Challenge #17: Systolic Array Bubble Sort - Documentation

# 1. User Prompts and Key Questions

- Give me the solution for challenge 17

- How to run the file

- How to include performance benchmarking

- Can you give documentation of this chat also include all the prompts I have asked and what are the results that didn't work and how did we overcome menyion each and every line of vibe coding do mention all the questions I have asked to get the final result in a docx file

# 2. Summary of Implemented Solution

We implemented a systolic-array-inspired Bubble Sort simulator in Python. The systolic model was emulated using a simple loop-based structure where each pass simulates a phase in which processing elements (PEs) compare and swap values if needed.

# 3. Vibe Coding Implementation Steps

1. Designed the systolic architecture where each element acts as a Processing Element (PE).

2. Implemented a basic Bubble Sort using loop constructs to simulate systolic data propagation.

3. Benchmarked the algorithm using various input sizes (10, 100, 1000, 2000).

4. Used `time.time()` for benchmarking and `matplotlib` for visualization.

5. Added optional CSV export of results to integrate with reports or further visualization.

# 4. Key Python Code Used

Function: systolic\_bubble\_sort

def systolic\_bubble\_sort(arr):  
 n = len(arr)  
 pe\_array = list(arr)  
 for phase in range(n):  
 for i in range(n - 1):  
 if pe\_array[i] > pe\_array[i + 1]:  
 pe\_array[i], pe\_array[i + 1] = pe\_array[i + 1], pe\_array[i]  
 return pe\_array

Benchmarking Function: benchmark\_systolic\_sort

def benchmark\_systolic\_sort(sizes, trials=3):  
 times = []  
 for size in sizes:  
 avg\_time = 0  
 for \_ in range(trials):  
 arr = np.random.randint(0, 1000000, size=size)  
 start = time.time()  
 systolic\_bubble\_sort(arr)  
 end = time.time()  
 avg\_time += (end - start)  
 avg\_time /= trials  
 print(f"Size: {size}, Avg Time: {avg\_time:.6f} sec")  
 times.append(avg\_time)  
 return times

# 5. Issues Encountered and Solutions

There were no syntax or logic errors encountered during this coding process. However, potential inefficiencies for large inputs were addressed by restricting the benchmark to moderate input sizes (up to 2000 elements), due to Bubble Sort's O(n^2) time complexity.