# Parallelization of K-Means and Spatial Join Algorithms on Heterogeneous Platforms Using Apache Spark and GPU Integration for Enhanced AI Information Management

Danqiong Wang
Information Office, University of Shanghai for Science and Technology, Shanghai 200093, China
E-mail: danqiongwang@163.com

**Keywords:** big data mining, K-means, spatial join algorithm, parallel computing, spark

**Received:** July 21, 2025

*In the era of artificial intelligence informatization, real-time mining of massive spatial data has become the core bottleneck of intelligent decision-making, and existing methods have problems of poor computational performance and high complexity. Therefore, this study proposes a novel solution based on heterogeneous computing platforms. The approach employs Apache Spark to design a hybrid system integrating Central Processing Units (CPUs) and Graphics Processing Units (GPUs). It achieves parallelization of the K-means algorithm through Spark's elastic distributed datasets and broadcast variables, optimizing both the initial cluster centre selection and new centre determination steps. Concurrently, upper and lower bound constraints alongside group filtering techniques are introduced to reduce computational complexity. For spatial join algorithms, the study achieves efficient spatial data mining and dynamic load balancing through spatial index partitioning and the Compute Unified Device Architecture (CUDA) dynamic parallelization strategy. Experiments have shown that the parallelized K-means algorithm exhibited significantly improved acceleration on different data dimensions. Especially with an acceleration ratio of 45.32 times on 90-dimensional data, the execution efficiency was 0.31 times higher than Spark MLlib. The parallel spatial join algorithm achieved optimal performance with 1,500 partitions, completing computations in just 37.5 seconds while maintaining a data mining accuracy of 94.2%, surpassing traditional algorithms. Its maximum data mining accuracy reached 94.2%, exceeding DBSCAN and GeoSpark by 3.7% and 4.4%, respectively. The research method effectively solves the real-time problem of spatial data mining in artificial intelligence information management, providing scalable technical support for scenarios such as smart cities and autonomous driving.*

*Povzetek: Heterogena CPU–GPU metoda za množično prostorsko rudarjenje podatkov močno pospeši K-means algoritem in prostorske združitve ter izboljša natančnost, kar omogoča učinkovito podporo sistemom v realnem času.*

## 1 Introduction

Technology is constantly advancing, especially with the expansion of the application scope of computer, communication, and sensor technologies, and the data of various things are growing rapidly [1]. The global data volume has exceeded 120 ZB and is expected to reach over 180 ZB by the end of 2025. The global data show an exponential leap, with a compound annual growth rate of about 40%-60% in the past decade and an expected annual growth rate of 30% in the future [2]. The current daily data generation is 463 EB, equivalent to producing 210 million high-definition power supplies every day. Among them, the data contribution rate of over 27 billion Internet of Things related devices worldwide exceeds 40%, and the demand for real-time data processing, such as drones and autonomous driving, is growing by over 35% annually [3]. The global market value of data processing and analysis can reach hundreds of billions of dollars, with Big Data Mining (BDM) and analysis expected to reach 115 billion dollars by 2025 [4]. The size of China's data analysis market is expected to reach 32 billion US dollars by 2025. Related policies, such as the "Action Plan for Building Digital China 2025", require that the data element market be basically established by 2025, with computing power exceeding 300EFLOPS, and the core industries of the digital economy accounting for over 10% of GDP [5]. Driven by policies and markets, the "East Data West Computing" project is accelerating the construction of data centers and computing power networks [6]. However, traditional data mining methods are prone to problems such as poor computational performance and high complexity. The existing solutions are usually parallel computing, but how to efficiently process data in parallel computing has become a key issue.

In response to the problem of limited computing resources in parallel computing, Mohajer et al. designed a novel dynamic optimization model to reduce the impact of user layer resources and computational limitations. This model maximized the total energy efficiency of the uplink and downlink while satisfying the quality-of-service requirements. The sub-gradient method was used for computing resource allocation, and the maximum minimum fairness problem was solved through successive

convex approximation and double decomposition methods. This model significantly improved the overall throughput of mobile computing services and enhanced system energy efficiency [7]. Liu et al. proposed a new heterogeneous parameter server framework to enhance the allocation efficiency of heterogeneous computing resources. This framework consisted of a distributed architecture and a Reinforcement Learning (RL)-based scheduling method, which efficiently scheduled workloads from each layer to appropriate computing resources utilizing RL-based methods. The throughput of this framework has increased by 14.5 times, and the economic cost has been reduced by 312.3%, significantly better than existing methods [8]. Zong et al. proposed a new cross-distributed cluster planning heterogeneous parallel strategy optimizer to enhance the computing power of interconnected heterogeneous clusters. This optimizer performed model partitioning between heterogeneous hardware and introduced a hierarchical search algorithm to solve optimization problems, using a mixed precision pipeline method to reduce the cost of inter-cluster communication. Compared to the state-of-the-art Metis, this optimizer has increased the average training throughput by 1.49 times [9]. The above research contributes to heterogeneous resource scheduling, but it mainly focuses on general computing tasks or machine learning frameworks, without deep optimization of specific computationally intensive algorithm kernels such as K-means or spatial connections. Moody et al. proposed a new, improved version of the K-Means Clustering algorithm (K-Means) to reduce the workload of big data processing. This algorithm used the distance from a certain point to its 2 Nearest Cluster Centers (NCCs) and its changes in the last 2 iterations. When points with a distance threshold superior to the equidistant index were excluded from distance calculation and remained stable in clustering, this algorithm could improve clustering quality by 41.85% in the best-case scenario, which had significant advantages for big data processing [10]. Djafri et al. proposed a dynamic distributed parallel Machine Learning (ML) algorithm to enable ML algorithms to run in both distributed and parallel modes simultaneously. They designed a distributed architecture that relies on random sampling, using a hierarchical random sampling method to extract representative learning bases at two levels, as well as shared learning bases, partial learning bases at the 1st level, and partial learning bases at the 2nd level. This algorithm had high efficiency without significantly reducing the classification results [11]. Vu et al. proposed a new ML-based Spatial Join (SJ) query optimization framework to address the high complexity of SJ operations in big data platforms. This framework defined a set of efficiently computable features to capture the complex details of SJ, and used these features to train different ML models for identifying the optimal SJ. The first two models were regression models utilized to estimate 2 vital indicators of SJ performance and serve as cost models. This framework had higher efficiency than baseline methods on both large-scale synthetic data and real data [12]. Yang proposed a new, improved K-Means to reduce the risk of user privacy leakage during BDM. The algorithm first generated a noise distance, then synthesized high-dimensional data records, and used bounded Laplacian method and clustering indistinguishable method to sample the noise distance, satisfying local differential privacy protection and local privacy protection, respectively. This algorithm effectively reduced the probability of information leakage without affecting the efficiency of data mining [13]. The improved K-means algorithm mentioned above effectively enhances the performance of a single machine, but integrating its optimization principles with distributed parallel computing models such as Spark and GPU is still an unexplored field.

In summary, existing research has explored the improvement and parallelization optimization methods of the BDM algorithm from multiple aspects and has achieved certain research results. However, the existing methods still have problems such as poor computational performance and high model complexity. Mainstream libraries such as Spark MLlib lack native support for GPU computing, thereby failing to harness its massive parallelism. Traditional spatial join algorithms are prone to data skew and load imbalance in distributed environments. Most optimization efforts target either Central Processing Unit (CPU) or Graphics Processing Unit (GPU) platforms in isolation, failing to fully realize the potential of CPU-GPU collaborative heterogeneous computing. Therefore, this study proposes a BDM analysis optimization method based on K-means and the SJ algorithm on heterogeneous platforms, and innovatively constructs a hybrid system of CPU and GPU using Apache Spark. The system uses Spark's elastic distributed dataset and broadcast variables to parallelize the K-means, with a focus on optimizing the selection steps of initial and new centers, and introducing upper and lower limit constraints and group filtering techniques. For the SJ algorithm, this study employs Spatial Index Partitioning (SIP) and Compute Unified Device Architecture (CUDA) dynamic parallel strategies to reduce communication volume through spatial index compression in Java Native Interface (JNI) transmission. This study aims to provide innovative ideas for the parallelization optimization of BDM and promote the application of Artificial Intelligence (AI) information management in data-intensive tasks. This study has three sections. Section 1 is about the parallelization improvement of data mining algorithms. Section 2 is about the experimental analysis of the BDM algorithm. Section 3 summarizes the entire text.

# 2 Methods and materials

## 2.1 Parallel design of BDM based on K-means algorithm

Parallel computing is a computational method that uses multiple computing resources simultaneously to solve a problem. Its core is to decompose a large task into multiple sub-tasks and allocate them to multiple different processing units for simultaneous execution. It can be

implemented through multi-core CPUs, GPUs, or distributed computing clusters. Parallel computing can effectively shorten computation time and improve throughput. The computing resources of parallel computing can be isomorphic. Multiple identical CPU cores can also be heterogeneous, such as mixing multiple CPUs and GPUs with different architectures, with a focus on parallel execution of tasks [14]. To systematically address performance bottlenecks in large-scale spatial data mining, a heterogeneous parallel computing framework based on Apache Spark and GPUs has been designed. The core concept involves offloading computationally intensive tasks to GPUs for acceleration, while leveraging Spark for efficient distributed data management and task scheduling. The overall research workflow, illustrated in Figure 1, comprises the following stages: (1) Data Preprocessing and Partitioning: Raw spatial data are loaded into Spark, undergoes cleaning and formatting, and is partitioned using spatial indexes to prepare for parallel computation; (2) CPU-GPU collaborative computation: Through the JNI interface, data are effectively transmitted to the GPU and optimized K-means or spatial connection algorithms are executed in parallel using the CUDA kernel; (3) Result aggregation and output: The GPU calculation results are returned to the Spark end for aggregation, and finally output data mining patterns and knowledge. The core innovation of this research lies in the parallelization, adaptation, and optimization strategies for both algorithms within this framework.

The K-Means is often applied to clustering tasks on large-scale datasets, and its core computational steps have natural characteristics such as parallelizability, data independence, and computational intensity, making it suitable for data parallelization processing. This study uses the elastic distributed dataset and broadcast variables in the Spark platform to parallelize K-means, with the main parallelization objects being the initial center and new center selection steps of K-means. There are two methods for parallelizing the original center of K-means, the first being random selection. The second method first randomly selects a cluster center, calculates the distance from the data point to the NCC, and then selects the next cluster center based on the distance. To overcome the potential issue of slow convergence caused by random initialization, the study employs sampling with probability proportional to the square of the distance. This ensures that the initial center points are dispersed as widely as possible, thereby significantly improving the clustering quality and convergence speed. The calculation is shown in equation (1) [15].

$$P(x_i) = \frac{D(x_i)}{\sum_{x_j \in X} D(x_j)} \tag{1}$$

In equation (1), $P(x_i)$ is the probability that data point $x_i$ is chosen as the next cluster center. $D(x_i)$ is the distance from $x_i$ to the NCC. $X$ is the collection of all data points. $x_j$ is the $j$-th data point. This process is repeated continuously until K cluster centers are chosen. This method is capable of finding near-optimal clustering results with a high degree of probability. By positioning the initial centers as far apart as possible, it provides the algorithm with a favourable starting point, thereby enhancing the quality of the final clustering. The calculation of $D(x_i)$ is shown in equation (2).

$$D(x_i) = \min_{c_j \in C} = \left\| x_i - c_j \right\|^2 \tag{2}$$

In equation (2), $c_j$ is the $j$-th cluster center, and $C$ is the set of all cluster centers. The platform used is a CPU-GPU hybrid system built on Apache Spark, aimed at accelerating Spark's data mining through the powerful computing power of GPUs. Spark-GPU integration first performs parallel loading, cleaning, and SIP on the Spark side. Subsequently, it utilizes JNI to batch-copy the flattened 1D data into GPU memory. On the GPU side, parent-child kernels are launched dynamically in parallel via CUDA, executing MBR filtering and precise geometric testing, with only index results transmitted back. Finally, the Spark Driver aggregates results from each partition to output the final spatial clustering or linkage outcomes, thereby achieving large-scale spatial data mining through CPU-GPU collaboration. The Spark GPU system mainly accelerates the parallel feature extraction and analysis of data mining algorithms, without affecting the progress of other processes in the platform. The running process of the K-means algorithm in the Spark GPU platform is shown in Figure 1.

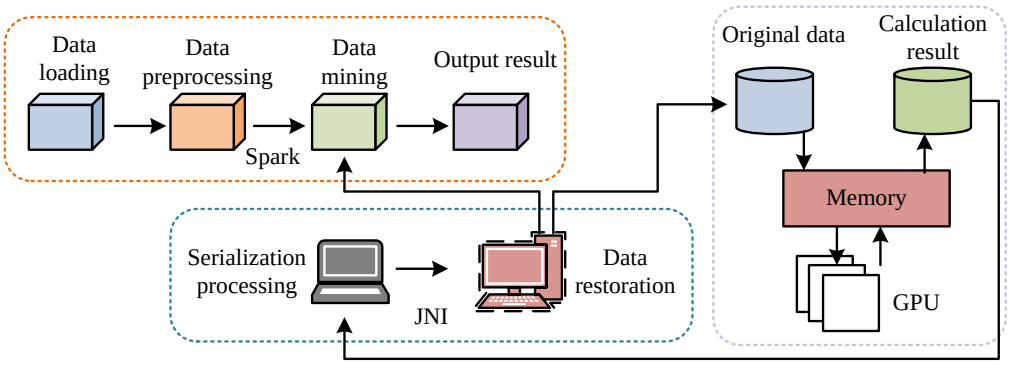


Figure 1: Operation process of the K-means in the Spark-GPU platform.

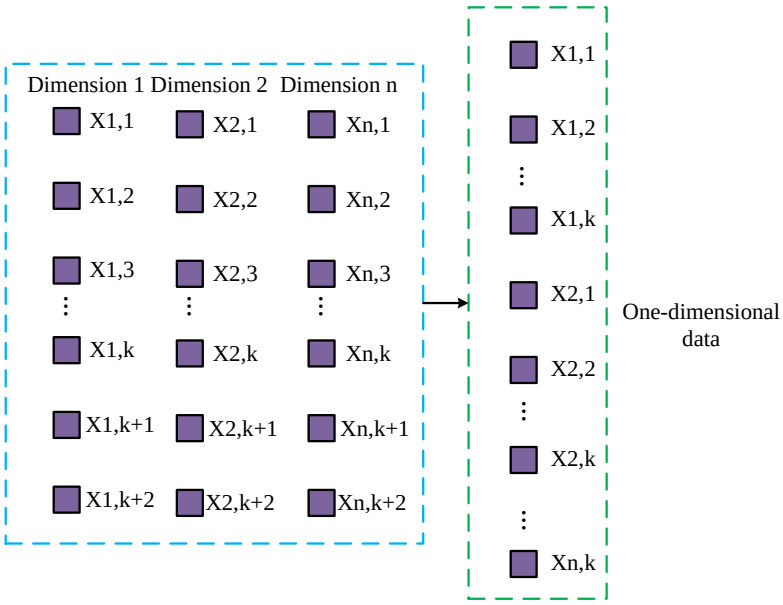


Figure 2: Data flattening processing.

In Figure 1, the data are preprocessed on the Spark platform by loading various data, such as removing duplicate data, handling missing values, and unifying formats. During preprocessing, missing attributes in the dataset are addressed by imputing with the mode or directly removing invalid records. Duplicate data are handled by eliminating identical records based on unique geometric IDs. Format standardization involves converting all geometric objects into the Well-Known Text (WKT) format, while Z-score standardization is applied to all numerical attributes to eliminate dimensional influences. The platform uses K-means to perform data mining tasks and obtains intermediate results through parallel computing to extract valuable knowledge or patterns from the data mining results, providing support for relevant decisions. After the data preparation on the Spark side is completed, this study uses JNI to transfer the data defined by the Java language to the GPU side. The GPU restores the received serialized data to the data structure defined in the C language. The results obtained after parallel computing on the GPU end are transmitted back to the Spark end through JNI for subsequent computation and analysis. GPU side graphics memory can effectively improve data computing efficiency, and multiple processors can simultaneously execute multiple threads. The speed of JNI technology during data transmission is greatly affected by the type of data, so this study flattens the data, as shown in Figure 2.

In Figure 2, the initial data type is multidimensional encapsulated data, and the transmission speed from Spark to GPU is slow. This study converts it into a one-dimensional array, which can effectively improve data transmission speed. At the same time, to reduce resource consumption during data transmission, the platform only transmits clustering result indexes instead of complete data in accordance with the pre-agreed data indexing order between Spark and GPU during JNI feedback, significantly reducing the amount of network transmission data. The primary computational overhead stems from directly calculating the distance between all data points and all cluster centers. To further reduce the computational complexity of the K-means algorithm during BDM, the study introduces Upper and Lower Bounds (U&L-B) constraints to avoid unnecessary distance computations. Its core principle leverages the triangle inequality, inferring potential changes in a data point's cluster membership by calculating the distance travelled by the centre point. The upper bound definition for the improved K-means algorithm is shown in equation (3).

$$U_l m(x) > d\left[x, m(x)\right] \tag{3}$$

In equation (3), $U_l m(x)$ is the upper bound boundary of the algorithm. $m(x)$ is the best center point that matches data point $x$. $d[\cdot]$ is the offset distance of the center point during iteration. The lower boundary of the improved K-means is shown in equation (4) [16].

$$L_l m(x) < d(x,c), \forall c \in C - m(x) \tag{4}$$

In equation (4), $L_l m(x)$ is the lower boundary of the algorithm, and $d(x,c)$ is the distance between $x$ and center point $c$. This study uses a global filtering method based on the U&L-Bs of the improved algorithm to enhance the performance of K-means. When the U&L-Bs of the algorithm satisfy equation (5), the clusters of the algorithm remain unchanged in the current iteration.

$$L_l m(x) - \max_{c \in C} \delta(c) \geq U_l m(x) + \delta(b) \tag{5}$$

In equation (5), $\delta(c)$ and $\delta(b)$ are the offset distances of the $c$ and $b$ in two iterations. Improved algorithms avoid unnecessary distance calculations through group filtering, achieving core acceleration. Group filtering first divides the cluster center into multiple groups, and the calculation is shown in equation (6).

$$cl = \{cl_1, cl_2, ..., cl_n\} \tag{6}$$

In equation (6), group filtering divides all cluster centers $cl$ into $n$ groups. After grouping, this study performs global filtering within each group, as shown in equation (7).

$$\begin{cases} L_l m(x, cl_i) < m(x, c), \forall c \in cl_i - m(x) \\ L_l m(x, cl_i) - \max \delta(c) \geq \underset{c \in cl_i}{U_l m(x)} + \delta(b) \end{cases} \quad (7)$$

In equation (7), the filtering effect is optimal when the number of groups $n$ is equal to one tenth of the number of cluster centers. Finally, local filtering conditions are introduced to further reduce the computational resource consumption of the algorithm, as shown in equation (8) [17].

$$d(x, c') < L_l m(x, cl_i) - \delta(c) \quad (8)$$

In equation (8), $c'$ is the center point in the next iteration. When $c'$ satisfies equation (8), the distance between $x$ and $c'$ is not the closest. When performing central updates in K-means, all points in the dataset will participate in the calculation. This study improves the center point update method of the algorithm, and the improved update formula is shown in equation (9).

$$c' = \frac{\left( c \cdot |A| - \sum_{c_i \in A - OA} c_i + \sum_{c_i' \in A' - OA} c_i \right)}{|A'|} \quad (9)$$

In equation (9), $A$ and $A'$ are the sets of center points for this iteration and the next iteration. $c_i$ is the $i$-th center point of this iteration. $c_i'$ is the $i$-th center point of the next iteration. $OA$ is the intersection of $A$ and $A'$. This study uses U&L-B sets to assist in implementing the above filtering steps. The upper boundary set is shown in equation (10).

$$U = \{U_l m(x_1), U_l m(x_2), ..., U_l m(x_n)\} \quad (10)$$

The lower boundary set is shown in equation (11).

$$L = \{L_l m(x_1, cl_1), L_l m(x_2, cl_2), ..., L_l m(x_n, cl_n)\} \quad (11)$$

The principle of the improved K-means running in CUDA is shown in Figure 3.

In Figure 3, the input data include the dataset size, point dimensions, and the number of center points. The algorithm is initialized to generate initial cluster centers. The algorithm uses CUDA for parallel initialization allocation, reallocating all data points to the NCC. When the algorithm meets the convergence requirements, it will output the result; otherwise, it will enter the iteration loop. The pseudo-code for the improved K-means algorithm is as follows:

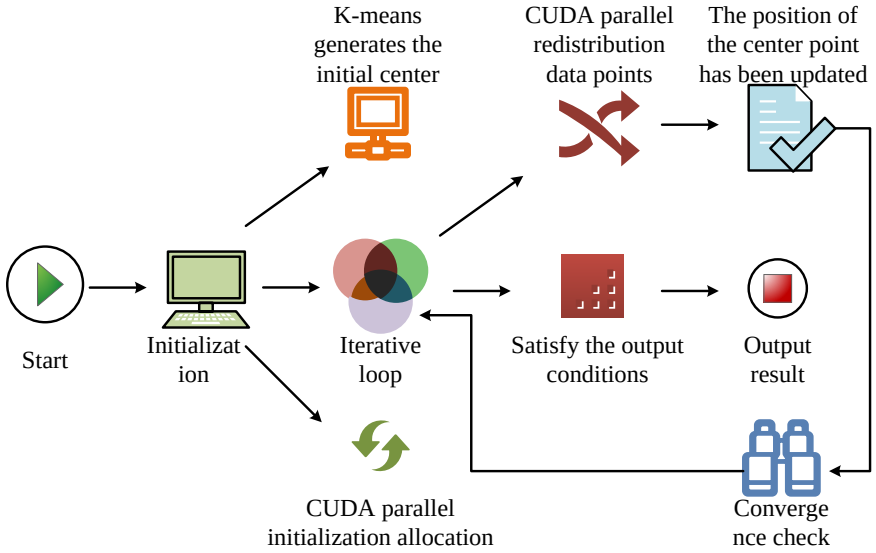


Figure 3: Specific process of the improved K-means running in CUDA.

Algorithm 1: Improved K-means with Bounds on Spark-GPU

Input: Dataset D, number of clusters K, max iterations M

Output: Final centroids C, cluster assignments A

1: // Spark Driver

2: C ← Initialize centroids using K-means++ strategy (Eq. 1-2)

3: Broadcast C to all Spark executors

4: Partition D into RDDs

5:

```
// Spark Executors (GPU accelerated)
for each partition in parallel do
    Transfer data partition to GPU memory via JNI
    Initialize lower bounds l(x) and upper bounds u(x) for each point x (Eq. 10-11)
    for iter = 1 to M do
        Kernel_GlobalFilter: Filter points using group filtering (Eq. 6-7)
        Kernel_LocalFilter: For remaining points, compute exact distances if needed (Eq. 8)
        Kernel_AssignPoints: Assign points to nearest centroid, update A
        Kernel_UpdateCentroids: Compute new centroids (Eq. 9)
        Kernel_UpdateBounds: Update l(x) and u(x) based on centroid movement (Eq. 3-5)
        if centroids converge then break
    end for
    Return assignments and partial sums to driver
end for

// Spark Driver
Aggregate results and compute final C
```

## 2.2 BDM based on SJ Algorithm Parallelization Design

The K-means-based BDM model is utilized in image segmentation and anomaly monitoring, but when facing massive spatial data such as geofencing analysis, autonomous driving, and remote sensing, SJ's BDM analysis performance is even better [18]. The SJ algorithm can associate objects in two spatial datasets based on spatial relationships (such as intersection, inclusion, adjacency, etc.), efficiently determining the topological relationships between geometric objects. The spatial relationship conditions for the SJ algorithm include intersection, containment, overlap, contact, and proximity within a specified distance. The goal of the SJ is to find all pairs of objects that satisfy spatial relationship conditions in two sets, as defined in equation (12) [19].

$$A \oplus B = \left\{ \left( a_i, b_i \right) \middle| a_i \in A, b_i \in B, S\left( a_i, b_i \right) = True \right\} \quad (12)$$

In equation (12), $A$ and $B$ are two sets. $a_i$ and $b_i$ denote the $i$ -th objects in sets $A$ and $B$ . $S\left( a_i, b_i \right) = True$ represents that $a_i$ and $b_i$ satisfy the spatial relationship condition. Spatial relationships include intersection, inclusion, and adjacency. The operation of the SJ algorithm in the Spark-GPU platform is shown in Figure 4.

In Figure 4, the SJ algorithm loads spatial data on the Spark side and performs preprocessing on the data, such as data cleaning and format conversion. The preprocessed data are partitioned to prepare for subsequent distributed computing. The processed data are transmitted to the GPU through JNI. On the GPU side, the minimum bounding rectangle is used for spatial data filtering, and possible spatial objects that can be added are screened by testing the intersection of edges and whether points are in polygons. After completing calculations in the GPU, the data are transmitted back to the Spark end, and the data mining results are integrated and output. The Spark-GPU-SJ algorithm can partition data through spatial indexing, effectively lowering the amount of data comparison and improving computational efficiency. The structure of SIP is shown in Figure 5.

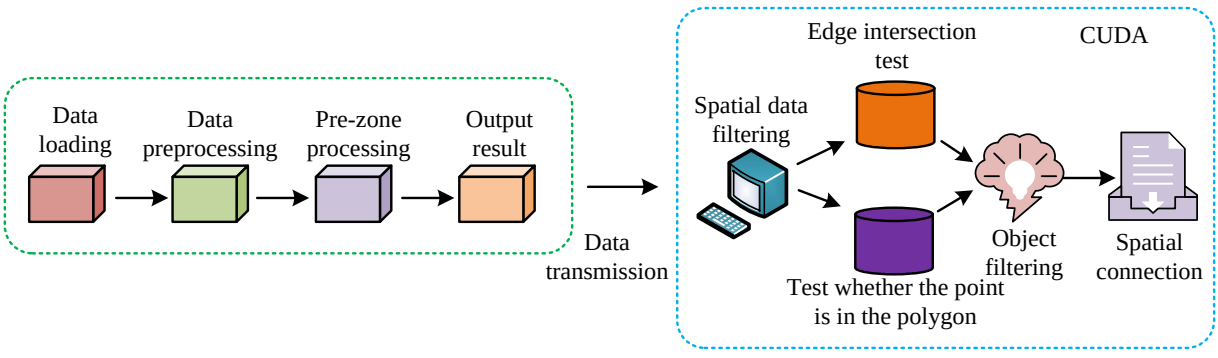


Figure 4: Operation process of the SJ algorithm in the Spark-GPU platform.

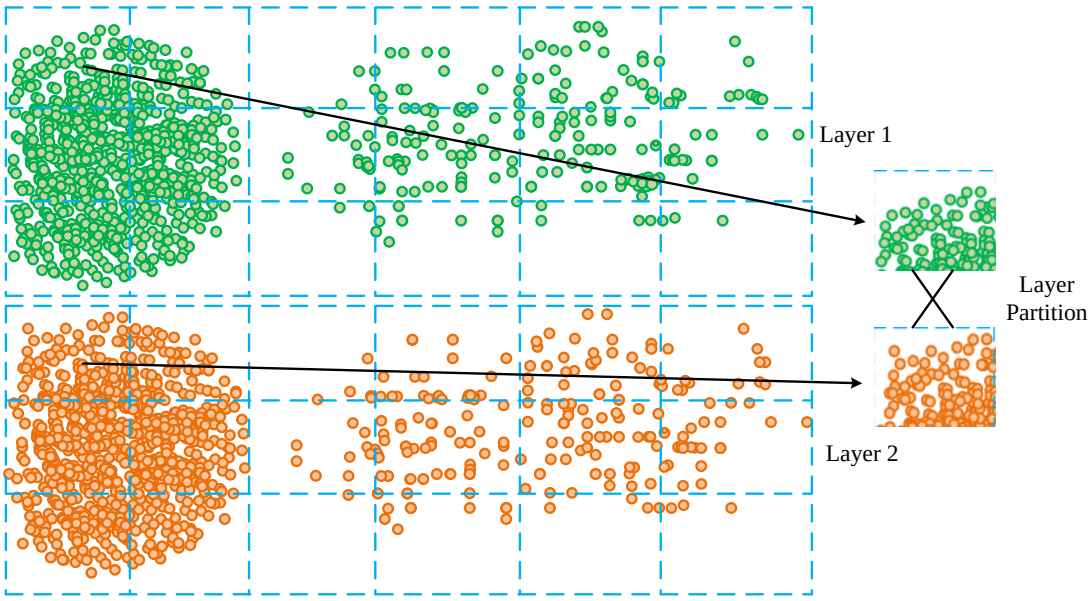


Figure 5: Schematic diagram of the structure of SIP.

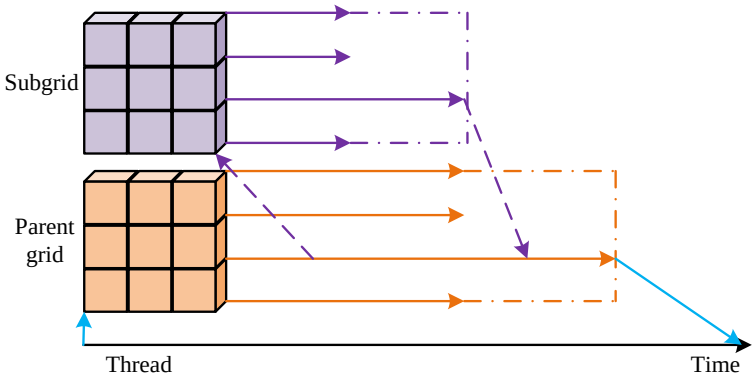


Figure 6: The basic principle of CUDA dynamic parallelism.

In Figure 5, the first step is to select a suitable spatial index structure, such as a K-dimensional tree, R-tree, quadtree, and grid index, and then determine the partitioning strategy, such as the number, shape, and size of partitions. The SJ algorithm divides all similar spatial objects into the same partition according to a determined strategy, and performs SJ calculations on objects in different partitions separately. When data are updated, their corresponding partition is separately calculated, and SJ is performed. To achieve dynamic load balancing and further improve the efficiency of data mining execution, this study introduces the CUDA dynamic parallel strategy. This strategy allows the CUDA kernel to directly create new jobs on the GPU and synchronize with them, without the need for CPU intervention, to start a new thread grid. The principle of CUDA dynamic parallelism is shown in Figure 6.

In Figure 6, the grid of the new kernel thread is started as the parent grid, and the grid of the newly started kernel is the sub-grid. The sub-grid inherits the memory space of the parent grid, but has its own execution scope and thread hierarchy. A single parent grid contains multiple sub-grids, which in turn have multiple sub-threads. The thread of the parent kernel will pass the startup configuration of the sub-kernel to the GPU scheduler. The GPU scheduler allocates resources and starts sub-kernels based on the startup configuration. The sub-kernel executes within its own thread hierarchy and can access the global memory of the parent kernel, but cannot directly access the local memory or shared memory of the parent kernel. The performance evaluation of parallel algorithms can be based on the Acceleration Ratio (Ar), execution efficiency, and scalability. The Ar calculation is shown in equation (13) [20].

$$Ar = \frac{T_1}{T_q} \tag{13}$$

In equation (13), $Ar$ is the Ar of the algorithm. $T_1$ is the computation time of a regular algorithm on a single processor. $T_q$ is the computation time of parallel algorithms on $q$ processors. The higher the $Ar$, the shorter the parallel running time of the algorithm. The execution efficiency of parallel algorithms is shown in equation (14).

$$Ee = \frac{Ar}{q} \tag{14}$$

In equation (14), $Ee$ is the execution efficiency of the algorithm, and its value is close to 1, indicating that the algorithm's performance is better. The scalability of parallel algorithms is shown in equation (15).

$$Sc = \frac{Q(z,1)}{Q(z \cdot q, q)} \tag{15}$$

In equation (15), $Sc$ means the scalability of the algorithm. $Q(z,1)$ is the computation time for each processor to independently mine the $z$ problem. $Q(z \cdot q, q)$ is the computation time for $q$ processors to mine the $z \cdot q$ problem. The larger $Sc$ value indicates better scalability of the algorithm. The pseudo-code for the SJ algorithm is as follows:

**Algorithm: Spark-GPU spatial join**

```
Input: Spatial datasets A, B; Spatial predicate θ
Output: Set R of object pairs satisfying θ

// Spark side
1. A_cleaned, B_cleaned = preprocess(A, B)  // Data cleaning & format conversion
2. index = create_spatial_index(A_cleaned ∪ B_cleaned)  // Create spatial index
3. partitions = partition_data(index)  // Spatial partitioning
4. flattened_data = flatten_data(partitions)  // Data flattening
5. transfer_to_gpu(flattened_data)  // Transfer to GPU

// GPU side parallel computation
6. FOR each partition p IN partitions PARALLEL DO:
7.   // Using CUDA dynamic parallelism
8.   parent_kernel<<<grid, block>>>(p, θ):
9.     FOR each object pair (a,b) IN p PARALLEL DO:
10.      IF mbr_filter(a, b, θ) THEN  // MBR quick filtering
11.        // Launch child kernel for precise test
12.        child_kernel<<<1, 128>>>(a, b, θ):
13.          IF exact_geometry_test(a, b, θ) THEN
14.            record_result(a.id, b.id)
15.          END IF
16.      END IF
17.    END FOR
18.  END parent_kernel

// Spark side result processing
19. transfer_to_spark(results)  // Return results
20. R = aggregate_results(results)  // Result aggregation
21. RETURN R
```

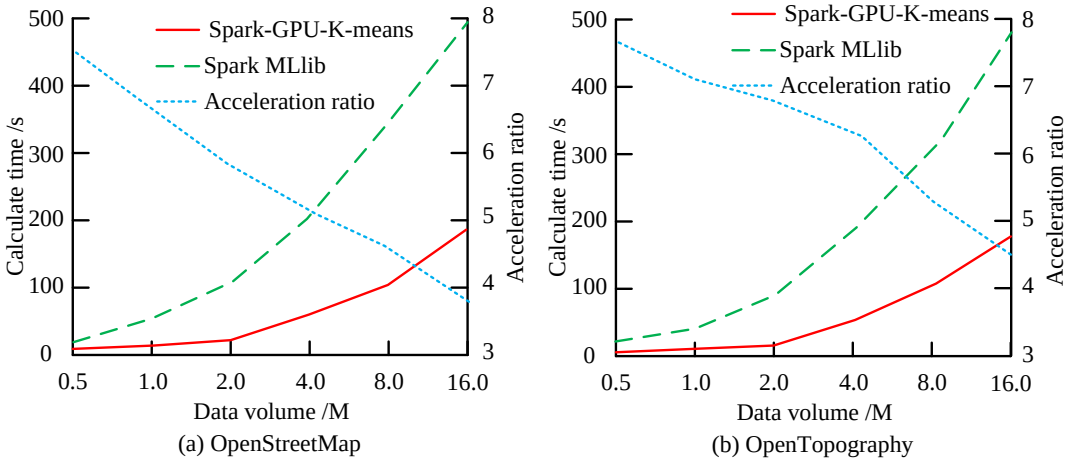


Figure 7: The influence of different dataset sizes on the Ar of the Spark-GPU-K-means.

# 3 Results

## 3.1 Experimental analysis of K-Means parallelization for BDM

The experiment employs a Spark cluster comprising four Dell PowerEdge R750xa×4 rack-mounted servers. Each node features an Intel Xeon Platinum 8380 CPU (2 sockets/40 cores/80 threads), an NVIDIA A100 80GB PCIe × 4 GPU supporting CUDA 12.2, 1TB DDR4 memory, and 200TB NVMe SSD primary storage. The number of clusters in K-means is 10, the maximum iterations are 100, and the depth of the K-dimensional tree used is 10. The datasets used include OpenStreetMap (including PB-level road, building, and natural terrain data) and OpenTopography (including surface elevation point cloud data). Due to the large amount of data in both datasets, this study extracts some data and randomly divides them into training and testing sets in an 8:2 ratio. The impact of various dataset sizes on the Ar of the Spark-GPU-K-means algorithm is shown in Figure 7.

In Figure 7 (a), as the dataset gradually increases, the running time of both algorithms increases, and the increase is getting larger, because as the dataset increases, the computational load of the algorithms also gradually increases. When the dataset is 16 M, the running time of Spark-GPU-K-means is 187.5 s, which is 312.1 s lower than Spark MLlib, and the Ar is between 3.80 - 7.49. The algorithm can effectively improve the efficiency of data clustering. In Figure 7 (b), facing the surface elevation point cloud data, the running time of Spark-GPU-K-means has decreased, and the minimum Ar has increased to 4.45. The impact of different cluster sizes on the performance of the algorithm is shown in Figure 8.

In Figure 8 (a), the computation time of both algorithms gradually increases with the increase of cluster size, and after exceeding 1,024 k, the increase in Spark MLlib suddenly increases. When the cluster size is 4,096 k, the running time of Spark MLlib is 465.2 s, which is 381.6 s higher than Spark-GPU-K-means, and the Ar is 6.98. In Figure 8 (b), the running time of Spark-GPU-K-means is consistently lower than that of Spark MLlib. At 4,096 k, the Ar of Spark-GPU-K-means is 44.32. The performance comparisons in different data dimensions are listed in Table 1.

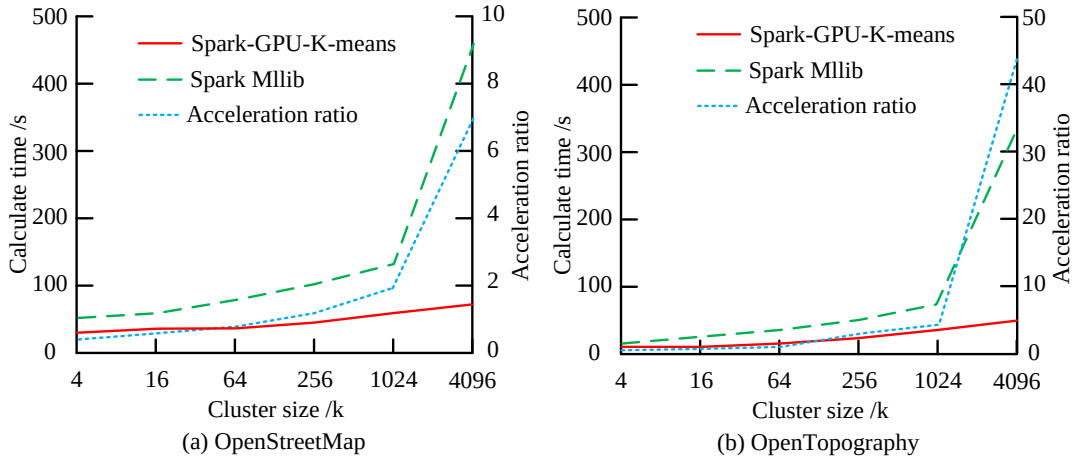


Figure 8: The influence of various cluster sizes on the algorithm's performance.

Table 1: Performance comparison of algorithms in different data dimensions.

| Data dimension /d | Algorithm | Acceleration ratio | Execution efficiency | Scalability |
|---|---|---|---|---|
| 18 | Spark-GPU-K-means | 4.25 | 0.97 | 1.27 |
| | Spark MLlib | | 0.73 | 0.84 |
| 28 | Spark-GPU-K-means | 7.13 | 0.95 | 1.22 |
| | Spark MLlib | | 0.68 | 0.80 |
| 68 | Spark-GPU-K-means | 12.68 | 0.94 | 1.54 |
| | Spark MLlib | | 0.65 | 0.78 |
| 90 | Spark-GPU-K-means | 45.32 | 0.91 | 1.36 |
| | Spark MLlib | | 0.60 | 0.75 |
| 18 | Spark-CPU | / | 0.54 | 0.65 |
| 28 | | / | 0.48 | 0.62 |
| 68 | | / | 0.46 | 0.59 |
| 90 | | / | 0.43 | 0.56 |

In Table 1, the dataset size is 4 M and the cluster size is 1,024 k. As the data dimension increases, the Ar of Spark-GPU-K-means gradually increases. When the data dimensions are 18, 28, 68, and 90, the Ars are 4.25, 7.13, 12.68, and 45.32. The execution efficiency of Spark-GPU-K-means is higher than that of Spark MLlib, and when the data dimension is 90, the execution efficiency is 0.31 higher. In different data dimensions, the scalability of Spark-GPU-K-means is greater than 1. The execution efficiency and scalability of Spark-GPU-K-means significantly surpass those of Spark-CPU. Across varying data dimensions, Spark-GPU-K-means achieves average execution efficiency and scalability of 0.94 and 1.35, respectively, exceeding the average values of Spark-CPU by 0.46 and 0.74. All test results in the experiment are validated via t-tests, with $p$ values below 0.05, demonstrating statistical significance.

## 3.2 Parallel BDM experimental analysis of SJ Algorithm

The experimental setup is identical to that described in Section 2.1. All experiments are conducted independently 5 times, and the results are averaged. The effect of different spatial partition numbers on the acceleration performance of Spark-GPU-SJ algorithm is shown in Figure 9.

In Figure 9 (a), the increase in spatial partitioning leads to an increase in the computation of data partitioning, and the computation time of the SJ in the GeoSpark platform continues to rise. The calculation time of Spark-GPU-SJ first decreases and then increases. Spark-GPU-SJ achieves optimal performance at 1,500 partitions, with a computation time of 37.5 seconds. This is because when the number of partitions is small, the computational load on a single GPU is high, but the parallelism of the algorithm is low. In Figure 9 (b), the Ar of Spark-GPU-SJ first increases, then decreases, and then increases again. At 2,500 partitions, the Ar is 3.82. The spatial data mining accuracy of Spark-GPU-SJ is shown in Figure 10.

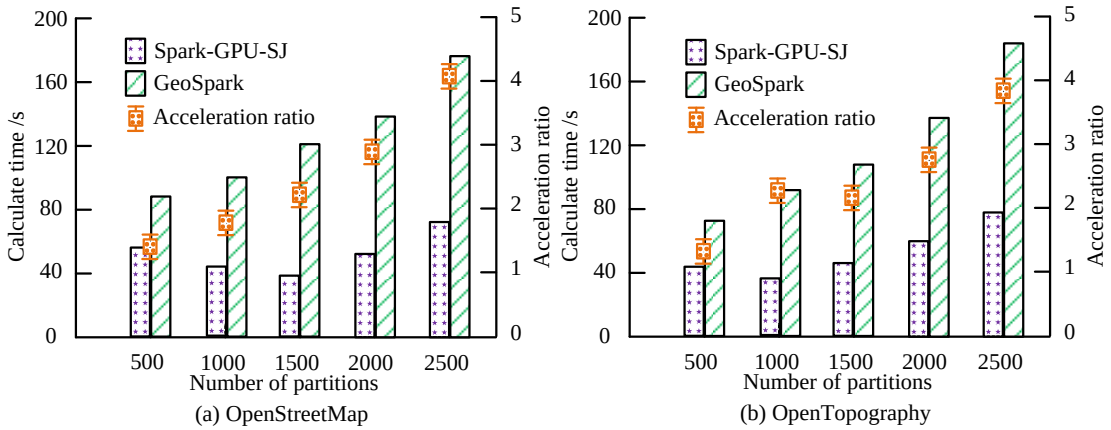


Figure 9: The influence of different numbers of spatial partitions on acceleration performance.

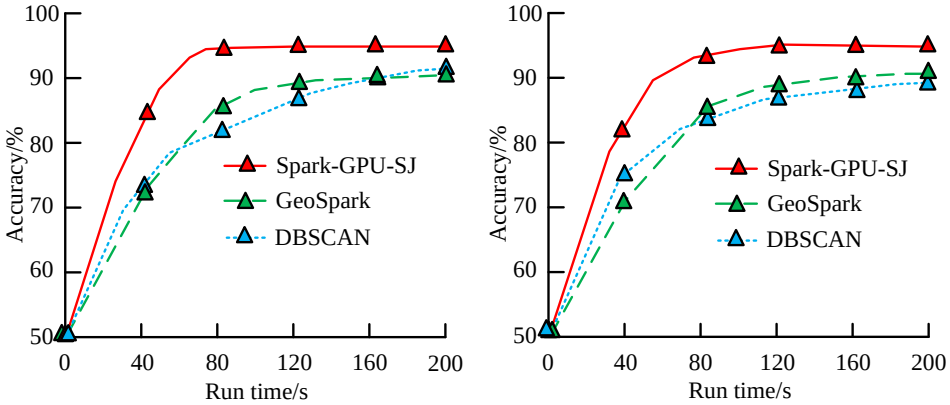


Figure 10: The spatial data mining accuracy rate of the Spark-GPU-SJ algorithm.

Table 2: Influence of CPU core numbers on the Spark-GPU-SJ's performance.

| Number of CPU cores /units | Algorithm | Ar | Execution efficiency | Scalability |
|---|---|---|---|---|
| 4 | Spark-GPU-SJ | 1.32 | 0.90 | 1.05 |
| | GeoSpark | | 0.83 | 0.83 |
| 8 | Spark-GPU-SJ | 1.45 | 0.94 | 1.20 |
| | GeoSpark | | 0.85 | 0.87 |
| 16 | Spark-GPU-SJ | 1.60 | 0.97 | 1.32 |
| | GeoSpark | | 0.86 | 0.90 |
| 32 | Spark-GPU-SJ | 1.83 | 0.92 | 1.21 |
| | GeoSpark | | 0.82 | 0.85 |

Table 3: Performance comparison of different algorithms on the HIGGS and TaxiNYC datasets.

| Dataset | Algorithm | Accuracy/% | Recall/% | F1 score/% |
|---|---|---|---|---|
| HIGGS | Spark-GPU-SJ | 95.7 | 94.9 | 96.3 |
| | GeoSpark | 90.3 | 89.5 | 90.5 |
| | DBSCAN | 88.2 | 87.3 | 88.7 |
| TaxiNYC | Spark-GPU-SJ | 93.8 | 92.6 | 94.5 |
| | GeoSpark | 87.6 | 85.2 | 86.8 |
| | DBSCAN | 85.4 | 83.0 | 85.9 |

In Figure 10 (a), Spark-GPU-SJ has the highest data mining accuracy, converging at a runtime of 80 seconds with a maximum accuracy of 94.2%, which is 3.7% and 4.4% higher than the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm and GeoSpark. In Figure 10 (b), except for DBSCAN, the mining accuracy of the other two algorithms has increased. The maximum accuracies of Spark-GPU-SJ and GeoSpark are 95.0% and 90.4%, while the accuracy of DBSCAN decreases by 1.1%. The performance impact of various CPU core numbers on Spark-GPU-SJ is shown in Table 2.

In Table 2, the partition numbers of SJ are all 1,000, and the computing performance of Spark-GPU-SJ gradually grows with the increase of CPU cores. When the cores are 4, 8, 16, and 32, the Ars of Spark-GPU-SJ method are 1.32, 1.45, 1.60, and 1.83. When the CPU cores are 16, the algorithm has the best execution efficiency and scalability, with values of 0.97 and 1.32. The performance comparison of different algorithms on the HIGGS and TaxiNYC datasets is shown in Table 3.

In Table 3, the Spark-GPU-SJ algorithm consistently achieves optimal BDM performance across different datasets. On the HIGGS dataset, its mining accuracy reaches 95.7%, surpassing GeoSpark and DBSCAN by 5.4% and 6.5%. Within the TaxiNYC dataset, the Spark-GPU-SJ algorithm achieves data mining accuracy, recall, and F1 scores of 93.8%, 85.2%, and 94.5%, respectively.

# 4 Conclusion

Given the problems of poor computational performance and high complexity that traditional data mining methods are prone to, this study proposed a BDM analysis method using K-means and SJ. The experiment showed that as the dataset gradually increased, the runtime of the algorithm gradually increased, and the increase was getting larger. When the dataset was 16 M, the runtime of Spark-GPU-K-means was 187.5 s, which was 312.1 s lower than Spark MLlib, and the Ar was between 3.80 - 7.49. Faced with surface elevation point cloud data, the runtime of Spark-GPU-K-means has decreased, and the minimum Ar has increased to 4.45. The computation time gradually increased with the growth of cluster size, and after 1,024 k, the increase in Spark MLlib suddenly increased. The Ar of Spark-GPU-K-means gradually increased with the increase of data dimension, and the execution efficiency was higher than Spark MLlib. When the data dimension was 90, the execution efficiency was 0.31 higher. In different data dimensions, the scalability of Spark-GPU-

K-means was greater than 1. The computation time of Spark-GPU-SJ first decreased and then increased with the increase of spatial partitions, achieving optimal performance at 1,500 partitions with a computation time of 37.5 seconds. This was because when the number of partitions was small, the computational load of a single GPU was large, but the parallelism of the algorithm was low. Spark-GPU-SJ had the highest data mining accuracy, converging at a runtime of 80 seconds with a maximum accuracy of 94.2%, which was 3.7% and 4.4% higher than DBSCAN and GeoSpark. When the number of CPU cores was 16, Spark-GPU-SJ had the best execution efficiency and scalability, with values of 0.97 and 1.32. This study also presents certain challenges, such as the relatively high energy consumption associated with BDM. Future investigations can explore technologies like dynamic voltage and frequency scaling, which will enable the GPU's frequency and voltage to be dynamically adjusted according to load conditions while maintaining computational performance. This method will promote green and energy-efficient computing, and further reduce the consumption of computing resources.

# References


[1] Yong Zhong, Liang Chen, Changlin Dan, and Amin Rezaeipanah. A systematic survey of data mining and big data analysis in internet of things. The Journal of Supercomputing, 78(17):18405-18453, 2022. https://doi.org/10.1007/s11227-022-04594-1

[2] Suprakash Mukherjee, Shashank Gupta, Oshin Rawlley, and Siddhant Jain. Leveraging big data analytics in 5G-enabled IoT and industrial IoT for the development of sustainable smart cities. Transactions on Emerging Telecommunications Technologies, 33(12):e4618-e4631, 2022. https://doi.org/10.1002/ett.4618

[3] Fouad Hammadi Awad, and Murtadha M. Hamad. Big data clustering techniques challenged and perspectives. Informatica, 47(6):75-92, 2023. https://doi.org/10.31449/inf.v47i6.4445

[4] A. Shaji George, and Dr T Baskar. Leveraging big data and sentiment analysis for actionable insights: A review of data mining approaches for social media. Partners Universal International Innovation Journal, 2(4):39-59, 2024. https://doi.org/10.4018/978-1-7998-8413-2.ch001

[5] Pengfei Zhang, Tianrui Li, Zhong Yuan, Chuan Luo, Keyu Liu, and Xiaoling Yang. Heterogeneous feature selection based on neighborhood combination entropy. IEEE Transactions on Neural Networks and Learning Systems, 35(3):3514-3527, 2022. https://doi.org/10.1109/tnnls.2022.3193929

[6] Madhu Kirola, Minakshi Memoria, Ankur Dumka, Amrendra Tripathi, and Kapil Joshi. A comprehensive review study on: Optimized data mining, machine learning and deep learning techniques for breast cancer prediction in big data context. Biomedical and Pharmacology Journal, 15(1):13-25, 2022. https://doi.org/10.13005/bpj/2339

[7] Amin Mohajer, Mahya Sam Daliri, A. Mirzaei, A. Ziaeddini, M. Nabipour, and Maryam Bavaghar. Heterogeneous computational resource allocation for NOMA: Toward green mobile edge-computing systems. IEEE Transactions on Services Computing, 16(2):1225-1238, 2022. https://doi.org/10.1080/08839514.2018.1486132

[8] Ji Liu, Zhihua Wu, Danlei Feng, Minxu Zhang, Xinxuan Wu, Xuefeng Yao, Dianhai Yu, Yanjun Ma, Feng Zhao, and Dejing Dou. Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. Future Generation Computer Systems, 148(4):106-117, 2023. https://doi.org/10.1016/j.future.2023.05.032

[9] Zan Zong, Minkun Guo, Mingshu Zhai, Yinan Tang, Jianjiang Li, and Jidong Zhai. Training large models on heterogeneous and geo-distributed resource with constricted networks. Big Data Mining and Analytics, 8(4):966-980, 2025. https://doi.org/10.26599/bdma.2025.9020031

[10] Fatemeh Moodi, and Hamid Saadatfar. An improved K-means algorithm for big data. IET Software, 16(1):48-59, 2022. https://doi.org/10.1049/sfw2.12032

[11] Laouni Djafri. Dynamic distributed and parallel machine learning algorithms for big data mining processing. Data Technologies and Applications, 56(4):558-601, 2022. https://doi.org/10.1108/dta-06-2021-0153

[12] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. A learning-based framework for spatial join processing: estimation, optimization and tuning. The VLDB Journal, 33(4):1155-1177, 2024. https://doi.org/10.1007/s00778-024-00836-1

[13] Mengmeng Yang, Longxia Huang, and Chenghua Tang. K-means clustering with local distance privacy. Big Data Mining and Analytics, 6(4):433-442, 2023. https://doi.org/10.26599/bdma.2022.9020050

[14] Omaima Essaad belhaj, Raheela zaib, and Ourlis Ourabah. Large scale data using K-means. Mesopotamian Journal of Big Data, 2023(7):36-45, 2023. https://doi.org/10.58496/mjbd/2023/006

[15] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Heterogeneous graph representation learning with relation awareness. IEEE Transactions on Knowledge and Data Engineering, 35(6):5935-5947, 2022. https://doi.org/10.1109/tkde.2022.3160208

[16] Mehdi Gheisari, Hooman Hamidpour, Yang Liu, Peyman Saedi, Arif Raza, Ahmad Jalili, Hamidreza Rokhsati, and Rashid Amin. Data mining techniques for web mining: A survey. Artificial Intelligence and Applications, 1(1):3-10, 2023. https://doi.org/10.47852/bonviewAIA2202290

[17] Dongxiao He, Chundong Liang, Cuiying Huo, Zhiyong Feng, Di Jin, and Liang Yang. Analyzing heterogeneous networks with missing attributes by unsupervised contrastive learning. IEEE

Transactions on Neural Networks and Learning Systems, 35(4):4438-4450, 2022. https://doi.org/10.1109/tnnls.2022.3149997

[18] Mohammad Annas, and Siti Norida Wahab. Data mining methods: K-means clustering algorithms. International Journal of Cyber and IT Service Management, 3(1):40-47, 2023. https://doi.org/10.34306/ijcitsm.v3i1.122

[19] Shu Yang. Strengthening accounting information systems with advanced big data mining algorithms: Innovative exploration of data cleaning and conversion automation. Informatica, 49(11):18-32, 2025. https://doi.org/10.31449/inf.v49i11.7302

[20] Md Mostafizur Rahman, Siful Islam, Md Kamruzzaman, and Zihad Hasan Joy. Advanced query optimization in SQL databases for real-time big data analytics. Academic Journal on Business Administration, Innovation & Sustainability, 4(3):1-14, 2024. https://doi.org/10.69593/ajbais.v4i3.77