Performance Comparison and Documentation

Efficiency Analysis:

Criteria	Copilot Version	Manual Version
Readability	✅ Very clean and Pythonic	❌ Less readable
Performance	✅ Uses optimized built-in sort	❌ $O(n^2)$ complexity
Edge-case safe	✅ Handles missing keys gracefully	✅ Also uses .get() with default
Developer effort	✅ Requires only 1 comment prompt	❌ Manual coding, debugging needed

---

GitHub Copilot's suggestion for sorting a list of dictionaries by a given key was concise, efficient, and Pythonic. It used Python's built-in sorted() function with a lambda function for key access, resulting in clean and performant code. Copilot's ability to understand intent from a natural-language comment (# Function to sort...) demonstrates its strength in rapid prototyping and eliminating boilerplate logic.

In contrast, the manual implementation used a bubble sort approach, which, while logically correct, is inefficient ($O(n^2)$ time complexity) and more verbose. This version was included for learning purposes but would not scale well with large datasets.

The AI-suggested code not only required less effort but also adhered to Python best practices, showcasing how such tools can significantly reduce development time and increase reliability — especially for common utility functions.

However, it's important to validate AI-suggested code to ensure correctness in more complex scenarios. Developers should use these

tools as assistants, not as blind replacements for understanding algorithms.