Distributed k-NN System Using gRPC - Project Report

# Introduction

This project implements a distributed k-Nearest Neighbors (k-NN) algorithm using gRPC for efficient computation across multiple servers. The system is designed to handle large datasets by distributing the data across several servers. Each server computes the k-nearest neighbors for a subset of the dataset, and the results are aggregated for the client query.

# System Design

## gRPC and Distributed Computing

gRPC facilitates distributed computing by enabling remote procedure calls (RPCs) between servers and clients across different machines. In this system, multiple gRPC servers hold partitions of the dataset. The client sends a k-NN query to these servers, and each server computes the nearest neighbors within its dataset subset. The results are then aggregated and returned to the client.

## Comparison: gRPC vs MPI

### Communication Models

gRPC uses a client-server model where clients make requests to remote servers using RPCs, and the server handles the requests and responds with the results. On the other hand, MPI (Message Passing Interface) uses peer-to-peer communication, where nodes exchange messages directly with each other without a central server.

### Usability

gRPC is easier to use for developing scalable, distributed systems due to its built-in support for multiple programming languages, easy service definition using Protocol Buffers, and automatic generation of client and server stubs. MPI is more suitable for scientific computing and tightly-coupled parallel applications but requires more manual handling of message passing.

### Scalability

gRPC scales well in large systems where services can be distributed over several servers, enabling easy load balancing and fault tolerance. MPI can also scale, but it is more efficient in tightly-coupled, high-performance computing environments.

# Performance Analysis

The performance of the distributed k-NN system was evaluated based on the size of the dataset and the value of k. For small datasets and low values of k, the system performs efficiently with minimal delay. As the dataset size increases, the distributed nature of the system helps maintain performance, as each server only computes nearest neighbors for its subset of data.

# Distance Calculation: Euclidean Metric

For this system, the Euclidean distance metric is used to calculate the distance between the query point and each data point in the dataset. The following dataset is used for demonstration:  
[[1.0, 2.0], [2.0, 3.0], [3.0, 4.0], [1.5, 2.5], [4.0, 5.0], [5.0, 6.0]]

Given a query point data\_point = [1, 2] and k = 2, the Euclidean distance between the query point and each point in the dataset is calculated as follows:

1. 1. Distance between [1, 2] and [1.0, 2.0]: 0
2. 2. Distance between [1, 2] and [2.0, 3.0]: 1.41
3. 3. Distance between [1, 2] and [3.0, 4.0]: 2.83
4. 4. Distance between [1, 2] and [1.5, 2.5]: 0.71
5. 5. Distance between [1, 2] and [4.0, 5.0]: 4.24
6. 6. Distance between [1, 2] and [5.0, 6.0]: 5.66

The two nearest neighbors for the query point [1, 2] with k = 2 are:  
• Neighbor 1: [1.0, 2.0] with distance 0  
• Neighbor 2: [1.5, 2.5] with distance 0.71