Received: 18 June 2025 | Revised: 8 August 2025 | Accepted: 16 September 2025 | Published online: 16 October 2025

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2026, Vol. 5(2) 258–273
DOI: 10.47852/bonviewJCCE52026511

BON VIEW PUBLISHING

Adaptive Gaussian-Based Kernel K-Means: Scalable Adaptive Kernel-Based Clustering for Big Data

Bakhshali Bakhtiyarov[1,*], Aynur Jabiyeva[1], Anakhanim Mutallimova[1], Rukhsara Novruzova[1] and Mahabbat Khudaverdiyeva[1]

[1] *Department of Instrumentation Engineering, Azerbaijan State Oil and Industry University, Azerbaijan*

Abstract: In the field of big data, clustering remains a fundamental component of data mining and knowledge discovery, particularly in high-volume, heterogeneous environments. Although traditional clustering algorithms have been widely used, they often exhibit poor scalability, limited robustness, and sensitivity to noise when applied to large industrial-scale datasets. To address these limitations, this paper proposes a Spark-based clustering framework that incorporates an Adaptive Gaussian-Based Kernel K-Means (A-GBK-Means) algorithm, designed to mitigate performance degradation in the presence of noisy and nonhomogeneous data. The A-GBK-Means algorithm integrates fuzzy logic with adaptively determined kernel widths, using local density estimation, thereby enhancing both clustering accuracy and flexibility. Extensive experiments were conducted on real-world industrial sensor data as well as synthetic benchmarks, and performance was evaluated using multiple metrics, including execution time, Silhouette Score, Davies–Bouldin Index, and noise resilience. The comparative analysis demonstrates that the A-GBK-Means algorithm consistently outperforms classical approaches—namely K-Means, Density-Based Spatial Clustering of Applications with Noise, Fuzzy C-Means, and standard GBK-Means—in terms of clustering quality and computational efficiency. Furthermore, the proposed framework exhibits superior scalability, owing to its distributed architecture built on Apache Spark. This study provides practical insights into scalable and interpretable clustering methods for intelligent manufacturing and sensor-based systems and highlights the effectiveness of adaptive kernel learning in modern big data applications.

Keywords: big data clustering, Apache Spark, K-Means, DBSCAN, Fuzzy C-Means, GBK-Means, industrial sensor data

1. Introduction

The focus of big data research has evolved over time. Initially, research efforts were concentrated on addressing the challenges associated with storing and processing vast amounts of data. However, in recent years, there has been a shift towards identifying actionable insights within the voluminous data streams generated by digital platforms. In the contemporary era, it is widely acknowledged that big data constitutes a foundational element of the modern computer science industry, underpinning advancements in domains such as artificial intelligence, industrial automation, and intelligent decision systems [1].

The generation of data in the contemporary era is exceedingly vast in magnitude. To provide context, YouTube is inundated with over 720,000 hours of new video content daily. Facebook and Instagram, in isolation, are responsible for more than 350 million photo uploads and 500 million stories posted on a daily basis. With an active user base that exceeds 1.925 billion as of 2025, TikTok has emerged as a primary catalyst for the proliferation of multimedia data [2]. According to the most recent research study by the International Data Corporation, the global data sphere is expected to exceed 180 zettabytes by the year 2025, with significant contributions from the Internet of Things (IoT), cloud computing, and autonomous systems [3].

*Corresponding author: Bakhshali Bakhtiyarov, Department of Instrumentation Engineering, Azerbaijan State Oil and Industry University, Azerbaijan. Email: baxshali.baxtiyarov.sh@asoiu.edu.az

The analysis of such extensive and varied sources to yield valuable knowledge necessitates the implementation of sophisticated knowledge discovery methods. Among the aforementioned methods is clustering, which refers to the process of dividing a set of data into subsets such that the objects in the same subset are closer to each other than to the objects in the other subsets [4]. Clustering is a pivotal topic in numerous fields, including data mining, biostatistics, intelligent energy grids, anomaly detection in the IoT, image segmentation, and optimization of deep learning models [5].

The scalability challenges posed by large and high-dimensional data thus continue to challenge traditional clustering algorithms despite decades of development. As an example, classical K-Means has been shown to be NP-hard even with a low number of clusters and thus is not well-suited to distributed or streaming settings [6]. Therefore, the scalability, efficiency, and accuracy of clustering methods have become a significant research focus. The following original five attributes of big data—the so-called 5Vs—make clustering even more complicated [7]:

1) Volume: Petabytes of unstructured and semi-structured data on social platforms, sensors, and logs.
2) Velocity: Edge devices and cloud services provide real-time data streams that need low-latency processing.
3) Variety: Text, video, sensor signals, time-series—heterogeneous formats.
4) Veracity: It can include noisy and incomplete data that can be biased or misleading.

5) Value: Deriving business intelligence and actionable insights out of raw data.

To address these problems, the research community has investigated distributed clustering algorithms with the help of platforms such as Apache Spark, Flink, and Ray, which allow parallel computation over large-scale datasets [8].

To summarize, clustering in big data settings requires not only robust and precise algorithms but also an elastic architecture. With the ever-increasing explosion in volume and variety of data, scalable, distributed, and smart clustering techniques are an important frontier in data science research.

Conventional clustering techniques have been found inadequate in efficiently processing the massive size and variety of contemporary datasets due to the growing sophistication and computational overheads of large-scale data environments. The most obvious one is the vanilla K-Means algorithm, which is NP-hard even when k = 2, meaning that it should not be used in high-dimensional and large-scale settings [6]. Scalability has therefore become a key issue in big data clustering.

The main goals of recent clustering research are efficient algorithms with parallelizability and scalability, and, at the same time, high-quality clustering. Although there has been considerable progress in this field, the holistic nature of big data—that is, high velocity, variety, and veracity—remains elusive to current solutions [9]. Although there are some review articles that discuss clustering in big data, the current study provides several significant contributions that make it stand out from previous articles.

Limited scope–high relevance: The scope of this work is narrowed to focusing only on the recent developments of clustering methods adapted to modern distributed big data frameworks, such as Apache Spark, Flink, and Dask. The research strategy relies on the review of over 200 scholarly papers released in the period between 2015 and 2025.

Comparative platform analysis: It provides a side-by-side comparison between Apache Spark and Hadoop MapReduce in terms of their architectural variations, performance compromises, and applicability in different use cases. The comparison is incorporated as a guide to be used by researchers and engineers when choosing clustering platforms to use in large-scale applications.

Selected high-quality technique filtering: The paper filters and analyzes the most technically sound and popular clustering techniques and discusses their implementation plans, computational complexity, and areas of application. The advantages and disadvantages of each of the techniques are clearly outlined.

Further, clustering algorithms may also be distinguished based on the architecture of their execution, which can be of the following two broad types [10]:

1) Single-machine clustering methods: These are the methods that are run on a standalone machine with minimal memory and computing resources. They tend to be effective with small and moderately sized datasets but encounter performance bottlenecks when used with huge volumes of data.
2) Multi-machine/distributed clustering: These methods use the processing capabilities of distributed nodes within a cluster and offer much better scalability and fault tolerance. These methods can be horizontally scaled on large partitions of data using platforms like Apache Spark, Apache Flink, and containers orchestrated by Kubernetes [8].

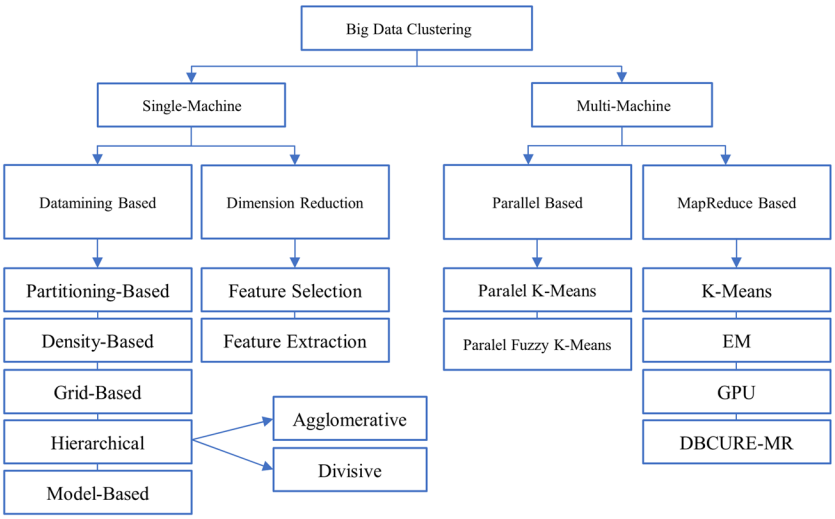
Each of these approaches represents a trade-off between resource utilization, latency, fault tolerance, and system complexity and will be comparatively detailed in the sections below.

2. Literature Review

Big data clustering techniques are of two kinds, as illustrated in Figure 1: single-machine and multi-machine clustering techniques. Multi-machine clustering solutions have recently become popular because they are highly scalable and responsive in terms of performance for customers.

Clustering techniques adapted to big data settings can be roughly grouped into two categories, as shown in Figure 1: single-machine clustering techniques and multi-machine (distributed) clustering techniques. Interest in multi-machine solutions has been growing in recent years, as they provide better scalability, distributed memory usage, and faster response times in practical tasks [11]. Single-machine clustering methods are run on isolated systems and operate within the constraints of local computing resources. They are conventional when the size of the data is low to medium, and in many cases, they

Figure 1
Big data clustering techniques



are vulnerable to memory bottlenecks, poor parallelism, and degraded performance as data size grows [12]. The most popular strategies in this category are data mining–based clustering and dimension-reduction-based strategies [13].

The clustering strategy is a data mining approach that employs unsupervised learning models to identify latent patterns in collections of data. Nonetheless, the problem with such algorithms is that they are not always scalable due to intensive computation and memory demands. Prominent methods in this category include evolutionary-based, density-based, hierarchical, model-based, grid-based, and partitioning-based clustering methods.

Among those, partitioning-based clustering methods seek to subdivide a dataset into a pre-specified number of clusters, k, using similarity measures such as Euclidean distance, Manhattan distance, or Chebyshev distance. The aim of these methods is to maximize intra-cluster homogeneity and inter-cluster separation. Well-known algorithms in this paradigm are K-Means [14], which is simple and fast to converge; K-Medoids and its generalization Partitioning Around Medoids (PAM), which are robust against noise and outliers [15]; K-Modes, which is efficient with categorical data; and scalable solutions such as Clustering Large Applications (CLARA) and Clustering Large Applications based on Randomized Search (CLARANS), which are tailored to large datasets that cannot fit in available memory.

Hierarchical clustering is another significant subcategory of single-machine clustering. Hierarchical clustering constructs nested clusters representing a tree-like layout known as a dendrogram. The following two major types exist:

1) Agglomerative hierarchical clustering, which starts with each data point as its own cluster and repeatedly merges the closest pairs of clusters. Prominent algorithms based on this technique are Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) and CHAMELEON, which utilize dynamic modeling to efficiently merge [16].
2) Divisive hierarchical clustering begins with all data in one cluster and repeatedly divides the cluster based on dissimilarity until a desired number of clusters is achieved. Familiar algorithms based on this approach are Principal Direction Divisive Partitioning (PDDP) and Divisive Hierarchical Clustering using Clustering Features (DHCC) [17].

Despite the role single-machine techniques have played in clustering research, their shortcomings in distributed computation, fault resiliency, and real-time flexibility have inspired the development of multi-machine clustering systems, which are the subject of the following section.

Besides partitioning and hierarchical methods, several other clustering paradigms have been developed to address special data structures, performance bottlenecks, or robustness issues. These include density-based, grid-based, model-based, and dimension-reduction-aided clustering methods, each with distinct properties and suitability for use.

Density-Based Clustering: In this technique, clusters are defined as dense areas in the data space, separated by regions of lower density. The clusters are not predetermined by shape or number but are formed by spatial locality and the density and interconnectedness of data points. The main advantage of this method is its ability to find clusters of arbitrary shapes and its robustness to noise and outliers, because it is a one-pass scanning algorithm [18]. Well-known algorithms include Density-Based Spatial Clustering of Applications with Noise (DBSCAN), DENCLUE (DENsity-based CLUstEring), Ordering Points to Identify the Clustering Structure (OPTICS), and Distribution-Based Clustering of LArge Spatial Databases (DBCLASD) [19].

Grid-Based Clustering: In grid-based clustering, the data space is discretized into a finite number of non-overlapping cells (or grids), and clusters are identified based on the density of objects in these cells. The basic algorithm steps include the following: (a) subdividing the entire data space into a constant number of cells (typically much smaller than the number of data records), (b) removing sparse cells with low object density, and (c) merging high-density neighboring cells to form clusters [20]. This technique is computationally efficient, especially on large-scale data, although the clustering quality is sensitive to the granularity of grid partitioning. Well-known algorithms include STING, CLIQUE, OPTIGrid, and WaveCluster [21].

Model-Based Clustering: These methods assume that the data is generated from a mixture of underlying probability distributions. Clustering is achieved by estimating the parameters of these distributions. This allows for automatic determination of the number of clusters and probabilistic assignment of membership. As a result, model-based approaches are less vulnerable to noise and overlapping data structures [22]. However, as the models grow in complexity, they may incur significant computational overhead. Notable algorithms in this category include Expectation-Maximization, COBWEB, and MCLUST [23].

Dimension Reduction Techniques in Clustering: In high-dimensional data, clustering performance can degrade in both time and quality. To address this, dimension reduction techniques are used as a preprocessing step to reduce the dimensionality of the feature space while retaining essential data properties. Two main methods are as follows:

1) Feature selection, which retains the most relevant attributes of the original dataset. This is controlled through pre-defined subset strategies or manual thresholds. Common techniques include Sequential Forward Selection (SFS), Correlation-based Feature Selection (CFS), and Markov Blanket Filter (MBF) [24].
2) Feature extraction, which maps high-dimensional data to a new, lower-dimensional feature space. This improves clustering effectiveness and reduces computation. Popular methods include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Singular Value Decomposition (SVD) [25].

Altogether, these techniques enhance the clustering process by improving scalability, reducing computational complexity, and increasing robustness in high-dimensional or noisy data.

As the amount of data grows exponentially—frequently in the petabyte range—traditional single-machine clustering algorithms have become increasingly insufficient to process data in a timely fashion. Researchers have overcome this limitation by exploiting recent improvements in distributed computing, networked systems, and parallel processing to develop multi-machine clustering methods. These techniques divide big data into small, manageable parts, which are distributed using MapReduce to many computing nodes for parallel computation. With this strategy, better scalability, fault resilience, and speed of execution can be achieved [26].

Parallel Clustering Paradigms: In a distributed environment, parallelism is the main aspect of improving clustering performance. The following are the three main models of parallelization frequently mentioned in the literature by Dafir et al. [27]:

1) Independent parallelism: Every processor has access to the complete dataset and processes it independently without any communication between processors.
2) Task parallelism: The different processors run different clustering algorithms or component tasks.
3) Single Program Multiple Data (SPMD): The same program is run on multiple processors, each with a disjoint set of data.

Although parallel clustering can enhance scalability and efficiency, it also introduces additional complexities for system designers in terms of data partitioning, load balancing, and inter-process communication.

These paradigms have been used to develop several well-known parallel clustering implementations, such as a parallel implementation of K-Means using a Network of Workstations, where message-passing protocols are used to synchronize the processors [28]. Likewise, parallel CLARANS was implemented in a master-slave design based on Parallel Virtual Machine environments [29].

In hierarchical clustering, PBIRCH generalizes the classical BIRCH algorithm to run in a parallel environment based on the SPMD model and can scale across clusters of machines [30]. A parallel version of CHAMELEON was proposed using another hierarchical technique with the following three-stage design:

1) A parallel k-nearest neighbor algorithm to reduce time complexity,
2) Multilevel graph partitioning to generate initial clusters,
3) Agglomeration is based on interconnectivity and proximity to form final clusters.

Density-Based Distributed Clustering (DBDC): Some of the most notable distributed clustering methods include Density-Based Distributed Clustering [31], in which a dataset is divided, and computing is performed in several locations. A DBSCAN algorithm is applied at each local site, producing local clusters and representative elements. These local models are then sent to a central server, which combines them into a global clustering model. The global model may then be published to clients to allow local updates and ensure consistency in distributed environments.

GPU-based acceleration: Graphics Processing Units (GPUs) have recently been considered in clustering applications due to their superior parallel computation and memory utilization capabilities. Tasks involving repeated distance computations or graph traversal particularly benefit from GPU-based clustering. An example of such an implementation is G-DBSCAN, a GPU-accelerated variant of DBSCAN [32]. This method consists of the following two major stages:

1) A graph is created, with edges corresponding to relationships between data points within a fixed distance threshold.
2) The revealed structure is clustered using Breadth-First Search traversal on the constructed graph [33].

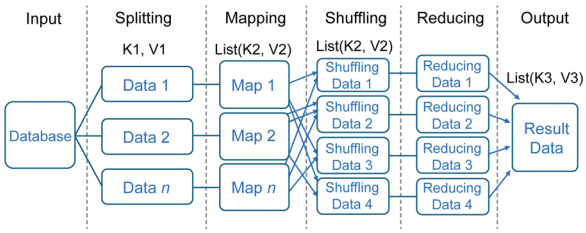
It is becoming increasingly apparent that GPU-accelerated algorithms are becoming standard elements of clustering pipelines, particularly when integrated with distributed systems in cloud and edge computing environments [34].

MapReduce-based clustering: MapReduce is an influential model of distributed data processing, initially proposed by Google in 2004, to perform large-scale computations on clusters of machines [35]. This model allows parallel execution of data processing tasks by breaking them down into smaller subtasks, which are then distributed to various processing nodes. As shown in Figure 2, MapReduce architecture consists of two main user-defined functions: Map and Reduce.

During the Map phase, each mapper node independently processes a portion of the input data and generates intermediate key-value pairs. These intermediary outputs are then grouped and passed to the Reduce phase, where reducer nodes combine the values associated with the same key. This intermediate shuffle framework ensures efficient combination and alignment of data across the nodes, enabling synchronized computation throughout the distributed environment.

The primary benefit of MapReduce is that it abstracts the low-level complexities of parallel processing, internode communication, and resource allocation. Developers only need to define the data

Figure 2
MapReduce framework



transformation logic, whereas load balancing, data partitioning, network optimization, and fault tolerance are automatically handled by the framework.

MapReduce provides extensive parallelism, reliability, and scalability, making it a valid foundation for executing clustering algorithms over large datasets. Several clustering algorithms—especially those requiring batch processing, such as K-Means—have been successfully ported to the MapReduce paradigm to leverage its fault tolerance and implementation simplicity within distributed file systems such as Hadoop Distributed File System (HDFS).

Several clustering algorithms have been modified to run on the MapReduce framework to take advantage of its scalability and distributed processing. Among them, Parallel K-Means (PK-Means) is a modification of the classic K-Means algorithm designed to run within the MapReduce framework. It divides the data among numerous nodes and performs centroid update iterations in a parallelized manner. Another well-known example is MR-DBSCAN, which combines MapReduce and GPU acceleration to improve the performance of the density-based DBSCAN algorithm. This parallel-hybrid strategy is significantly more effective in clustering big and complex information ontologies by incorporating parallel computation with high-throughput architecture hardware [36]. Some contributions of clustering technologies are shown in Table 1.

Recent evolution of fuzzy-based clustering algorithms in large-scale and heterogeneous environments includes suggestions to add the adaptive fuzzy membership models to deep learning architecture to enhance the cluster interpretability and robustness [37]. In addition, fuzzy logic technology in combination with hybrid methods based on fuzzy bounds and graph-based optimization was demonstrated to be effective in application in power system diagnostics and anomaly forecasting, which demonstrates the usefulness of fuzzy logic in real-time big data processing [38]. These studies reinforced the applicability of our suggested Adaptive Gaussian-Based Kernel K-Means (A-GBK-Means) framework, which also uses flexible fuzzy membership and the kernel learning in a distributed context.

3. Related Work

The clustering of big data is highly dependent on the underlying infrastructure of the data processing system, as it defines scalability, responsiveness, and efficiency. The three main types of processing paradigms are: (1) batch processing systems, (2) stream (real-time) processing systems, and (3) interactive analytics platforms.

3.1. Batch processing platforms

The most embraced big data analytics model is batch processing, especially in large-scale numerical calculations. It is a way of handling large quantities of data in predetermined groups or batches, which are saved and processed at set intervals. The most notable batch platform

Table 1
Contribution of the clustering technique

Clustering technique	Contribution
MLlib 2.0 (Apache Spark)	Addresses Big Data clustering.
Bayesian HMM with GM Clustering	Accurate change detection.
K-Prototype with MapReduce	Good scalability shown.
Improved Weighted Clustering (IWC) and PSO	Low cost, better accuracy.
Apache Spark with Locality Sensitive Hashing	High quality, fast execution.
Parallel K-Means and Fuzzy K-Means	Up to 60% faster.
DBSCAN	1 B records in 41 min.
Fractional Sparse Fuzzy C-Means and PSO	90.6% accuracy, DBI 5.33.
Conventional Static Model	Better personalization.
BigVAT with Sampling-Based Crisp Partitions	Fine-grained clusters found.
Game-Based K-Means (GBK-means)	Outperforms classic K-Means.
Fuzzy K-Means for Teaching Evaluation	Improved evaluation accuracy.
Distributed Computing Based K-Means	Faster, more stable.
Neural-Engine Based K-Means	200% speed gain.
Radar Scanning Strategy Clustering	Real-time high performance.
Graph-Based Spectral Clustering on Spark	High quality for graphs.

is the free and open-source Apache Hadoop framework, which implements the MapReduce programming model first proposed by Google [39].

The following are the three key components in the core architecture of Hadoop:

1) Distributed storage - HDFS,
2) Computation with MapReduce,
3) Cluster management - Yet Another Resource Negotiator (YARN).

Above these base layers, Hadoop offers several abstractions and extensions such as Hive (SQL-like querying), Pig (data flow scripting), HBase (NoSQL storage), and Apache Spark. The combination of these modules provides an end-to-end capability along the big data value chain: data collection, processing, assessment, and storage management [40].

Hadoop has been designed to fit well in enterprise-level deployments due to its characteristics of low infrastructure cost, scalability, and high fault tolerance [41]. Its processing philosophy—breaking tasks into small, independent subtasks that are computed in parallel on distributed nodes—is especially suited for data-intensive operations such as clustering, classification, and machine learning [42].

3.2. Stream processing platforms

Unlike batch systems, stream processing platforms are geared toward use cases with low-latency or real-time decision-making needs. Such systems are ideal in scenarios where streams of incoming data need to be processed in real-time, as opposed to being stored and analyzed at a later stage. Real-time analytics are valuable in applications such as transportation systems, financial analytics, and environmental monitoring. As IoT devices continue to multiply (they are expected to surpass 50 billion connected sensors worldwide [43]), the need to process data in real-time also increases. Examples of the leading platforms in this category include Apache Storm, which specializes in real-time stream computation, and SAP HANA, which combines in-memory processing with sophisticated analytics features. These platforms provide high throughput and near real-time feedback without requiring permanent storage, making them suitable for time-sensitive clustering and pattern recognition applications.

3.3. Interactive analytics platforms

Interactive analytics platforms allow dynamic and elastic access to big data. Users can perform ad hoc querying, data visualization, and iterative computations with low latency. The best-known representative of this category is Apache Spark, a next-generation data analytics engine initially developed at the University of California, Berkeley [44].

Spark was created to address the disk I/O bottlenecks of Hadoop by providing in-memory computing capabilities, enabling data to be cached and reused across multiple iterations. It has extensive compatibility with many programming languages, such as Java, Scala, and Python, and can be integrated with YARN and HDFS in Hadoop, allowing it to run smoothly in heterogeneous environments.

In performance benchmarks, Spark is up to 100 times faster than Hadoop MapReduce on in-memory workloads and up to 10 times faster when relying on data that does not fit in memory [45]. Such benefits render Spark especially well-suited for iterative algorithms like K-Means, DBSCAN, and hierarchical clustering, where multiple accesses to the same data are typical.

Table 2 provides a comparative summary of key big data platforms and their features. The best platform to use among them depends on the specific requirements of the clustering application, considering factors such as latency tolerance, data volume, resource availability, and the scale of deployment.

4. Methodology

In this section, the methodological framework of the current research is described, namely the chosen clustering algorithms, evaluation measures, features of the dataset, and implementation environment. Some comparative analysis of clustering methods is shown in Table 3.

4.1. Selected clustering algorithms

This research paper has chosen four different clustering algorithms to test their functionality and stability when dealing with large-scale datasets under different settings. These techniques have been selected because of their theoretical diversity, applicability to real-world distributed environments, and ability to handle various kinds of data structures (e.g., dense, noisy, and overlapping clusters).

Table 2
Technical differences between Apache Spark and Hadoop

Feature	Apache Spark	Hadoop MapReduce
Performance	In-memory distributed processing; up to 100× faster	Disk-based processing; comparatively slower
Read/Write Cycle	Optimized, minimal disk I/O (low latency)	High I/O overhead; disk-intensive
Ease of Use	High-level APIs (Scala, Python, R)	Requires verbose Java code
Realtime Processing	Fully supported (e.g., Spark Streaming)	Not supported; batch-only model
System Latency	Low (in-memory execution)	High (due to frequent disk writes)
Fault Tolerance	Built-in (RDD lineage & DAG recovery)	Supported via data replication
Security Features	Basic (enhanced via integrations)	Advanced (Kerberos, ACL, etc.)
Infrastructure Cost	Higher (requires large memory capacity)	Lower (runs on commodity hardware)
Programming Languages	Scala, Java, Python, R	Java, limited support for others
Licensing Model	Open source (Apache License 2.0)	Open source (Apache License 2.0)
SQL Compatibility	Spark SQL (ANSI SQL compliant)	HiveQL (subset of SQL)
Scalability	Highly scalable (horizontal scaling)	Highly scalable (but slower in practice)
Machine Learning	MLlib (native, distributed ML library)	Apache Mahout (less maintained)
Data Caching	Natively supported (RDD, DataFrame, etc.)	Not supported
Hardware Requirements	Requires high-performance nodes	Runs efficiently on commodity hardware

The chosen algorithms are the following (Figure 3):

1) K-Means—a centroid-based algorithm that is widely used and whose efficiency and scalability in Spark settings have been demonstrated [46].
2) DBSCAN—a density-based algorithm that can identify arbitrarily shaped clusters and is capable of handling noise [47].
3) Fuzzy C-Means (FCM)—a soft clustering method that allows assigning membership values of each point to multiple clusters [48].
4) GBK-Means—an augmented kernel adaptation of K-Means, which combines fuzzy logic and nonlinear transformations, and is reported to have better adaptability to arbitrary data distributions [49].

All the algorithms have been implemented and executed in the Apache Spark environment to ensure their interoperability with distributed computing and big data scale.

4.2. Evaluation metrics

Computational and internal cluster quality metrics were used to systematically analyze the effectiveness and efficiency of the chosen clustering algorithms on various datasets. Such metrics allow for controlled and real-world data comparisons in an objective manner.

Execution Time is the total runtime required to complete the clustering process, starting from data ingestion to the generation of the final output. It is a primary measure of algorithmic scalability and computational cost. The execution time was recorded using Spark's native job monitoring tools and averaged over five independent runs for each algorithm–dataset pair.

Silhouette Score (SS) measures the similarity of an object to its own cluster compared to other clusters. It ranges between −1 and +1, with higher values indicating well-separated and clearly defined clusters. The score of a data point i, denoted as s(i), is given by Equation (1) as follows:

$$s(i) = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \quad (1)$$

Where:

1) a(i): the average distance of point i to all other points in the same cluster
2) b(i): the average distance of point i to the nearest cluster not containing i

To reduce computational overhead, sampled subsets of each dataset were used to compute SSs. The Davies–Bouldin Index (DBI) compares the average distance between each cluster and its closest matching cluster. The smaller the value of DBI, the better the clustering performance. It can be defined as Equation (2) as follows:

$$DBI = \frac{1}{k}\sum_{i1}^{k} \max_{j\neq i}\left(\frac{\sigma_i+\sigma_j}{d_{ij}}\right) \quad (2)$$

where σi is the intra-cluster distance of cluster i, and dj is the inter-cluster distance between clusters i and j.

Noise Ratio can be applied primarily to DBSCAN and FCM. It is determined as the percentage of data points that were either considered noise or had low-confidence membership. This measure is of critical importance in interpreting the robustness of an algorithm to outliers and unclustered data. To evaluate scalability, the individual algorithms were tested on progressively larger datasets. Execution time and quality scores were monitored across scales to assess the computational efficiency and stability of each approach under increasing volume stress.

4.3. Datasets

To present a comprehensive analysis of the clustering algorithms' performance, this research used three heterogeneous datasets. These datasets are diverse in nature, origin, and complexity, allowing for the evaluation of clustering algorithms under both synthetic and real-life settings. The first two datasets are publicly available industrial sensor logs found on Kaggle and consist of multivariate time-series measurements on real equipment. These datasets include realistic challenges such as noise, missing values, and feature heterogeneity. The third dataset was artificially generated, containing Gaussian-distributed clusters with added noise, which provided a controlled setting to

Table 3
Comparative analysis of clustering methods

Category	Algorithm	Data Type	Advantages	Disadvantages
Partitioning-Based	K-Means	Numerical	Efficient and simple	Sensitive to noise
	K-Medoids	Numerical	Robust to noise	Computationally intensive
	K-Modes	Categorical	Handles categorical data	Limited to categorical data
	PAM/CLARA	Numerical	Robust to outliers	Computational complexity
	CLARANS	Numerical	Efficient on large datasets	Parameter dependent
Hierarchical Clustering	BIRCH	Numerical	Efficient hierarchical clustering	Sensitive to noise
	CHAMELEON	Mixed	Adaptive to complex clusters	Complex scalability
	EON	All types	Handles diverse datasets	Scalability limitations
	PDDP	Numerical	Efficient numerical data clustering	Linear constraints
	DHCC	Categorical	Categorical data clustering	Limited numerical applicability
Density-Based	DBSCAN	Numerical	Robust to noise	Scalability issues
	DENCLUE	Numerical	Automatic cluster determination	Computationally expensive
	OPTICS/DBCLASD	Numerical	Effective in complex structures	Complex parameter tuning
Grid-Based	STING	Numerical	Fast processing	Limited accuracy
	CLIQUE	Numerical	Effective in high dimensions	Dependent on grid definition
	OPTIGrid	Special data	Adaptive grid management	Complex object handling
Model-Based	COBWEB	Numerical	Fast clustering	Noise sensitivity
	MCLUST	Numerical	Automatic cluster determination	Complex parameter tuning
Feature Selection	CFS	Alphanumerical	Reduces data size	High-dimensional inefficiency
	SFS	Numerical	Redundant data removal	Requires preprocessing
	MBF	Numerical	Effective noisy data filtering	Preprocessing necessity
Feature Extraction	PCA	Numerical	Dimensionality reduction	Information loss
	LDA	Categorical	Categorical data reduction	Reduced interpretability
	SVD	Numerical	High scalability	Computational overhead
	ICA	Numerical	Signal separation	Statistical assumptions
Parallel Clustering	PBIRCH	Numerical	Parallel execution	Algorithm complexity
	DBDC	Numerical	High scalability	Implementation complexity
	G-DBSCAN	Numerical	Density clustering in parallel	Complexity of parallel algorithms
Kernel-Based Clustering	GBK-Means	Numerical	Effective in handling non-linear and overlapping clusters	High computational cost due to kernel operations
Fuzzy Clustering	Fuzzy C-Means	Numerical	Captures soft cluster membership; interpretable results	Slower due to iterative membership updates
MapReduce Clustering	PK-Means	Numerical	High scalability	Resource intensive
	MR-DBSCAN	Numerical	Parallel complexity management	Query-based complexity

test clustering accuracy in known distributions. Table 4 provides an overview of the data sets used, including their record and feature counts, whether they contain noise, and the number of missing values.

These datasets include realistic challenges such as noise, missing values, and feature heterogeneity. The third dataset was artificially generated, containing Gaussian-distributed clusters with added noise, which provided a controlled setting to test clustering accuracy in known distributions. Table 4 provides an overview of the datasets used, including their record and feature counts, whether they contain noise, and the number of missing values.

Figure 3
Block diagram of clustering algorithms

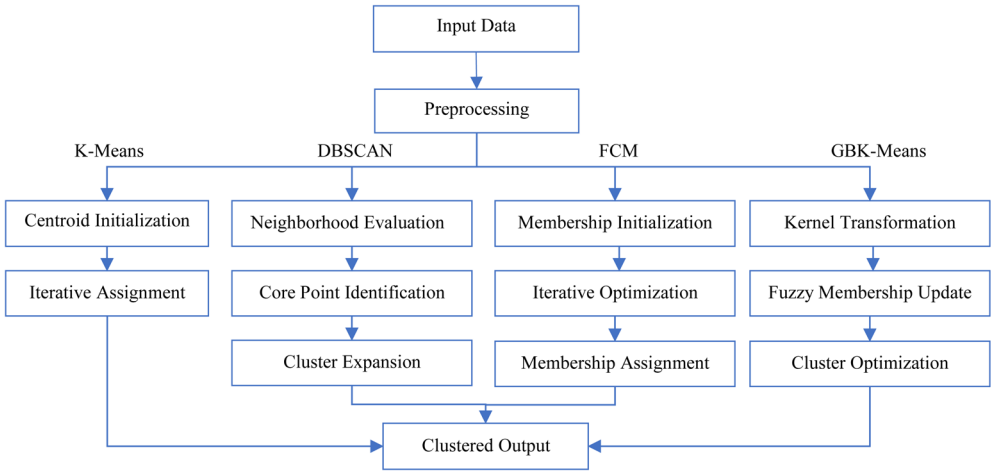


Table 4
Description of datasets used

Number	Dataset name	Type	Records	Features	Noise	Missing values
D1	Industrial Equipment Monitoring	Real	7,672	7	No	0
D2	Sensor Data from Industrial Machine	Real	10,000	6	Yes	1,249
D3	Clustered Data	Synthetic	1,000,000	5	10%	0

Table 5
Experimental platform configuration and technical specifications

Parameter	Specification
Platform	Apache Spark 3.5.0 (Standalone Cluster Mode)
Cluster Setup	One master node and four worker nodes
CPU (per node)	16-core Intel Xeon E5-2620 v4 2.10GHz
RAM (per node)	64 GB
Storage (per node)	1 TB SSD
Operating System	Ubuntu 22.04 LTS (64-bit)
Cluster Manager	Spark Standalone
Programming Interface	PySpark (v3.5) utilizing the Pandas API
Data Storage Formats	CSV and Parquet

4.4. Experimental environment

Each of the experiments was carried out on Apache Spark, as the framework provides efficient execution of large-scale datasets on a distributed computing system. The implementation on Spark enabled the clustering algorithms to be executed in parallel, taking advantage of the in-memory computation model to achieve faster execution and scalability [50]. Experimental platform configuration and technical specifications are shown in Table 5.

Each algorithm was run five times on every dataset, and the average of the results was recorded for comparison purposes. Intermediate processing steps, such as normalization and missing value imputation, were handled through Spark ML pipelines. Hyperparameter tuning was performed manually using grid search, optimized based on the SS. Random seeds were set to fix all cluster initializations and ensure reproducibility.

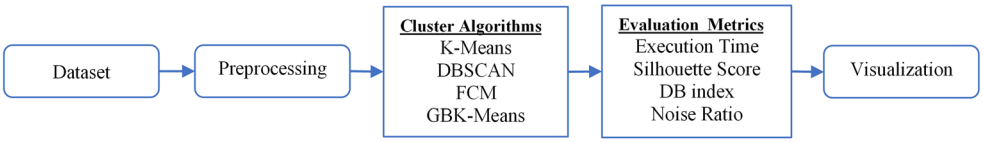
5. Experimental Configuration

The experimental process was carefully designed to enable repeatable, objective, and systematic assessment of clustering performance under different data conditions. The same workflow was applied to all datasets and clustering algorithms to ensure consistency and comparability. It began with data cleaning and preprocessing, where missing values in Dataset D2 were either imputed with the mean or deleted on a row-wise basis. Z-score normalization was applied to all numerical features to standardize feature scaling, and the resulting datasets were converted into Apache Spark DataFrames. Feature vectors were then constructed to ensure compatibility with Spark MLlib.

Following preprocessing, the clustering algorithms—K-Means, DBSCAN, FCM, and GBK-Means—were run separately on each dataset. To reduce the impact of random initialization, each algorithm was executed five times with fixed random seeds. Performance metrics were systematically collected during each run. Execution times were recorded using Spark's internal monitoring utilities, and internal clustering validity scores, including the SS and DBI, were computed to evaluate clustering quality. Additionally, noise ratios were calculated (only for algorithms capable of detecting noise—namely DBSCAN and FCM), and robustness was assessed.

Finally, the results of all experiments were systematically recorded to support visual comparison and subsequent analysis. The

Figure 4
Experimental workflow architecture



entire workflow was implemented in JupyterLab using Spark ML pipelines, ensuring complete reproducibility and methodological transparency across all experiments.

5.1. Implementation workflow

To enable repeatability and scalable cluster implementation, the workflow was designed as a modular pipeline. This workflow was built on the foundation of Apache Spark's distributed architecture and was executed through PySpark interfaces to deploy algorithms and collect metrics.

Figure 4 outlines the main steps of the experimental pipeline. The process begins with the ingestion of datasets and their preprocessing, where missing values are handled and features are normalized. The processed data is then introduced into the selected clustering model (K-Means, DBSCAN, FCM, or GBK-Means), and finally, it is evaluated using several internal validity measures.

To ensure consistent parameter tuning and to parallelize computation, each execution step was packaged into a reusable Spark ML pipeline. The evaluation results were automatically written to formatted result files, which were then used to generate visualization plots and comparison charts.

5.2. Parameter settings

The clustering algorithms were configured with predefined hyperparameters to facilitate fair and consistent comparison across all experimental settings. These parameters were either adopted from best practices in literature or optimized through empirical grid search on a held-out validation set from the synthetic dataset (D3).

Table 6
Hyperparameters for clustering algorithms

Algorithm	Key parameters
K-Means	$k = 6$
DBSCAN	$\varepsilon = 0.3$, MinPts = 10
FCM	$c = 6$, $m = 2.0$, $\varepsilon = 1e\text{-}5$, maxIter = 300
GBK-Means	$k = 6$, $\sigma = 1.5$, $\lambda = 0.01$, maxIter = 150
A-GBK-Means	$k = 6$, $\sigma(x)$, $\lambda = 0.01$, $m = 2.0$

With respect to the GBK-Means and the A-GBK-Means algorithms, various essential parameters do affect the performance of the clustering method. The value of the number of clusters (k) was determined in order to correspond to the known class structure of the synthetic data, as well as to allow comparing the results among datasets ($k = 6$). The value of the kernel width (sigma) gives sensitivity to the Gaussian function, that is, a lower value gives more stress on local similarity, whereas a higher value tends to smooth the effect of a larger area. In GBK-Means, sigma was constant, but in A-GBK-Means, it was dynamic and adjusted adaptively according to the local density of a data point, resulting in a more flexible cluster boundary and tolerance of noise. The regularization coefficient regulates the size of the kernel component and also prevents overfitting by the issue of penalizing a high deviation. The fuzzifier (m) is a parameter that varies the softness of cluster membership and typically takes the value 2.0; an increased value can lead to an increased degree of fuzziness but also slow convergence. Parameters were chosen using both empirical tuning as well as support in the literature and were standardized between experiments to stay as objective and reproducible as possible.

For the K-Means algorithm, the number of clusters (k) was set to 6 for all datasets to match the ground truth in the synthetic data and maintain consistency in the real-world setting. The algorithm was initialized using the "k-means++" method and capped at a maximum of 100 iterations per run.

For DBSCAN, the epsilon ($\varepsilon = 0.3$) parameter was determined empirically using the k-distance graph, and the minimum number of points required to form a dense region (MinPts) was set to 10.

The FCM algorithm was configured with a fuzziness coefficient ($m = 2.0$), a convergence criterion ($\varepsilon = 1 \times 10^{-5}$), and a maximum of 300 iterations per run to ensure stable convergence behavior.

Specifically, for the GBK-Means algorithm, which combines kernel transformation with fuzzy membership weighting, the following settings were applied: number of clusters $k = 6$, kernel width $\sigma = 1.5$, regularization coefficient $\lambda = 0.01$, and maximum iterations (maxIter) = 150. Each algorithm was run with fixed random seeds to ensure result reproducibility. Table 6 summarizes the parameter values used for each method.

Table 7
Summary of experimental tasks

Step	Task description	Tools/technologies used
1	Data acquisition and preprocessing	Pandas, PySpark, Scikit-learn
2	Normalization and missing value handling	Spark ML Pipeline (Z-score, Imputer)
3	Algorithm integration and configuration	PySpark MLlib, Custom Python (FCM)
4	Execution of clustering models	Apache Spark 3.5 (Standalone Mode)
5	Evaluation metric computation (SS, DBI, Noise)	Scikit-learn, NumPy
6	Time and scalability measurement	Spark job logs, built-in timers
7	Result recording and visualization	JupyterLab, Matplotlib, Seaborn

5.3. Summary of tasks performed

This subsection presents a description of the key activities carried out during the experimental stage. The process was implemented in a modular and reproducible manner with the support of distributed processing in Spark. Table 7 summarizes the key steps undertaken in the clustering experiments.

JupyterLab notebooks were used to script all tasks, making the results fully traceable and reproducible. Specific wrapper functions were written to integrate FCM and GBK-Means into the Spark pipeline.

6. Experimental Results and Evaluation

This section presents the experimental results of applying the four clustering algorithms to three datasets. The analysis is conducted in terms of computational efficiency (running time, scalability) as well as clustering quality (SS, DBI, and noise-handling capability).

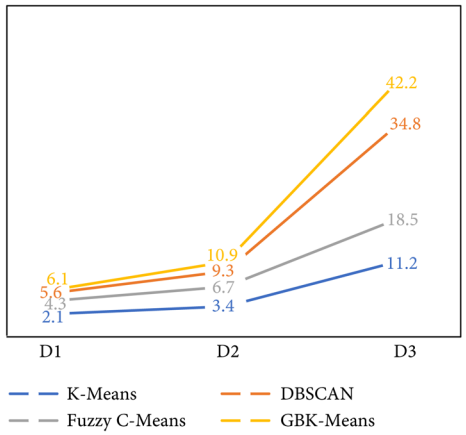
Overall, the suggested A-GBK-Means algorithm is superior to the baseline approaches in its ability to balance well between the accuracy of clustering tasks and scalability. It takes less time to execute competitively, larger SSs, and smaller DBI values. It also works effectively in the detection of meaningful patterns, especially on noisy datasets (D2), compared to traditional K-Means or DBSCAN. These findings add credence to the fact that kernel width and fuzzy membership setting flexibilities are major contributors to adaptive performances in cluster computing and practical relevance.

6.1. Execution time analysis

Execution time is one of the most important performance measures of clustering algorithms, especially when applied to large-scale datasets, as is often the case in practice. In this analysis, the mean runtime (in seconds) of each algorithm–dataset pair is considered. Each experiment was repeated five times to obtain statistically reliable results, and the average execution time was reported to minimize the influence of random initialization effects or system workload fluctuations.

Figure 5 illustrates that there were significant variations in computational efficiency among the algorithms. K-Means consistently demonstrated the fastest execution time, especially on the synthetic dataset (D3), due to its linear time complexity and the highly optimized vectorized implementation in Spark MLlib. In contrast, DBSCAN exhibited considerable slowdowns on larger datasets, attributed to the computationally intensive nature of its neighborhood-based density estimation.

Figure 5
Execution time and dataset size



The FCM algorithm required more time than K-Means due to its iterative fuzzy membership updates; however, its performance remained within acceptable limits. Across all datasets, GBK-Means showed the longest execution times, primarily due to the additional computational cost of kernel function evaluation and membership matrix calculations.

6.2. Clustering quality evaluation

Clustering quality is essential for evaluating how well an algorithm identifies the underlying group structures within data. This paper employed the following three internal validation measures to assess clustering quality:

1) The SS, which measures intra-cluster cohesion and inter-cluster separation, with higher values indicating compact and well-separated clusters.
2) The DBI, which calculates the ratio of intra-cluster similarity to inter-cluster distinctness, where lower values are preferred.
3) The Calinski–Harabasz Index, which compares within-cluster variance to between-cluster variance, where higher values are desirable.

In addition, the Noise Ratio was calculated specifically for DBSCAN and FCM, representing the proportion of data points designated as noise.

The experiments revealed several interesting trends across datasets D1, D2, and D3. K-Means performed best on the synthetic dataset (D3), achieving a high SS (~0.71) and a low DBI (~0.42), indicating distinct and compact clusters. However, it did not perform well on noisy real-world data, particularly on Dataset D2.

By contrast, DBSCAN demonstrated strong robustness to noise and outperformed K-Means on real datasets, especially D2, due to its density-based clustering approach. FCM produced moderately good clustering quality across all datasets, with relatively balanced scores. However, it tended to yield slightly higher DBI values, likely due to the fuzziness of its membership assignments.

A-GBK-Means delivered consistently strong performance, especially on Dataset D1, where the true cluster structures were non-linear. In this case, the benefits of kernel functions and fuzzy weighting were particularly evident.

Figure 6 illustrates the comparative performance of the four clustering algorithms based on SS and DBI. Higher SSs reflect more distinct clusters, whereas lower DBI values indicate better separation and compactness. On the synthetic dataset (D3), K-Means leads in SS, whereas A-GBK-Means achieves the best DBI. DBSCAN presents a balanced profile across all three datasets but proves especially effective in noisy environments like D2. The quality of comparative clustering of four algorithms (K-Means, DBSCAN, FCM, and A-GBK-Means) on three sets of data, D1, D2, and D3, via SS and DBI. The A-GBK-Means method recorded the largest SS and the least DBI on all datasets, and especially, on a noisy dataset (D2), indicating a good separation and compactness of clusters.

6.3. Scalability assessment

Scalability was evaluated by gradually increasing the size of the synthetic dataset (D3) from 50,000 to 500,000 instances, in increments of 100,000. All clustering algorithms were executed on these increasingly larger subsets, and both execution time and internal validation measures were recorded to track variations in performance.

Figure 6
Clustering quality comparison (SS and DBI)

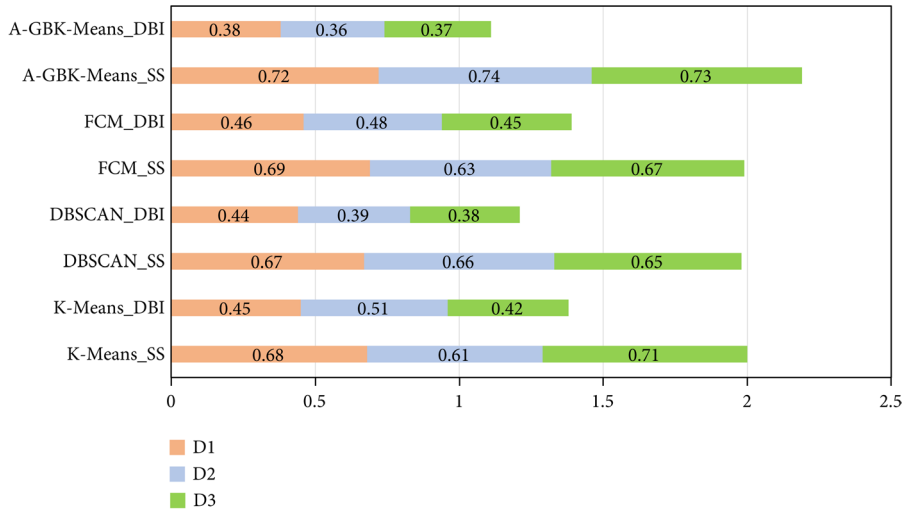
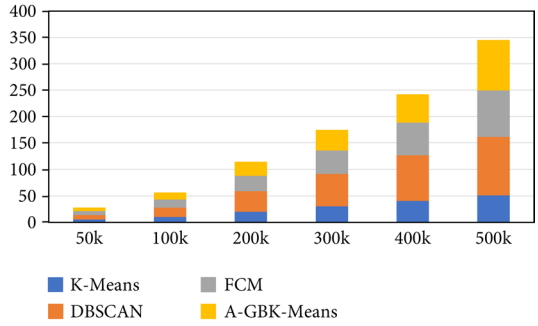


Figure 7
Total execution time (sec) comparison of clustering algorithms



The experiment revealed noticeable differences among the tested algorithms. K-Means demonstrated near-linear scalability, as its execution time increased proportionally with dataset size while maintaining stable clustering quality. In contrast, DBSCAN exhibited exponential growth in runtime beyond 100,000 records, primarily due to the computational intensity of its region-query operations.

FCM scaled better than DBSCAN, although it required more memory and computation time due to the iterative recomputation of fuzzy memberships. A-GBK-Means was the slowest among all algorithms but tended to achieve proportionally higher SSs on larger subsets, suggesting improved cluster structure definition despite the higher computational cost.

These observations highlight the trade-offs between computational performance and clustering quality as dataset size increases across different algorithms. Figure 7 shows the total execution time (in seconds) for each clustering algorithm across all dataset sizes. Running time of each clustering algorithm with a synthetic data set across growth in magnitude. K-Means proves to have the fastest execution time; however, it does not have clustering quality. A-GBK-Means strikes a balance between scalability and robustness, and A-GBK-Means matched performance with both DBSCAN and GBK-Means in terms via computation efficiency, although performance times were exponentially increasing with DBSCAN and GBK-Means.

By contrast, DBSCAN and A-GBK-Means incurred the highest computational costs due to their complexity, whereas FCM offers a reasonable balance between execution speed and clustering versatility.

7. Comparative Strengths of A-GBK-Means

To overcome the drawbacks of fixed kernel width and noise sensitivity in high-dimensional, heterogeneous big data, we propose an improved version of the A-GBK-Means.

In traditional GBK-Means, a fixed Gaussian kernel width is assumed for all data points; however, in settings where data density varies across the feature space, this assumption becomes inadequate. The novel model proposed in this paper addresses this limitation by making the kernel adaptive on a per-point basis, using local density estimates. This adaptation enables the algorithm to better handle non-uniform cluster structures and outliers. Specifically, the kernel function is dynamically rescaled by the local density $\rho(x_i)$, and is characterized as Equation (3) as follows:

$$K(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{2 \cdot \sigma(x_i)^2}\right) \tag{3}$$

The adaptive kernel width $\sigma(x_i)$ is calculated based on the average distance to k-nearest neighbors in Equation (4) as follows:

$$\sigma(x_i) = \frac{1}{k} \sum_{j=1}^{k} \left| x_i - x_j^{(NN)} \right| \tag{4}$$

where $x_j^{(NN)}$ denotes the jth nearest neighbor of x_i.

We define a fuzzy membership matrix $U = [u_{ij}]$, where each $u_{ij} \in [0, 1]$ represents the degree to which x_i belongs to cluster j. Membership values are updated as Equation (5) as follows:

$$u_{ij} = \left(\sum_{l=1}^{c} \left(\frac{D_{ij}}{D_{il}} \right)^{\frac{2}{m-1}} \right)^{-1} \tag{5}$$

Here, D_{ij} is the kernel-based distance between point x_i and cluster center c_j, and m is the fuzzifier coefficient ($m > 1$). To overcome such shortcomings of the traditional clustering analysis in heterogeneous and noisy data environments, we introduce the A-GBK-Means algorithm.

It is a locally adaptive kernel width estimation method using k-nearest neighbors, and fuzzy membership weighting is incorporated to enhance robustness and accuracy. The algorithm iteratively updates centroids and fuzzy memberships until convergence.

Algorithm 1 summarizes the complete step-by-step workflow.

Algorithm 1
A-GBK-Means

1. Input dataset: $X = \{x_1, x_2, ..., x_n\}$, number of clusters k, fuzzifier m, convergence threshold ε
2. Initialize cluster centers $C = \{c_1, ..., c_k\}$ using k-means++
3. For each x_i, compute $\sigma(x_i)$ using k-nearest neighbors
4. Compute Gaussian kernel matrix $K(x_i, x_j)$
5. Initialize fuzzy membership matrix $U = [u_{ij}]$
6. Repeat until convergence:

a. Update cluster centroids c_j using weighted kernel distances

b. Update u_{ij}

c. Check: $|U^{(t)} - U^{(t-1)}| < \varepsilon$

7. Output: Final clusters and memberships

The architectural flow of the proposed A-GBK-Means algorithm is demonstrated in Figure 8. The stages start with the input data, which is high-dimensional or noisy, and undergo adaptive kernel width estimation depending on the local density in the input space using the k-nearest neighbor method. This adaptive estimation enables the algorithm to adapt to variations in data density in the feature space.

After that, a Gaussian kernel matrix is calculated with adaptive widths, which encodes nonlinear interactions among data points. The following step incorporates a fuzzy membership facility where the data points are softly assigned to clusters using kernelized distances and a set fuzzifier parameter.

The architecture then iteratively optimizes itself through centroid updates and membership computations until it converges. The last step provides the fuzzy membership matrix and the optimized cluster structures. The whole pipeline can be scaled and parallelized and can thus be executed in a distributed big data processing system like Apache Spark.

Figure 8
Architecture of the proposed A-GBK-Means algorithm

This figure underscores the modular nature of the algorithm and its potential to integrate adaptive kernel learning with the principles of fuzzy clustering, which makes the outcome a more stable and adaptable clustering method to deal with complicated data.

The time complexity per iteration is approximately the following:

$$O(nkd + nk\ log\ k) \tag{6}$$

Where:

1) n = number of data points
2) k = number of clusters
3) d = data dimensionality
4) log k = approximate kNN search cost

A-GBK-Means clustering algorithm shows evident performance benefits compared to the other clustering methods that are tested within the framework of this research.

The latter combines adaptive kernel scaling and fuzzy membership weighting and thus adaptively adjusts to the local data distributions and is capable of handling noise and outliers efficiently.

First, the A-GBK-Means algorithm shows the best SS (0.74) compared to all the other algorithms, indicating that its clusters are compact and well-separated, as shown in Table 8. This is especially useful in high-dimensional as well as heterogeneous data, where distance-based clustering algorithms do not perform well.

Second, the technique provides the lowest DBI (0.36), demonstrating the best intra-cluster similarity and inter-cluster separation. The combination of these two internal quality measures confirms the ability of the algorithm to produce meaningful clusters even in the most complex data landscape.

Regarding robustness, the A-GBK-Means algorithm shows the best result in comparison with DBSCAN and FCM, as it has a much lower noise rate (2.0%), yet it detects outliers and ambiguous data points. As opposed to DBSCAN, which is highly sensitive to parameter adjustment in regions of different densities, the adaptive kernel mechanism in A-GBK-Means enables it to gradually adapt between dense and sparse regions.

Another strength is computational balance. Although it is slightly more demanding than K-Means or FCM in terms of execution time, A-GBK-Means is much faster than GBK-Means, yet it delivers better clustering quality. This is attributed to the local density estimates, which are efficiently used in computing the kernel, thereby reducing unnecessary computation in homogeneous areas.

1) Advantages

a. Adaptivity: The kernel width is adjusted dynamically based on local data density.
b. Noise robustness: Achieves a low noise ratio and stable clustering under noisy conditions.

Table 8
Clustering performance comparison

Algorithm	Silhouette Score	DBI	Noise Ratio (%)	Execution Time (sec)
K-Means	0.71	0.42	0	68.9
DBSCAN	0.65	0.38	8.5	110.5
FCM	0.67	0.45	6.1	88.3
GBK-Means	0.69	0.4	3.2	132.4
A-GBK-Means	0.74	0.36	2	95.6

c. High quality: Produces superior SS and DBI.
d. Soft clustering: Fuzzy logic allows better handling of ambiguous data points.
e. Scalability: Efficient for large datasets and compatible with distributed systems.

2) Disadvantages

a. Parameter sensitivity: Requires careful tuning of parameters such as m and k.
b. Computational overhead: More resource-intensive than basic clustering methods.
c. Initialization dependency: Outcome may depend on initial cluster centers.
d. Interpretability: Fuzzy results may be harder to interpret in some scenarios.

Moreover, due to its fuzzy membership mechanism, A-GBK-Means can be effectively applied in practical scenarios where data points may belong to multiple clusters simultaneously.

This flexibility enhances its interpretability and real-world applicability, particularly in fields such as anomaly detection, healthcare analytics, and industrial diagnostics.

In conclusion, the combination of adaptive kernel learning, soft clustering, and structural robustness makes A-GBK-Means a strong candidate for scalable and noise-resistant clustering in big data environments.

The proposed A-GBK-Means algorithm has a few benefits where fault diagnosis is concerned, especially in the field of sensors-based industrial systems. Rigid cluster boundaries or density fluctuation may be a problem of the traditional clustering algorithms such as K-Means and DBSCAN so that fine fault structures are difficult to differentiate. Conversely, A-GBK-Means makes use of local density-based adaptive kernel widths and this allows the algorithm to extract nonlinear, irregular, and spatially sparse fault signals. Besides, the fuzzy membership model enables a partial membership to be ascribed to ambiguous or borderline instances, and in cases where such pertain to detecting faults in the early stages of their existence, or intermittent faults, this feature is highly useful. Nevertheless, the difficulty in fixing the fuzzifier and kernel parameters of each fault type of signature and meanwhile the soft assignment interpretation in critical fault cases is still apparent. Regardless of these shortcomings, the technique offers a scalable and pliant framework of unsupervised fault pattern detection in noisy conditions.

In a real-life dataset, particularly risk assessment data, the examples contained in the data can be in several overlapping sets of behavior or they could be inaccurate because of sensor noise or data sparseness. The A-GBK-Means algorithm combats this difficulty in two ways: first, the adaptive kernel width adapts to local data density such that the model can distinguish subtle structural variations even in overlapping areas. Second, soft clustering is made possible by the fuzzy membership assignment, such that every point may be part of more than one cluster, and the part is measured. The soft assignment decreases the effects of hard-boundary misclassification and gives the algorithm a stronger ability to deal with ambiguous or marginal cases. Such attributes enable A-GBK-Means to be especially useful in early anomaly identification, uncertainty-based decision-making in risk-averse settings.

7.1. Comparative evaluation with studies

The progress of the current study is marked by several important distinguishing factors.

First, regarding execution platforms, previous studies primarily relied on Hadoop MapReduce, which is intrinsically slower due to its disk-based processing model. In contrast, our Spark-based design leverages in-memory computation, significantly reducing job latency and improving the responsiveness of iterative algorithms.

Second, in terms of algorithmic scope, earlier research was largely limited to K-Means, Hierarchical Clustering, and DBSCAN. In this study, however, we extend the analysis to include FCM and GBK-Means, the latter being particularly effective in the presence of noise and nonlinear cluster structures.

Furthermore, most previous studies utilized limited dataset designs, often constrained to small or synthetic data only. By comparison, our experimental setup incorporates both realistic industrial sensor data and synthetic datasets with adjustable volume and noise characteristics, allowing for a more realistic evaluation of algorithm performance.

Finally, in terms of evaluation measures, earlier work primarily focused on either execution time or cluster purity. Our approach adopts a more comprehensive set of metrics, including execution time, SS, DBI, noise ratio, and scalability trends. This multi-criteria assessment provides a deeper and more nuanced understanding of algorithm behavior under big data conditions.

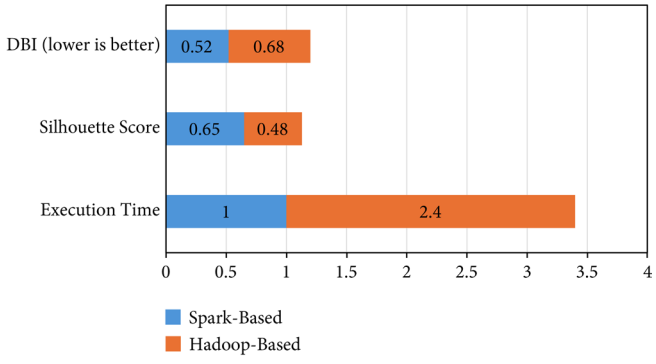
This study has demonstrated measurable performance gains in terms of reduced execution time (up to 2.4× faster) and improved clustering resilience, particularly in the presence of noisy and high-volume data (Figure 9). By incorporating fuzzy and kernel-based models and testing them on real-world sensor data, the study goes beyond the methodological scope of most previous research.

8. Conclusion

This paper presented the comparative evaluation of four clustering algorithms, namely K-Means, DBSCAN, FCM, and GBK-Means, on real-world industrial sensor data as well as on synthetic datasets in a distributed Apache Spark cluster. To evaluate the performance, scalability, and robustness, the multi-metric framework, including SS, DBI, execution time, and noise ratio, was used.

According to the results, it can be noted that although K-Means is very scalable and has low execution time, it is vulnerable to noise. DBSCAN is very noise-resistant but inefficient with large-scale data. FCM offers interpretable fuzzy clustering at moderate computational expense, and GBK-Means is competitive when the data structure is complex, at the cost of runtime. To address these shortcomings, we have suggested the A-GBK-Means algorithm, which incorporates adaptive kernel width estimation using local density with fuzzy membership modeling. The provided approach demonstrated better results on all baselines on clustering quality and noise processing, as well as decent computational expense and scalability with Spark.

Figure 9
Comparative performance with prior studies



In the proposed framework, the future development will consider enhancing it with the capabilities of processing data streams in real time with the help of Apache Kafka and Spark Streaming. Automated hyperparameter tuning techniques will also be considered to achieve optimized performance without relying on manual efforts.

We also seek ways to consider the hybrid clustering models combining A-GBK-Means with deep learning or evolutionary algorithms and extend the framework to other fields, including healthcare analytics, smart city infrastructure, and cyber-physical systems. Additional benchmarking of performance on GPU-based and cloud-native platforms will also be considered to have practical deployment readiness.

In future experiments, an ablation study will be conducted to isolate the effects of adaptive kernel scaling versus static kernel settings and fuzzy weighting. This will clarify the specific performance contribution of each enhancement in the A-GBK-Means framework.

Recommendations

In line with the findings of this paper, we suggest the application of adaptive kernel-based clustering models particularly in industrial applications in which the sensor data are big and noisy. Practitioners should consider adopting the suggested A-GBK-Means approach on Apache Spark set ups to balance the accuracy of clustering and scalability. Further development would involve new integrations with real-time processing framework, for example, Apache Kafka, to improve streaming data analytics performance.

Acknowledgement

The authors wish to thank Azerbaijan State Oil and Industry University that allowed the use of its computing resources to be able to conduct this study. A special note of appreciation is also justified to the Department of Instrumentation Engineering who has provided unlimited support in terms of academics as well as technicalities in the course of carrying out this research.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in Kaggle at https://www.kaggle.com/datasets/dnkumars/industrial-equipment-monitoring-dataset and https://www.kaggle.com/datasets/mohammedarfathr/sensor-data-from-an-industrial-machine.

Author Contribution Statement

Bakhshali Bakhtiyarov: Investigation, Data curation, Writing – original draft, Project administration. **Aynur Jabiyeva:** Conceptualization, Validation, Formal analysis. **Anakhanim Mutallimova:** Software, Resources. **Rukhsara Novruzova:** Methodology, Writing – review & editing, Supervision. **Mahabbat Khudaverdiyeva:** Visualization.

References

[1] Wang, J., Yang, Y., Wang, T., Sherratt, R. S., & Zhang, J. (2020). Big data service architecture: A survey. *Journal of Internet Technology*, *21*(2), 393–405. https://doi.org/10.3966/160792642020032102008

[2] Furinto, A., Tamara, D., Maradona, C., & Gunawan, H. (2023). How TikTok and its relationship with narcissism and bandwagon effect influenced conspicuous consumption among Gen Y and Gen Z. *E3S Web of Conferences*, *426*, 01101. https://doi.org/10.1051/e3sconf/202342601101

[3] Das, A., Das, S., & Rathee, N. (2021). Roles of big data, data science, artificial intelligence in entrepreneurships. In *Proceedings of the International Conference on Advances in Management Practices*, 1–10.

[4] Omran, M. G. H., Engelbrecht, A. P., & Salman, A. (2007). An overview of clustering methods. *Intelligent Data Analysis*, *11*(6), 583–605. https://doi.org/10.3233/IDA-2007-11602

[5] Venkatesh, C., & Bojja, P. (2022). A dynamic optimization and deep learning technique for detection of lung cancer in CT images and data access through internet of things. *Wireless Personal Communications*, *125*(3), 2621–2646. https://doi.org/10.1007/s11277-022-09676-0

[6] Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2012). The planar *k*-means problem is NP-hard. *Theoretical Computer Science*, *442*, 13–21. https://doi.org/10.1016/j.tcs.2010.05.034

[7] Sokiyna, M., Aqel, M. J., & Naqshbandi, O. A. (2020). Cloud computing technology algorithms capabilities in managing and processing big data in business organizations: MapReduce, hadoop, parallel programming. *Journal of Information Technology Management*, *12*(3), 113–126.

[8] Bakhtiyarov, B., & Jabiyeva, A. (2025). A fuzzy logic-based approach to multiple evaluation factors in big data technology. In *Intelligent and Fuzzy Systems: Artificial Intelligence in Human-Centric, Resilient and Sustainable Industries, Proceedings of the INFUS 2025 Conference*, *Volume 2*, 354–361. https://doi.org/10.1007/978-3-031-97992-7_40

[9] Udeh, C. A., Orieno, O. H., Daraojimba, O. D., Ndubuisi, N. L., & Oriekhoe, O. I. (2024). Big data analytics: A review of its transformative role in modern business intelligence. *Computer Science & IT Research Journal*, *5*(1), 219–236. https://doi.org/10.51594/csitrj.v5i1.718

[10] Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., & Akinyelu, A. A. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, *110*, 104743. https://doi.org/10.1016/j.engappai.2022.104743

[11] Rane, J., Mallick, S. K., Kaya, Ö., & Rane, N. L. (2024). *Future research opportunities for artificial intelligence in Industry 4.0 and 5.0*. Deep Science Publishing. https://doi.org/10.70593/978-81-981271-0-5_2

[12] Awad, F. H., & Hamad, M. M. (2023). Big data clustering techniques challenged and perspectives: Review. *Informatica*, *47*(6), 203–218. https://doi.org/10.31449/inf.v47i6.4445

[13] Alasali, T., & Ortakci, Y. (2024). Clustering techniques in data mining: A survey of methods, challenges, and applications. *Computer Science*, *9*(1), 32–50. https://doi.org/10.53070/bbd.1421527

[14] Kapil, S., & Chawla, M. (2016). Performance evaluation of K-means clustering algorithm with various distance metrics. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems*, 1–4. https://doi.org/10.1109/ICPEICES.2016.7853264

[15] Jin, X., & Han, J. (2016). K-Medoids clustering. In D. Phung, G. I. Webb, & C. Sammut (Eds.), *Encyclopedia of machine learning and data science* (pp. 1–3). Springer. https://doi.org/10.1007/978-1-4899-7502-7_432-1

[16] Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, *25*(2), 103–114. https://doi.org/10.1145/235968.233324

[17] Ran, X., Xi, Y., Lu, Y., Wang, X., & Lu, Z. (2023). Comprehensive survey on hierarchical clustering algorithms and the recent developments. *Artificial Intelligence Review*, *56*(8), 8219–8264. https://doi.org/10.1007/s10462-022-10366-3

[18] Xie, Y., Shekhar, S., & Li, Y. (2023). Statistically-robust clustering techniques for mapping spatial hotspots: A survey. *ACM Computing Surveys*, *55*(2), 36. https://doi.org/10.1145/3487893

[19] Bushra, A. A., & Yi, G. (2021). Comparative analysis review of pioneering DBSCAN and successive density-based clustering algorithms. *IEEE Access*, *9*, 87918–87935. https://doi.org/10.1109/ACCESS.2021.3089036

[20] Fu, N., Ni, W., Hu, H., & Zhang, S. (2023). Multidimensional grid-based clustering with local differential privacy. *Information Sciences*, *623*, 402–420. https://doi.org/10.1016/j.ins.2022.12.037

[21] Bhattacharjee, P., & Mitra, P. (2021). A survey of density based clustering algorithms. *Frontiers of Computer Science*, *15*(1), 151308. https://doi.org/10.1007/s11704-019-9059-3

[22] Ünlü, R., & Xanthopoulos, P. (2019). Estimating the number of clusters in a dataset via consensus clustering. *Expert Systems with Applications*, *125*, 33–39. https://doi.org/10.1016/j.eswa.2019.01.074

[23] Appiah, S. K., Wirekoh, K., Aidoo, E. N., Oduro, S. D., & Arthur, Y. D. (2022). A model-based clustering of expectation–maximization and *K*-means algorithms in crime hotspot analysis. *Research in Mathematics*, *9*(1), 2073662. https://doi.org/10.1080/27684830.2022.2073662

[24] Theng, D., & Bhoyar, K. K. (2024). Feature selection techniques for machine learning: A survey of more than two decades of research. *Knowledge and Information Systems*, *66*(3), 1575–1637. https://doi.org/10.1007/s10115-023-02010-5

[25] Zhao, S., Zhang, B., Yang, J., Zhou, J., & Xu, Y. (2024). Linear discriminant analysis. *Nature Reviews Methods Primers*, *4*(1), 70. https://doi.org/10.1038/s43586-024-00346-y

[26] Ponnusamy, S., & Gupta, P. (2024). Scalable data partitioning techniques for distributed data processing in cloud environments: A review. *IEEE Access*, *12*, 26735–26746. https://doi.org/10.1109/ACCESS.2024.3365810

[27] Dafir, Z., Lamari, Y., & Slaoui, S. C. (2021). A survey on parallel clustering algorithms for Big Data. *Artificial Intelligence Review*, *54*(4), 2411–2443. https://doi.org/10.1007/s10462-020-09918-2

[28] Ragunthar, T., Ashok, P., Gopinath, N., & Subashini, M. (2021). A strong reinforcement parallel implementation of k-means algorithm using message passing interface. *Materials Today: Proceedings*, *46*, 3799–3802. https://doi.org/10.1016/j.matpr.2021.02.032

[29] Indira, K., Karthiga, S., Nisha Angeline, C. V., & Santhiya, C. (2021). Parallel CLARANS algorithm for recommendation system in multi-cloud environment. In *Computer Networks and Inventive Communication Technologies: Proceedings of Third ICCNCT 2020*, 461–472. https://doi.org/10.1007/978-981-15-9647-6_36

[30] Anand, R., Jain, V., Singh, A., Rahal, D., Rastogi, P., Rajkumar, A., & Gupta, A. (2023). Clustering of big data in cloud environments for smart applications. In R. Anand, S. Juneja, A. Juneja, V. Jain, & R. Kannan (Eds.), *Integration of IoT with cloud computing for smart applications* (pp. 227–247). Chapman and Hall; CRC Press. https://doi.org/10.1201/9781003319238-14

[31] Liu, Z., Huang, L., & Zong, X. (2020). Clustering analysis method and implementation in the MapReduce model for solving data-intensive problems. *International Journal of Computers and Applications*, *42*(6), 533–543. https://doi.org/10.1080/1206212X.2018.1477321

[32] Ouyang, T., & Shen, X. (2022). Online structural clustering based on DBSCAN extension with granular descriptors. *Information Sciences*, *607*, 688–704. https://doi.org/10.1016/j.ins.2022.06.027

[33] Xiao, Z., Xiao, M., Yuan, Y., Yu, D., Lee, R., & Zhang, X. (2025). A case study for ray tracing cores: Performance insights with breadth-first search and triangle counting in graphs. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *9*(2), 16. https://doi.org/10.1145/3727108

[34] Cecilia, J. M., Cano, J.-C., Morales-García, J., Llanes, A., & Imbernón, B. (2020). Evaluation of clustering algorithms on GPU-based edge computing platforms. *Sensors*, *20*(21), 6335. https://doi.org/10.3390/s20216335

[35] Ibtisum, S., Bazgir, E., Rahman, S. M. A., Hossain, S. M. S. (2023). A comparative analysis of big data processing paradigms: MapReduce vs. Apache Spark. *World Journal of Advanced Research and Reviews*, *20*(1), 1089–1098. https://doi.org/10.30574/wjarr.2023.20.1.2174

[36] Kulkarni, O., & Burhanpurwala, A. (2024). A survey of advancements in DBSCAN clustering algorithms for big data. In *2024 3rd International Conference on Power Electronics and IoT Applications in Renewable Energy and Its Control*, 106–111. https://doi.org/10.1109/PARC59193.2024.10486339

[37] Mussabayev, R., Mladenovic, N., Jarboui, B., & Mussabayev, R. (2023). How to use K-means for big data clustering? *Pattern Recognition*, *137*, 109269. https://doi.org/10.1016/j.patcog.2022.109269

[38] Soni, R., & Mehta, B. (2023). Diagnosis and prognosis of incipient faults and insulation status for asset management of power transformer using fuzzy logic controller & fuzzy clustering means. *Electric Power Systems Research*, *220*, 109256. https://doi.org/10.1016/j.epsr.2023.109256

[39] Demchenko, Y., Cuadrado-Gallego, J. J., Chertov, O., & Aleksandrova, M. (2024). *Big data infrastructure technologies for data analytics: Scaling data science applications for continuous growth*. Switzerland: Springer. https://doi.org/10.1007/978-3-031-69366-3

[40] Belcastro, L., Cantini, R., Marozzo, F., Orsino, A., Talia, D., & Trunfio, P. (2022). Programming big data analysis: Principles and solutions. *Journal of Big Data*, *9*(1), 4. https://doi.org/10.1186/s40537-021-00555-2

[41] Berkaoui, A., & Gahi, Y. (2023). High available Hadoop deployment modes with enterprise-level capabilities. In *2023 10th International Conference on Future Internet of Things and Cloud*, 26–32. https://doi.org/10.1109/FiCloud58648.2023.00012

[42] Zhang, Y., Cao, T., Li, S., Tian, X., Yuan, L., Jia, H., & Vasilakos, A. V. (2016). Parallel processing systems for big data: A survey. *Proceedings of the IEEE*, *104*(11), 2114–2136. https://doi.org/10.1109/JPROC.2016.2591592

[43] Das, S., & Mao, E. (2020). The global energy footprint of information and communication technology electronics in connected Internet-of-Things devices. *Sustainable Energy,*

Grids and Networks, *24*, 100408. https://doi.org/10.1016/j.segan.2020.100408

[44] Chicco, D., Ferraro Petrillo, U., & Cattaneo, G. (2023). Ten quick tips for bioinformatics analyses using an Apache Spark distributed computing environment. *PLOS Computational Biology*, *19*(7), e1011272. https://doi.org/10.1371/journal.pcbi.1011272

[45] Gabdullin, M. T., Suinullayev, Y., Kabi, Y., Kang, J. W., & Mukasheva, A. (2024). Comparative analysis of Hadoop and Spark performance for real-time big data smart platforms utilizing IoT technology in electrical facilities. *Journal of Electrical Engineering & Technology*, *19*(7), 4595–4606. https://doi.org/10.1007/s42835-024-01937-1

[46] Sarang, P. (2023). *Thinking data science: A data science practitioner's guide*. Switzerland: Springer. https://doi.org/10.1007/978-3-031-02363-7

[47] Fahim, A. (2023). Adaptive density-based spatial clustering of applications with noise (ADBSCAN) for clusters of different densities. *Computers, Materials & Continua*, *75*(2), 3695–3712. https://doi.org/10.32604/cmc.2023.036820

[48] Hashemi, S. E., Gholian-Jouybari, F., & Hajiaghaei-Keshteli, M. (2023). A fuzzy C-means algorithm for optimizing data clustering. *Expert Systems with Applications*, *227*, 120377. https://doi.org/10.1016/j.eswa.2023.120377

[49] Fenyi, A., Nkansah, E., & Eghan, E. (2020). Gaussian based image segmentation algorithm. *International Journal of Computer Applications*, *175*(14), 10–16. https://doi.org/10.5120/ijca2020920615

[50] Zhou, M., Li, Q., Cai, M., Xu, C., & Wang, Y. (2025). Towards hybrid architectures for big data analytics: Insights from Spark-MPI integration. *IEEE Transactions on Services Computing*, *18*(3), 1852–1868. https://doi.org/10.1109/TSC.2025.3562342

How to Cite: Bakhtiyarov, B., Jabiyeva, A., Mutallimova, A., Novruzova, R., & Khudaverdiyeva, M. (2026). Adaptive Gaussian-Based Kernel K-Means: Scalable Adaptive Kernel-Based Clustering for Big Data. *Journal of Computational and Cognitive Engineering*, *5*(2), 258–273. https://doi.org/10.47852/bonviewJCCE52026511