4. Analysis & Comparison

4.1 Route Quality Comparison

The Traveling Salesman Problem (TSP) was solved using two different approaches:

1. A **Classical Algorithm** (e.g., Dynamic Programming, Branch-and-Bound, or Nearest Neighbor).
2. A **Self-Organizing Map (SOM)** approach, which is a neural network-based heuristic method.

Method	**Final Route**	**Total Distance**
Classical TSP Algorithm	(Exact or near-optimal route)	(Shortest distance possible)
SOM-Based Approach	(Heuristic-based route)	(May be slightly longer but faster to compute)

- The classical method guarantees an optimal or near-optimal solution, depending on the algorithm used.
- The SOM approach approximates a good solution but might not always be optimal.

4.2 Complexity Discussion

- **Classical TSP Algorithm Complexity**:
    - **Brute Force**: O(n!) (Exponential, infeasible for large n).
    - **Dynamic Programming (Held-Karp Algorithm)**: ($n^2\ 2^n$) (Still exponential but better).
    - **Branch-and-Bound or Nearest Neighbor**: More efficient but may not always find the optimal path.
- **SOM-Based TSP Complexity**:
    - Approximate time complexity is O(N·T), where:
        - N = Number of neurons (cities).
        - T = Number of training iterations.
    - Slower for small n, but scales better for large instances.

4.3 Practical Considerations

Scenario	Best Approach	Reason
Small Number of Cities (≤ 10)	**Classical Algorithm**	Finds an exact solution in reasonable time.
Medium-Sized Problem (10 - 100 cities)	**Depends on Constraints**	If accuracy is critical, classical methods may be better. If speed is needed, SOM is preferable.
Large-Scale TSP (> 100 cities)	**SOM or Metaheuristics**	Classical methods become too slow, and heuristics provide a good approximation.

4.4 Extensions and Improvements

1. **Hybrid Approach**:
    - Use a **Genetic Algorithm (GA)** to refine the SOM-generated route for better accuracy.
    - Use a **Simulated Annealing (SA)** step after SOM to fine-tune the solution.
2. **Improved SOM Parameters**:
    - Adjust learning rate and neighborhood function for better convergence.
    - Implement an **adaptive learning rate** that decreases over time.
3. **Parallelization**:
    - Speed up the process by running **SOM iterations on multiple cores**.
    - Use **GPU acceleration** for large-scale problems.