
struct Node {
	int data;
	struct Node* next;
};

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

GPT 4.1 mini (Model 1):

Certainly! I'll implement the node deletion by value in the given linked list and explain the implementation step-by-step.

Here is the complete function implementation:

```
void deleteNode(struct Node** head, int key) {
    // Step 1: Store head node
    struct Node* temp = *head;
    struct Node* prev = NULL;

    // Step 2: 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;
    }

    // Step 3: Search for the key to be deleted, keep track of previous node
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // Step 4: If key was not present in linked list
    if (temp == NULL) return;

    // Step 5: Unlink the node from linked list
    prev->next = temp->next;

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

Step-by-step explanation:

1. Initialize pointers:
    - temp is initialized to the head of the list (*head).
    - prev is initialized to NULL. It will keep track of the previous node during traversal.
2. Check if head node itself holds the key:
    - If the list is not empty (temp != NULL) and the data in the head node matches the key, then:
        - Update the head pointer to point to the next node (*head = temp->next).
        - Free the memory allocated to the old head node (free(temp)).
        - Return immediately because deletion is complete.
3. Traverse the linked list to find the node to delete:
- Use a while loop to move through the list until:
    - You reach the end (temp == NULL), or
    - You find the node whose data matches the key.
- During traversal, update prev to the current node before moving temp forward.
4. If the key was not found:
    - If temp becomes NULL (end of list) and the key was not found, simply return doing nothing.
5. Delete the node:
    - If the node was found (temp is not NULL), unlink it by making prev->next point to temp->next.
    - Free the memory allocated to the node to be deleted (free(temp)).

This function safely deletes the first node in the linked list that contains the specified key. It handles deletion at the head, middle, or end of the list, as well as the case where the key is not found.

Codestral-2501 (Model 2):

To implement node deletion by value in the given singly-linked list, we need to traverse the list, find the node with the specified value, and then remove it. There are a few scenarios to consider:

1. The node to be deleted is the head node.
2. The node to be deleted is in the middle or end of the list.
3. The node to be deleted does not exist in the list.

Here is the complete implementation of the deleteNode function:

```
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 as we need to change 'prev->next'
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

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

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

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

Step-by-Step Explanation:

1. Initialization:
    - We initialize a pointer temp to point to the head of the list. This will be used to traverse the list.
    - We also initialize a pointer prev to NULL. This will be used to keep track of the previous node during the traversal.
2. Check if the head node needs to be deleted:
    - If the head node itself holds the key to be deleted, we update the head pointer to point to the next node in the list.
    - We then free the memory allocated for the old head node and return from the function.
3. Traverse the list to find the node to delete:
    - We use a while loop to traverse the list. The loop continues as long as temp is not NULL and the data in the current node (temp->data) does not match the key.
    - Inside the loop, we update prev to point to temp (the current node) and then move temp to the next node in the list.
4. Check if the key was not found:
    - If temp becomes NULL during the traversal, it means the key was not found in the list. In this case, we simply return from the function.
5. Unlink and delete the node:
    - If the key is found (i.e., temp points to the node with the key), we update the next pointer of the previous node (prev->next) to skip the current node (temp).
    - Finally, we free the memory allocated for the node to be deleted.

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

<u>Task 14:</u> (Max tokens: 800, Temperature: 0.8)

<u>Few-Shot:</u>
Prompt:
“Example 1:
Input: 5
Output: 5

Example 2:
Input: 7
Output: 13

Complete the recursive Python function for Fibonacci:
~~~
def fibonacci(n):
        # TODO: Base cases and recursive call pass
```”

GPT 4.1 mini (Model 1):

Here's the completed recursive Python function for Fibonacci:

```
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)
~~~

Explanation:
fibonacci(0) returns 0
fibonacci(1) returns 1
For n > 1, the function returns the sum of the two previous Fibonacci numbers.
Testing with your examples:

```
print(fibonacci(5))  # Output: 5
print(fibonacci(7))  # Output: 13
```

Codestral-2501 (Model 2):

To complete the recursive Python function for the Fibonacci sequence, you need to define the base cases and the recursive call. The Fibonacci sequence is defined as follows:

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

```
Here is the complete implementation of the fibonacci function:
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(5))  # Output: 5
print(fibonacci(7))  # Output: 13
```

```
Explanation:
   1. Base Cases:
         ● If n is 0, the function returns 0.
         ● If n is 1, the function returns 1.
   2. Recursive Call:
         ● For any other value of n, the function returns the sum of the two
           preceding Fibonacci numbers, fibonacci(n - 1) and fibonacci(n - 2).
This implementation correctly computes the Fibonacci number for any
non-negative integer n.
```

Prompt Chaining:

Prompt 1:

“Complete the recursive Python function for Fibonacci. Return only code:

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

GPT 4.1 mini (Model 1):

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

Codestral-2501 (Model 2):

```
def fibonacci(n):
  if n <= 0:
    return 0
  elif n == 1:
    return 1
```

```
  else:
    return fibonacci(n-1) + fibonacci(n-2)
```

Prompt 2:

"Explain how to optimize the above Fibonacci code."

GPT 4.1 (Model 1):

To optimize the recursive Fibonacci function, you can use memoization to avoid redundant calculations. The naive recursive approach recalculates the same Fibonacci numbers multiple times, leading to exponential time complexity (O(2^n)).

How memoization helps:

- Store previously computed Fibonacci values in a cache (like a dictionary).
- When you need a Fibonacci number, first check if it's in the cache.
- If yes, return it directly instead of recomputing.
- This reduces time complexity to linear (O(n)).

Example using memoization:

```
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 0:
        memo[n] = 0
    elif n == 1:
        memo[n] = 1
    else:
        memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]
```

Alternatively, you can use dynamic programming with iteration to compute Fibonacci numbers efficiently, which also has (O(n)) time complexity and (O(1)) space complexity if you only keep track of the last two numbers.

Codestral-2501 (Model 2):

The original Fibonacci code provided is a straightforward recursive implementation, but it has an exponential time complexity due to redundant calculations. To optimize it, we can use several techniques:

1. Memoization: Store the results of expensive function calls and reuse them when the same inputs occur again. This can significantly reduce the time complexity from exponential to linear.
2. Dynamic Programming: Use an iterative approach to build up the solution in a bottom-up manner, storing the results of subproblems in a table.

Here's how you can optimize the Fibonacci function using memoization:

```
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
        return memo[n]
```

In this optimized version, a dictionary memo is used to store the results of Fibonacci numbers that have already been computed. This way, each Fibonacci number is computed only once, and subsequent calls use the stored results, significantly improving performance.
Alternatively, you can use dynamic programming with an iterative approach:

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1

    fib = [0, 1]
    for i in range(2, n + 1):
        fib.append(fib[i-1] + fib[i-2])

    return fib[n]
```

In this iterative approach, a list fib is used to store the Fibonacci numbers up to n. The loop calculates each Fibonacci number based on the previous two numbers, and the result is stored in the list. This method is more efficient and avoids the overhead of recursive calls and memoization.

Task 15: (Max tokens: 400, Temperature: 1.0)

Self-Consistency:

Prompt:

“Complete the following Python class constructor, returning only code:

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

GPT 4.1 mini (Model 1):

Output 1:

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

Output 2:

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

Output 3:

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

Codestral-2501 (Model 2):

Output 1:

```
class Person:
    def __init__(self, name, age, email=""):
        self.name = name
        self.age = age
        self.email = email
```

Output 2:

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

Output 3:

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

<u>Chain of Thought:</u>

Prompt:

"Complete the following Python class constructor. Explain the constructor step-by-step:

```
class Person:
	def __init__(self):
```

                    # TODO: Add name, age, and optional email pass
```"

## GPT 4.1 mini (Model 1):

Sure! Here's how to complete the __init__ constructor for the Person class to include name, age, and an optional email attribute.

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

Step-by-step explanation:

1. Add parameters to the constructor
   The constructor __init__ method needs to accept name and age as required parameters, since these are essential attributes to create a Person.
   The email parameter should be optional, so we set a default value email=None.
2. Assign parameters to instance variables
   Inside the constructor, we assign the input parameters to instance variables with self.name = name, self.age = age, and similarly for email.
   This allows every Person object to store these values individually.
3. Use default argument for optional email
   By using email=None, we allow the creation of a Person object without specifying an email address. If no email is provided, it will default to None.

This completes the constructor with the required attributes name, age, and an optional email.

Codestal-2501 (Model 2):

To complete the __init__ method for the Person class, we need to add parameters for name, age, and an optional email. Here's a step-by-step process for achieving this:

1. Define the Parameters: Specify parameters for name, age, and email within the __init__ method.
2. Initialize Instance Variables: Assign the parameter values to instance variables.
3. Make email Optional: Provide a default value for email (None is a common choice for optional parameters).

Here is the completed Person class:

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

Detailed Explanation:

1. Define the Parameters:
```

    - name: The name of the person.
    - age: The age of the person.
    - email: An optional email address.
2. Initialize Instance Variables:
    - self.name = name: Assign the name parameter to the instance variable self.name.
    - self.age = age: Assign the age parameter to the instance variable self.age.
    - self.email = email: Assign the email parameter to the instance variable self.email. If no email is provided, it defaults to None.
3. Make email Optional:
    - By setting email=None as the default value in the method signature, we ensure that the email parameter is optional. If no email is provided when creating a Person object, self.email will be None.

This allows the Person class to be instantiated with either the name and age, or with all three parameters (name, age, and email). Here are some examples of instantiating Person objects: [**TOKEN LIMIT REACHED**]

**Task 16:** (Max tokens: 500, Temperature: 0.9)

**Zero-Shot:**

**Prompt:**

“Complete the following Java binary search implementation, returning only code:

```
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;
}
```
”

GPT 4.1 mini (Model 1):

```
if (arr[mid] == target) {
  return mid;
} else if (arr[mid] < target) {
  left = mid + 1;
} else {
  right = mid - 1;
}
```

Codestral-2501 (Model 2):

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

Prompt Chaining:

Prompt 1: “Complete the following Java binary search implementation, returning only code:

```
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;
}
```
”

GPT 4.1 mini (Model 1):

```
if (arr[mid] == target) {
  return mid;
} else if (arr[mid] < target) {
  left = mid + 1;
} else {
  right = mid - 1;
}
```

Codestral-2501 (Model 2):

```
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;
}
```

Prompt 2:

"Create example inputs and outputs to test the above binary search implementation."

GPT 4.1 mini (Model 1):

```
// Example inputs and expected outputs for testing binarySearch

int[] arr1 = {1, 3, 5, 7, 9};
System.out.println(binarySearch(arr1, 5));  // Output: 2
System.out.println(binarySearch(arr1, 1));  // Output: 0
System.out.println(binarySearch(arr1, 9));  // Output: 4
System.out.println(binarySearch(arr1, 6));  // Output: -1

int[] arr2 = {};
System.out.println(binarySearch(arr2, 1));  // Output: -1

int[] arr3 = {2};
System.out.println(binarySearch(arr3, 2));  // Output: 0
System.out.println(binarySearch(arr3, 3));  // Output: -1

int[] arr4 = {1, 2, 2, 2, 3};
System.out.println(binarySearch(arr4, 2));  // Output: 1 (or 2 or 3, any index
with value 2)
```

Codestral-2501 (Model 2):

```
Certainly! Here are some example inputs and expected outputs to test the binary
search implementation:

Input:
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Target: 5
Output:
4

Input:
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Target: 11
Output:
-1

Input:
Array: []
```

```
Target: 5
Output:
-1

Input:
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Target: 1
Output:
0

Input:
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Target: 10
Output:
9

Input:
Array: [1, 3, 5, 7, 9, 10, 12, 13, 15, 18, 20, 25, 30, 31, 32, 35, 38, 40]
Target: 18
Output:
9

Input:
Array: [1, 3, 5, 7, 9, 10, 12, 13, 15, 18, 20, 25, 30, 31, 32, 35, 38, 40]
Target: 19
Output:
-1
```

You can use these inputs and expected outputs to verify if the binary search implementation is working correctly.

Task 17: (Max tokens: 500, Temperature: 0.9)

Zero-Shot:

Prompt: “The following C++ method does not do what the method name implies. Fix the method code so that it accurately reflects the name:

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

GPT 4.1 (Model 1):

The current implementation returns true when x is even (because x % 2 == 0), which is the opposite of what isOdd should do.
To correctly check if a number is odd, it should return true when x % 2 != 0.
Here's the corrected method:

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

Codestral-2501 (Model 2):

The method name isOdd implies that the method should return true if the integer x is odd and false if it is even. However, the current implementation returns true if x is even and false if it is odd. To fix this, you need to reverse the condition in the method.
Here is the corrected code:

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

This way, isOdd will return true if x is odd and false if it is even.

<u>Few-Shot:</u>

Prompt:

“Example 1:

~~~
```
bool isNegative(int x) {
	return x >= 0;
}
```
~~~

This method is incorrect because it returns true when x is positive, not negative.

Example 2:

~~~
```
bool isTen(int x) {
	return x == 11;
}
```
~~~

This method is incorrect because it returns true when x is 11, not 10.

Fix the following C++ method code so that it accurately reflects the name:

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

GPT 4.1 mini (Model 1):

The given code:

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

```
is incorrect because it returns true when x is even, not odd.
Corrected version:
```

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

Codestral-2501 (Model 2):

```
The provided method isOdd is incorrectly named because it returns true when x
is even, and false when x is odd. To fix this, you need to negate the condition
to accurately reflect the method name. Here is the corrected method:
```

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

```
This method will return true when x is odd and false when x is even.
```

<u>Task 18:</u> (Max tokens: 500, Temperature: 0.8)

<u>Prompt Chaining:</u>

Prompt 1:

"Identify the bug in the following Javascript function without fixing it:
~~~
function isEven(n) {
	return n % 2;
}
```"

GPT 4.1 mini (Model 1):

```
The bug is that the function returns the remainder of n % 2 (which is 0 or 1),
rather than a boolean indicating whether n is even. Specifically, it returns 0
for even numbers (which is falsy) and 1 for odd numbers (which is truthy), so
the function's return value does not correctly represent "is even" as a
boolean.
~~~

Codestral-2501 (Model 2):

```
The bug in the given JavaScript function isEven(n) lies in the return
statement. The function uses the modulus operator % to check if the number n is
even. However, the modulus operator % returns the remainder of the division of
n by 2.
```

For an even number, the remainder is 0, and for an odd number, the remainder is 1. The current implementation returns the remainder directly, which means:

- If n is even, it returns 0 (which is falsy).
- If n is odd, it returns 1 (which is truthy).

This behavior is correct in terms of checking if the number is even or odd, but it does not follow the conventional way of returning true for even numbers and false for odd numbers.

To identify the bug, one would notice that the function does not explicitly return true or false but rather the result of the modulus operation. This can lead to confusion if the function's purpose is to specifically return boolean values.

Prompt 2:

“Fix the identified bug.”

GPT 4.1 mini (Model 1):

Here is the fixed version of the function:

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

This returns true if n is even, and false otherwise.

Codestral-2501 (Model 2):

The corrected JavaScript function should return true if the input is an even number and false if it's an odd number. Here is the fixed function:

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

In this corrected version, the function checks if n % 2 equals 0. If it does, the function returns true, indicating that n is an even number. Otherwise, it returns false, indicating that n is an odd number.

<u>Self-Consistency:</u>

Prompt:

“Identify and fix the bug in the following Javascript function:

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

GPT 4.1 mini (Model 1):

Output 1:

The bug in the function is that it returns the remainder of n % 2, which will be 0 for even numbers and 1 for odd numbers. However, since 0 is treated as false and any non-zero number is true in JavaScript, the function will return false for even numbers and true for odd numbers, which is the opposite of what the function name isEven implies.
To fix the bug, the function should return true if n is even, i.e., if n % 2 === 0, and false otherwise.
Here is the corrected function:

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

This way, isEven(4) will return true, and isEven(5) will return false.

Output 2:

The bug in the function is that it returns n % 2, which gives the remainder when n is divided by 2 — this will be 0 for even numbers and 1 for odd numbers. However, the function is expected to return a boolean indicating whether n is even or not.
How to fix:
Return a boolean expression that evaluates to true if n is even, and false otherwise. You can do this by checking if n % 2 equals 0.

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

Now, isEven(4) will return true, and isEven(3) will return false.

Output 3:

The bug in the function is that it returns the remainder of n % 2, which will be 0 for even numbers and 1 (or -1 for negative odd numbers) for odd numbers. Since 0 is falsy and any non-zero number is truthy in JavaScript, the function currently returns false for even numbers and true for odd numbers, which is the opposite of the intended behavior.
To fix the bug:
You need to return true when n is even (i.e., when n % 2 equals 0), and false otherwise. This can be done by explicitly comparing the remainder to 0:

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

Explanation:
n % 2 === 0 evaluates to true if n is even.
n % 2 !== 0 evaluates to true if n is odd.
Now isEven(4) will return true, and isEven(5) will return false as expected.

Codestral-2501 (Model 2):

Output 1:

The given JavaScript function isEven is intended to determine whether a number n is even. However, the current implementation has a bug. The function returns n % 2, which returns 0 for even numbers and a non-zero value for odd numbers. To fix this bug, you should return true for even numbers and false for odd numbers.
Here is the corrected function:

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

In this corrected version:
n % 2 computes the remainder when n is divided by 2.
n % 2 === 0 checks if the remainder is zero, which means n is even.
The function returns true if n is even and false otherwise.

Output 2:

The provided JavaScript function isEven(n) is intended to check if a number n is even. However, it contains a bug. The function returns n % 2, which returns 0 if n is even and a non-zero value if n is odd. Instead, the function should return true if n is even and false if n is odd.
To fix the bug, you should modify the function to return true when n % 2 is 0 and false otherwise. Here's the corrected version of the function:

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

In this corrected version:
n % 2 checks if n is divisible by 2.
n % 2 === 0 ensures that the function returns true when n is even and false when n is odd.
Now, you can use this function to check if a number is even:

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

Output 3:

The provided JavaScript function isEven is intended to determine whether a number n is even. However, the bug in the function is that it returns the remainder of the division of n by 2, rather than a boolean value indicating whether n is even.
To fix this, you should return a boolean value, specifically true if n % 2 === 0 (i.e., n is even), and false otherwise. Here’s the corrected function:

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

This way, the function will correctly determine if the number n is even and return true or false accordingly.

<u>Task 19:</u> (Max tokens: 500, Temperature: 0.8)

<u>Chain of Thought:</u>

Prompt:

“Decompose the high-level comment/summary into logical steps according to the method:
// Function that validates an input, calculates square, and returns result
```
int process(int x) {
	if (x < 0)
		return -1;
	return x * x;
}
```
”

GPT 4.1 mini (Model 1):

Certainly! Here's the decomposition of the high-level comment into logical steps according to the method shown in the code:

High-level comment:

// Function that validates an input, calculates square, and returns result

Logical steps:

1. Receive an integer input x.
2. Validate the input by checking if x is negative.
3. If x is less than 0, return -1 to indicate invalid input.
4. If the input is valid (non-negative), calculate the square of x.

Return the squared value as the result.

Codestral-2501 (Model 2):

To decompose the high-level comment/summary into logical steps according to the provided method, we can break down the process into the following stages:

1. Input Validation:
    - Check if the input x is valid.
    - If x is less than 0, return -1 to indicate an invalid input.
2. Calculation:
    - If the input x is valid (i.e., x is non-negative), calculate the square of x.
3. Return Result:
    - Return the calculated square of x.

Here is the corresponding code implementation of the process function with these logical steps:

```
int process(int x) {
    // Step 1: Input Validation
    if (x < 0) {
        return -1;  // Return -1 for invalid input
    }

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

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

Explanation of the steps:

1. Input Validation: The function checks if x is less than 0. If true, it immediately returns -1, indicating that the input is invalid.
2. Calculation: If x is non-negative, the function calculates x * x and stores the result in the result variable.
3. Return Result: Finally, the function returns the calculated square of x.

This decomposition makes the function easier to understand and maintain.

Zero-Shot:

Prompt:

"Use the given high-level comment/summary to explain the process of the method:

// Function that validates an input, calculates square, and returns result

```
int process(int x) {
	if (x < 0)
		return -1;
	return x * x;
}
```
"

GPT 4.1 mini (Model 1):

The method process takes an integer input x and performs two main tasks:

1. Validation: It first checks if the input x is negative. If x is less than zero, the method considers this invalid input and immediately returns -1 to indicate an error or invalid value.
2. Calculation and Return: If the input is valid (i.e., zero or positive), the method calculates the square of x by multiplying x by itself, and then returns this squared value as the result.

In summary, the method ensures the input is non-negative before proceeding to compute and return its square, using -1 as a signal when the input does not meet the validation criteria.

Codestral-2501 (Model 2):

Certainly! The provided method, process, follows a sequence of steps to validate an input, calculate the square of the input if valid, and return the result. Here is a detailed explanation of the process:

1. Input Validation:
    - The method takes a single integer input x.
    - It immediately checks if the input x is less than 0.
    - If x is negative, the method returns -1, indicating an invalid input.
2. Square Calculation:

- If the input x is not negative, the method proceeds to the next step.
- It calculates the square of x by multiplying x by itself (x * x).

3. Return Result:
    - The computed square of x is then returned as the result of the method.

Here is the method with a breakdown of each step:

```
int process(int x) {
    // Step 1: Input Validation
    if (x < 0) {
        return -1;  // Return -1 if x is negative (invalid input)
    }

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

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

Summary of the Process

1. Input Validation: Check if x is less than 0.
    - If true, return -1.
2. Square Calculation: Compute x * x.
3. Return Result: Return the calculated square.

This method ensures that only non-negative inputs are processed, and it returns -1 for any invalid (negative) inputs. For valid inputs, it returns the square of the input.

Task 20: (Max tokens: 500, Temperature: 1.0)

Prompt Chaining:

Prompt 1:

“Determine the function of the following Python method based on intent. Do not return code:

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

GPT 4.1 mini (Model 1):

The function is intended to calculate and return the average value of the numbers provided in the scores list.

Codestral-2501 (Model 2):

Based on the method name "calculate_average" and the presence of a variable named "total" initialized to 0, as well as the presence of a comment indicating the intent to "return average pass," the function is intended to take a list or collection of scores, calculate the sum of these scores, and then compute and return the average of these scores.

Prompt 2:

“Implement the calculate_average function with the described purpose.”

GPT 4.1 mini (Model 1):

Here is the implementation of the calculate_average function that calculates and returns the average of the values in the scores list:

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

Codestral-2501 (Model 2):

Here is the completed calculate_average function:

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

This function takes a list of scores as input, sums them up, and then returns the average by dividing the total by the number of scores. The len(scores) function is used to determine the number of elements in the list scores.

Prompt 3:

“Identify any potential bugs in the created code.”

GPT 4.1 mini (Model 1):

Potential issues in the provided code include:

1. Empty list handling:
   The function returns 0 when the input list scores is empty, which might be acceptable in some contexts, but it could also be better to raise an exception or handle it differently depending on use case.
2. Division by zero:
   The code uses a conditional expression to avoid division by zero (if scores else 0), so it safely handles empty lists and prevents a ZeroDivisionError.

3. Data type assumptions:
   The code assumes all elements in scores are numeric (integers or floats). If non-numeric values are present, it will raise a TypeError during addition.

No outright bugs exist in the code but input validation or error handling could be improved to make the function more robust.

Codestral-2501 (Model 2):

There are a few potential issues with the provided calculate_average function:

1. Division by Zero: If the input list scores is empty, the function will attempt to divide by zero, which will result in a ZeroDivisionError.
2. Non-numeric Input: If the input scores list contains non-numeric values, such as strings or other types, the function will raise a TypeError when attempting to add them to total.
3. Return type: The name (calculate_average) suggests returning a float, but if the scores list is all integers, the function will return an integer instead.

To address these issues, we can add checks and ensure the function handles these cases appropriately.

Self-Consistency:

Prompt:

“Determine the function of the following Python method based on intent. Do not return code:

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

GPT 4.1 mini (Model 1):

Output 1:

The function is intended to calculate and return the average value of the elements in the scores list.

Output 2:

The purpose of the function is to compute and return the average value of the numerical values contained in the input list or collection named scores.

Output 3:

The function is intended to calculate and return the average value of the elements in the scores list.

Codestral-2501 (Model 2):

Output 1:

Based on the method name calculate_average and the variable name scores, the intent of this Python method appears to be calculating the average of a list or array of numerical scores. The total variable is initialized to 0, presumably to accumulate the sum of the scores. The comment # TODO: Complete to return

average pass suggests that the method needs to iterate through the scores list, sum the values, then compute and return the average.
To complete this method:

1. Iterate through each score in the scores list.
2. Add each score to the total variable.
3. Compute the average by dividing the total by the number of scores.
4. Return the computed average.

Output 2:

The function calculate_average(scores) is intended to calculate the average of a list of scores. The incomplete part indicates that the method needs to iterate over the scores list, sum them, and then divide the total by the number of scores to get the average. Finally, it should return the computed average.

Output 3:

The function calculate_average(scores) is intended to compute the average of a list of scores. It initializes a total variable to zero, indicating that it will accumulate the sum of the scores. To complete the function, the programmer needs to iterate through the list of scores, add each score to the total, and then calculate the average by dividing the total by the number of scores. Finally, it should return the computed average.

<u>Task 21:</u> (Max tokens: 500, Temperature: 1.0)

<u>Prompt Chaining:</u>

Prompt 1:

“Analyze the following Python utility script. Identify any potential logic issues or design flaws. Do not return code:

~~~
```
#utils.py - Script to parse and summarize numeric CSV files
import csv
def read_csv(filepath):
    with open(filepath, ’r’) as f:
        return [row for row in csv.reader(f)]

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

def main():
    filepath = ’data.csv’
    data = read_csv(filepath)
    total, avg = summarize_column(data, 1)
~~~

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

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

GPT 4.1 mini (Model 1):

~~~
Here are some potential logic issues and design flaws in the provided Python
utility script:

1. **Use of Non-Standard Quotation Marks**
   The script uses typographic (curly) quotes (`’` and `‘`) instead of standard
straight quotes (`'` or `"`). This will cause syntax errors when running the
code. For example, the open statement uses `’r’` instead of `'r'`, and the `if
__name__ == ’__main__’: ` line uses curly quotes.

2. **Lack of Input Validation**
   - The `summarize_column` function assumes that all values in the targeted
column can be converted to float without error. If the CSV contains any
non-numeric strings, this will raise a `ValueError`.
   - It also assumes that the row has enough columns (i.e., `index` exists in
all rows). There is no check for missing or malformed rows.

3. **No Handling of Empty Data or Empty Columns**
   If the CSV file is empty or contains only headers, `values` will be an empty
list, leading to a division by zero error when calculating the average.

4. **Hardcoded Filepath and Column Index**
   - The filepath is hardcoded as `'data.csv'`, reducing flexibility.
   - The column index is hardcoded as `1` in the call to `summarize_column`.
This is not clear or configurable by the user.

5. **No Use of CSV Header for Column Selection**
   The script skips the header row but never uses it. It would be more robust
and user-friendly to allow referencing the column by name instead of by index.

6. **No Error Handling for File I/O**
   The script assumes the file exists and is readable. No try/except blocks
handle `FileNotFoundError`, permission errors, or parsing errors.

7. **Reading Entire CSV into Memory**
   The entire CSV file is read into memory as a list of rows, which might be an
issue with very large files. A streaming or generator approach might be more
memory efficient.

8. **Minimal Documentation and No Logging**
   The functions lack docstrings or comments explaining expected input/output,
and there is no logging of errors or process steps, which reduces
maintainability.
~~~

9. **No Support for Different Delimiters or CSV Variants**
   The `csv.reader` is used with default parameters. If the CSV uses other delimiters or quoting conventions, this might cause issues.

Overall, the script is a minimal example but lacks robustness, configurability, and proper error handling required for production use.

Codestral-2501 (Model 2):

Here are some potential logic issues and design flaws in the provided Python utility script:

1. **File Path Hardcoding**: The file path (`"data.csv"`) is hardcoded in the `main()` function. This makes the script inflexible and not reusable for different CSV files. It would be better to accept the file path as an argument or from user input.

2. **Error Handling**: There is no error handling for file operations or data conversion. If the file does not exist, the script will raise a `FileNotFoundError`. If the CSV contains non-numeric data, the script will raise a `ValueError`. Proper error handling should be implemented to handle such cases gracefully.

3. **Column Indexing**: The column index (1 in this case) is hardcoded in the `main()` function. It would be better to make this configurable, perhaps through command-line arguments or user input.

4. **Header Row Handling**: The script assumes that the header row is present and should be skipped. If the CSV file doesn't have a header row, this will lead to incorrect data processing. The script should validate whether the header is present.

5. **Data Type Checking**: The script assumes all values in the specified column are numeric. If non-numeric data is present, the script will raise a `ValueError`. Proper validation should be done to handle such cases.

6. **Separation of Concerns**: The script should ideally separate the concerns of reading the CSV, processing the data, and presenting results. This would make it easier to test and maintain.

7. **Code Formatting**: The use of single quotes (`’`) instead of standard single quotes (`'`) in the `open` function call and the `if __name__ == ’__main__’:` check is a typographical error.

8. **Import Statements**: Although this is minor, it is good practice to put import statements at the top of the file.

These issues can be addressed to make the script more robust, flexible, and maintainable.

Prompt 2:
“Refactor the code above for maximum readability, correctness, and safety. Provide only the refactored code.”

GPT 4.1 mini (Model 1):

```
import csv
import sys

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

def summarize_column(data, column_index):
    """
    Calculate the total and average of numeric values in the specified column.

    Args:
        data (list of list): CSV data including header row.
        column_index (int): Index of the column to summarize.

    Returns:
        tuple: (total, average) or (None, None) if calculation is not possible.
    """
    if not data or len(data) < 2:
        print("No data to summarize.", file=sys.stderr)
        return None, None

    values = []
    for i, row in enumerate(data[1:], start=2):  # start=2 to indicate the CSV
line number excluding header
        try:
            value = float(row[column_index])
            values.append(value)
        except IndexError:
            print(f"Warning: Row {i} is missing column {column_index}. Skipping.",
file=sys.stderr)
        except ValueError:
            print(f"Warning: Non-numeric value in row {i}, column {column_index}:
'{row[column_index]}'. Skipping.", file=sys.stderr)
```

```
    if not values:
        print("No valid numeric data found in the specified column.",
file=sys.stderr)
        return None, None

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

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

    # Attempt to use header to find column index for better readability
    header = data[0]
    column_name = 'value'  # Replace with actual column name if known
    if column_name in header:
        column_index = header.index(column_name)
    else:
        # Fall back to column index 1 if column name not found
        column_index = 1
        print(f"Column '{column_name}' not found in header. Using column index
{column_index}.", file=sys.stderr)

    total, avg = summarize_column(data, column_index)
    if total is not None and avg is not None:
        print(f"Total: {total}")
        print(f"Average: {avg}")
    else:
        print("Summary could not be calculated due to data issues.",
file=sys.stderr)

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

Codestral-2501 (Model 2):

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

def read_csv(filepath):
    """Reads CSV file and returns its data."""
    try:
        with open(filepath, 'r') as f:
            return list(csv.reader(f))
    except FileNotFoundError:
        print(f"Error: The file {filepath} does not exist.")
        sys.exit(1)
```

```
def summarize_column(data, index):
    """Summarizes a column in the CSV data."""
    if not data:
        print("Error: No data to process.")
        sys.exit(1)

    try:
        # Skip the header and convert values to float
        values = [float(row[index]) for row in data[1:]]
        total = sum(values)
        avg = total / len(values)
        return total, avg
    except (ValueError, IndexError):
        print(f"Error: Non-numeric data found in column {index} or the column
index is out of range.")
        sys.exit(1)

def main():
    """Main function to execute the script."""
    if len(sys.argv) != 3:
        print("Usage: python utils.py <filepath> <column_index>")
        sys.exit(1)

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

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

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

Zero-Shot:

Prompt:

“Analyze the following Python utility script. Identify any potential logic issues or design flaws, and refactor the code for readability, correctness, and safety. Return only the final code:

```
#utils.py - Script to parse and summarize numeric CSV files
import csv
def read_csv(filepath):
        with open(filepath, ’r’) as f:
```

~~~
        return [row for row in csv.reader(f)]

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

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

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

GPT 4.1 mini (Model 1):

```
# 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 content as a list of rows."""
    with open(filepath, 'r', newline='') as f:
        return list(csv.reader(f))

def summarize_column(data: List[List[str]], index: int) -> Tuple[float, float]:
    """
    Summarizes a numeric column by calculating the total and average,
    skipping the header row.
    """
    if len(data) < 2:
        raise ValueError("CSV data must contain at least one data row.")

    values = []
    for i, row in enumerate(data[1:], start=2):
        if index >= len(row):
            raise IndexError(f"Row {i} does not have index {index}.")
        try:
            value = float(row[index])
        except ValueError:
            raise ValueError(f"Row {i} has non-numeric value '{row[index]}' at
index {index}.")
```

```
        values.append(value)

    if not values:
        raise ValueError("No valid numeric data found in the specified column.")

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

def main():
    filepath = 'data.csv'
    try:
        data = read_csv(filepath)
        total, avg = summarize_column(data, 1)
    except (FileNotFoundError, ValueError, IndexError) as e:
        print(f"Error processing file '{filepath}': {e}")
        return
    print("Total:", total)
    print("Average:", avg)

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

Codestral-2501 (Model 2):

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

def read_csv(filepath):
    """Reads a CSV file and returns its content as a list of rows."""
    with open(filepath, 'r', newline='') as f:
        return list(csv.reader(f))

def summarize_column(data, index):
    """Summarizes a numeric column from the data, returning the total and 
average."""
    if not data:
        raise ValueError("The data list is empty.")

    # Assuming the first row is the header
    header = data[0]
    if len(header) <= index or not header[index].strip():
        raise IndexError(f"The column at index {index} does not exist or is not a 
valid column name.")

    values = []
    for row in data[1:]:
        try:
            value = float(row[index])
        except (ValueError, IndexError):
            raise ValueError(f"'Invalid number found at row index {data.index(row)
+ 1}, column index {index}")

        values.append(value)

    if not values:
        raise ValueError("No valid numeric values found in the specified column.")

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

def main():
    """Main function to execute the CSV reading and summarization."""
    filepath = 'data.csv'
    try:
        data = read_csv(filepath)
        if not data:
            raise ValueError("The CSV file is empty.")

        total, avg = summarize_column(data, 1)
        print("Total:", total)
        print("Average:", avg)
    except FileNotFoundError:
        print(f"The file '{filepath}' does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")

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

<u>Task 22:</u> (Max tokens: 400, Temperature: 1.0)

<u>Self-Consistency:</u>

Prompt:

“Complete the following file-processing script. The method to implement will clean each line, remove punctuation, and count word frequencies correctly. Return only the clean_line method code:

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

import string

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

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

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

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

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

GPT 4.1 mini (Model 1):

Output 1:

```
def clean_line(line):
    return line.translate(str.maketrans('', '', string.punctuation)).lower()
```

Output 2:

```
def clean_line(line):
    line = line.lower()
    return line.translate(str.maketrans('', '', string.punctuation))
```

Output 3:

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

Codestral-2501 (Model 2):

Output 1:

```
Certainly! To complete the clean_line method, we need to remove punctuation
from each line and convert it to lowercase. Here's the implementation for the
```

clean_line method:

```
def clean_line(line):
    # Remove punctuation using `str.translate` with `string.punctuation`
    translator = str.maketrans('', '', string.punctuation)
    line = line.translate(translator)

    # Convert to lowercase
    line = line.lower()

    return line
```

By applying these adjustments, the script will correctly process the lines, removing punctuation and converting all text to lowercase before counting word frequencies.

Output 2:

Certainly! Below is the implementation of the clean_line method to remove punctuation and convert the line to lowercase:

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

This clean_line method uses str.translate to remove all punctuation and then converts the line to lowercase to ensure uniformity in word counting.

Output 3:

To complete the clean_line method, we need to remove punctuation from each line and convert all the characters to lowercase. Here is the clean_line method code:

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

This function uses str.translate to remove all punctuation from the string and str.lower to convert the string to lowercase.

Chain of Thought:

Prompt:

“Complete the following file-processing script. The method to implement will clean each line, remove punctuation, and count word frequencies correctly. Return only the clean_line method code and explain it step-by-step:

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

import string

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

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

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

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

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

GPT 4.1 mini (Model 1):

```
def clean_line(line):
    # Step 1: Convert the entire line to lowercase to ensure consistent counting
    line = line.lower()
    # Step 2: Create a translation table that maps all punctuation characters to None
    translator = str.maketrans('', '', string.punctuation)
    # Step 3: Use the translate method to remove all punctuation from the line
    line = line.translate(translator)
    # Step 4: Strip leading/trailing whitespace and return the cleaned line
    return line.strip()
```

Explanation, step-by-step:

1. Convert to lowercase:
   `line = line.lower()`
   This ensures all words are counted in a case-insensitive manner, so “Apple” and “apple” count as the same word.
2. Create a translation table:
   `translator = str.maketrans('', '', string.punctuation)`
   This creates a mapping that will remove every punctuation character (from string.punctuation) when used with translate.
3. Remove punctuation:
   `line = line.translate(translator)`
   This applies the translation table to the line, effectively deleting all punctuation marks (e.g., commas, periods, exclamation points, etc.).
4. Strip whitespace:
   `return line.strip()`
   This removes any leading or trailing whitespace characters from the cleaned line before returning it.

This method ensures that each line is cleaned properly before word counting, improving the accuracy of the word frequency analysis.

Codestral-2501 (Model 2):

Here is the implementation of the clean_line method along with an explanation of how it works:

```
def clean_line(line):
  # Remove leading and trailing whitespace
  line = line.strip()
  # Convert to lowercase
  line = line.lower()
  # Remove punctuation using str.translate and str.maketrans
  translator = str.maketrans('', '', string.punctuation)
  line = line.translate(translator)
  return line
```

Step-by-Step Explanation:

1. Remove Leading and Trailing Whitespaces:

`line = line.strip()`

- This method removes any leading and trailing whitespace characters (spaces, tabs, newlines) from the line. This step is crucial to ensure that any extra spaces around the actual text do not interfere with processing.

2. Convert to Lowercase:

`line = line.lower()`

- Converting the entire line to lowercase makes the word counting case-insensitive, meaning "Word" and "word" will be treated as the same word.

3. Remove Punctuation:

```
translator = str.maketrans('', '', string.punctuation)
line = line.translate(translator)
```

- str.maketrans('', '', string.punctuation) creates a translation table that maps each punctuation character to None. This effectively removes all punctuation marks.
- line.translate(translator) then uses this translation table to replace punctuation characters in the line with None, thereby removing them.

Complete Code with clean_line Method

Here is the complete file_processor.py script with the clean_line method included:

```
# 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 leading and trailing whitespaces
  line = line.strip()
  # Convert to lowercase
  line = line.lower()
  # Remove punctuation using str.translate and str.maketrans
  translator = str.maketrans('', '', string.punctuation)
  line = line.translate(translator)
  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()
```

l

This script should now correctly clean each ine of the input file by removing punctuation and converting the text to lowercase, and count the word frequencies accordingly.