In Challenge #17, the objective was to explore the implementation of bubble sort using a systolic array architecture and to evaluate its performance across different input sizes. The first step involved understanding the architecture required for such a system. Since bubble sort operates by repeatedly comparing and swapping adjacent elements, a 1D systolic array with one processing element (PE) per data element was designed. Each PE was capable of holding a value and performing a compare-and-swap operation with its immediate neighbor. During each clock cycle (or pass), the values propagate through the array, and over multiple cycles, the largest elements "bubble" to the end, mimicking the behavior of traditional bubble sort in hardware.

The software implementation of this design was done in Python. A ProcessingElement class was created to simulate the systolic behavior, where each PE acted independently in a coordinated fashion during alternating even and odd passes. This simulated the hardware-like data movement and local processing found in systolic arrays. To evaluate its performance, the bubble sort simulation was tested on datasets of varying sizes — specifically 10, 100, 1000, and 10000 elements. The time taken to complete sorting was measured and plotted using matplotlib to visualize the scalability of the systolic array approach.

The resulting graph demonstrated the expected quadratic time complexity of bubble sort, with execution time increasing sharply, particularly between the 1000 and 10000 element tests. The steep curve confirmed the inefficiency of bubble sort for large data sizes, even in a systolic framework. One of the main challenges faced during this project was simulating the behavior of a systolic array — which is inherently parallel and hardware-timed — in a sequential software environment. Additionally, initial doubts regarding the use of TPUs or GPUs for acceleration were clarified using GPT, which explained that this implementation was better suited to CPU execution unless rewritten in a highly parallelized framework.

GPT played a crucial role in accelerating the development process. It helped design the systolic array structure, provided a clean and modular Python implementation, and assisted in visualizing performance trends. Moreover, it provided insight into whether a hardware accelerator like a TPU would benefit the implementation, reinforcing the understanding that this bubble sort variant, being inherently sequential, did not lend itself well to such acceleration. Overall, this challenge deepened the understanding of systolic architectures and offered practical experience in simulating hardware-like data flows in software.