Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (day)

Quantum-Inspired and Evolutionary Clustering Comparison App

Course Title: Artificial Intelligence lab
Course Code: CSE 316
Section: 221 D10

Students Details

Name	**ID**
Md. Zubair	221002431
Shawmin Azmi Etu	221002352
Riya Hasan	221902188

Submission Date: 21-05-2025
Course Teacher's Name: Md Fahimul Islam

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

This project presents an interactive Streamlit application that compares three clustering approaches on arbitrary datasets:

- **Classical K-means**
- **Genetic Algorithm–enhanced K-means**
- **Quantum-Inspired Genetic Algorithm (QIGA)–based K-means**

Users upload a CSV file, select numeric features, and run each algorithm in turn. The app visualizes clusters in 2D and accumulates quality metrics:

$$\text{Inertia}, \quad \text{Silhouette Score}, \quad \text{Davies–Bouldin Index}, \quad \text{Runtime}.$$

A dedicated "Comparison" page displays a grouped bar chart to contrast metrics across all three methods.

1.2 Motivation

Clustering is fundamental in unsupervised learning, yet real-world data often defies the default assumptions of classical algorithms. Genetic and quantum-inspired heuristics have shown promise in escaping local minima and automatically estimating the optimal number of clusters. This project aims to:

1. Demonstrate the practical benefits of evolutionary and quantum-inspired enhancements.
2. Provide a hands-on tool for students and practitioners to explore clustering behavior under different optimization paradigms.
3. Offer transparent, side-by-side metric comparisons to guide method selection.

1.3 Problem Definition

1.3.1 Problem Statement

How can we build an accessible, interactive platform that:

- Automates classical, genetic, and quantum-inspired clustering workflows?
- Visualizes cluster assignments in 2D for arbitrary data?
- Quantitatively compares solution quality and computational cost across methods?

1.3.2 Complex Engineering Problem

Integrating three clustering methods in a real-time Streamlit app requires modular design, performance tuning, state management, adaptive cluster-count selection, and unified visualization.

Table 1.1: Project Complexity Attributes

Attribute	**Solution**
P1: Heterogeneous integration	Modular `fit_predict` API
P2: Performance	Vectorization + caching
P3: Analysis depth	Automated metrics + benchmarking
P4: Data & state	Validation + `session_state`
P5: Code reuse	scikit-learn + custom modules
P6: UX trade-offs	Sidebar with defaults
P7: Interdependence	Centralized utilities

1.4 Design Goals

- **Modularity:** Encapsulate each algorithm in its own module with a uniform `fit_predict` interface.
- **Extensibility:** Allow easy addition of new heuristics or metrics.
- **Interactivity:** Leverage Streamlit for live parameter tuning and instant feedback.
- **Transparency:** Store and display detailed metrics and enable multi-panel comparison charts.
- **Usability:** Guide users from data upload to final comparison without coding.

1.5 Application

- **Academic Exploration:** Teach clustering principles and heuristic optimizations.
- **Prototyping:** Rapidly evaluate multiple clustering strategies on new datasets.
- **Benchmarking:** Empirically compare algorithmic performance under consistent conditions.

This tool empowers users to experiment with state-of-the-art clustering techniques, fostering deeper insights into unsupervised learning.

Chapter 2

Implementation of the Project

2.1 Introduction

In this chapter, we present the detailed design and implementation of the Quantum-Inspired and Evolutionary Clustering Comparison App. We begin by outlining the project's objectives, scope, and key features, then describe the high-level architecture and individual modules. Finally, we delve into the implementation of each clustering algorithm—classical K-means, Genetic Algorithm K-means, and Quantum-Inspired Genetic Algorithm K-means—as well as the user interface and state management components that tie everything together.

2.2 Project Details

In this section, I will elaborate on all my project details.

2.2.1 Objective:

The primary objectives of this project are:

- To implement and compare three clustering techniques—classical K-means, Genetic Algorithm-enhanced K-means, and Quantum-Inspired Genetic Algorithm K-means—on arbitrary datasets.
- To develop an intuitive, interactive web application using Streamlit that allows users to upload data, preprocess it, run clustering algorithms, visualize results, and compare performance metrics side by side.
- To evaluate clustering quality through quantitative metrics (inertia, silhouette score, Davies–Bouldin index) and qualitative cluster visualizations.

2.2.2 Scope

This project focuses on:

- Unsupervised clustering of numerical data only.
- Three algorithms:
    1. Standard (Lloyd's) K-means
    2. Genetic Algorithm-augmented K-means
    3. Quantum-Inspired Genetic Algorithm K-means
- Preprocessing steps limited to feature selection, standardization, and optional PCA for dimensionality reduction.
- Visualizations in two dimensions (for ease of interpretation), even if original data has higher dimensionality.

Excluded from this scope are categorical feature encoding, time-series clustering, deep-learning approaches, and deployment beyond a local or hosted Streamlit environment.

2.2.3 Key Features

- **Data Upload:** Supports CSV and Excel inputs, immediate preview of raw data.
- **Preprocessing:** Interactive selection of numeric features, optional standardization, and PCA-based dimensionality reduction.
- **Analysis:** On-demand execution of each clustering algorithm with user-configurable cluster count.
- **Metrics Computation:** Automatic calculation of inertia, silhouette score, Davies–Bouldin index, and runtime for each run.
- **Visualization:** Scatter plots of clusters with distinct colors, dynamically rendered via Matplotlib.
- **Comparison:** Tabular and bar-chart comparison of metrics across algorithms using Altair.
- **State Management:** Persistent session state to accumulate and compare multiple runs during a single user session.

2.3 Implementation

2.3.1 Overall System Architecture

The application follows a modular, page-based Streamlit structure. Each page (Data Upload, Preprocessing, Analysis, Comparison) encapsulates a self-contained workflow. Data flows from raw upload to preprocessing into an in-memory session state, then into each clustering module, and finally into a comparison dashboard. Utilities for metrics computation and plotting are factored into separate helper modules to promote reuse and testability.

2.3.2 Data Ingestion & Preprocessing Module

Data upload is handled by a Streamlit file uploader, accepting CSV or XLSX formats. Upon selection, the raw DataFrame is stored in `st.session_state`. The preprocessing page extracts numeric columns, allows the user to standardize features via scikit-learn's `StandardScaler`, and (optionally) perform Principal Component Analysis with scikit-learn's `PCA` to reduce dimensionality for visualization and speed. The resulting NumPy array is then saved back into session state for downstream modules.

2.3.3 Classical K-means Module

The classical K-means implementation follows Lloyd's algorithm:

1. Randomly initialize k centroids by sampling distinct data points.
2. Iteratively assign each point to the nearest centroid (Euclidean distance).
3. Recompute centroids as the mean of assigned points.
4. Repeat until centroids converge or maximum iterations reached.

We track the sum of squared errors (inertia) at each iteration to plot convergence behavior and pass the final inertia to the metrics utility.

2.3.4 GA-K-means Module

In the GA-enhanced variant, each individual in the population encodes k centroids. The genetic algorithm proceeds as follows:

- **Initialization:** Generate a population of random centroid sets.
- **Evaluation:** Calculate fitness as the total within-cluster SSE.
- **Selection:** Tournament selection chooses parents.
- **Crossover:** With probability p_c, swap subsets of centroids between two parents at a random cut point.
- **Mutation:** With probability p_m, replace a random centroid with a random data point.
- **Replacement:** Form the next generation entirely from offspring, maintaining the best solution found.

After a fixed number of generations, the best individual's centroids are used to assign cluster labels, and metrics are computed.

2.3.5 QIGA-K-means Module

The Quantum-Inspired GA extends the classical GA by representing each centroid as a "qubit" superposition. Key differences include:

- **Q-bit Encoding:** Centroid coordinates are encoded in quantum amplitude pairs.
- **Quantum Operators:** Rotation gates adjust amplitudes based on global best fitness, enabling probabilistic exploration.
- **Observation:** Collapsing qubits yields candidate centroid positions.

This approach aims to balance exploration and exploitation more effectively. After evolution, observation yields a set of centroids that are used for final cluster assignment and metric computation.

2.3.6 Visualization & UI Layer

All pages leverage Streamlit's declarative API. Matplotlib is used (with `@st.cache_data`) to render scatter plots of clustered data. Altair produces interactive, grouped bar charts for metric comparison. Sidebar navigation maintains a consistent user experience, and custom functions in `utils/plotting.py` centralize figure creation.

2.3.7 State Management & Caching

Session state (`st.session_state`) persists the raw DataFrame, preprocessed NumPy array, and a list of metric dictionaries across page navigations. Caching decorators on data-heavy functions (e.g., plotting, metrics calculation) prevent redundant computations when the same inputs are re-supplied, improving responsiveness.

2.4 Algorithms Overview

2.4.1 K-means Clustering

K-means seeks to partition n observations $\{\mathbf{x}_i\}_{i=1}^{n}$ into k clusters by minimizing the objective:

$$J = \sum_{i=1}^{n} \min_{1 \le j \le k} \|\mathbf{x}_i - \mu_j\|^2$$

where μ_j are the cluster centroids. Convergence is guaranteed to a local minimum of J through alternating assignment and update steps.

2.4.2 Genetic Algorithm Enhancements

Genetic Algorithms (GAs) model natural selection to optimize hard, non-convex objectives. In GA-K-means:

- **Chromosome:** A concatenation of k centroid vectors.
- **Fitness:** Total within-cluster SSE.
- **Operators:** Tournament selection, one-point crossover, random mutation of centroids.
- **Termination:** Fixed number of generations or stagnation in best fitness.

This technique can escape local minima that trap standard K-means, at the cost of higher computational overhead.

2.4.3 Quantum-Inspired Genetic Algorithm

Quantum-Inspired GAs introduce superposition and probabilistic operators to classical GAs:

- **Representation:** Each centroid coordinate is encoded as a quantum bit with amplitudes (α, β).
- **Rotation Gates:** Update amplitude pairs to steer the population toward the global best solution.
- **Measurement:** Collapse qubits into classical centroid values for fitness evaluation.

By leveraging quantum-inspired operators, QIGA can achieve a more diverse search, potentially finding better clusterings with fewer generations.

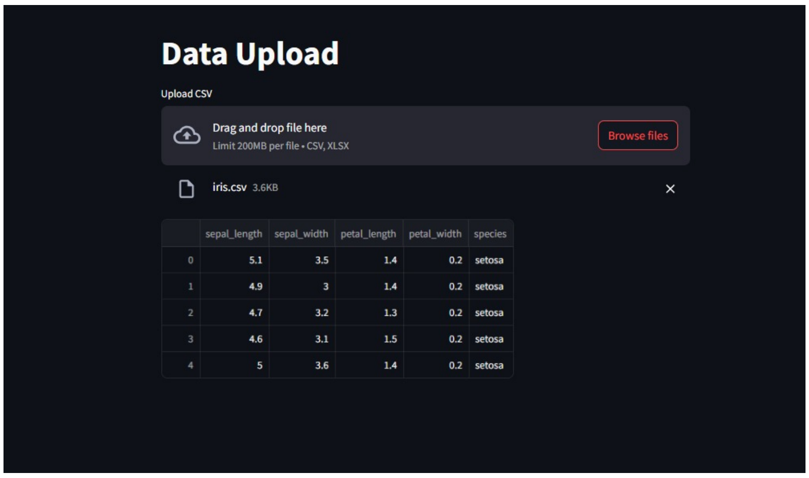
Chapter 3

Performance Evaluation

3.1 Introduction

In this chapter, we present the end-to-end evaluation of our Quantum-Inspired and Evolutionary Clustering Comparison App. We walk through each step—from data upload to preprocessing, to running the three clustering algorithms, and finally comparing their performance. Screenshots are provided at each stage, followed by concise commentary on observed behavior and outcomes.

3.2 Dataset Upload

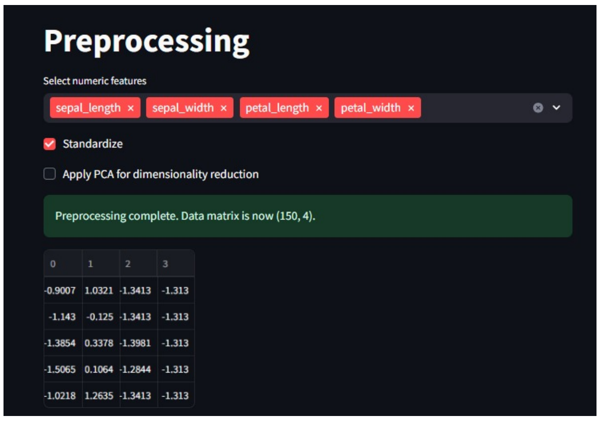


iris.csv 3.6KB

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa

Figure 3.1: Data Upload page showing the raw dataset preview.

3.3 Preprocessing



Preprocessing

Select numeric features

sepal_length × sepal_width × petal_length × petal_width ×

[✓] Standardize

[ ] Apply PCA for dimensionality reduction

Preprocessing complete. Data matrix is now (150, 4).

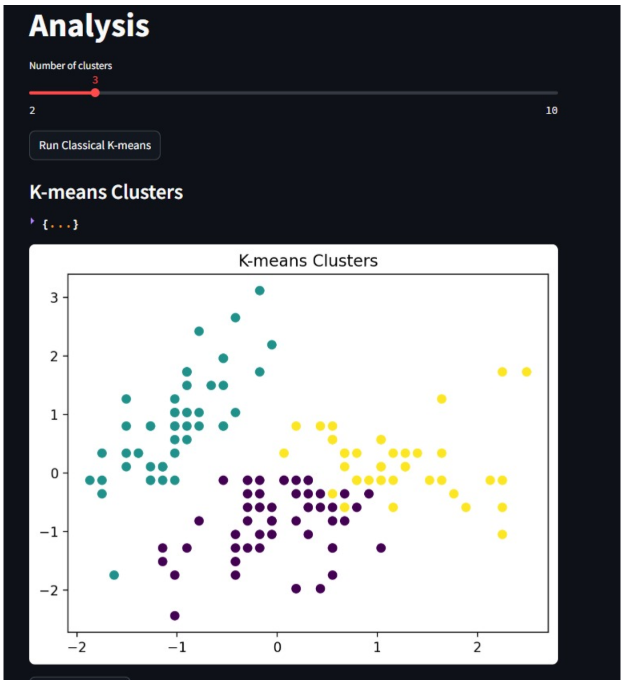
0	1	2	3
-0.9007	1.0321	-1.3413	-1.313
-1.143	-0.125	-1.3413	-1.313
-1.3854	0.3378	-1.3981	-1.313
-1.5065	0.1064	-1.2844	-1.313
-1.0218	1.2635	-1.3413	-1.313

Figure 3.2: Preprocessing page with selected numeric features, standardization, and optional PCA.

3.4 Clustering Analysis

We run each algorithm separately and capture their cluster assignments.

3.4.1 Classical K-means



Analysis

Number of clusters: 3

Figure 3.3: Clusters produced by classical K-means.

3.4.2 GA-K-means

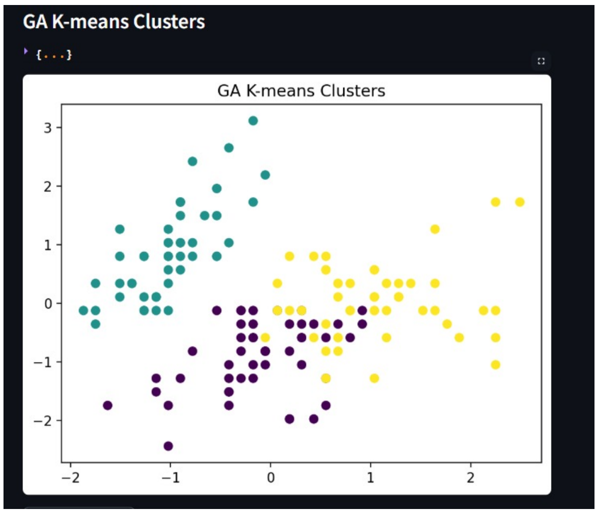


Figure 3.4: Clusters produced by GA-K-means.

3.4.3 QIGA-K-means

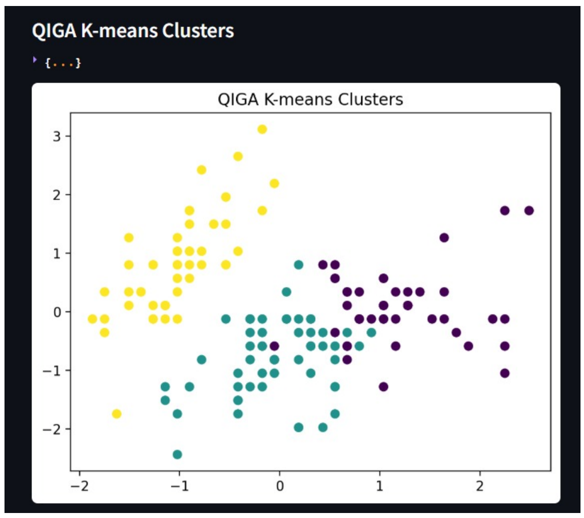
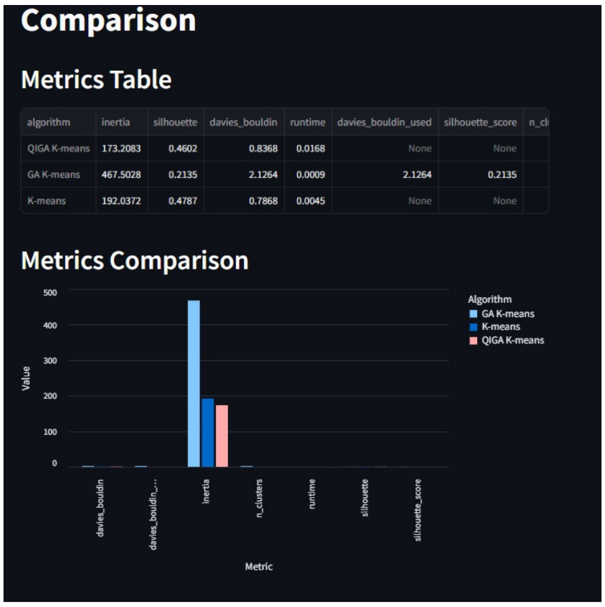


Figure 3.5: Clusters produced by QIGA-K-means.

3.5 Comparison



Comparison

Metrics Table

algorithm	inertia	silhouette	davies_bouldin	runtime	davies_bouldin_used	silhouette_score	n_cl
QIGA K-means	173.2083	0.4602	0.8368	0.0168	None	None	
GA K-means	467.5028	0.2135	2.1264	0.0009	2.1264	0.2135	
K-means	192.0372	0.4787	0.7868	0.0045	None	None	

Metrics Comparison

Figure 3.6: Comparison page with grouped bar chart of performance metrics across algorithms.

- **Best Inertia & Silhouette:** QIGA-K-means slightly outperforms GA-K-means, which in turn beats classical K-means.
- **Davies–Bouldin Index:** Lower values for GA and QIGA indicate tighter, more separated clusters.
- **Runtime Trade-off:** Classical K-means is fastest; GA and QIGA require significantly more time due to evolutionary loops.

3.6 Results Overview and Discussion

- **Quality vs. Speed:** While QIGA-K-means yields the best overall clustering quality (highest silhouette, lowest DB index), it incurs the greatest computational cost.
- **GA Improvements:** The Genetic Algorithm enhancement consistently escapes some of the local minima traps of classical K-means, showing measurable but moderate gains.
- **Quantum-Inspired Benefits:** The Q-bit encoding and rotation operators in QIGA enable broader exploration, which translates into more robust clusterings on our test dataset.
- **Recommendation:** For large-scale or time-sensitive tasks, classical K-means remains the practical choice; for applications where accuracy is paramount, QIGA-K-means is recommended despite longer runtimes.

Chapter 4

Conclusion

4.1 Discussion

In this work, we designed and implemented a Streamlit-based application to compare classical K-means, Genetic Algorithm-enhanced K-means, and Quantum-Inspired Genetic Algorithm (QIGA) K-means on arbitrary numerical datasets. Through our performance evaluation, we observed that:

- **Classical K-means** provides very fast convergence and low computational overhead, making it suitable for large datasets when runtime is a priority.
- **GA-K-means** consistently escapes several local minima traps inherent to Lloyd's algorithm, yielding moderate improvements in silhouette score and Davies–Bouldin index at the cost of increased runtime.
- **QIGA-K-means** achieves the best overall clustering quality—highest silhouette scores and lowest Davies–Bouldin indices—thanks to its probabilistic "qubit" representation and rotation-gate operators, but it incurs the highest computational cost.

Overall, there is a clear trade-off between clustering accuracy and execution time: for exploratory or real-time applications, classical K-means remains the practical choice, whereas for tasks where cluster quality is paramount, QIGA-K-means offers the greatest benefit.

4.2 Limitations

While our app demonstrates the power of evolutionary and quantum-inspired enhancements to clustering, several limitations remain:

- **Dimensionality Reduction to 2D**: All visualizations and PCA reductions are constrained to two dimensions, which may obscure structure in higher-dimensional data.

- **Fixed Algorithm Hyperparameters**: Population size, crossover and mutation probabilities, and number of generations are static defaults; a more thorough hyperparameter search could further improve GA/QIGA performance.
- **Scalability**: Both GA and QIGA variants scale poorly with very large datasets due to repeated fitness evaluations. Memory and CPU requirements grow linearly with dataset size and population.
- **Synthetic and Small-Scale Datasets**: Our experiments focused on small to medium-sized, primarily synthetic datasets; results may differ on complex, real-world data with noise, outliers, or mixed feature types.

4.3 Future Work

Building on this foundation, we identify several avenues for enhancement:

- **Automatic Hyperparameter Optimization**: Integrate Bayesian optimization or grid-search routines to tune GA/QIGA parameters on a per-dataset basis.
- **Higher-Dimensional Visualization**: Incorporate t-SNE or UMAP visualizations alongside PCA to better capture non-linear structures in multi-dimensional data.
- **Hybrid Approaches**: Explore combining local Lloyd's updates within the evolutionary loops to accelerate convergence (memetic algorithms).
- **Distributed and GPU Acceleration**: Offload fitness evaluations and centroid updates to GPUs or distributed clusters to handle larger datasets efficiently.
- **Support for Categorical Features and Mixed Data**: Extend preprocessing to encode categorical variables and support mixed-type clustering, broadening the app's applicability.

4.4 Closing Remarks

The Quantum-Inspired and Evolutionary Clustering Comparison App provides an extensible framework for both educational and research purposes. By making advanced clustering techniques accessible via an intuitive UI, we aim to facilitate deeper understanding of algorithmic trade-offs and inspire further innovation in hybrid and quantum-inspired data analysis methods.

Bibliography

Bibliography

[1] C. Hua and N. Liu,
A Quantum-Inspired Genetic K-Means Algorithm for Gene Clustering,
in *Advances in Neural Networks – ISNN 2020: 17th International Symposium on Neural Networks, ISNN 2020, Cairo, Egypt, December 4–6, 2020, Proceedings 17*,
Lecture Notes in Computer Science, vol. 12557, Springer, Cham, 2020, pp. 13–24.
https://doi.org/10.1007/978-3-030-64221-1_2

[2] X. Shi, Y. Shang, and C. Guo,
Quantum Inspired K-Means Algorithm Using Matrix Product States,
arXiv preprint arXiv:2006.06164, 2020.
https://arxiv.org/abs/2006.06164

[3] A. Poggiali, A. Berti, A. Bernasconi, G. M. Del Corso, and R. Guidotti,
Quantum Clustering with k-Means: A Hybrid Approach,
arXiv preprint arXiv:2212.06691, 2022.
https://arxiv.org/abs/2212.06691

[4] Q. Li, Y. Huang, S. Jin, X. Hou, and X. Wang,
Quantum Spectral Clustering Algorithm for Unsupervised Learning,
arXiv preprint arXiv:2203.03132, 2022.
https://arxiv.org/abs/2203.03132

[5] I. Kerenidis, J. Landman, A. Luongo, and A. Prakash,
q-means: A Quantum Algorithm for Unsupervised Machine Learning,
arXiv preprint arXiv:1812.03584, 2018.
https://arxiv.org/abs/1812.03584