InPrime: Indonesian Journal of Pure and Applied Mathematics
Vol. 8, No. 1 (2026), pp. 85 – 105, doi: 10.15408/inprime.v8i1.50999
p-ISSN 2686-5335, e-ISSN: 2716-2478

Scalability and Cost Efficiency Analysis of Distributed Logistic Regression on Low-Power Clusters Using Apache Spark

Arjuananta Malik, Taufik Edy Sutanto, Muhaza Liebenlito*, and Mohamad Irvan Septiar Musti
Department of Mathematics, Faculty of Science and Technology, UIN Syarif Hidayatullah Jakarta,
Tangerang Selatan, Banten, Indonesia
Email: *muhazaliebenlito@uinjkt.ac.id

Abstract

The limited availability of large-scale computing infrastructure, combined with the high acquisition, operational, and energy costs of conventional GPU-based servers, remains a significant challenge for AI model training and big data learning in resource-constrained educational and research environments. To address this issue, this study proposes and evaluates a horizontally scalable low-power cluster architecture based on Apache Spark for distributed logistic regression training. The novelty of this research lies in the integrated evaluation of scalability performance and computational cost efficiency of Single Board Computer (SBC)-based clusters under varying data complexity scenarios. Experiments were conducted using binary and multiclass datasets by varying the number of observations, feature dimensionality through feature hashing, and the number of classes, with cluster configurations ranging from one to four nodes. Performance was assessed using training time, speedup, parallel efficiency, and electricity cost metrics. Results show that increasing dataset size and feature dimensionality substantially increases computational workload; however, horizontal scaling effectively reduces training time, particularly for high-dimensional and multiclass datasets. The best scalability performance was achieved on the multiclass dataset, where training time decreased by up to 46.98% when scaling from one to four nodes. Although parallel efficiency declined due to communication and synchronization overhead, the system maintained low energy consumption and operational cost. These findings demonstrate that low-power distributed clusters provide a scalable and cost-efficient alternative infrastructure for AI and big data learning in institutions with limited computational resources.
Keywords: Distributed logistic regression; Efficiency; Horizontal scalability.

Abstrak

Keterbatasan ketersediaan infrastruktur komputasi skala besar, yang disertai dengan tingginya biaya pengadaan, operasional, dan konsumsi energi dari server berbasis GPU konvensional, masih menjadi tantangan utama dalam pelatihan model kecerdasan buatan (AI) dan pembelajaran big data pada lingkungan pendidikan dan penelitian yang memiliki sumber daya terbatas. Untuk mengatasi permasalahan tersebut, penelitian ini mengusulkan dan mengevaluasi arsitektur klaster berdaya rendah yang dapat diskalakan secara horizontal berbasis Apache Spark untuk pelatihan regresi logistik terdistribusi. Kebaruan penelitian ini terletak pada evaluasi terpadu terhadap kinerja skalabilitas dan efisiensi biaya komputasi pada klaster berbasis Single Board Computer (SBC) di bawah berbagai skenario kompleksitas data. Eksperimen dilakukan menggunakan dataset biner dan multikelas dengan memvariasikan jumlah observasi, dimensi fitur melalui teknik feature hashing, serta jumlah kelas, pada konfigurasi klaster yang terdiri dari satu hingga empat node. Kinerja sistem dievaluasi berdasarkan waktu pelatihan, speedup, efisiensi paralel, dan biaya listrik. Hasil penelitian menunjukkan bahwa peningkatan ukuran dataset dan dimensi fitur secara signifikan meningkatkan beban komputasi. Namun, penskalaan horizontal mampu mengurangi waktu pelatihan secara efektif, terutama pada dataset berdimensi tinggi dan multikelas. Kinerja skalabilitas terbaik diperoleh pada dataset multikelas, dengan penurunan waktu pelatihan hingga 46,98% ketika jumlah node ditingkatkan dari satu menjadi empat. Meskipun efisiensi paralel menurun akibat overhead komunikasi dan sinkronisasi, sistem tetap mempertahankan konsumsi energi serta biaya

*) Corresponding author
Submitted April 30th, 2026, Revised March 30th, 2026,
Accepted for publication March 31st, 2026, Published Online May 31st, 2026

operasional yang rendah. Temuan ini menunjukkan bahwa klaster terdistribusi berdaya rendah merupakan alternatif infrastruktur yang skalabel dan hemat biaya untuk pembelajaran AI dan big data pada institusi dengan sumber daya komputasi yang terbatas.
Kata Kunci: *Regresi Logistik Terdistribusi; Efisiensi; Skalabilitas horizontal.*

2020MSC: 68T09, 68W10.

1. INTRODUCTION

The training of artificial intelligence (AI) models in Indonesia, particularly in educational and research institutions, continues to face significant limitations in computing infrastructure and funding availability [1]. Most universities and small-scale research organizations still depend on vertical scalability infrastructure or single-machine systems, typically consisting of one or two local servers equipped with 1–4 consumer-grade GPUs such as RTX 2060–3090 [2], [3]. Unlike large industrial environments that utilize distributed computing clusters for large-scale AI training and big data processing, these institutions often operate under constrained computational resources and short-term funding schemes. The acquisition cost of a conventional AI server with multiple GPUs can reach IDR 250–500 million, excluding annual electricity and cooling expenses estimated at IDR 30–70 million [2]. Consequently, AI training and experimentation in academic environments remain limited in scale and computational capability.

To address these constraints, low-power distributed computing architectures based on Single Board Computers (SBC) have emerged as a cost-effective alternative for scalable computation [4]. SBC clusters offer horizontal scalability, where computational capacity is increased by adding additional nodes instead of upgrading a single machine. This approach is attractive for educational and research institutions because it provides flexible scalability with lower acquisition and operational costs [5]. However, despite their affordability and low energy consumption, SBC clusters possess limited hardware specifications, making their practical effectiveness for distributed machine learning workloads still uncertain, particularly when handling increasing data complexity and communication-intensive computations.

The relevance of low-power distributed computing is closely related to the concept of Green AI [6], which emphasizes that machine learning systems should be evaluated not only from predictive performance but also from computational efficiency, energy consumption, and operational cost. In distributed environments, scalability improvements are often accompanied by communication overhead, synchronization delays, and resource bottlenecks that may reduce parallel efficiency [7]. Therefore, evaluating distributed machine learning systems requires not only measuring runtime acceleration but also analyzing the trade-off between scalability and computational cost efficiency.

Several previous studies have investigated scalability and distributed machine learning from different perspectives. Cloud-based distributed systems have demonstrated the ability of horizontal scaling to maintain performance under increasing data volumes [8], while large-scale distributed machine learning frameworks have shown improvements in training efficiency through parallel computation [9]. Systematic reviews further identify scalability as one of the major strengths of distributed machine learning architectures, although communication overhead remains a critical limitation affecting system efficiency [10]. In the context of distributed logistic regression, previous works have explored scalability performance under high-dimensional and multiclass datasets, reporting improvements in computational speed under specific distributed configurations [11], [12].

However, these studies primarily focus on computational acceleration and predictive scalability without evaluating the relationship between scalability behavior and operational cost efficiency.

Research on scalability using SBC-based clusters has also been widely explored. Basford [13] evaluated horizontal scalability on Raspberry Pi and Odroid clusters with up to 16 nodes using High Performance Linpack (HPL) benchmarks. Although the study demonstrated better energy efficiency compared to conventional systems, the results also revealed that scalability gains were non-linear due to network bottlenecks and hardware limitations. Similarly, Justus [14] analyzed machine learning computational complexity using GPU-based systems and showed that theoretical FLOPs-based complexity metrics fail to fully capture practical training costs because of memory transfer and synchronization overhead. In low-power environments, Azka [15] demonstrated that SBC clusters achieved higher energy efficiency than PC clusters for Singular Value Decomposition (SVD) workloads, despite slower execution time. Noer [16] further reported energy savings of up to 93% for distributed unsupervised learning using Apache Spark on SBC clusters. Meanwhile, Razaba [4] evaluated Apache Spark-based machine learning workloads on SBC clusters and reported training time reductions of up to 59.7% with relatively stable power consumption.

Despite these advances, several limitations remain in the current state-of-the-art literature. First, most previous studies evaluate scalability performance, energy efficiency, or distributed architecture independently rather than integrating scalability and computational cost efficiency into a unified analytical framework. Second, prior studies generally focus on infrastructure benchmarking or generic distributed machine learning workloads without systematically analyzing how computational factors such as dataset size, feature dimensionality, and number of classes jointly influence scalability behavior and operational cost. Third, research specifically investigating distributed logistic regression on low-power Apache Spark clusters remains limited, particularly in educational-scale infrastructures where computational resources are highly constrained.

Based on these gaps, this study proposes a comprehensive evaluation of distributed logistic regression on a low-power SBC cluster using Apache Spark by integrating scalability analysis and computational cost efficiency within a single experimental framework. The novelty of this research lies in the combined analysis of runtime scalability, speedup, parallel efficiency, and electricity-based computational cost under varying computational complexities, including the number of observations, feature dimensionality through feature hashing, and multiclass classification scenarios. Unlike previous studies that primarily focus on either system performance or energy efficiency independently, this research examines the interaction between horizontal scalability and cost-efficient computation in resource-constrained distributed machine learning environments.

Therefore, this study aims to analyze the scalability performance and computational cost efficiency of distributed logistic regression on a low-power cluster using Apache Spark. The evaluation is conducted by systematically varying dataset size, feature dimensionality, and number of classes while measuring runtime, speedup, parallel efficiency, and electricity consumption. However, this study focuses specifically on computational scalability and operational electricity cost; therefore, factors such as network bandwidth analysis, detailed communication overhead modeling, hardware maintenance cost, and long-term infrastructure reliability are beyond the scope of this research. Through this approach, the study seeks to provide empirical insights into the feasibility of low-power distributed clusters as scalable and cost-efficient infrastructure for AI and big data learning in educational and research environments with limited computational resources.

2. METHOD

2.1. Data

The initial data used in this study were obtained from publicly available sources on the internet (Kaggle), which are widely used in machine learning research, namely the Credit Card Fraud Detection Dataset [17] and the Forest Cover Type (Covtype) Dataset [18]. Both datasets represent real-world classification problems. The Credit Card Fraud Detection Dataset involves binary classification with two classes, while the Covtype dataset represents a multiclass classification problem with seven classes. These datasets are ready-to-use, as they are already labeled, allowing them to be directly utilized for experimental evaluation.

Table 1. Credit Card Fraud Detection Dataset

Time	V1	V2	...	V28	Amount	Class
0	-1.3598	-0.0727	...	-0.021	149.62	0
0	0.1918	0.2661	...	0.0147	2.692	0
1	-1.3583	-1.34016	...	-0.059	378.66	0
1	-0.9662	-1.8552	...	0.0614	123.5	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
172192	-0.5334	-0.1897	...	0.0136	217	1

Table 1 presents a dataset of credit card transactions from European cardholders, used for fraud detection. The dataset consists of 284,807 transaction records with 31 features, where each row represents a single credit card transaction. The features labeled V1 to V28 are anonymized variables, as they have been transformed to protect sensitive information. The *Time* feature indicates the elapsed time since the first recorded transaction, while the *Amount* feature represents the monetary value of the transaction. The *Class* feature is a binary label indicating whether a transaction is fraudulent (*fraud = 1*) or non-fraudulent (*non-fraud = 0*). This dataset is publicly available on the Kaggle platform and is widely used as a benchmark in anomaly detection and machine learning research.

Table 2. Forest Cover Type Dataset

Elevation	Aspect	Slope	...	Cover Type
2596	51	3	...	5
2590	56	2	...	5
2804	139	9	...	2
2785	155	18	...	2
⋮	⋮	⋮	⋮	⋮
2383	165	13	...	3

Table 2 is derived from the U.S. Forest Service (USFS) and is available through the UCI Machine Learning Repository as well as Kaggle. This dataset represents forest cover conditions in the Roosevelt National Forest, Colorado, USA, collected using remote sensing data and Geographic Information Systems (GIS). The dataset consists of 581,012 samples with 54 features, including one target feature with seven classes. The numerical features describe topographical and geographical characteristics, such as *Elevation* (altitude in meters), *Aspect* (slope direction), *Slope* (degree of slope), and other related variables. The dataset also includes categorical features encoded using one-hot encoding, representing four wilderness areas (*Wilderness_Area1* to *Wilderness_Area4*). Additionally, there are 40 one-hot

encoded categorical features representing different soil types (*Soil_Type1* to *Soil_Type40*). The target variable consists of seven classes representing forest cover types: (1) *Spruce/Fir*, (2) *Lodgepole Pine*, (3) *Ponderosa Pine*, (4) *Cottonwood*, (5) *Aspen*, (6) *Douglas-fir*, and (7) *Krumholz*.

2.2. Feature Hashing

Feature hashing in this study is used to generate variations in feature dimensionality by expanding or reducing the number of features beyond the original dataset, enabling the simulation of different computational workloads without altering the source data. This approach is suitable for evaluating system scalability, as it allows the analysis of logistic regression performance under various data size scenarios in a distributed cluster. In the experiments, feature hashing is applied as a transformation method by controlling the output dimension (d), affecting both feature expansion and reduction. It is chosen because it can convert both numerical and categorical features into a fixed-dimensional vector representation without requiring one-hot encoding for high-cardinality variables. This makes it particularly efficient for large-scale and distributed processing, as it reduces memory usage and accelerates computation [19]. The hash function is defined as follows equation (1):

$$h(k) = k \bmod d, \tag{1}$$

where h represents a function that maps each integer k into the range $[0, d-1]$, where k denotes an integer corresponding to the original key or feature, and d indicates the dimensionality of the resulting feature space after the hashing process.

Although feature hashing improves computational efficiency, it also introduces the possibility of hash collisions, where multiple original features are mapped into the same hashed index. Collision effects become more significant when the output dimensionality (d) is relatively small, potentially causing information overlap between unrelated features. In logistic regression, excessive collisions may reduce feature separability and negatively affect model performance because multiple independent feature contributions are aggregated into the same vector position. Conversely, increasing the feature dimensionality reduces collision probability and preserves more feature information, but also increases computational complexity, memory usage, and communication cost in distributed environments. Therefore, feature hashing introduces a trade-off between computational efficiency and representational accuracy.

In the context of this study, feature hashing is primarily utilized to simulate varying computational workloads rather than to optimize predictive accuracy. By systematically varying the feature dimensionality, the experiments can evaluate how distributed logistic regression behaves under different levels of sparsity, computational complexity, and communication requirements. Higher-dimensional feature spaces produce larger parameter vectors and more intensive gradient computations, which increase both processing time and distributed synchronization overhead during model training.

The output of the feature hashing process is a fixed-dimensional vector (for example, 30, 200, or 1000 features) that can be directly used for model training. For experimental purposes, both datasets are modified into several versions based on combinations of the number of observations (n) and feature dimensionality (d). The datasets are sampled or oversampled to produce new datasets with different sizes, where both the Credit Card Fraud Detection dataset and the Forest Cover Type dataset are adjusted to $d =$ 100,000, 284,000, and 1,000,000 according to predefined parameters. For feature

transformation, each dataset is converted into multiple versions using feature hashing, resulting in feature dimensionalities of $d = 30$, 200, and 1,000. These modifications produce several dataset variations that are used in the experiments, as presented in Table 3.

Table 3. Dataset Variations for Experiments

Dataset	Number of Observations (n)	Number of Features (d)	Number of Classes (K)
creditcard_n1000k_d1000	1000000	1000	2
creditcard_n1000k_d200	1000000	200	2
creditcard_n1000k_d30	1000000	30	2
creditcard_n100k_d1000	100000	1000	2
creditcard_n100k_d200	100000	200	2
creditcard_n100k_d30	100000	30	2
creditcard_n284k_d1000	284000	1000	2
creditcard_n284k_d200	284000	200	2
creditcard_n284k_d30	284000	30	2
covtype_n1000k_d1000	1000000	1000	7
covtype_n1000k_d200	1000000	200	7
covtype_n1000k_d30	1000000	30	7
covtype_n100k_d1000	100000	1000	7
covtype_n100k_d200	100000	200	7
covtype_n100k_d30	100000	30	7
covtype_n284k_d1000	284000	1000	7
covtype_n284k_d200	284000	200	7
covtype_n284k_d30	284000	30	7

2.3. Apache Spark

Apache Spark [20] is a big data processing framework designed to perform high-speed parallel computation on computer clusters. Spark operates on a distributed programming model that enables data to be partitioned across multiple worker nodes and processed in parallel, thereby improving efficiency and scalability. It was developed as a faster alternative to Hadoop MapReduce due to its ability to perform in-memory computation [21], which stores data in main memory rather than repeatedly writing it to disk storage. With this design, Spark is particularly well-suited for iterative workloads such as machine learning model training, which require repeated computations on the same dataset. The illustration of the architecture and internal workflow of Apache Spark is presented in Figure 1.

The machine learning model in Spark MLlib used in this study is implemented using a data parallelism approach. The dataset is divided into multiple partitions, and each worker node computes statistical information, such as local gradients, based on the data within its partition. The results from each node are then aggregated by the driver program using mechanisms such as treeAggregate, producing a global gradient that is used to update the model parameters. This process is repeated iteratively until the model reaches convergence.

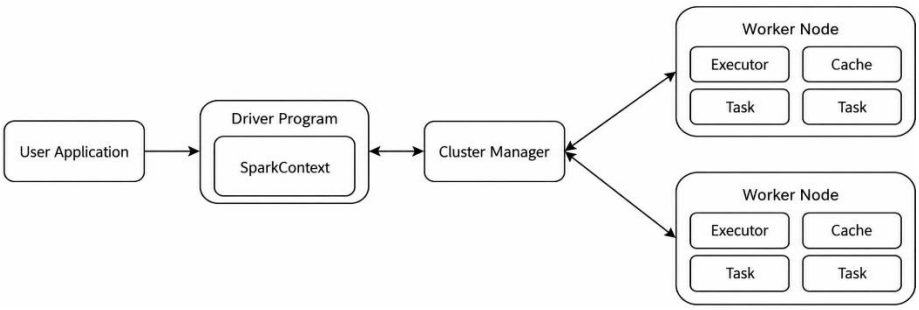

Figure 1. Architecture and internal workflow of Apache Spark

In this study, the experiments are conducted using Apache Spark version 3.5.3, running in a distributed multi-node environment with a Standalone cluster configuration, which simplifies cluster setup and management [22]. Several experimental scenarios are considered based on the number of nodes, where each node is configured with four CPU cores. The system utilizes four cores on each node, and since multiple nodes are involved, a data parallelism strategy is applied. The training data is partitioned across the cluster, and each node processes its assigned data in parallel using the same model. This task distribution ensures efficient utilization of computational resources across all nodes.

The cluster resource configuration is controlled through environment settings to ensure balanced computation across nodes. Each worker is configured with *SPARK_WORKER_CORES* = 4, providing four CPU cores, and *SPARK_WORKER_MEMORY* = 3 GB, limiting memory allocation to avoid over-allocation on resource-constrained SBC devices. On the executor side, *SPARK_EXECUTOR_MEMORY* is set to the default value of 1 GB per executor, while *SPARK_EXECUTOR_CORES* uses the default configuration of one core per executor. With this setup, each executor runs as a single Java Virtual Machine (JVM) process utilizing one CPU core and 1 GB of memory, allowing multiple executors to run within a worker as long as the total resource allocation does not exceed the predefined limits.

2.4. Logistic Regression

Logistic Regression is a binary classification algorithm used to model the probability of an event based on one or more input variables. Unlike linear regression, which produces continuous outputs, logistic regression applies the logistic (sigmoid) function to map the output into a probability value between 0 and 1. Mathematically, the relationship between the input variables and the probability of an event is expressed using the logit function as in equation (2).

$$logit(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k, \tag{2}$$

where p represents the probability of the event, $x_1, \ldots, x_k$ denote the input features, and $\beta_0, \ldots, \beta_k$ are the model parameters. The training process is carried out using optimization methods such as Stochastic Gradient Descent (SGD)/L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) to minimize the log-loss function. In Apache Spark, logistic regression is implemented in a distributed manner using the Spark MLlib library. The dataset is partitioned into multiple subsets and distributed across several nodes in the cluster, where each node performs a portion of the computation.

2.5. Multiclass Logistic Regression

For classification problems with more than two classes, such as the cover type dataset, where the number of classes K exceeds two, logistic regression is extended from a binary model to a softmax regression model (also known as multinomial logistic regression). The objective of this model is to predict the probability that an observation $x_k \in \mathbb{R}^d$ belongs to one of the classes $c \in \{1,2,\dots,K\}$. Mathematically, the probability that a sample x_k belongs to class c is defined by the softmax function, as shown in Equation (3):

$$P(y = c|x_k; \boldsymbol{W}) = \frac{\exp(\boldsymbol{w}_c^T x_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^T x_k)}, \tag{3}$$

where $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K]$ denotes the parameter matrix of size $d \times K$, where w_c is the parameter vector for class c, x_k represents the feature vector of the k sample, and K is the number of classes.

2.6. Distributed Logistic Regression

In Apache Spark, the training process of distributed logistic regression is performed using a data-parallel approach, where the training dataset is partitioned into multiple subsets and distributed across nodes in the cluster. Let the training dataset be denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, which is partitioned into p subsets $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_p$, where each subset is processed by a single node. At each training iteration, each node j computes a local gradient based on its data partition $\mathcal{D}_j$ using the same model parameter w, defined as:

$$\nabla \ell_j(\boldsymbol{w}) = \sum_{(x_i, y_i) \in \mathcal{D}_j} (\sigma(\boldsymbol{w}^T x_i) - y_i) x_i, \tag{4}$$

where $\mathcal{D}_j$ represents the subset of data processed by node j, while w is the shared model parameter used across all nodes. Each observation in the partition contributes to the gradient through the difference between the model prediction and the actual label, weighted by the feature vector. This local gradient represents the direction and magnitude of parameter updates based on the data available at each node.

$$\nabla \ell(\boldsymbol{w}) = \sum_{j=1}^{p} \nabla \ell_j(\boldsymbol{w}). \tag{5}$$

Equation (5) describes the aggregation process of local gradients to obtain the global gradient in distributed computation. Here, p denotes the number of nodes in the cluster. Spark aggregates the gradients by summing the contributions from each node as defined in Equation (4), so that the global gradient reflects information from the entire dataset processed in parallel. This global gradient is then used to update the model parameters synchronously at each training iteration.

In Spark, to obtain the parameter w that minimizes the loss function, the L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) algorithm is used, which belongs to the family of quasi-Newton methods [23]. This algorithm approximates the inverse Hessian matrix using gradient change information from previous iterations. This information helps capture the characteristics of the loss function, enabling more efficient determination of the parameter update direction.

2.7. Low-Power Cluster Architecture

This study utilizes a low-power cluster that implements a distributed computing approach. The cluster is built using Single Board Computers (SBC), as demonstrated in prior energy measurement experiments by Razaba [4], where the system consumes an average power of less than 15 watts when operating in a multicore configuration. Distributed computing refers to a system in which two or more machines connected within the same network share and execute computational workloads collaboratively. In this study, the experimental architecture is illustrated in Figure 2.

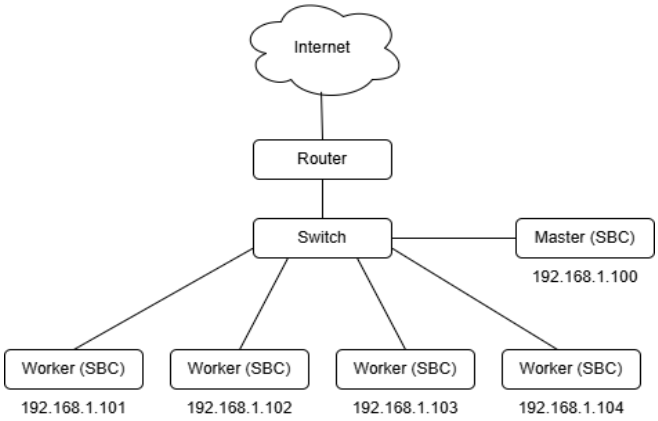

Figure 2. Low-Power Cluster Architecture

Figure 2 illustrates the architecture of the low-power cluster, which consists of four SBC units, specifically Orange Pi Zero 3 devices, responsible for executing computational tasks in parallel. The master node is also implemented using an SBC, namely the Orange Pi 3B, which manages task distribution, resource allocation, and coordination of data exchange among nodes. All devices are connected through a network switch and configured within a local network using static Internet Protocol (IP) addresses for each node. This configuration ensures that both the master and worker nodes reside within the same subnet and can communicate efficiently. The detailed hardware specifications used in this study are presented in Table 4.

Table 4. Hardware Specifications of the Master and Worker Nodes in the Low-Power Cluster

Component	Master Node	Worker Node (SBC Cluster)
Processor	Rockchip RK3566 1.8 GHz	Allwinner H618 quad-core Cortex-A53 1.5 GHz
Number of Nodes	1	4
Memory (RAM)	8 GB (LPDDR4)	4 GB per node (LPDDR4)
Storage	NVMe SSD (128 GB)	MicroSD (64 GB)
Operating System	Ubuntu Image ver. 1.0.8 (OrangePiOS)	Ubuntu 22.04.4 LTS

2.8. Measurement of Scalability and Time Efficiency in Parallel Systems

The experiments are performed on cluster configurations with varying numbers of nodes, ranging from 1 to 4 nodes. For each configuration, the model training process is executed with identical

parameters, while the total execution time (runtime) is recorded as the primary indicator of time efficiency.

2.8.1. Runtime and Time Efficiency

Runtime refers to the total time required by the system to complete the training process for each node configuration. The training time serves as the basis for evaluating time efficiency, where a reduction in training time indicates improved performance due to parallelization. The measurement of training time is conducted from the start of the training process until the model is fully trained. These values are then compared across different node configurations to observe the impact of increasing computational resources on execution time.

2.8.2. Speedup

Speedup is used to measure the acceleration of computational time resulting from parallel computing. It is defined as the ratio between the execution time on a single node and the execution time on N nodes, as expressed in Equation (6):

$$S(N) = \frac{T_1}{T_N}, \tag{6}$$

where T_1 is the training time using one node, T_N is the training time using N nodes, and $S(N)$ represents the speedup value for N nodes. The ideal speedup is linear, $S(N) = N$. However, in distributed systems, the achieved speedup is typically lower due to additional communication overhead between nodes.

2.8.3. Parallel Efficiency

Parallel efficiency is used to evaluate how effectively parallel resources are utilized. This metric is calculated as the ratio between speedup and the number of nodes, as defined in Equation (7):

$$E(N) = \frac{S(N)}{N}. \tag{7}$$

The efficiency value ranges from 0 to 1 (or 0% to 100%), where higher values indicate more optimal utilization of additional nodes.

2.9. Measurement of Scalability and Time Efficiency in Parallel Systems

In this study, power consumption is measured under two system configurations: single-node (serial) and multi-node scenarios. In the single-node configuration, the average power consumption during training is 13.5 *watts*. In contrast, in the multi-node configuration, the average power consumption increases to 14.93 *watts*, reflecting additional energy usage due to multiple nodes operating simultaneously. These average power values are used as parameters in the energy and cost estimation model. Power (P) is expressed in watts, while training time (t) is measured in seconds. The energy E, initially in joules, is then converted into *kilowatt-hours* (kWh), the standard unit of electrical energy consumption, as shown in Equation (8):

$$E_{kWh} = \frac{P \times t}{3600 \times 1000}. \quad (8)$$

After obtaining the energy value in kWh, the electricity cost is calculated using the non-subsidized household electricity tariff issued by PLN on December 12, 2025, which is IDR 1,440.7 per kWh [24]. If τ represents the electricity tariff per kWh (IDR/kWh), the total cost for a single training process is defined as equation (9):

$$C = E_{kWh} \times \tau. \quad (9)$$

3. RESULTS

In this study, the distributed logistic regression model is trained using the built-in library from Spark MLlib, namely LogisticRegression. The model is configured with several key parameters, including *maxIter* = 100 as the maximum number of optimization iterations, *regParam* = 0.0 and *elasticNetParam* = 0.0 to disable regularization, and *tol* = 1e-6 as the convergence tolerance threshold. In addition, *fitIntercept* = True is used to allow the model to learn the bias term, while *standardization* = True, ensures that features are normalized before the optimization process. The parameter *threshold* = 0.5 is set as the decision boundary for binary classification, and the option f*amily* = "auto" allows Spark to automatically select the appropriate logistic regression formulation based on the number of classes in the dataset. All configurations are applied consistently across all datasets and experimental scenarios, ensuring that model evaluation results can be compared fairly and systematically.

3.1. Analysis of Distributed Model Performance with Respect to the Number of Observations

This subsection discusses the effect of the number of observations (n) on the training execution time of logistic regression in a distributed computing environment based on Apache Spark. The analysis is conducted by comparing several dataset size variations presented in Table 3 across different numbers of nodes, with the aim of evaluating execution time scalability and identifying computational efficiency limits in the low-power cluster.

Figures 3 illustrate the relationship between n and training time (runtime) in logistic regression experiments using the Credit Card Fraud and Covtype datasets. The experiments are conducted with fixed feature dimensionality (d) within each subplot, specifically $d = 30, 200$ and 1,000. The results indicate that increasing the number of observations leads to higher computational time. In Figure 3 (left), the Credit Card Fraud dataset shows that the most notable reduction in training time due to node scaling occurs for n=1,000,000 and d=200, where training time decreases from 131.7 seconds on one node to 107.3 seconds on four nodes, corresponding to a time efficiency improvement of 18.53%. In contrast, Figure 3 (right) shows that the Covtype dataset achieves the largest reduction in training time at n=1,000,000 and d=200, with a decrease of approximately 140 seconds, corresponding to an efficiency improvement of 46.98% on four nodes.

Figure 3 show speedup versus number of nodes for binary (left) and multiclass (right) datasets. Based on this figures, increasing the number of observations also affects the speedup values in both binary and multiclass classification scenarios. Across all feature configurations, the dataset with the largest number of observations (n=1,000,000) consistently achieves the highest speedup, exceeding 1.5× on four nodes. This indicates that larger computational workloads can reduce the relative impact of communication overhead (additional time required for coordination among nodes or the driver

without performing computation). The multiclass scenario shows a generally higher increase in speedup compared to the binary dataset.

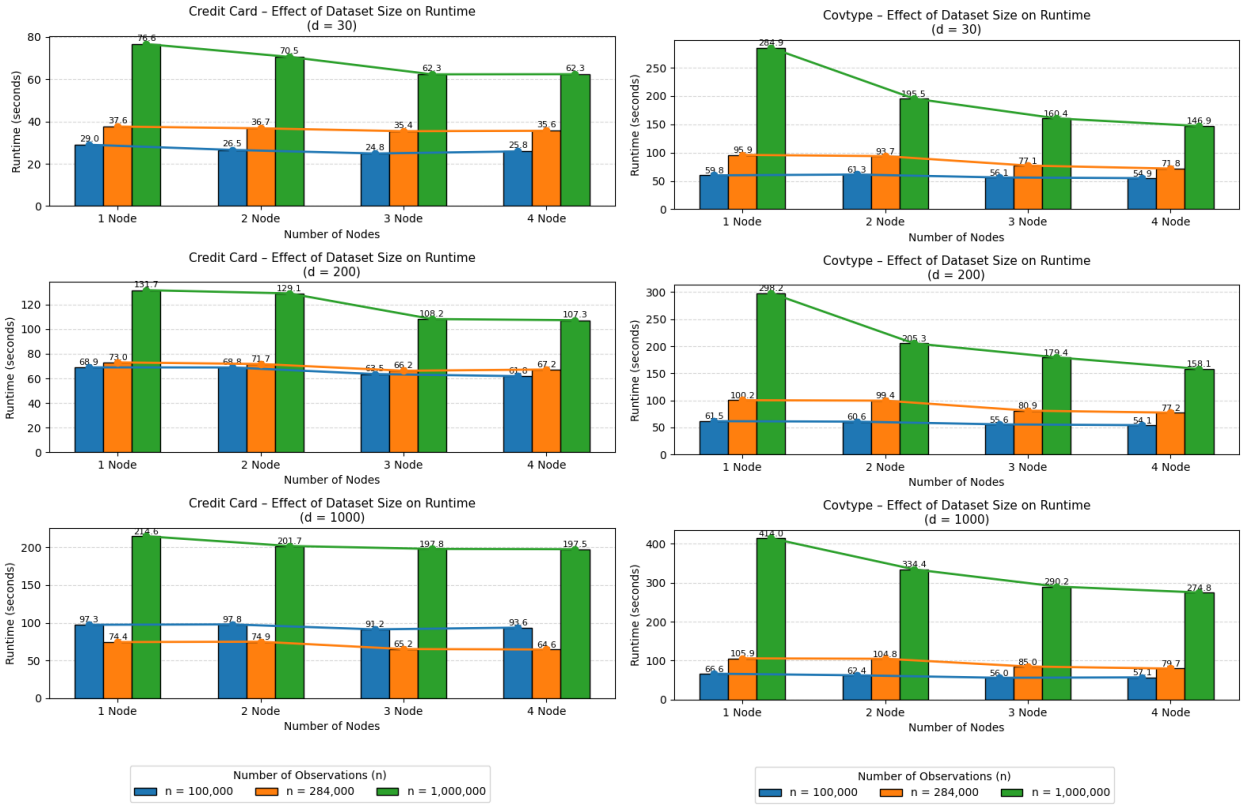

Figure 4. Runtime versus number of observations for the credit card fraud (left) and forest cover type (right) datasets.

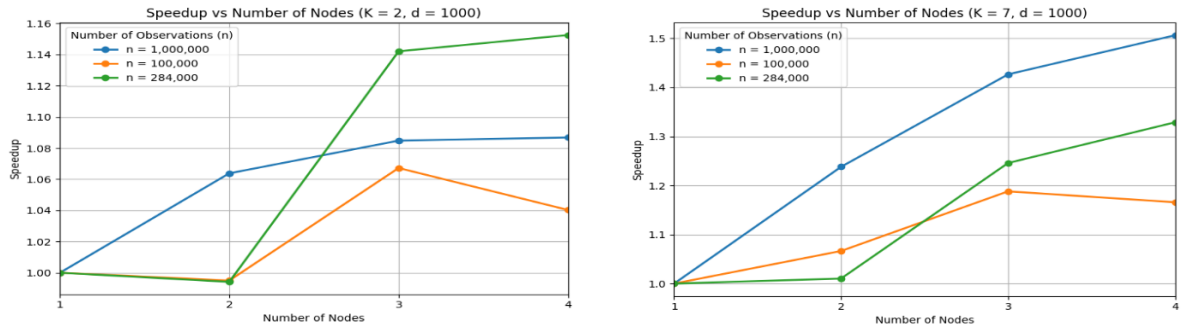

Figure 5. Speedup versus number of nodes for binary (left) and multiclass (right) datasets.

Figure 6 show parallel efficiency with respect to the number of nodes in the binary dataset scenario (left), and parallel efficiency with respect to the number of nodes in the multiclass dataset scenario (right). Based on Figures 5, parallel efficiency exhibits a consistently decreasing trend as the number of nodes increases across all configurations of feature dimensionality and dataset size, for both binary and multiclass classification scenarios. The most significant drop in efficiency occurs when

scaling from one to two nodes, indicating the emergence of communication and synchronization costs in the distributed system. The dataset with the largest number of observations (n=1,000,000) maintains relatively higher efficiency compared to smaller datasets when using four nodes. In contrast, for smaller datasets (n=100,000 and n=284,000), efficiency declines significantly to below 20% when scaling to four nodes.

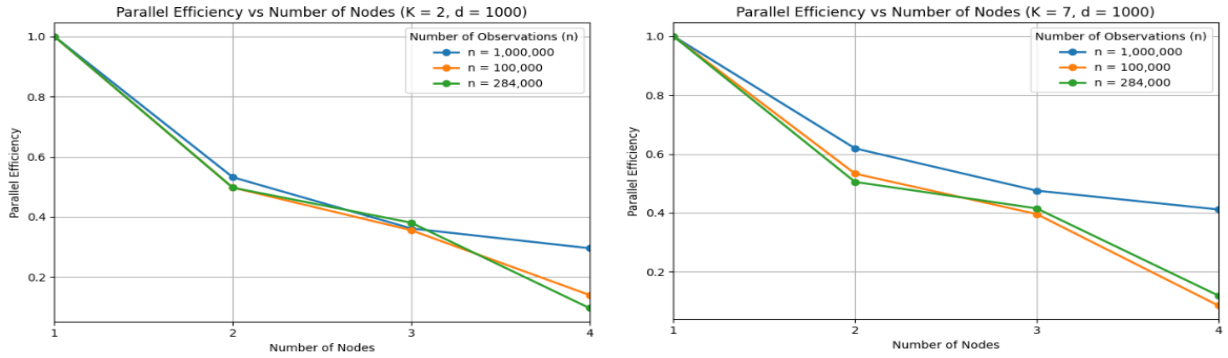

Figure 7. Parallel efficiency with respect to the number of nodes in the binary dataset scenario (left), and in the multiclass dataset scenario (right).

3.2 Analysis of Distributed Model Performance with Respect to the Number of Features

In addition to varying the number of observations, the number of features used in the experiments is also varied to analyze its effect on model training time in Apache Spark. Increasing the number of features can increase data complexity, as the training process involves operations on higher-dimensional feature vectors across all observations. Therefore, this analysis aims to evaluate the impact of feature dimensionality on training time and to assess how effectively the Apache Spark-based distributed computing approach can handle the increased computational workload.

Figure 8 show the effect of the number of features on training time for the credit card fraud (left) and forest cover type (right) dataset. This figures show that model training time is also influenced by feature dimensionality (d), where an increase in the number of features leads to higher computational cost and longer training time. In Figure 6 (a), the Credit Card Fraud dataset shows that the most notable reduction in training time due to node scaling occurs at n=1,000,000 and d=1000, where training time decreases from 214.6 seconds on one node to 197.5 seconds on four nodes, corresponding to a time efficiency improvement of 7.9%. In Figure 6 (b), the Covtype dataset achieves the largest reduction in training time at n=1,000,000 and d=1000, with a reduction of approximately 140 seconds, corresponding to an efficiency improvement of 33.62% on four nodes.

Figure 7 show the speedup as a function of the number of nodes for binary (a) and multiclass (b) datasets. Based on Figures 7, variations in feature dimensionality significantly affect speedup behavior in both dataset scenarios, with clearer patterns observed as the number of observations increases. For biggest datasets (n=1,000,000), speedup improvements are relatively limited and fluctuate across feature configurations, indicating that communication overhead dominates the computation process. However, the effect of feature dimensionality becomes more pronounced. In several cases, lower-dimensional configurations (d=30 and d=200) achieve higher speedup compared to d=1000, in both binary and multiclass classification scenarios. The multiclass scenario exhibits more stable and higher speedup growth compared to the binary case.

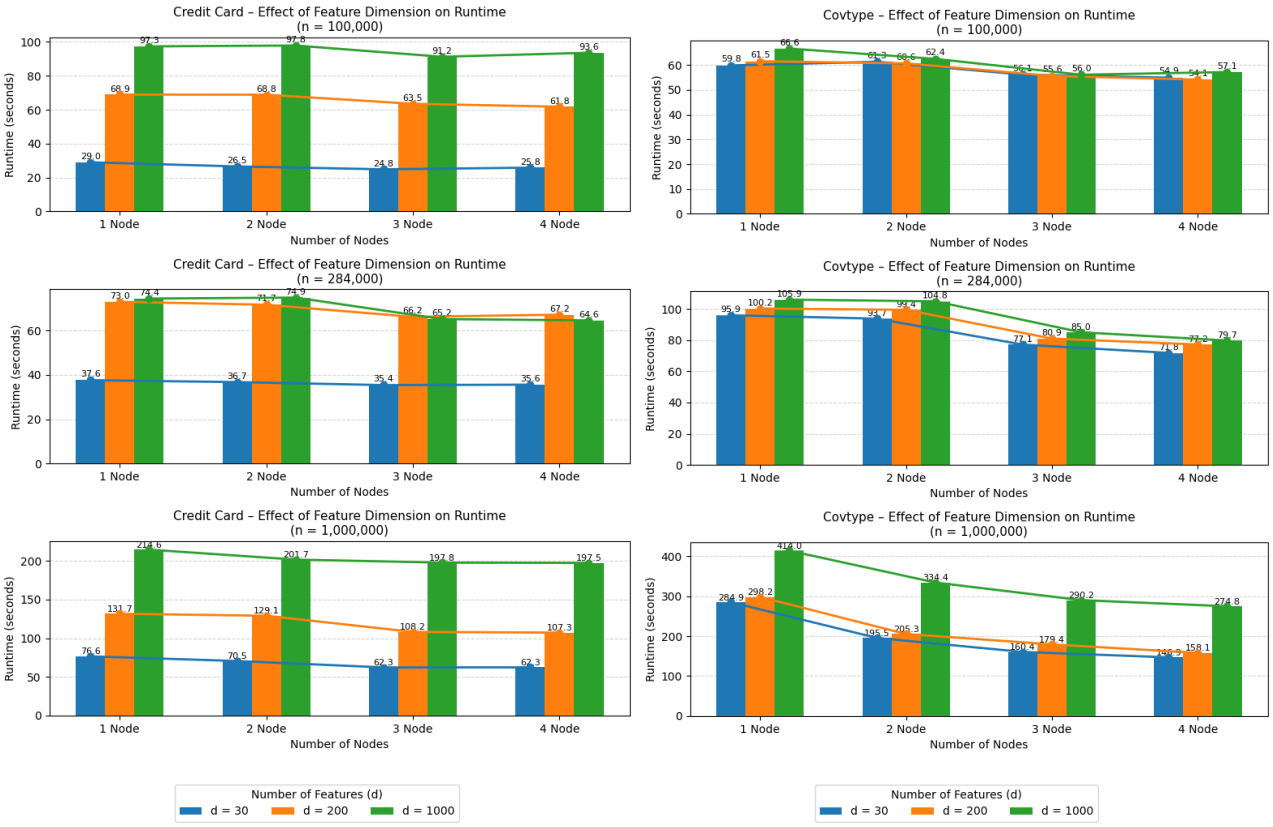

Figure 9. Effect of the number of features on training time for the credit card fraud (left) and forest cover type (right) dataset.

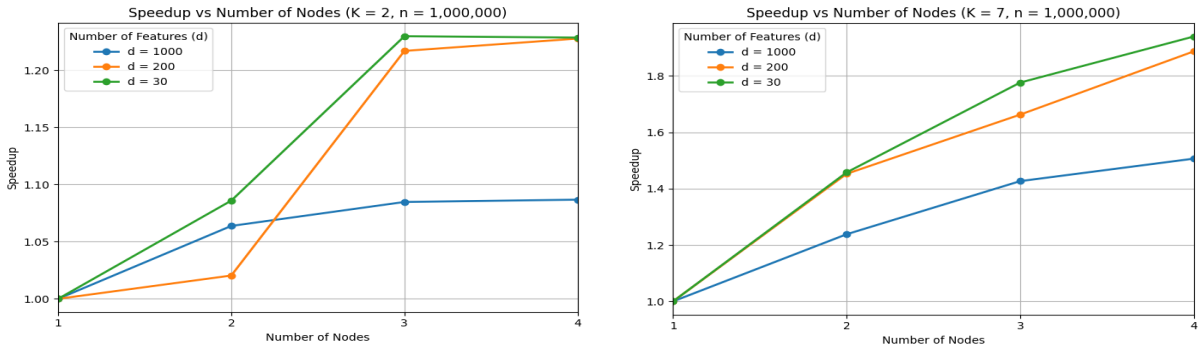

Figure 10. Speedup as a function of the number of nodes for binary (left) and multiclass (right) datasets.

Figure 11 show parallel efficiency versus number of nodes for binary (left) and multiclass (rght) datasets. Based on Figure 8, parallel efficiency in the multiclass classification scenario shows a consistent decreasing trend as the number of nodes increases across all configurations of feature dimensionality and dataset size. For the largest dataset (n=1,000,000), the impact of feature dimensionality becomes more evident, where lower-dimensional configurations (d=30 and d=200)

maintain higher efficiency up to the third node compared to d=1000. Across all scenarios, efficiency drops sharply to below 20% when scaling to the fourth node.

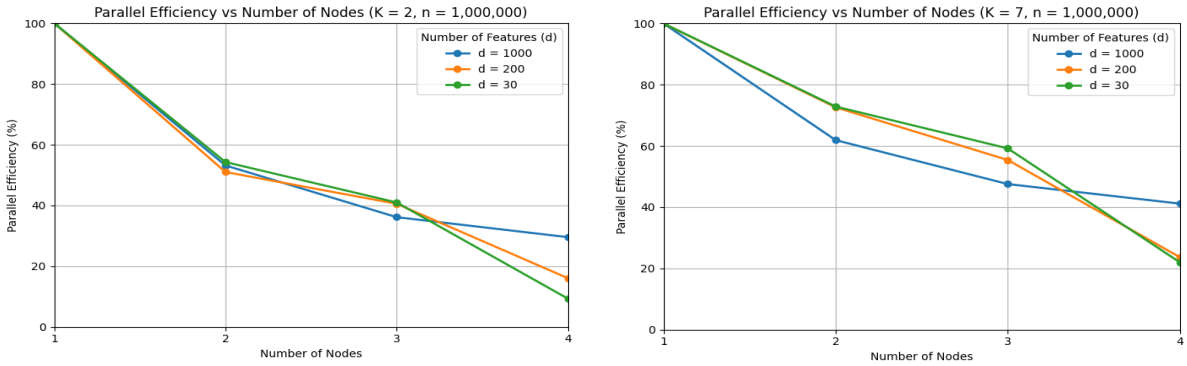

Figure 12. Parallel efficiency versus number of nodes for binary (a) and multiclass (b) datasets.

3.2. Analysis of Distributed Model Performance with Respect to the Number of Classes

The number of classes is also an important factor influencing data complexity, particularly in multiclass logistic regression. This analysis focuses on the Covtype dataset while applying feature hashing to transform numerical features into a fixed-dimensional feature space, there by representing a high-dimensional data scenario. The number of classes is varied from 2 to 7, while the dataset size is kept constant at n=581,000 and d=200. Under this setup, increasing the number of classes does not change the total number of observations, allowing the impact of class variation on training time to be analyzed more precisely. Additionally, the system is evaluated across multiple node configurations to examine the extent to which adding nodes can reduce training time.

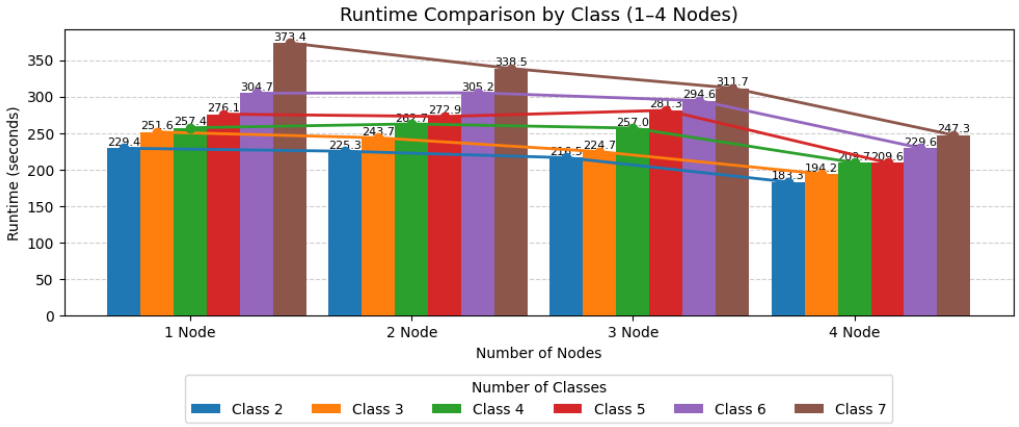

Figure 13. Effect of the Number of Classes on Training Time for the Forest Cover Type (Covtype) Dataset

Figure 9 presents a comparison of multiclass logistic regression training time across different numbers of classes using one to four nodes. In general, training time decreases as the number of computational nodes increases, although scenarios with a larger number of classes still require longer training time. In this experiment, the total dataset size remains constant across all class variations; thus, increasing the number of classes reduces the number of samples per class. However, training

time still increases because the number of gradient computations grows with the number of classes, particularly due to the use of the softmax function as defined in equation (3). This condition increases the overall computational workload and enlarges the ratio between computation and communication overhead, allowing Spark's parallelization mechanism to be utilized more effectively and resulting in more significant reductions in training time. The highest time efficiency is achieved at the maximum number of classes and nodes, specifically in the 7-class scenario, where training time decreases from 373.4 seconds on one node to 247.3 seconds on four nodes, corresponding to an efficiency improvement of approximately 33.7%. This reduction indicates that adding computational nodes effectively distributes the gradient computations in multiclass scenarios as complexity increases.

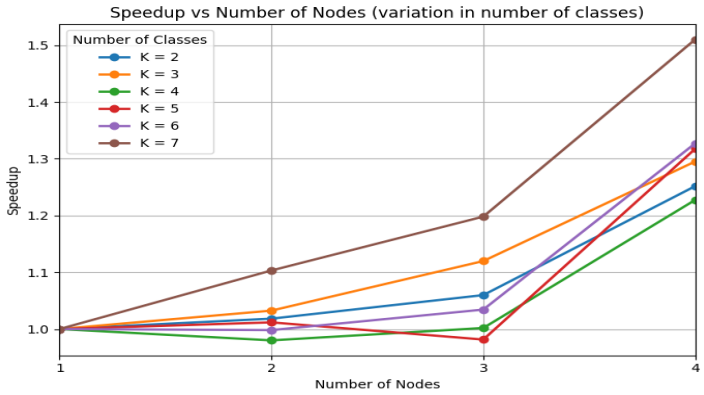

Figure 14. Speedup with respect to the number of nodes across class variations (K = 2–7)

Based on Figure 10, increasing the number of classes tends to produce higher speedup values as the number of nodes increases, with the scenario K=7 showing the greatest acceleration on four nodes. This indicates that increased computational complexity due to more classes provides sufficient workload to utilize parallelization more effectively. In contrast, for smaller numbers of classes, speedup improvements are relatively limited and tend to fluctuate, indicating the dominance of communication overhead in lighter workloads.

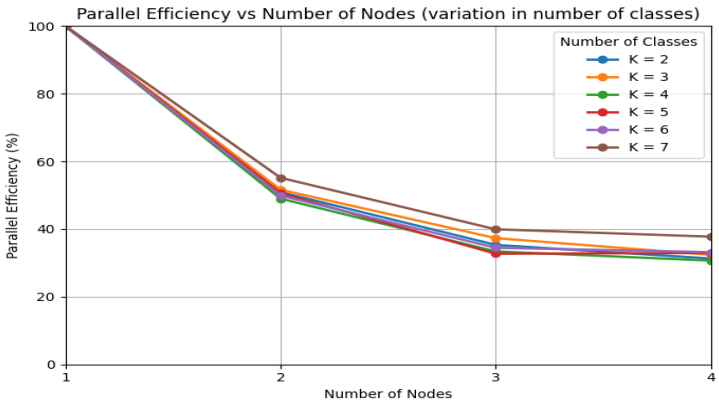

Figure 15. Parallel efficiency with respect to the number of nodes across class variations (K = 2–7)

Based on Figure 11, parallel efficiency decreases as the number of nodes increases across all class configurations, with the most significant drop occurring when scaling from one to two nodes. Across

all configurations, efficiency values are relatively similar for K=2 to K=6, while K=7 shows slightly higher efficiency at the third and fourth nodes. This suggests that increasing the number of classes leads to higher computational workload, thereby increasing the ratio between computation time and communication cost. However, efficiency still declines as more nodes are added, indicating increased communication overhead in the distributed system. These results show that although adding nodes improves speedup, it does not result in proportional gains in parallel efficiency.

3.3. **Cost Model Analysis in Distributed Models on Low-Power Infrastructure**

The cost model analysis in this subsection focuses on the most computationally intensive scenario, namely using the largest dataset size (n=1,000,000) and the highest feature dimensionality (d=1000), which represent the longest training time observed in the distributed logistic regression experiments. This approach is selected to evaluate system cost efficiency under worst-case conditions, ensuring that the conclusions reflect the capability of the low-power cluster in handling large computational workloads. This section not only evaluates training time acceleration but also examines whether adding nodes in a low-power infrastructure provides economic benefits for distributed logistic regression training. The following graphs illustrate the cost incurred based on the conducted experiments, calculated using training time and the cost model defined in Equation (9).

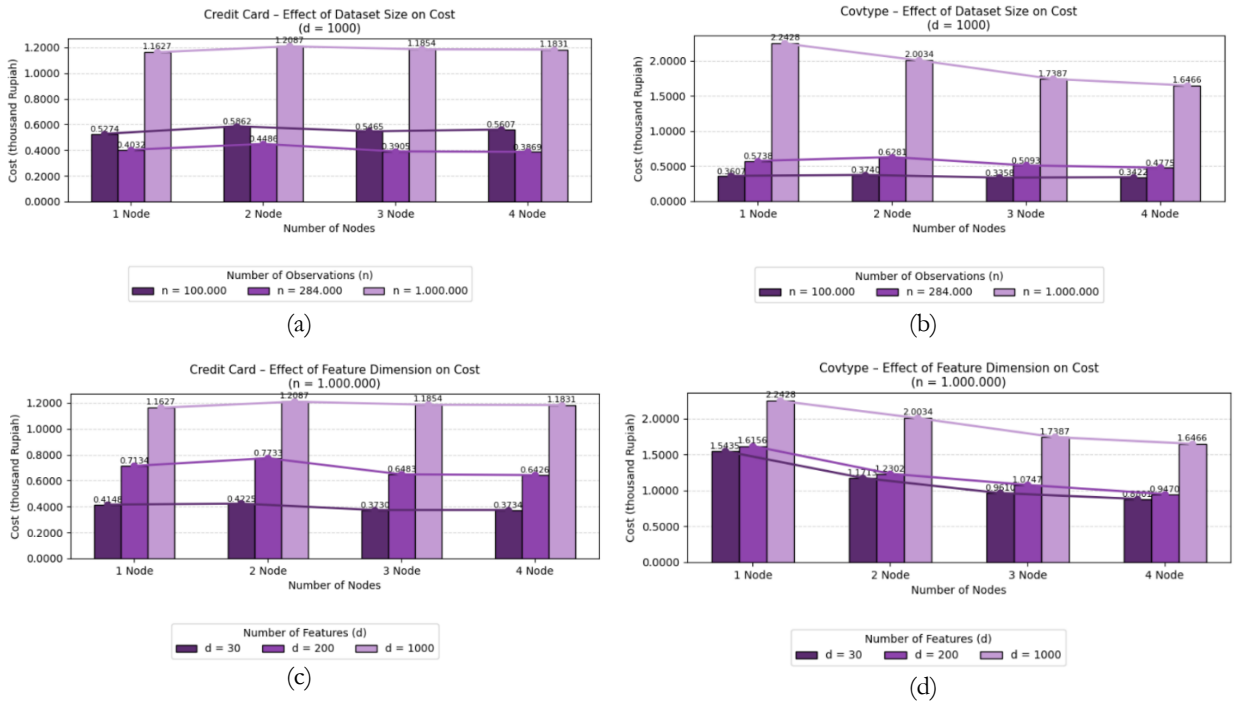

Figure 16. Computational cost versus number of observations and feature dimensionality for the Credit Card ((a), (b)) and Cover Type ((c), (d)) datasets.

Figures 12 present examples of cost model results for experiments with the largest number of observations and feature dimensionality (n=1,000,000 and d=1000). These configurations are selected based on previous analyses, which indicate that the highest computational workload occurs under these conditions. The figures illustrate cost variations resulting from increasing the number of nodes, where the results are strongly dependent on the observed training time, as described in equation (9). For the Credit Card dataset, training costs range from approximately IDR 1,162 to IDR 1,183 across

one to four nodes, showing relatively stable values with a slight decrease as the number of nodes increases. In contrast, for the Covtype dataset under the same configuration, training costs decrease more significantly, from approximately IDR 2,242 on one node to around IDR 1,646 on four nodes.

From Figures 12 (b) and (d), the distributed system demonstrates strong cost efficiency in large-scale computational scenarios. For the Covtype dataset, the highest cost is approximately IDR 2,200 on one node, which decreases to around IDR 1,650 on four nodes, representing a cost reduction of more than 20% as the number of nodes increases. Conversely, for the Credit Card dataset under the same configuration, computational costs remain within the range of IDR 1,100 to IDR 1,200 and do not show a significant decrease with additional nodes. These findings indicate that low-power clusters are most effective for workloads with high data complexity, such as multiclass classification. Although multiclass scenarios incur higher training time and cost compared to binary classification, both metrics can be significantly reduced as the number of nodes increases in a distributed environment.

4. DISCUSSION

This study evaluated the scalability performance and computational cost efficiency of distributed logistic regression on a low-power Apache Spark cluster. The experimental results demonstrate that horizontal scaling through the addition of worker nodes can effectively reduce training time, particularly for datasets with large numbers of observations, high feature dimensionality, and multiclass structures. However, the scalability gains observed in this study are not linear, indicating the presence of communication and synchronization overhead inherent in distributed computing environments.

The analysis of dataset size shows that larger datasets consistently achieved higher speedup values than smaller datasets. In particular, datasets with (n=1,000,000) observations exhibited the most substantial performance improvements when additional nodes were introduced. This finding suggests that the computation-to-communication ratio becomes more favorable as workload size increases. Similar observations have been reported in large-scale distributed computing systems, where scalability improves when computational workload dominates communication costs [25]. Consequently, distributed processing becomes increasingly beneficial for large datasets, whereas smaller datasets tend to suffer from limited scalability because communication overhead represents a larger proportion of total execution time.

A similar trend was observed in the analysis of feature dimensionality. Increasing the number of features substantially increased training time due to the higher computational complexity of gradient calculations and parameter updates. Nevertheless, higher-dimensional datasets did not always achieve proportionally better speedup. In several scenarios, lower-dimensional configurations ((d=30) and (d=200)) demonstrated better scalability than (d=1000). This result indicates that increasing feature dimensionality not only increases computational workload but also enlarges the amount of information exchanged during distributed optimization, thereby increasing synchronization costs. Such behavior is consistent with previous studies on distributed machine learning, which emphasize that computational complexity alone does not guarantee improved parallel efficiency when communication overhead remains significant [26].

The multiclass classification experiments further support the importance of workload complexity in distributed environments. As the number of classes increased, training time also increased due to the additional gradient computations required by the softmax optimization process. However, higher-class scenarios achieved greater speedup and slightly better parallel efficiency than lower-class scenarios. This finding indicates that increased model complexity provides sufficient computational

workload to offset part of the communication overhead. Similar behavior has been reported in distributed implementations of machine learning algorithms, where larger optimization problems generally exhibit more favorable computation-to-communication ratios and better resource utilization [27].

Although speedup increased as more nodes were added, parallel efficiency consistently decreased across all experimental scenarios. The most pronounced decline occurred when scaling from one to two nodes, indicating that communication and synchronization costs emerge immediately once distributed execution is introduced. This phenomenon is consistent with Amdahl's Law, which states that the achievable speedup of a parallel system is fundamentally constrained by the serial portion of the computation and associated overhead [28]. Similar non-linear scalability behavior has also been observed in SBC-based clusters, where network bandwidth limitations and resource coordination reduce the effectiveness of adding more computational nodes [29]. Therefore, while horizontal scaling improves overall performance, the efficiency gains diminish as cluster size increases.

From an economic perspective, the results demonstrate that low-power clusters provide meaningful cost advantages for computationally intensive machine learning workloads. The multiclass Covtype dataset showed a reduction in computational cost exceeding 20% when scaling from one to four nodes. This outcome indicates that the reduction in runtime compensates for the additional energy consumption associated with operating multiple nodes. In contrast, the binary Credit Card dataset exhibited only marginal cost reductions because the workload was insufficient to fully exploit distributed computation. These findings reinforce the concept of Green AI, which emphasizes that machine learning systems should be evaluated not only in terms of predictive capability but also with respect to energy consumption and computational cost [30].

Overall, the findings extend previous studies on low-power distributed computing by demonstrating that SBC-based Apache Spark clusters can simultaneously provide scalability and cost-efficiency benefits for machine learning workloads. While earlier studies primarily focused on energy efficiency or runtime performance independently, the present work integrates both perspectives and shows that the effectiveness of horizontal scaling strongly depends on workload characteristics, including dataset size, feature dimensionality, and classification complexity. These results suggest that low-power distributed infrastructures constitute a practical and sustainable alternative for educational institutions and research laboratories with limited computational resources.

5. CONCLUSIONS

This study investigated the scalability and computational cost efficiency of distributed logistic regression on a low-power Apache Spark cluster. The results show that increasing the number of observations, feature dimensionality, and class complexity increases computational workload, while horizontal scaling effectively reduces training time and improves speedup, particularly for large-scale and multiclass datasets. The highest training-time reduction reached 46.98% when scaling from one to four nodes. Although speedup improved with additional nodes, parallel efficiency decreased because of communication and synchronization overhead. Nevertheless, the low-power cluster maintained low operational costs and achieved cost reductions exceeding 20% for computationally intensive multiclass workloads. These findings demonstrate that SBC-based Apache Spark clusters provide a scalable and cost-efficient alternative for AI and big data learning in resource-constrained educational and research environments, while supporting the principles of Green AI through reduced energy consumption and operational cost. Future research may explore communication-cost analysis,

larger cluster configurations, and additional machine learning algorithms to further evaluate scalability–efficiency trade-offs in distributed learning systems.

REFERENCES

[1] M. Mustopa, N. Nasikhin, R. Chamami, H. Nihayah, M. R. Habibullah, and A. Manshur, "Challenges in Artificial Intelligence Development in Higher Education in China, India, and Indonesia: International Students' Perspectives," 2024. doi: 10.26803/ijlter.23.2.17.

[2] M. F. Nasution, "Desain Arsitektur Berbasis Teknologi AI untuk Peningkatan Efisiensi Energi," *arsitek*, vol. 1 no 10, pp. 1–10, 2024.

[3] K. R. Ririh *et al.*, "Studi Komparasi dan Analisis SWOT pada Implementasi Kecerdasan Buatan (Artificial Intelligence) di Indonesia," vol. 15, no. 2, pp. 122–133, 2020.

[4] S. Rusfi, M. L. Liebenlito, and T. Edy, "Kajian Performa Efisiensi Infrastruktur Big Data Hemat Energi Menggunakan Single Board Computer dan Framework Apache Spark," *J. Nas. UNAND*, vol. 01, pp. 152–160, 2025, doi: 10.25077/TEKNOSI.v11i2.2025.152-160.

[5] S. Bourhnane, M. R. Abid, K. Zine-dine, N. Elkamoun, and D. Benhaddou, "Cluster of Single-Board Computers at the Edge for Smart Grids Applications," *Appl. Sci.*, vol. 11, no. 22, 2021, doi: 10.3390/app112210981.

[6] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, 2020, doi: 10.1145/3381831.

[7] M. Li *et al.*, "Scaling Distributed Machine Learning with the Parameter Server," *Proc. OSDI*, pp. 583–598, 2014, [Online]. Available: http://www.mzi.gov.si/si/medijsko_sredisce/novica/article/799/8867/

[8] C. A. A. Beauchemin and A. Handel, "A review of mathematical models of influenza A infections within a host or cell culture: Lessons learned and challenges ahead," *BMC Public Health*, vol. 11, no. SUPPL. 1, pp. 1–15, 2011, doi: 10.1186/1471-2458-11-S1-S7.

[9] A. Pradhan, S. K. Bisoy, S. Kautish, M. B. Jasser, and A. W. Mohamed, "Intelligent Decision-Making of Load Balancing Using Deep Reinforcement Learning and Parallel PSO in Cloud Environment," 2022. doi: 10.1109/access.2022.3192628.

[10] W. Gan, Z. Ma, and S. Liu, "Dimensionality Reduction for Tensor Data Based on Projection Distance Minimization and Hilbert-Schmidt Independence Criterion Maximization1," *J. Intell. Fuzzy Syst. Appl. Eng. Technol.*, vol. 40, no. 5, pp. 10307–10322, 2021, doi: 10.3233/jifs-202582.

[11] X. Li, X. Zhu, and H. Wang, "Distributed Logistic Regression for Massive Data With Rare Events," *Stat. Sin.*, 2025, doi: 10.5705/ss.202022.0242.

[12] S. Wu, J. Zhou, K. Xu, and H. Wang, "Class-Distributed Learning for Multinomial Logistic Regression With High Dimensional Features and a Large Number of Classes," 2024. doi: 10.1080/10618600.2024.2362230.

[13] P. J. Basford *et al.*, "Performance Analysis of Single Board Computer Clusters," 2020. doi: 10.1016/j.future.2019.07.040.

[14] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the Computational Cost of Deep Learning Models," 2018, *Arxiv*. doi: 10.1109/bigdata.2018.8622396.

[15] S. Azka, M. Liebenlito, and T. E. Sutanto, "Analisis Performa Klaster Single Board Computer Dalam Implementasi Singular Value Decomposition," 2025. doi: 10.37905/euler.v13i3.33367.

[16] D. P. Noer, M. Liebenlito, E. Sutanto, D. P. Noer, M. Liebenlito, and E. Sutanto, "Analisis Kinerja dan Efisiensi Energi k-means dan Gaussian Mixture Model Terdistribusi pada Klaster

Single Board Computer dan Personal Computer dengan Apache Spark.," *Jambura J. Ilm. Mat.*, 2026, doi: 10.37905/jjom.v8i1.33549.

[17] "Credit Card Fraud Detection," 03 May 2021. Accessed: Dec. 03, 2025. [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[18] U. M. Learning, "Forest Cover Type Dataset," Kaggle. Accessed: Dec. 08, 2025. [Online]. Available: https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset

[19] K. Weinberger, J. Attenberg, and A. S. Org, "Feature hashing for large scale multitask learning," no. Icml, 2009.

[20] "Apache Spark." Accessed: Jan. 10, 2026. [Online]. Available: https://spark.apache.org/

[21] M. Zaharia *et al.*, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," 2012. doi: 10.5555/2228298.2228301.

[22] "Apache Spark, 'Cluster Mode Overview', Apache.org." Accessed: Nov. 26, 2025. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview

[23] "Linear Methods Documentation Spark." Accessed: Jan. 04, 2026. [Online]. Available: https://dist.apache.org/repos/dist/release/spark/docs/1.5.2/mllib-linear-methods.html

[24] J. M. Harbangan, "Detail Tarif Listrik Berdasarkan Golongan Daya dan Jenisnya," PLN. Accessed: Jan. 10, 2026. [Online]. Available: https://web.pln.co.id/cms/media/2025/12/tarif-listrik/

[25] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.

[26] G. C. Fox, J. Qiu, S. Jha, J. Ekanayake, and A. Luckow, "High performance data analytics on clouds and HPC," Future Gener. Comput. Syst., vol. 36, pp. 13–27, Jul. 2014, doi: 10.1016/j.future.2013.07.001.

[27] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, et al., "MLlib: Machine learning in Apache Spark," J. Mach. Learn. Res., vol. 17, no. 34, pp. 1–7, 2016.

[28] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in Proc. AFIPS Spring Joint Computer Conf., vol. 30, Atlantic City, NJ, USA, 1967, pp. 483–485, doi: 10.1145/1465482.1465560.

[29] P. J. Basford, S. J. Johnston, C. S. Perkins, H. Herry, and F. P. Tso, "Performance analysis of single board computer clusters," Future Gener. Comput. Syst., vol. 102, pp. 278–291, Jan. 2020, doi: 10.1016/j.future.2019.07.040.

[30] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," Commun. ACM, vol. 63, no. 12, pp. 54–63, Dec. 2020, doi: 10.1145/3381831.