Journal of
Computer & Electrical and Electronics Engineering Sciences
Original Article
DOI: 10.51271/JCEEES-0029

Performance comparison of compression algorithms for log data in compliance with Turkish Law No. 5651

Melih Karahan, Erdal Erdal, Atilla Ergüzen
Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Kırıkkale University, Kırıkkale, Turkiye

Cite this article as: Karahan, M., Erdal E., & Ergüzen, A. (2025). Performance comparison of compression algorithms for log data in compliance with Turkish Law No. 5651. *J Comp Electr Electron Eng Sci, 3*(2), 54-60.

Received: 01.08.2025 • **Accepted**: 09.09.2025 • **Published**: 03.10.2025

ABSTRACT

Aims: The aim of this study is to evaluate the performance of various lossless compression algorithms for storing log data in compliance with Turkish Law No. 5651.

Methods: In the study, compression and decompression processes were performed on 100 MB and 10 GB of FortiGate log data that was generated synthetically from sample data using the Zstandard, Brotli, GZIP, LZ4, and Snappy algorithms. The algorithms were evaluated based on the amount of data compressed, compression ratio, time, speed, decompression time, and speed.

Results: The experimental studies reveal that the Zstd algorithm provides a balanced performance between compression ratio and speed. While the Brotli algorithm provides the highest compression ratio, it has the longest compression time and the slowest compression speed. The LZ4 and Snappy algorithms, on the other hand, have provided the best results in terms of compression speed and time, but their compression rates have lagged behind those of other algorithms.

Conclusion: The study results show that compression algorithms reveal significant differences in terms of performance and efficiency in large-scale log systems. The selection of the appropriate algorithm should be determined based on criteria such as the system's speed, storage requirements, and processing time. The findings obtained reveal that compression strategies play a critical role in fulfilling legal log retention obligations.

Keywords: Turkish Law No. 5651, big data, data storage, lossless compression, firewall logs

INTRODUCTION

Web 2.0 technology has significantly changed the way businesses, people, and communities interact through web-based environments and digital technology. One of the most important features of Web 2.0 technology is that it enables the widespread distribution of user-generated material. Additionally, the features associated with this technology have introduced various complexities. These features have led to new discussions about issues such as the legal management of user-generated materials, the protection of data privacy, and the oversight of produced content (Wang et al., 2024).

Data protection and privacy regulations (DPPR) have become widespread and continue to be developed worldwide. However, when the concept of the Internet and privacy is discussed, it becomes difficult to evaluate this issue as a whole and to manage it with a single law. Therefore, the laws on this subject have been classified under five main headings by the United Nations Conference on Trade and Development (UNCTAD). 195 countries around the world; 90% of countries have e-transactions laws, 79% of countries have data protection and privacy laws, 90% of countries have cybercrime laws, 60% of countries have consumer protection laws, and 50% of countries have indirect taxation laws (Unctad, 2025a). The laws and their respective status worldwide are shown in Table 1.

Table 1. Legal adoption status of cyberlaw areas worldwide

Law area	Legislation (%)	Draft legislation (%)	No legislation (%)	No data (%)
Electronic transactions	90	2	7	1
Data protection and privacy	79	3	17	1
Cybercrime	90	3	7	1
Consumer protection	60	2	29	9
Indirect taxation	50	0	50	0

Data sourced from Global Cyberlaw Tracker by UNCTAD (2025)

Turkiye's situation; when all these processes are examined for Turkiye, legislation exists for all the mentioned areas (Unctad, 2025b). The laws enacted for each aspect of the law in Turkiye and their details are presented in Table 2.

Although the five legal regulations presented in Table 2 cover the five specific topics identified by UNCTAD, they are insufficient for the collection and recording of Internet data. Therefore, a new law titled "Regulation on the procedures and principles regarding the regulation of publications made

Corresponding Author: Melih Karahan, karahanmelih@yahoo.com

Table 2. Laws adopted for each specific cyberlaw area in Turkiye

Law area	Name of law	Law no	Date
E-transactions	Electronic signature law	5070	23.01.2004
Data protection and privacy	Personal data protection law	6698	24.03.2016
Consumer protection	Law on consumer protection	6502	07.11.2013
Cybercrime	Turkish penal code Law	5377	29.06.2005
Indirect taxation	Communique No. 17 on VAT	3065	31.01.2018

Data sourced from Cyberlaw Tracker: the case of Turkiye by UNCTAD (2025)

on the internet environment" (Law No. 5651) was enacted (Republic of Turkiye, 2007). The aim and scope of this law are to define the duties and liabilities of content, hosting, access, and mass use providers and to set the rules and procedures for addressing crimes committed on the Internet via these providers.

Under the specified law, content may be removed from websites, and, if necessary, internet traffic data may be requested from the relevant authorities. Internet traffic data includes information such as the parties involved, time, duration, type of service used, amount of data transferred, and connection points related to Internet access. Article 6 further stipulates that access providers are required to store the specified traffic data for a minimum of six months and a maximum of two years and to ensure the accuracy, integrity, and confidentiality of this data. However, as these logs accumulate rapidly, especially in big data environments, they pose significant challenges in terms of storage, retrieval, and cost efficiency.

Data compression plays a crucial role in managing large-scale log repositories by significantly reducing disk usage while retaining valuable information for subsequent analysis and security audits. Selecting the right compression strategy directly impacts both storage efficiency and the operational costs associated with long-term log data retention (Li et al., 2024). Various compression algorithms, such as GZIP, LZ4, Snappy, Zstandard (Zstd), and Brotli, have been extensively studied for their effectiveness in general-purpose data compression. These algorithms are also widely adopted in key–value store systems to handle large volumes of unstructured data efficiently (Jaranilla and Choi, 2023). However, the performance of these algorithms on log data generated under specific legal constraints, such as those imposed by Turkish Law No. 5651, has not been sufficiently explored.

Literature Review

Big data: The concept of big data describes large data sets that are difficult or nearly impossible to manage using traditional analysis and data storage techniques (Raban and Gordon, 2020). The Big Data concept is frequently described using the four Vs: velocity, variety, volume, and veracity (Butts-Wilmsmeyer et al., 2020). However, the big data concept has recently been characterized by 7V (Moharm, 2019), as detailed in Table 3.

Big data storage: Although database systems continue to evolve in the big data era, the growing data volume generates challenges for many industries. When data sizes reach petabyte levels, databases experience performance and efficiency problems, despite optimization techniques (Chen et al., 2022).

Table 3. Big data 7Vs and details

Character	Basic description
Value	The valuable information when the data is properly analyzed
Volume	The actual amount of data collected and stored
Variety	The data is in an unstructured form or a mixture of structured and unstructured data
Velocity	New data values are generated at frequent intervals, usually in a constant flow
Veracity	The obtained data be healthy, that is, the uncertainty in its accuracy
Visualization	The visualization and readability of this obtained data
Variability	The variability of this data obtained

Data sourced from Moharm, K. (2019)

Traditional storage methods are inadequate in terms of performance and scalability. Therefore, various data storage technologies and architectures have been reshaped according to the needs of big data (Theodorakopoulos et al., 2024).

Across these technologies, Hadoop represents an open-source version of the cloud computing infrastructure developed by Google. Hadoop Distributed File System (HDFS) is a distributed file system that runs on standard hardware. The number of clusters can be increased as needed to meet the application requirements. Data are stored in block-structured small files. Its primary advantages include high scalability, reliability, and fault tolerance (Jing et al., 2018).

Additionally, while NoSQL does not have a precise definition, it refers to a database paradigm that offers non-relational solutions and avoids SQL (de Oliveira et al., 2021). There are four types of NoSQL approaches: Document-based, which stores each piece of data in an independent document. This structure is especially used for accessing, storing, and processing semi-structured data (e.g., MongoDB). Key-value-based, where each piece of data is kept in the form of a unique key and key-value pair, and these pairs are organized through a hash table. These values can be text, numbers, HTML, XML, JSON, videos, and images (e.g., Redis). Column-based, each data item is recorded in cell structures positioned within columns. These columns are classified into groups called column families to maintain a logical structure (e.g., Apache Cassandra). In graph-based data representation and storage operations, relationships and nodes that show the interaction between data are used (Ahmed et al., 2018).

Furthermore, cloud computing is a model that enables the on-demand and convenient distribution of computing resources such as storage, servers, applications, networks, and services over the Internet, with easy management and minimal involvement from the service provider (Rashid and Chaturvedi, 2019). In the public cloud, enterprises externally store their data by using services provided by cloud storage providers (e.g., Alibaba Cloud and Amazon Web Services) without establishing their infrastructure. It is preferred because of its scalability, low cost, and flexibility. The personal cloud is a type of public cloud, but it provides services for individual users. In the private cloud, enterprises establish and manage the infrastructure within their environment, which is managed by professional IT teams. Data are completely under the organization's control and are highly secure. However, due to the high cost, this is more convenient for storing sensitive data. The hybrid cloud is a mix of private and public cloud

storage. Sensitive data is stored in the private cloud, while other data is stored in the public cloud. It offers a flexible and balanced solution for enterprises. The community cloud is designed to address the shared requirements of several enterprises within the same sector (e.g., finance). The costs of infrastructure and management duties are distributed among community members (Yang et al., 2020).

Data compression: Compression techniques are categorized into two types; lossless and lossy methods. Lossy compression refers to a technique in which some portion of the original data is lost during the process. In some cases, such as real-time video streaming, a limited amount of data loss, such as reduced graphics or color depth, can be tolerated. This type of compression is applied to multimedia formats including JPEG, MP3, MPEG, and various video formats. Lossless compression is a method that decreases the size of data without losing quality or causing damage. When the data compressed with this method is restored, the original data decompresses precisely. It is widely used in file types such as spreadsheets and text documents, where data integrity is critical. While lossless compression is very successful in terms of quality, lossy compression can reach better compression ratios. Lossless compression techniques can be commonly categorized into two types; entropy-based encoding and dictionary-based encoding (Gupta and Nigam, 2021).

The Huffman coding algorithm is one of the entropy-based compression techniques and is named after D.A. Huffman, who developed the algorithm in the 1950s. The Huffman compression algorithm forms the basis of the most common algorithms and is implemented behind many programs, such as JPEG and GZIP. When the symbols that need to be coded and the possibility of their occurrence are calculated, this algorithm minimizes the number of bits required, thus providing optimal coding. Shorter codes are assigned to appear more frequently in characters, while longer codes are assigned to appear less frequently in characters.

The algorithm is created by the combination of the lowest characters, and this process only remains until the probability of only two combined characters is formed. Consequently, the Huffman tree is generated, and the code tree yields Huffman (Baidoo, 2023).

Arithmetic coding is a near-optimal coding technique that falls into the entropy coding category. Arithmetic coding involves assigning a decimal number in the range [0, 1] to a text. Both encoding and decoding processes are based on recursive range division. The encoder takes a character string and an expected probability distribution as input and produces a compressed structure. The decoder takes the compressed structure and expected probability distribution as input and produces the original string. The expected probability distribution determines the entropy and therefore affects the compression efficiency (Liu et al., 2019).

The run length encoding algorithm, an entropy-based coding algorithm, is the most basic of data compression algorithms. In this approach, consecutive character sequences are called "runs," while all other sequences are classified as "non-runs." This algorithm handles repeated characters. For example, the string "CDCDDDDE" acts as a source for compression; the first three letters are classified as non-runs of length three, while the next four letters are defined as a run of length four due to the repetition of the D character. The main purpose of this algorithm is to detect runs in the source file and document the character and length of each run. This algorithm compresses the runs in the original source file but does not compress the non-runs (Kodituwakku and Amarasinghe, 2010).

The Lempel-Ziv algorithm, which is a dictionary-based coding algorithm, was developed by the first members of the replacement compression, Abraham Lempel and Jakob Ziv, in 1977. LZW works with practically all kinds of data as a compression technique. LZW compression creates a structure of frequently occurring strings within the compressed data, replacing the real data with pointers to this structure. The same dictionary is created during compression and decompression, ensuring that the original data is accurately restored. LZW compression does not perform text analysis. Instead, it simply adds each new character sequence encountered to a set of character structures and replaces the character with a single code. Compression starts with a "dictionary" containing all single characters indexed from 0 to 255. Afterwards, it initiates the process of expanding the dictionary during data transmission. Duplicate strings will be compressed into a single byte; hence, compression is completed (Btoush and Dawahdeh, 2018).

The LZ77 algorithm uses the probability of repeating phrases and words in a text to compress data. Repeated data is represented by a number indicating the match length and a pointer pointing to a previous location. This method is a very simple, source-neutral, and adaptable compression method. In LZ77, the dictionary is obtained from a specific part of the previously encoded data sequence. The encoder uses a sliding window system structure to scan the data sequence. This window consists of two parts: the look-ahead buffer-the new data to be encoded-and the search buffer-the previously encoded data. This method searches for the longest match with the beginning of the data, and if a match is found, it indicates this match as <length, offset, char>. If no match is found, it outputs <0.0, char>. Length and offset values are defined using specific bit lengths. Usually, the match length is 8 bits, and the offset length is between 12 and 16 bits. The compression ratio is directly affected by these limits. Since multiple symbols are represented by a single reference, compression processes are rapid. However, searching for the longest match takes a long time. The decompression process is relatively fast by directly replacing each reference (Senthil and Robert, 2011).

METHODS

Dataset Description

The datasets used in this study consist of web traffic logs from the FortiGate firewall produced in compliance with Law No. 5651. This log data explains port scanning behaviors and is taken from the port scanning example presented in the source (Security, 2023). Although synthetic data was used, it was generated entirely from real FortiGate log structures, ensuring that all records preserved the exact format and field patterns of operational data. Log records include fields such as timestamp, source and destination IP addresses, requested URL, response size, HTTP method, and status code.

The datasets, the 10 GB dataset and the 100 MB dataset, have been created in plain text (TXT) format, with each line representing an HTTP request. These data were synthetically generated using the Java programming language and do not contain any personal information. In the synthetic data

generation process, fields such as srcip, dstip, srcport, dstport, duration, sentbyte, rcvdbyte, service, sessionid, and time were created randomly but within meaningful ranges, referencing the FortiGate device log structure. In order to replicate real network traffic, timestamps have been generated in a sequential and consistent manner; port and IP information have been configured according to the dynamics of internal and external networks. Some fixed identifier fields (devid, vd, policytype, etc.) have been used without modification. All compression algorithms were run on the same datasets under the same conditions.

Compression Algorithms

Zstandard (Zstd): Zstd is a lossless data compression algorithm developed by Meta. It can operate at fast execution speeds and high compression ratios (Facebook, 2025a).

In this study, the open-source library zstd-jni was used for compression and decompression via Java Maven. The code structure of Zstd includes both Huffman and FSE (Finite State Entropy)-based entropy structures. There are compression levels from 1 to 22. The default compression level of 3 was used in this study. Higher levels provide a higher compression ratio, while lower levels allow for faster results (Facebook, 2025a; Facebook, 2025b). Additionally, Zstd has dictionary-based compression support that generates efficient results in compressing repetitive data structures or small files (Facebook, 2025b). Zstd's compression format supports streams, allowing it to compress data without storing it in memory and to process it in parts. This functionality makes Zstd useful for applications requiring real-time data streaming or low-memory systems (Facebook, 2025a).

Brotli: Brotli is a modern algorithm developed by Google that provides lossless data compression. It uses dictionary-based (LZ77), entropy-based (Huffman) approaches, and second-order contextual modeling together. Brotli divides the text into small blocks, and these divided blocks are compressed separately (Prayogo et al., 2024). Brotli's encoding structure consists of dynamic Huffman coding, a static dictionary, context modeling, and a sliding window algorithm. Huffman tables are dynamically generated based on the compressed data. Brotli can also provide highly efficient results, especially for small-sized files and frequently repeated data structures, thanks to its support for static dictionaries (IETF, 2016).

The Brotli algorithm offers compression levels ranging from 0 to 11. Low levels (0-5) provide rapid compression; high levels (6-11) offer higher compression rates but increase processing (Google, 2025a). In this study, the open-source library brotli4j was used for compression and decompression via Java Maven. In the compression processes, the default compression level for Brotli, which is 5, was used.

GZIP: ZIP is an open-source algorithm used for file compression. This approach, created by Mark Adler and Jean-loup Gailly, is based on the LZ77 algorithm, which results in lossless data compression. Adding Huffman coding improves the results. GZIP divides the data into blocks, and each block is encoded separately. This format provides effective results, especially in text files. The GZIP compressed file contains not only the compressed data but also the filename and the date and time information (GNU Project, 2025; Syed and Soomro, 2018).

In this study, java.util.zip, which is part of the Java SDK, was used for compression and decompression operations in the Java environment without requiring external dependencies. The GZIP algorithm offers compression levels ranging from 1 to 9. Low levels (1-3) offer rapid compression but with a low compression ratio; high levels (4-9) provide slow compression but with a high compression ratio (Syed and Soomro, 2018). The default compression level has been used (this level is 6).

LZ4: LZ4 is a dictionary-based data compression technique that operates on the principle of identifying repeating trends in data sets and encoding them with the references to previous locations. This technique provides remarkable compression efficiency, especially in large datasets. A key characteristic of the algorithm is that it offers the user flexibility between compression ratio and speed. Depending on the application requirements, the user can choose to increase the compression speed at the expense of the compression ratio, or vice versa. As a member of the LZ4 family, LZ4HC (High Compression) offers better compression ratios by using higher processing power. This variant is particularly suitable for scenarios where compression efficiency is more important than speed (Jaranilla and Choi, 2023).

In this study, the open-source Java library lz4-java was used for the LZ4 algorithm. No adjustments were made regarding compression; it was used as is. Compression and decompression operations have been carried out via Maven.

Snappy: Snappy is a fast and lightweight data compression library developed by Google and offered as open source on GitHub. This algorithm is designed with a focus on speed rather than a high compression ratio (Google, 2025b). While working with dictionary-based techniques based on LZ77, it also includes some Huffman coding elements. Each compressed block may optionally contain a checksum for verification purposes. One of the most important advantages of Snappy is its outstanding performance in both compression and decompression processes (Jaranilla and Choi, 2023).

In this study, the Snappy algorithm was implemented using the open-source official library snappy-java integrated with the Java language. No special configuration was made for the compression processes; the default settings were used. Compression and decompression operations were performed using Maven.

All compression algorithms in this study were used with their default settings (Zstd at level 3, Brotli at level 5, GZIP at level 6, and no manual tuning for LZ4/Snappy). This choice ensures a fair comparison and avoids potential imbalances that could arise from using different tuning levels. Although higher or lower compression levels could reveal additional trade-offs between speed and compression ratio, the purpose of this study was to evaluate baseline performance under standard conditions.

RESULTS

Experimental Settings

All experiments were performed on a machine operating Windows 10 (22H2) with an intel (R) core (TM) i7-7500U CPU [2.70 gigahertz (GHz)], 16 gigabytes (GB) of RAM, and 256 GB solid-state drive (SSD) storage. The algorithms were executed using IntelliJ IDEA 2024 (community edition 2.1) with Java version 21.0.7. For compilation and configuration,

the maven-compiler-plugin version 3.11.0 was used, along with the central repository at repo1.maven.org. The compression libraries utilized in this study are zstd-jni version 1.5.7-3 for Zstandard, brotli4j version 1.18.0 for Brotli, java.util.zip.* from JDK 21 for GZIP, lz4-java version 1.8.0 for LZ4, and snappy-java version 1.1.10.1 for Snappy.

Dataset Specifications

The format of each log record is plain text, containing space-separated key-value pairs. For the experimental studies, synthetic datasets containing 172,270 log records (100 megabytes) and 17,639,965 log records (10 gigabyte) were utilized.

The content of the log records created by modification includes the following: The main time information includes codedate (YYYY-MM-DD) and time (HH:MM:SS), representing the log's creation timestamp, and it has been created to simulate a time distribution of approximately 27.7 hours (up to 100,000 seconds before creation). Device-specific identifiers, such as devname (e.g., FORTIGATE-02132, containing a random 5-digit number) and devid (e.g., FGT5510460331346, containing a random 13-digit number), uniquely identify the logging device. Each record also contains a unique logid.

Network addresses and ports are detailed with srcip and dstip (randomly generated from the internal ranges 192.168.x.x and the private range 10.0.x.x) and srcport and dstport (randomly generated in the range 1024-61023). Interface information is provided by randomly selecting srcintf and dstintf from internal, dmz, wan1, and vpn. Geographic contexts are captured with dstcountry and srccountry randomly selected from reserve, Turkiye, Germany, the USA, and the United Kingdom. The operating system version (OSVersion) is randomly selected from Microsoft Windows 10, Windows Server 2016, Linux, and Ubuntu 20.04. Finally, to complete the network context, MAC addresses (mastersrcmac, srcmac, dstmac) are generated randomly.

Experimental Results

This section presents the results of a comparison of five compression algorithms; Zstandard (Zstd), Brotli, GZIP, LZ4, and Snappy. The evaluation focuses on six performance metrics: compressed output size, compression ratio, compression time, decompression time, compression speed, and decompression speed. The experiments were conducted using synthetic FortiGate log files with sizes of 100 MB and 10 GB. Table 4, 5, and Figure display the results.

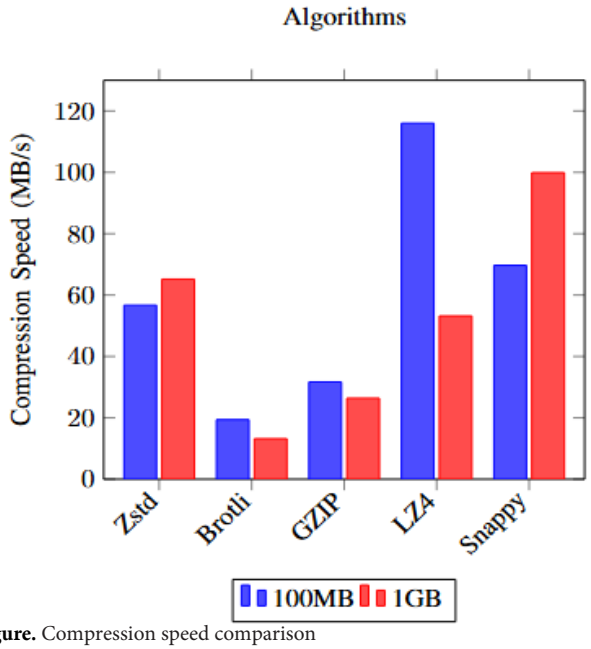

Figure. Compression speed comparison

Blue bars represent the 100 MB log dataset; red bars represent the 1 GB log dataset

DISCUSSION

In this study, the performances of different compression algorithms on 100MB and 10GB sized log data were compared. The results obtained have shown that the algorithms provide similar compression rates regardless of the data size, but their processing times and speeds vary according to the data size. Although including system performance metrics such as CPU and memory usage would have made the study stronger, the datasets used were archival data, and the primary focus of this study was on compression performance; therefore, CPU and memory usage metrics were considered out of scope.

Table 4. Compression performance results for the 100MB FortiGate log dataset

Algo	Comp size (MB)	Comp ratio (%)	Comp time (ms)	Decomp time (ms)	Comp speed (MB/s)	Decomp speed (MB/s)
Zstd	15.10	84.9	1765	541	56.7	184.8
Brotli	13.51	86.5	5142	454	19.4	220.3
GZIP	14.53	85.5	3154	1100	31.7	90.9
LZ4	25.68	74.3	862	373	116.0	268.1
Snappy	28.60	71.4	1435	475	69.7	210.5

Algo: Algorithm, Comp: Compression, Decomp: Decompression, MB: Megabyte, %: Percentage, ms: Milliseconds, MB/s: Megabyte per second

Table 5. Compression performance results for the 10GB FortiGate log dataset

Algo	Comp size (MB)	Comp ratio (%)	Comp time (ms)	Decomp time (ms)	Comp speed (MB/s)	Decomp speed (MB/s)
Zstd	1547.67	84.9	106463	53267	96.2	192.2
Brotli	1383.79	86.5	350703	49386	29.2	207.3
GZIP	1488.24	85.5	248491	79719	41.2	128.5
LZ4	2631.93	74.3	48950	57994	209.2	176.6
Snappy	2930.79	71.4	48476	58682	211.2	174.5

Algo: Algorithm, Comp: Compression, Decomp: Decompression, MB: Megabyte, %: Percentage, ms: Milliseconds, MB/s: Megabyte per second

The compression ratios obtained from Table 4 and 5 show significant differences in the data compression capabilities of the algorithms. In both datasets, the Brotli algorithm achieved the highest compression ratio (86.5%), followed by the GZIP (85.5%) and Zstd (84.9%) algorithms. These rates indicate that the compression algorithms provide similar levels of efficiency regardless of the data size. On the other hand, LZ4 and Snappy achieved lower compression rates (74.3% and 71.4%). However, the main advantage of these algorithms lies in their processing speeds.

Compression time and speed are important metrics in systems with real-time log processing or high data throughput. Table 4, 5, and Figure demonstrate the high compression speeds offered by the LZ4 and Snappy algorithms. In the 100 MB dataset, LZ4 achieved the fastest compression speed of 116 MB/s, followed by Snappy with 69.7 MB/s. Additionally, despite Zstd (56.7 MB/s), Brotli (19.4 MB/s), and GZIP (31.7 MB/s) having high compression ratios, their compression speeds remain slow.

In a 10 GB dataset size, there has been a decrease in the compression speeds of the algorithms, and the compression times have increased. Snappy achieved 211.2 MB/s as the fastest algorithm, while LZ4's compression speed fell to third place at 209.2 MB/s. Brotli, although it has a high compression ratio, is considered the slowest algorithm. It has 100 MB of data compressed in a time of approximately 5.1 seconds and around 350.7 seconds for 10 GB of data.

When evaluated in terms of decompression times, LZ4 achieved the fastest decompression with 268.1 MB/s (100 MB data) and 176.6 MB/s (10 GB data). Snappy, Brotli, and Zstd have also demonstrated high-speed decompression performance. However, GZIP has a low decompression speed in both datasets.

According to these results, LZ4 and Snappy are suitable for applications that require high speed despite low compression ratios. Zstd has been the most balanced algorithm in terms of speed and compression ratio. Brotli is well-suited for situations that require high-density compression and are not time-sensitive. GZIP has not been able to provide superiority in either speed or compression ratio.

Limitations

This study evaluated the performance of compression algorithms using raw log data.

Due to legal restrictions in Turkiye, the use of real production data was not possible. However, when deployed in a production environment, the same compression methods will be applied directly to real log streams. The experiments were conducted on datasets of 100 MB and 10 GB to ensure manageable yet non-trivial test sizes; however, extending the evaluation to larger datasets (e.g., 1 TB, 100 TB) would provide stronger evidence of scalability and improve the generalizability of the results. In future work, testing with real operational log data (once legally permissible) and integrating benchmark studies or industrial application reports will further validate the findings and strengthen the practical relevance of this research.

CONCLUSION

This research assessed five lossless compression algorithms (Zstd, Brotli, GZIP, LZ4, and Snappy) were evaluated for the purpose of compressing log data that must be retained under Turkish Law No. 5651. Synthetic log datasets of 100MB and 10GB were used to compare the compression ratio, compression time and speed, and decompression time and speed metrics of each algorithm.

The results indicate that Zstd provides an optimal balance between compression ratio and speed, making it a strong candidate for general-purpose log compression tasks. Brotli achieved the highest compression ratio for both datasets but incurred significant processing time, making it more suitable for archival datasets where time is not critical. Contemporary alternatives are gradually approaching GZIP, which displayed modest performance in both compression ratio and speed. LZ4 and Snappy offered the fastest compression and decompression speeds, which makes them appropriate for real-time or high-throughput logging systems, albeit at the cost of lower compression ratios.

As a result of these comparisons, it has been observed that the algorithm to be used for compressing logs should be selected according to the system's needs. If storage space is limited, algorithms with a high compression ratio should be used; however, if data needs to be processed and updated in real-time, speed-priority algorithms should be used. As the data size increases, the performance difference between algorithms becomes more noticeable. Therefore, general solutions do not provide the desired performance in the large datasets produced as a result of Law No. 5651. Because obligations of 5651 introduce design constraints that are not typically addressed in general compression studies: (i) strictly lossless and deterministic decoding across software versions; (ii) efficient, auditable retrieval of narrow time windows; and (iii) the ability to deliver legally requested periods as single-file archives without sacrificing verifiability. Evaluating compression in this legal–operational setting therefore requires re-examining algorithm trade-offs with compliance-aligned metrics rather than generic throughput alone. For this reason, the necessity of developing a compression algorithm for texts at higher rates has been identified and targeted as future work. In addition, Hadoop and AWS have been considered as part of the planned future work to provide an optimized solution specifically addressing this problem.

ETHICAL DECLARATIONS

Referee Evaluation Process

Externally peer-reviewed.

Conflict of Interest Statement

The authors have no conflicts of interest to declare.

Financial Disclosure

The authors declared that this study has received no financial support.

Author Contributions

All of the authors declare that they have all participated in the design, execution, and analysis of the paper, and that they have approved the final version.

Acknowledgments

The author acknowledges the assistance of the QuillBot tool, which was used for improving the clarity and fluency of the English in this manuscript.

REFERENCES

Abdulazeez, F. A., Ahmed, I. T., & Hammad, B. T. (2024). Examining the performance of various pretrained convolutional neural network models in malware detection. *Appl Sci, 14*(6), 2614. doi:10.3390/app14062614

Ahmed, M. R., Khatun, M. A., Ali, M. A., & Sundaraj, K. (2018). A literature review on NoSQL database for big data processing. *Int J Eng Technol, 7*(2), 902-906. doi:10.14419/ijet.v7i2.12113

Almomani, I., Alkhayer, A., & El-Shafai, W. (2022). An automated vision-based deep learning model for efficient detection of android malware attacks. *IEEE Access, 10*, 2700-2720. doi:10.1109/ACCESS.2022.3140341

Baidoo, P. K. (2023). Comparative analysis of the compression of text data using Huffman, arithmetic, run-length, and Lempel Ziv Welch coding algorithms. *J Adv Mathemat Comput Sci, 38*(9), 144-156. doi:10.9734/jamcs/2023/v38i91812

Btoush, M. H., & Dawahdeh, Z. E. (2018). A complexity analysis and entropy for different data compression algorithms on text files. *J Comput Communicat, 6*(1), 301. doi:10.4236/jcc.2018.61029

Butts-Wilmsmeyer, C. J., Rapp, S., & Guthrie, B. (2020). The technological advancements that enabled the age of big data in the environmental sciences: a history and future directions. *Curr Opin Environment Sci Health, 18*, 63-69. doi:10.1016/j.coesh.2020.07.006

Chen, Y., Zhang, F., Hong, Y., et al. (2022). Taming the big data monster: managing petabytes of data with multi-model databases. Proceedings of the 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (pp. 283-292). IEEE.

de Oliveira, V. F., Pessoa, M. A. O., Junqueira, F., & Miyagi, P. E. (2021). SQL and NoSQL databases in the context of Industry 4.0. *Machines, 10*(1), 20. doi:10.3390/machines10010020

Facebook. (2025a). Zstandard (Zstd) [Computer software]. GitHub. Retrieved June 20, 2025, from https://github.com/facebook/zstd

Facebook. (2025b). *Zstandard API manual*. Retrieved June 20, 2025, from https://facebook.github.io/zstd/doc/api_manual_latest.html#Chapter1

Google. (2025a). Brotli compression tool manual page (brotli.1). Retrieved June 20, 2025, from https://github.com/google/brotli/blob/master/docs/brotli.1

Google. (2025b). Snappy - a fast compressor/decompressor [computer software]. GitHub. Retrieved June 20, 2025, from https://github.com/google/snappy

GNU Project. (2025). Gzip [Software]. GNU Operating System. Retrieved June 20, 2025, from https://www.gnu.org/software/gzip/manual/gzip.html

Gupta, A., & Nigam, S. (2021). A review on different types of lossless data compression techniques. *Int J Sci Res Comput Sci, 7*(1), 50-56. doi:10.32628/cseit217113

IETF. (2016). Brotli compressed data format (RFC 7932). https://datatracker.ietf.org/doc/html/rfc7932

Jaranilla, C., & Choi, J. (2023). Requirements and trade-offs of compression techniques in key-value stores: a survey. *Electronics, 12*(20), 4280. doi:10.3390/electronics12204280

Jing, W., Tong, D., Chen, G., Zhao, C., & Zhu, L. (2018). An optimized method of HDFS for massive small files storage. *Comput Sci Informat Syst, 15*(3), 533-548. doi:10.2298/CSIS170926034J

Kodituwakku, S. R., & Amarasinghe, U. S. (2010). Comparison of lossless data compression algorithms for text data. *Indian J Comput Sci Eng, 1*(4), 416-425.

Republic of Turkiye. (2007). Law No. 5651: Regulation on the procedures and principles regarding the regulation of publications made on the internet environment. https://www.mevzuat.gov.tr/mevzuat?MevzuatNo=5651&MevzuatTur=1&MevzuatTertip=5

Li, X., Zhang, H., Le, V. H., & Chen, P. (2024, February). Logshrink: Effective log compression by leveraging commonality and variability of log data. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (pp. 1-12).

Liu, Q., Xu, Y., & Li, Z. (2019, February). DecMac: a deep context model for high efficiency arithmetic coding. In 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC) (pp. 438-443). IEEE.

Moharm, K. (2019). State of the art in big data applications in microgrid: A review. *Adv Eng Informat, 42*, 100945. doi:10.1016/j.aei.2019.100945

Prayogo, A. I., Nugraha, A., Kurniawan, J. C., Kusuma, E. J., & Wibowo, A. (2024). Enhancing least significant bit steganography image fidelity using Brotli compression. *Sinkron J Res Informat Technol, 8*(1), 285-295. doi:10.33395/sinkron.v9i1.13186

Raban, D. R., & Gordon, A. (2020). The evolution of data science and big data research: a bibliometric analysis. *Scientometrics, 122*(3), 1563-1581. doi:10.1007/s11192-019-03332-4

Rashid, A., & Chaturvedi, A. (2019). Cloud computing characteristics and services: a brief review. *Int J Comput Sci Eng, 7*(2), 421-426. doi:10.26438/ijcse/v7i2.421426

Security, R. (2023). Detecting attacks using Fortigate firewall logs with log samples [Blog post]. Medium. Retrieved June 20, 2025, from https://readsecurity.medium.com/detecting-attacks-using-fortigate-firewall-logs-with-log-samples-ad0e7ee57903

Senthil, S., & Robert, L. (2011). Text compression algorithms: a comparative study. *J Commun Technol, 2*(4), 444-451. doi:10.21917/ijct.2011.0062

Syed, Z., & Soomro, T. (2018). Compression algorithms: Brotli, Gzip and Zopfli perspective. *Indian J Sci Technol*, *11*(45), 1-4. doi:10.17485/ijst/2018/v11i45/117921

Theodorakopoulos, L., Theodoropoulou, A., & Stamatiou, Y. (2024). A state-of-the-art review in big data management engineering: real-life case studies, challenges, and future research directions. *Eng, 5*(3), 1266-1297. doi:10.3390/eng5030071

UNCTAD. (2025a). Summary of adoption of e-commerce legislation worldwide. United Nations Conference on Trade and Development. Retrieved June 20, 2025, from https://unctad.org/topic/ecommerce-and-digital-economy/ecommerce-law-reform/summary-adoption-e-commerce-legislation-worldwide

UNCTAD. (2025b). Cyberlaw tracker - Turkiye. United Nations Conference on Trade and Development. Retrieved June 20, 2025, from https://unctad.org/page/cyberlaw-tracker-country-detail?country=tr

Wang, S., Jiang, X., & Khaskheli, M. B. (2024). The role of technology in the digital economy's sustainable development of Hainan Free Trade Port and genetic testing: cloud computing and digital law. *Sustainability, 16*(14), 6025. doi:10.3390/su16146025

Yang, P., Xiong, N., & Ren, J. (2020). Data security and privacy protection for cloud storage: a survey. *IEEE Access, 8*, 131723-131740. doi:10.1109/ACCESS.2020.3009876