Comparative Analysis of Persistence Storage Levels in Spark with Case Study

THET HSU AUNG[1], AYE MYAT MYAT PAING[2]
[1, 2] *Faculty of Computer Science, University of Information Technology*

Abstract- ***This study conducts a comparative analysis of the training times for Long Short-Term Memory (LSTM) networks on Apache Spark, evaluating three different persistence storage levels: Disk-Only, Memory-Disk, and Memory-Only. The analysis is performed with and without a proposed sampling algorithm designed to address the issue of imbalanced datasets. The study specifically focuses on the Credit Card Fraud Detection dataset across varying dataset sizes. The results indicate that the Memory-Only storage level achieves the shortest training times. Considering this case study, the amount of the dataset influences the storage level selection. Therefore, this dataset indicates that memory_only is the best. Memory_disk_only is the second-best option, if memory becomes insufficient due to the growing dataset. Therefore, the choice of storage level affects performance, especially in memory usage and computation speed. Furthermore, the application of the sampling algorithm significantly enhances model performance metrics, including precision, recall, and F1-score, particularly in scenarios involving imbalanced data. These findings provide crucial insights for improving LSTM training on large-scale imbalanced datasets, highlighting the importance of selecting appropriate storage configurations and preprocessing techniques in big data environments.***

Indexed Terms- ***Long Short-Term Memory, Disk-Only, Memory-Only, Memory-Disk***

I. INTRODUCTION

In the era of big data and machine learning, the efficient training of deep learning models, such as Long Short-Term Memory (LSTM) networks, is essential for various applications that involve sequential data. LSTM networks are particularly adept at handling tasks like time series prediction, natural language processing, and anomaly detection. However, training these models on large-scale datasets poses significant computational challenges, often requiring substantial time and resources. Apache Spark, a powerful distributed computing system, offers a promising solution to these challenges by enabling the parallel processing of large datasets, thereby accelerating the training process of machine learning models.

One critical aspect of improving LSTM training on Apache Spark involves the configuration of storage levels, which dictate how data is cached during computations. Apache Spark provides various storage levels, such as Disk-Only, Memory-and-Disk, and Memory-Only, each with distinct performance characteristics. The Disk-Only storage level stores data on disk, which conserves memory but can increase data retrieval times. The Memory-and-Disk storage level stores data in memory as much as possible, spilling over to disk only when memory is insufficient. The Memory-Only storage level keeps data entirely in RAM, which reduces access times but requires ample memory capacity. Understanding the impact of these storage levels on training times is crucial for improving the performance of LSTM models.

Another significant challenge in machine learning is dealing with imbalanced datasets. Imbalanced datasets, where some classes are underrepresented, can lead to biased models that perform poorly on minority classes. The Credit Card Fraud Detection dataset is a prominent example of an imbalanced dataset, where fraudulent transactions are much rarer than legitimate ones. Addressing data imbalance is essential for developing robust and reliable machine learning models.

To tackle the issue of data imbalance, this study proposes a Sampling Algorithm for Imbalanced Datasets. This algorithm aims to improve the representation of minority classes, thereby enhancing

the performance and reliability of the trained models. The primary objectives of this study are:

To compare the training times of LSTM models on Apache Spark with Disk-Only, Memory-and-Disk, and Memory-Only storage levels.

To evaluate the effectiveness of the proposed Sampling Algorithm for Imbalanced Datasets in reducing training time and improving model performance.

To analyze the performance improvements achieved through the combination of different storage levels and the sampling algorithm with different dataset size.

This study conducts a comprehensive comparison analysis of LSTM training times on Apache Spark, focusing on three different storage configurations: Disk-Only, Memory-and-Disk, and Memory-Only. The analysis is applied to the Credit Card Fraud Detection dataset, an imbalanced dataset that presents a realistic and challenging scenario for model training with different dataset sizes. The findings offer valuable insights into the most efficient storage configurations, aiding practitioners in choosing the appropriate storage level to balance resource usage and training speed. Moreover, the evaluation demonstrates how the algorithm enhances model performance and training efficiency, providing a deeper understanding of how data preprocessing techniques can mitigate the issues associated with imbalanced datasets.

The remainder of this paper is structured as follows. Section 2 reviews related work on LSTM networks, Apache Spark storage configurations, and techniques for handling imbalanced datasets. Section 3 theorical background. Section 4 presents Persistence Storage Levels in Spark. Section 5 describes the case study including the Credit Card Fraud Detection dataset and the proposed Sampling Algorithm for Imbalanced Datasets. Section 6 outlines the experimental setup, including the implementation of LSTM models on Apache Spark and the configurations tested. Section 7 presents the results of the experiments, comparing training times for different dataset sizes and model performance across different configurations. Finally, Section 8 concludes the paper and suggests directions for future research.

II. LITERATURE REVIEW

Kim, Lee, and Yoo (2019) [1] explored the integration of big data analytics with in-memory databases in the context of smart manufacturing. The authors discuss the challenges and opportunities presented by big data in the manufacturing sector, emphasizing the need for real-time data processing and analysis to enhance operational efficiency and decision-making. By leveraging in-memory databases, which allow data to be stored and processed directly in the main memory rather than on traditional disk storage, the study demonstrates significant improvements in data retrieval speeds and analytics performance. The research highlighted how in-memory database technologies can support various smart manufacturing applications, including predictive maintenance, quality control, and production optimization. The authors present case studies and experimental results that showcase the effectiveness of their approach, illustrating how it can lead to more responsive and adaptive manufacturing systems. The paper concludes with a discussion on the future directions for big data analytics in smart manufacturing, underscoring the potential for further innovations and the importance of continued research in this area.

In the paper, Park, Kim, and Yoo [2] presented a hybrid machine learning approach designed to enhance financial early warning systems (FEWS). The authors combined various machine learning techniques, including decision trees, support vector machines, and neural networks, to create a robust ensemble model. This hybrid approach leverages the strengths of each individual method, mitigating their respective weaknesses and resulting in more precise predictions. The study involved extensive experimentation with real-world financial data to validate the effectiveness of the proposed model. The results demonstrated that the hybrid model outperforms traditional single method approaches in terms of predictive accuracy and early detection capabilities. By providing more accurate and timely warnings, the proposed system has the potential to

assist financial institutions and regulators in taking proactive measures to prevent financial crises.

The authors [3] introduced a distributed deep learning framework that integrates edge computing, which involves processing data closer to the data source, thereby reducing the reliance on centralized cloud resources. This method is particularly beneficial for IoT environments where real-time data processing and low latency are critical. The framework partitions the deep learning tasks across multiple edge devices, allowing for parallel processing and reducing the load on central servers. The paper details the architecture of the proposed system, including the mechanisms for data partitioning, model synchronization, and task scheduling. Through extensive experimental evaluations, the authors demonstrate that their edge computing-based approach significantly improves the efficiency and speed of deep learning tasks compared to traditional cloud-based methods. The results show notable reductions in latency and bandwidth usage, making the approach suitable for various IoT applications, including smart cities, autonomous vehicles, and industrial automation.

The authors proposed a parallel data mining algorithm designed to efficiently handle big data using the Hadoop framework [4]. The core objective of the research is to address the challenges associated with mining large-scale datasets by leveraging the parallel processing capabilities of Hadoop, an open-source framework that facilitates distributed storage and processing of big data. Experimental evaluations are conducted to compare the performance of the proposed parallel algorithm against traditional, non-parallel approaches. The results demonstrate substantial improvements in processing speed and scalability, validating the effectiveness of the Hadoop-based solution for big data mining tasks.

The authors [5] identified the inefficiencies in existing cluster computing frameworks, such as MapReduce, which often rely on disk-based storage, leading to high latency and complexity in managing fault tolerance. To address these issues, they propose RDDs, which are distributed memory abstractions that enable users to perform computations on large datasets more efficiently. RDDs are immutable, partitioned collections of objects that can be operated on in parallel. They are designed to be fault-tolerant by maintaining lineage information that allows lost data to be recomputed, thus eliminating the need for costly replication strategies. This approach significantly reduces the overhead associated with fault tolerance. To evaluate the performance of RDDs, the authors conduct experiments comparing Spark with Hadoop, demonstrating that Spark can be up to 100 times faster in memory and 10 times faster on disk for certain applications. The results underscore the efficiency and scalability of RDDs in handling iterative and interactive queries on large datasets.

Boosting and bagging are widely used ensemble learning techniques that aim to improve model accuracy by aggregating predictions from multiple base learners. Boosting focuses on sequential training models to correct the errors of its predecessors, while bagging involves training models on different subsets of the data and aggregating their predictions through averaging or voting. The authors [6] conducted a comparative analysis of boosting (specifically AdaBoost) and bagging (specifically Random Forests) across several experimental setups using benchmark datasets with varying degrees of noise and class imbalance. They evaluate the performance of these techniques using standard metrics such as accuracy, precision, recall, and F1-score. Key findings from the study highlight those boosting methods, particularly AdaBoost, exhibit superior performance compared to bagging methods like Random Forests when dealing with noisy and imbalanced data. AdaBoost demonstrates robustness in handling noisy features and class imbalance, achieving higher classification accuracy and more balanced performance across different evaluation metrics.

The paper [7] provided a comprehensive review of techniques and methodologies proposed in the literature to mitigate the impact of class imbalance on machine learning models. They discussed the strengths, limitations, and applicability of each approach based on experimental results and empirical studies reported in the literature. They emphasize the importance of evaluating model performance using metrics beyond accuracy, such as precision, recall, F1-score, and area under the receiver operating

characteristic curve (AUC-ROC), which are more informative in imbalanced settings.

III. THEORICAL BACKGROUND

This section describes the theorical background of the proposed system.

A. Apache Spark Architecture

The Apache Spark framework manages data in the form of Resilient Distributed Datasets (RDDs), which store compute objects across networked cluster nodes. RDDs are stored in distributed memory (RAM) to optimize performance and enable fault tolerance by persisting intermediate results on disk drives or SSDs [8]. Spark jobs are executed in two phases—ShuffleMapStage and ResultStage—to organize tasks and gather computed results, respectively. However, data skewness, where partitions unevenly distribute data, can challenge operations, especially in join procedures, impacting performance. In enhancing reliability for real-time messaging systems, fault tolerance is crucial. Apache Kafka serves as a popular solution, offering flexible and reliable message delivery. Kafka ensures messages are delivered reliably across distributed processing environments like Apache Spark, where data partitioning and processing independence can lead to varying processing times across partitions, affecting overall task completion speed.

B. Imbalanced Data on Apache Spark

Imbalanced datasets present significant challenges in machine learning, especially within Big Data frameworks like Apache Spark. In such frameworks, data is divided into partitions across multiple machines within a cluster, potentially resulting in unequal data distribution among partitions. This phenomenon, known as data skew, can lead to performance issues, including degraded model performance, unstable training, misleading evaluation metrics, and inefficient resource allocation. To mitigate these challenges, effective balancing methods tailored for Big Data environments are crucial. These methods should leverage high-performance computing architectures such as CPU clusters and GPUs to ensure scalability and efficiency in handling large and imbalanced datasets. Techniques like sampling algorithms are commonly employed to balance datasets before applying machine learning tasks. Addressing data imbalance is essential for improving classification performance and enhancing the reliability of data processing activities in Apache Spark. By implementing robust balancing strategies, machine learning models can achieve better accuracy and stability, thereby improving overall system performance in Big Data applications.

C. Oversampling

Oversampling, also known as upsampling, is a method used to adjust the class distribution of a dataset by increasing the number of instances in the minority class [7]. This is done by randomly replicating instances from the minority class until it reaches a desired balance with the majority class. The goal is to alleviate the bias towards the majority class and improve the model's ability to learn from the minority class instances.

D. Long Short-Term Memory

A Long Short-Term Memory (LSTM) network represents a specialized type of recurrent neural network (RNN) designed to manage long-range dependencies and sequential data effectively. Unlike conventional RNNs, LSTMs are equipped with memory cells capable of retaining information over extended periods, which enables them to capture intricate patterns in time series and sequential data. The architecture of an LSTM comprises crucial components such as the input gate, forget gate, output gate, and memory cell. These elements work in tandem to regulate the flow of information into and out of the memory cell. By selectively retaining or discarding information as per task requirements, LSTMs can mitigate the vanishing gradient problem that commonly hampers traditional RNNs. This capability makes LSTMs well-suited for training on sequences with prolonged dependencies. LSTMs find applications across diverse domains, including natural language processing (NLP), speech recognition, time series forecasting, healthcare for medical time series analysis, and finance for predicting market trends. Their proficiency in learning and remembering patterns within sequential data has established LSTMs as a pivotal tool in contemporary machine learning research and practical applications.

IV. PERSISTENCE STORAGE LEVELS IN SPARK

In Apache Spark [9, 10], persistence storage levels refer to the strategies used to store RDDs (Resilient Distributed Datasets) across the nodes of a cluster. These storage levels are crucial for ***improving*** performance by balancing memory usage, computation speed, fault tolerance, and disk I/O. Here's a detailed description of the common persistence storage levels available in Spark as shown in Figure 1.

- MEMORY_ONLY: This is the default storage level in Spark. It stores RDDs as deserialized Java objects in the JVM heap memory of the executor nodes. This allows for fast access to data but is limited by the available memory size. If an RDD does not fit in memory, recomputation will be necessary.
- MEMORY_AND_DISK: This storage level stores RDD partitions that do not fit in memory on disk. The partitions that fit in memory stay in memory, while the remaining ones are spilled to disk. This level provides better tolerance for large datasets that exceed the available memory capacity but may incur higher access latency due to disk reads.
- MEMORY_ONLY_SER and MEMORY_AND_DISK_SER: These levels store RDDs as serialized Java objects (binary data) in memory (or on disk for MEMORY_AND_DISK_SER). Serialization reduces the memory usage compared to storing objects directly but adds CPU overhead for serialization and deserialization. This can be beneficial when dealing with large objects or when memory resources are limited.
- DISK_ONLY: RDDs are stored only on disk, which is useful when RDDs are too large to fit in memory. This level trades off computation speed for increased fault tolerance and the ability to handle very large datasets. Disk reads are slower compared to memory access, so performance may be impacted.
- MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.: These are variants of the above storage levels that replicate each partition on two nodes to provide data redundancy and fault tolerance. This redundancy helps in recovering lost data partitions due to node failures but increases memory usage and storage requirements.

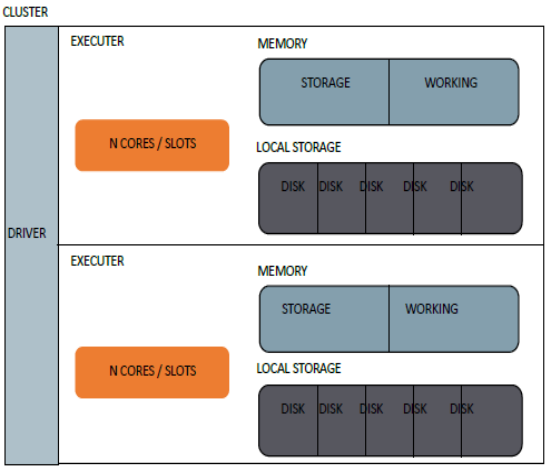

Figure 1. Storage Levels

Choosing the right persistence storage level depends on several factors including the size of the dataset, available memory in the cluster, computation speed requirements, and fault tolerance considerations. By selecting an appropriate storage level, Spark users can optimize performance, reduce recomputation overhead, and effectively manage large-scale data processing tasks in distributed environments [11, 12].

V. CASE STUDY

In this system, training unbalanced dataset: Credit card dataset is used. The proposed system workflow is shown in Figure 2. The process begins with loading the credit card dataset into Apache Spark, followed by applying the proposed sampling algorithm to generate balanced datasets. During the model training phase, the LSTM model is trained on the dataset using three different storage levels: Disk-Only, Memory-Disk, and Memory-Only. Training is conducted with both the original imbalanced dataset and the balanced dataset created by the sampling algorithm. The performance comparison involves analyzing training times across the various storage configurations, evaluating the impact of dataset size on training efficiency, and assessing the benefits of the sampling algorithm in terms of training speed and convergence.

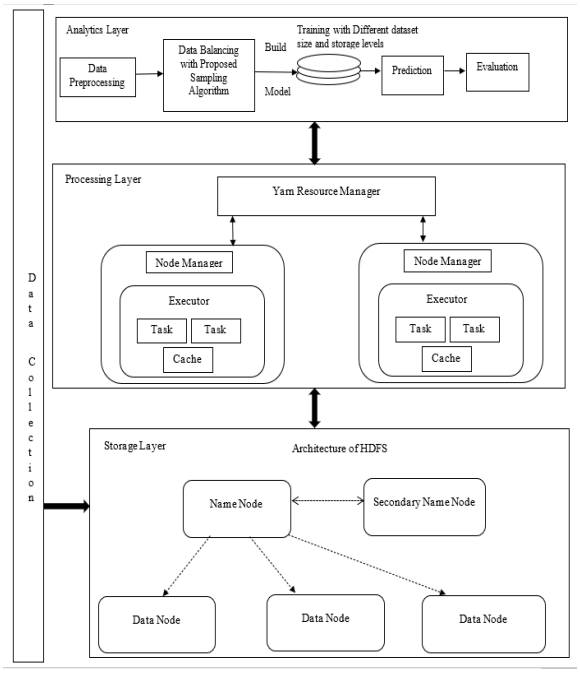

Figure 2. Workflow on Apache Spark Architecture

A. Dataset

The credit card dataset sourced from a European financial institution serves as a crucial resource for studying fraud detection in electronic transactions while safeguarding cardholders' privacy. Spanning a two-day period, the dataset captures both legitimate and fraudulent transactions, anonymized to protect sensitive details like card numbers and identities. Key attributes include transaction time, amount, and anonymized features derived from PCA to preserve confidentiality [13]. The dataset's scale enables robust analysis despite its inherent imbalance, with fraudulent transactions significantly fewer than legitimate ones. Its imbalanced nature poses challenges, requiring specialized techniques to ensure accurate classification. By leveraging anomaly detection methods and robust classification algorithms, analysts can uncover fraudulent patterns and enhance transaction security. This dataset's value lies in its practical application, facilitating research into fraud prevention strategies and evaluation of detection algorithms within real-world transaction environments. This dataset is shown in figure 3.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.06692
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.33984
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.68928
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.17557
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.14126
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.50934
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.01622
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.64013
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.12320
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.00679

284807 rows × 31 columns

Figure 3. Credit Card Transaction in Dataset

The balance of class labels in the dataset are displayed in Table 1. 99% of the data samples that were evaluated for the dataset correspond to the valid class "0." This data is very imbalanced.

Table 1. Distribution of Sample in Credit Card Dataset

Class Label	No. of Class
Non-Fraud	284315
Fraud	492

B. Proposed Sampling Algorithm

Class imbalance, where one class significantly outnumbers the other, poses challenges in machine learning. Traditional approaches include oversampling, which duplicates minority class instances, but this can lead to overfitting. Conversely, undersampling, which reduces the majority class, risks losing critical data. To address these issues, the proposed system introduces an innovative sampling algorithm.

To achieve a balanced dataset from the training data, follow these detailed steps:

- It begins with training data and the corresponding labels. It determines the counts of fraud and non-fraud instances in training data.
- It identifies which class, fraud or non-fraud, has a higher count. This will be used to balance the dataset by generating additional instances of the minority class.
- In Balancing Process, if the count of the fraud class is greater than the count of the non-fraud class,

- The difference (num) between the counts of the two classes is calculated.
- a feature vector is extracted from the non-fraud class.
- The last element of the extracted feature vector is removed.
- A single feature from the adjacent non-fraud instance is obtained.
- This feature is appended to the modified feature vector.
- Then, the updated feature vector is appended to the non-fraud dataset. After that, label 1 is appended to the abnormal labels.
- The difference (num) between the counts of the two classes.
- A feature vector is extracted from the fraud class.
- The last element of the extracted feature vector is removed.
- A single feature from the adjacent fraud instance is obtained.
- This feature is appended to the modified feature vector.
- Then, the updated feature vector is appended to the fraud dataset.
- After that, the label 0 is appended to the fraud labels.
- Finally, the result is a balanced dataset where the number of instances for each class is equalized.

This flowchart is presented in Figure 4.

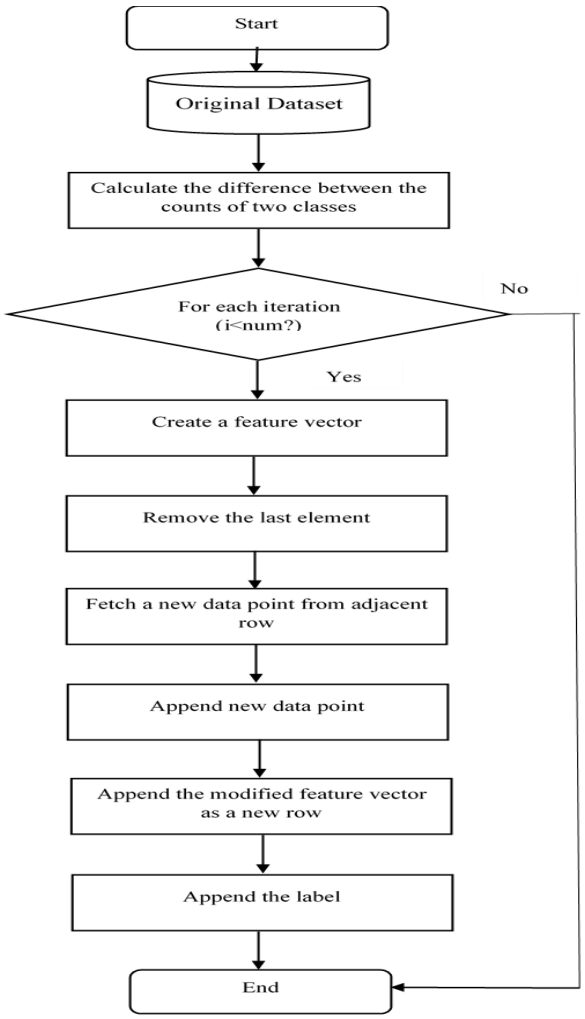

Figure 4. Flowchart of Proposed Sampling Algorithm

It can be applied to various types of data (numerical, categorical) and can be adjusted based on specific needs by modifying the way synthetic samples are generated. By adding synthetic samples rather than duplicating existing ones, this method can help reduce the risk of overfitting.

VI. EXPERIMENTAL SETUP

In this experiment, Apache Spark cluster is developed and is used for integrating with deep learning like LSTM. The system specification and necessary software components of the system are presented in Table 2. In this experiment, the parameters for LSTM for loss is "categorical_crossentropy," batch size is "128," and epochs is "100" for LSTM, activation function "Softmax," and optimizer "Adam." The Apache

Spark model provides the least training time. The main benefit of applying Spark is that the Spark cluster is constructed with commodity hardware. The nodes are situated within the same rack.

Table 2. System Specification

Operating System	Ubuntu 20.04 LTS
Host Specification	Intel ® Core i7-11800H
	CPU @ 2.30GHz
	16GB Memory
	512 SSD
VM Specification	8 GB RAM
	100 GB Hard Disk
Software Components	Hadoop 3
	Spark 3.3.3 Python 3.10
	Elephas 4.0.1

VII. EVALUATION

In the system evaluation, LSTM model is trained on Apache Spark using different storage levels and dataset size. The training is conducted with both the original and balanced datasets to evaluate the impact of the sampling algorithm. Table 3, 4, and 5, describe the performance results with the storage levels: memory-only, disk-only, and memory-disk on original and balanced dataset.

Two dataset sizes were used for evaluation: 280000 and 560000 instances. The training was conducted with both the original and balanced datasets to evaluate the impact of the sampling algorithm. The performance results were evaluated using different storage-levels. For each run, the metrics (Precision, Recall, and f1-Score) were calculated and the final reported metrics are the average values across all runs. Despite variations in dataset sizes, the LSTM model consistently achieved perfect scores of 1 for precision, recall and F1-score across different storage levels. This indicates that the model's robustness and the effectiveness of the training and proposed sampling methodologies employed. The following tables illustrate the consist results.

Table 3. Performance Results of Memory-Only

	Precision	Recall	F1-Score
Dataset size:280000	1	1	1
Dataset size:560000	1	1	1

Table 3 presents the performance results of the Long Short-Term Memory (LSTM) network when using the Memory-Only storage level with proposed sampling algorithm, measured across two different dataset sizes: 280,000 and 560,000. The metrics evaluated are precision, recall, and F1-score, each of which is crucial for assessing the effectiveness of the model, especially in the context of credit card fraud detection.

Therefore, Table 3 highlights the efficacy of using the Memory-Only storage level for training LSTM networks on the Credit Card Fraud Detection dataset. The perfect precision, recall, and F1-score across varying dataset sizes underscore the model's robustness and accuracy, reinforcing the suitability of the Memory-Only storage level for high-performance LSTM training in big data environments.

Table 4. Performance Results of Disk-Only

	Precision	Recall	F1-Score
Dataset size:280000	0.98	0.98	0.98
Dataset size:560000	0.95	0.95	0.95

Table 4 illustrates that the LSTM network performs well using Disk-Only storage, achieving high precision, recall, and F1-score. However, there is a noticeable decrease in performance as the dataset size increases, indicating that Disk-Only storage may not be as efficient or effective for larger datasets compared to Memory-Only storage. This underscores the importance of selecting appropriate storage configurations to optimize model performance in big data environments.

Table 5. Performance Results of Memory-Disk

	Precision	Recall	F1-Score
Dataset size:280000	1	1	1
Dataset size:560000	0.99	0.99	0.99

Table 5 demonstrates that the LSTM network achieves excellent performance with the Memory-Disk storage level, maintaining high precision, recall, and F1-score across different dataset sizes. While perfect scores are attained for the smaller dataset, there is only a slight decrease in metrics for the larger dataset, highlighting the effectiveness and robustness of Memory-Disk storage in handling larger datasets. This underscores the importance of selecting appropriate storage configurations to optimize LSTM training and performance in big data environments.
In the evaluation with persistent storage level: memory-disk, the proposed sampling algorithm provides the more accurate results. Therefore, more accurate results are provided by the proposed sampling algorithm on three persistent levels of storage: memory, disk, and memory-disk. Table 5 presents the training time of three storage levels on different dataset sizes with proposed sampling algorithm. It presents the training time of three storage levels on different dataset sizes with proposed sampling algorithm.

By highlighting the consistent performance metrics despite variations in dataset sizes and storage conditions, this paper emphasizes the reliability and robustness of the LSTM model. This consistency across different experimental setups underscores the model's suitability for real-world applications where dataset sizes and storage configurations may vary.

Table 6. Training Time Comparison of Three Storage Levels on Different Dataset size

Time (seconds)	Memory-Only	Disk-Only	Memory-Disk
Dataset size:280000	603	640	649
Dataset size:560000	1177	2149	1316

This table 6 compares the training times (in seconds) of Long Short-Term Memory (LSTM) networks using three different persistence storage levels: Memory-Only, Disk-Only, and Memory-Disk. The training times are evaluated across two different dataset sizes: 280,000 and 560,000 records. Memory-Only storage consistently shows the shortest training times across both dataset sizes. This indicates that training the LSTM network is most efficient when all data can be kept in memory, minimizing latency associated with data access. Disk-Only storage has the longest training times, particularly for the larger dataset.

This substantial increase in training time for the larger dataset highlights the inefficiencies introduced by relying on disk I/O operations, which are slower compared to memory access. Memory-Disk storage performs better than Disk-Only but is not as fast as Memory-Only storage. This storage level provides a compromise, offering better performance than Disk-Only by utilizing memory where possible and spilling to disk when necessary.

Therefore, Table 6 illustrates that the choice of storage level significantly impacts the training time of LSTM networks. Memory-Only storage provides the best performance, especially as dataset size increases, due to its fast data access speeds. Disk-Only storage, while potentially necessary for very large datasets that exceed memory capacity, results in considerably longer training times due to slower disk I/O operations. Memory-Disk storage offers a middle ground, balancing memory usage with disk storage, and provides a reasonable compromise in training time performance. These insights emphasize the importance of selecting an appropriate storage configuration to optimize training efficiency in big data environments.

CONCLUSION

This study presents a comparative analysis of the training time of Long Short-Term Memory (LSTM) networks on Apache Spark, focusing on three persistent storage levels: Disk-Only, Memory-Disk, and Memory-Only. The analysis utilized a credit card fraud detection dataset of varying sizes, both with and without a proposed sampling algorithm to

address class imbalance. The findings indicate that Disk-Only storage resulted in the longest training times due to slow disk I/O operations, while Memory-Disk showed moderate training times with occasional disk spills when memory was insufficient. Memory-Only storage achieved the fastest training times by keeping data entirely in memory, thus avoiding I/O delays. Training times increased with dataset size across all storage configurations, with Memory-Only handling larger datasets more efficiently. As dataset size increases and memory constraints arise, the memory-disk storage level serves as a viable alternative, balancing memory usage and computation speed. The proposed sampling algorithm balanced the dataset, leading to more efficient training and faster convergence, whereas training on the imbalanced dataset without the algorithm was slower and less efficient. The study concludes that Memory-Only storage is preferred for minimizing training times in resource-sufficient environments for this case study. Implementing the sampling algorithm is crucial for handling imbalanced datasets effectively, thereby enhancing training efficiency and model performance. Future work should explore dynamic and hybrid storage strategies to further enhance Spark's performance and scalability.

REFERENCES

[1] J. Kim, D. Lee, and C. Yoo, “Big data analytics on in-memory database for smart manufacturing”, Journal of Manufacturing Systems, 2019.

[2] R. Y. Park, J. H. Kim, J. Y. Yoo, “A hybrid machine learning approach to financial early warning systems”, Expert Systems with Applications, 2020.

[3] X. Huang, J. Liu, W. Wu, and J. Xie, “Efficient distributed deep learning using edge computing for IoT”, IEEE Internet of Things Journal, 2017.

[4] D. Jiang, Z. Xu, Z. Lin, and J. Guo, “A parallel data mining algorithm on Hadoop for big data. Journal of Parallel and Distributed Computing, 2018.

[5] M. Zaharia, M. Chowdhury, J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, USENIX NSDI, 2012.

[6] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, “Comparing boosting and bagging techniques with noisy and imbalanced data”, IEEE Transactions on Systems, Man, and Cybernetics, 2013.

[7] https://www.geeksforgeeks.org/stratified-random-sampling-an-overview/#what-is-stratified-random-sampling

[8] https://statusneo.com/solving-data-skewness-in-apache-spark-techniques-and-best-practices/

[9] https://sparktpoint.com/spark-persistence-storage-levels/

[10] https://sparkbyexamples.com/spark/spark-difference-between-cache-and-persist/

[11] https://medium.com/data-engineer/cache-vs-persist-in-apache-spark-a-detailed-comparison-cce43c529599

[12] https://www.kaggle.com/datasets/mlgulb/creditcardfraud

[13] https://medium.com/apache-spark-performance-with-caching-and-persistence