Part 2: Practical Implementation

Task 1: AI-Powered Code Completion

Python function to sort a list of dictionaries by a specific key

AI-Suggested Code (GitHub Copilot):

Python

```
def sort_list_of_dictionaries_ai(data, key_to_sort_by, reverse=False):
    """
    Sorts a list of dictionaries by a specified key using AI suggestion.

    Args:
        data (list): The list of dictionaries to sort.
        key_to_sort_by (str): The key to sort the dictionaries by.
        reverse (bool, optional): If True, sort in descending order.
Defaults to False.

    Returns:
        list: A new sorted list of dictionaries.
    """
    return sorted(data, key=lambda x: x[key_to_sort_by], reverse=reverse)

# Example Usage:
my_list_ai = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35}
]

sorted_by_age_ai = sort_list_of_dictionaries_ai(my_list_ai, "age")
print("AI-Suggested - Sorted by age (ascending):", sorted_by_age_ai)

sorted_by_name_desc_ai = sort_list_of_dictionaries_ai(my_list_ai, "name",
reverse=True)
print("AI-Suggested - Sorted by name (descending):",
sorted_by_name_desc_ai)
```

Manual Implementation:

Python

```
def sort_list_of_dictionaries_manual(data, key_to_sort_by, reverse=False):
    """
    Sorts a list of dictionaries by a specified key using manual
implementation.

    Args:
        data (list): The list of dictionaries to sort.
        key_to_sort_by (str): The key to sort the dictionaries by.
        reverse (bool, optional): If True, sort in descending order.
Defaults to False.
```

```
    Returns:
        list: A new sorted list of dictionaries.
    """
    # Create a copy to avoid modifying the original list if needed
    sorted_data = list(data)

    # Use the list's sort() method with a lambda function as the key
    sorted_data.sort(key=lambda x: x[key_to_sort_by], reverse=reverse)
    return sorted_data

# Example Usage:
my_list_manual = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35}
]

sorted_by_age_manual = sort_list_of_dictionaries_manual(my_list_manual,
"age")
print("Manual Implementation - Sorted by age (ascending):",
sorted_by_age_manual)

sorted_by_name_desc_manual =
sort_list_of_dictionaries_manual(my_list_manual, "name", reverse=True)
print("Manual Implementation - Sorted by name (descending):",
sorted_by_name_desc_manual)
```

Analysis:

Both the AI-suggested code and the manual implementation achieve the desired outcome of sorting a list of dictionaries by a specific key. However, the **AI-suggested version is generally more efficient and Pythonic.**

The AI-suggested code directly uses Python's built-in `sorted()` function. This function returns a *new* sorted list, leaving the original list unchanged. It's often preferred for its immutability, which can prevent unintended side effects in larger codebases. The `sorted()` function internally uses Timsort, a highly optimized hybrid sorting algorithm (a combination of merge sort and insertion sort) that performs very well on various real-world data sets, offering an average and worst-case time complexity of O(NlogN).

The manual implementation, while functionally correct, uses the `list.sort()` method. While `list.sort()` also uses Timsort and has the same time complexity (O(NlogN)), it sorts the list *in-place*, modifying the original list directly and returning `None`. To match the behavior of `sorted()` (returning a new list), a copy of the list (`list(data)`) must be created first. This extra step of creating a copy adds a slight overhead in terms of both time and memory.

For most typical scenarios and data sizes, the performance difference between the two approaches (after considering the copying for `list.sort()`) will be negligible. However, the `sorted()` function (as used by the AI) is often considered more idiomatic Python for returning a sorted *copy*, leading to cleaner and potentially safer code as it avoids accidental modification of

the input data. If in-place sorting is explicitly desired, `list.sort()` is the direct and efficient choice. In summary, the AI-suggested code is marginally more efficient due to avoiding the explicit list copy and often aligns better with functional programming principles of immutability.