

Adaptive Data Compression Framework for Network Transmission Optimization Based on Entropy and Bandwidth Analysis

Liubov Oleshchenko*

Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics, Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine

E-mail: oleshchenkoliubov@gmail.com

ORCID iD: <https://orcid.org/0000-0001-9908-7422>

*Corresponding author

Zhengbing Hu

School of Computer Science, Hubei University of Technology, Wuhan, 430079, China

E-mail: drzbhu@gmail.com

ORCID iD: <https://orcid.org/0000-0002-6140-3351>

Andrii Dychka

Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics, Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine

E-mail: andriydychka@gmail.com

ORCID iD: <https://orcid.org/0000-0003-0578-2788>

Received: March 26, 2026; Revised: April 16, 2026; Accepted: April 26, 2026; Published: August 8, 2026

Abstract: This paper addresses the problem of efficient data transmission under dynamically changing network and computational conditions by proposing an adaptive data compression method based on context-aware selection of compression algorithms and their parameters. Unlike conventional static approaches, the proposed method performs real-time analysis of data characteristics, network bandwidth, latency, and available computational resources, enabling dynamic selection of the optimal compression strategy through multi-criteria optimization. The scientific novelty of the work lies in the integration of data-driven and environment-aware adaptation within a unified decision-making framework that simultaneously minimizes end-to-end transmission delay while balancing compression ratio and processing overhead. Experimental evaluation was conducted using both synthetic datasets and the standard Silesia Corpus benchmark. The synthetic datasets included repetitive low-entropy text (repeated.txt), structured JSON data (structured.json), moderate-complexity text (example.txt), and high-entropy binary streams (random.bin), representing realistic web content and raw data transmission scenarios. The Silesia Corpus, containing approximately 200 MB of heterogeneous real-world files, including text, binaries, and images, was used for validation and benchmarking. The proposed method was evaluated using compression algorithms such as LZ4, Zstandard, Brotli, and ZSTD under different network conditions and system loads. Experimental results show that the adaptive approach reduces total transmission time by an average of 23%, improves compression efficiency by 16%, and decreases computational resource consumption by 13% compared to conventional static compression methods. The software implementation is based on a modular service-oriented architecture that supports real-time monitoring, dynamic algorithm switching, and scalable deployment in distributed, cloud, streaming, and Internet of Things environments.

Index Terms: Adaptive Data Compression Framework, Entropy, Compression Algorithms, LZ4, ZSTD, Brotli, Software Implementation, Real-Time Data Analysis

1. Introduction

The rapid growth of data volumes in modern digital systems creates significant challenges for efficient data transmission. The exponential increase in network traffic, driven by cloud computing, streaming services, and Internet

of Things (IoT) applications, leads to increased latency, higher operational costs, and reduced quality of service. In this context, data compression plays a critical role in optimizing network performance. The foundations of modern data compression were established in classical works on information theory and redundancy reduction. The universal compression algorithm proposed by J. Ziv and A. Lempel [1] introduced the concept of dictionary-based encoding, which became the basis for many subsequent methods. Further development of this approach led to the LZW algorithm proposed by T. Welch [2], which significantly improved compression efficiency in practical applications. In parallel, alternative approaches such as block-sorting transformations were introduced, notably the Burrows–Wheeler transform [3], which enabled more effective preprocessing of data prior to entropy coding. These theoretical and algorithmic developments formed the basis for widely used compression standards. The DEFLATE algorithm [4], combining LZ77 and Huffman coding, became one of the most common methods for lossless compression. Later, more advanced compression systems were developed, including JPEG2000 for image compression [5] and comprehensive frameworks described in classical handbooks [6]. Despite their widespread adoption, these algorithms operate with predefined parameters and static configurations. As described in the paper [8], traditional compression techniques are not designed to adapt to dynamic changes in data characteristics or network conditions. This limitation becomes critical in modern distributed systems, where data streams are heterogeneous and network environments are highly variable.

Recent research demonstrates a shift toward adaptive compression methods that dynamically adjust their behavior based on data properties and system constraints. Comprehensive surveys highlight that modern compression systems increasingly rely on adaptive, data-aware, and context-sensitive approaches [10]. In particular, adaptive compression methods for stream processing enable real-time adjustment of compression strategies, significantly improving performance in dynamic environments [11]. Further advancements include adaptive compression techniques for distributed intelligent systems, where compression parameters are adjusted based on information constraints [12]. In addition, entropy-based approaches such as relative entropy coding provide more efficient data representation by dynamically modeling probability distributions [13]. High-performance compression architectures based on probabilistic models and parallel dataflow further improve scalability and processing speed [14].

The integration of machine learning techniques into compression systems has opened new prospects for improving compression efficiency. Neural and hybrid models enable adaptive feature extraction and compression for complex data types, including multimedia and communication signals [15–17]. Despite significant progress, existing methods still do not fully address the problem of optimizing data transmission under varying network conditions. Most approaches focus on either data characteristics or compression efficiency, while insufficient attention is given to the joint consideration of entropy, bandwidth, and system resources. Thus, the development of an adaptive data compression method that integrates entropy analysis with network-aware optimization remains an important research problem.

The goal of this article is to develop a method and software for adaptive data compression that dynamically selects optimal compression strategies based on data characteristics, entropy properties, and network conditions to improve transmission efficiency.

1.1. Problem Statement

Given a stream of data packets or data blocks D_k transmitted over a network with time-varying characteristics, the task is to determine an optimal compression strategy for each data segment in real time.

The inputs to the system include estimated data entropy $H(D_k)$, available bandwidth $B(t)$, network latency $L(t)$, system resource utilization $R(t)$ (CPU, memory), predefined constraints such as latency thresholds and computational budgets. The expected outputs are selected compression mode $C_k \in \{\text{no compression, lightweight, high-ratio}\}$, estimated transmission time T_k , resulting compression ratio ρ_k . The system must operate under streaming conditions, ensuring bounded decision latency (on the order of milliseconds), reproducibility of decision logic, and adaptability to rapidly changing network environments. Additionally, the framework should minimize unnecessary computation while maintaining acceptable transmission efficiency.

1.2. Motivation

The growing need for efficient data transmission in distributed and real-time systems highlights the limitations of traditional compression approaches. Most existing methods rely on static configurations or limited adaptive mechanisms, which do not fully capture the complexity of modern network environments. As a result, systems often experience inefficient trade-offs, such as excessive compression overhead under high-bandwidth conditions or insufficient compression under constrained networks. Recent research trends emphasize end-to-end optimization, where both computation and communication are considered jointly. However, practical implementations of such approaches remain limited, particularly in scenarios requiring real-time decision-making and deployment-ready architectures.

This work is motivated by the need to bridge the gap between theoretical adaptive compression models and practical system-level implementations. The proposed framework addresses this gap by combining data-driven decision-making with system-aware optimization, enabling more efficient utilization of network and computational resources. This paper addresses these challenges by proposing a context-aware adaptive compression framework that integrates multiple decision factors within a unified control loop. The framework combines entropy-based data analysis, real-time network monitoring, and resource-aware optimization to dynamically select compression strategies. Unlike

purely algorithmic contributions, the proposed approach is primarily system-oriented, focusing on the integration and orchestration of existing techniques into a coherent and deployable architecture.

The main idea of the proposed framework is to model compression selection as a multi-criteria decision problem, where the system continuously evaluates trade-offs between compression ratio, transmission delay, and computational cost. The decision module operates on a set of measurable inputs, including estimated data entropy, available bandwidth, network latency, and system resource utilization. Based on these inputs, the framework dynamically selects one of several compression modes, including no compression, lightweight compression, or high-ratio compression.

The presented software architecture is implementation-oriented and supports deployment in cloud, edge, and hybrid environments. The system is designed as a modular pipeline consisting of data acquisition, feature extraction, decision-making, and compression execution components, enabling scalability and extensibility in real-world scenarios.

The main contributions of this paper are fourfold. First, it introduces a multi-parameter decision model for adaptive compression that jointly considers entropy, bandwidth, latency, and computational resources within a single control loop. Second, it proposes a context-aware decision module that dynamically selects compression strategies based on real-time system observations. Third, it presents a modular software architecture suitable for deployment in distributed environments, including edge and cloud systems. Fourth, it provides an experimental evaluation framework that analyzes the trade-off between compression ratio and transmission delay under varying simulated conditions.

2. Related Work

Data compression is a fundamental technique aimed at reducing the size of digital data by eliminating redundancy while preserving essential information. Classical approaches to data compression are based on theoretical principles of information entropy and redundancy reduction, as described in foundational works [6], [8]. Early compression methods were primarily based on dictionary-based encoding techniques. The Lempel–Ziv algorithms [1] and their extensions, such as LZW [2], introduced adaptive dictionary construction mechanisms that allow efficient encoding of repeated data patterns. These methods became the foundation for many practical compression algorithms used today. Another important class of methods includes transformation-based techniques. The Burrows–Wheeler transform [3] enables efficient data reordering, improving the performance of subsequent entropy coding stages. These approaches are widely used in modern compression algorithms due to their ability to exploit data locality and redundancy. The combination of dictionary-based and entropy coding techniques led to the development of widely used compression standards such as DEFLATE [4], Brotli [7], and Zstandard [9]. These algorithms provide a good balance between compression efficiency and computational complexity, making them suitable for a wide range of applications, including network data transmission. Despite their effectiveness, these traditional algorithms are characterized by static configurations and predefined parameters. As a result, they are not able to adapt to changing data characteristics or dynamic network conditions. This limitation reduces their efficiency in modern distributed and heterogeneous environments. Modern developments also emphasize entropy-based approaches, which aim to approach theoretical limits of compression efficiency. Relative entropy coding methods dynamically adjust encoding strategies based on probability distributions, resulting in improved compression performance [13]. In addition, high-performance compression architectures based on probabilistic models and parallel data processing enable efficient handling of large-scale data streams [14]. These approaches are particularly relevant for modern distributed systems and cloud environments. Despite the rapid development of adaptive compression techniques, most existing methods focus primarily on data characteristics and computational efficiency, while insufficient attention is given to network conditions such as bandwidth and latency. As a result, they do not fully optimize the overall data transmission process.

Therefore, modern research highlights the need for environment-aware adaptive compression methods that integrate entropy analysis, data characteristics, and network conditions into a unified framework. This forms the basis for the proposed approach to adaptive data compression for network transmission optimization.

3. Comparative Analysis of Data Compression Algorithms

Data compression is defined as the process of reducing the size of digital information in order to optimize storage and transmission. By eliminating statistical, structural, and entropy redundancy, compression techniques significantly decrease memory usage and transmission time, thereby improving the overall efficiency of information systems.

Traditionally, compression methods are divided into lossless and lossy approaches. Lossless compression ensures complete reconstruction of the original data after decompression, which makes it suitable for applications requiring strict data integrity, such as databases, telemetry systems, and executable code. In contrast, lossy compression achieves higher compression ratios by discarding less significant information, which results in irreversible quality degradation and is therefore primarily applied to multimedia data, including images, audio, and video (Table 1).

Table 1. Comparison of Lossless and Lossy Compression Methods.

Characteristic	Lossless compression	Lossy compression
Data recovery	100%	85–99% (depending on codec and settings)
Compression ratio	1.5–3.5×	5–50×
Data quality	100% preserved	70–98% perceptual quality
Typical data types	Text, databases, code	Image, audio, video
Typical latency	0.1–10 ms/MB	1–50 ms/MB

Compression methods can be analyzed according to their level of adaptivity. Static compression algorithms operate with fixed parameters that remain unchanged throughout execution, which limits their efficiency in environments with variable data characteristics. Such methods demonstrate optimal performance only under stable and homogeneous conditions. Adaptive compression methods dynamically adjust their internal models based on statistical properties of the input data stream. This allows them to better capture local patterns and improve compression efficiency under heterogeneous and time-varying conditions. Entropy-based adaptive methods approach theoretical limits by assigning shorter codes to more frequent symbols while continuously updating probability distributions during processing.

Dictionary-based adaptive techniques rely on identifying repeating patterns and replacing them with references to dynamically maintained dictionaries, enabling efficient representation of redundant data structures. More advanced approaches extend adaptivity beyond the data itself by incorporating environmental factors. In such systems, compression strategies are selected based not only on input data characteristics but also on network conditions and available computational resources, allowing for improved overall system performance. The evaluation of compression algorithms requires a comprehensive analysis that considers multiple performance metrics. The selection of an appropriate algorithm depends on achieving a balance between compression efficiency and computational overhead (Table 2).

Table 2. Key Performance Metrics.

Metric	Typical range
Compression ratio	1.5–5× (general-purpose), up to 50× (multimedia)
Compression speed	20–800 MB/s
Decompression speed	100–4000 MB/s
CPU usage	5–95% (depending on algorithm and level)
Memory usage	1–1024 MB

Since no single algorithm provides optimal performance across all scenarios, practical implementations rely on trade-offs between speed and compression ratio (Table 3).

Table 3. Comparative Analysis of Modern Compression Algorithms.

Algorithm	Compression ratio	Compression speed (MB/s)	Decompression speed (MB/s)	CPU usage (%)	Memory usage (MB)
LZ4	1.8–2.2	400–800	2000–4000	5–20	1–16
Snappy	2.0–2.4	250–600	1000–2000	10–30	8–32
Gzip	2.5–3.0	50–150	200–500	20–60	32–128
Brotli	2.8–3.5	100–400	500–1500	40–80	64–256
Zstandard	2.5–3.5	200–600	500–2000	20–70	32–256
LZMA	3.0–4.5	10–50	100–300	70–95	128–1024

In real-world network environments, the effectiveness of compression is determined by end-to-end latency, which includes compression time, transmission delay, and decompression time. The relative contribution of each component depends on network bandwidth, latency, and system constraints. An optimal strategy requires balancing computational cost and data size reduction. In high-bandwidth environments, faster algorithms with lower compression ratios may provide better performance, whereas in constrained networks, higher compression efficiency may be preferable despite increased computational overhead.

Modern network-oriented compression solutions are predominantly proprietary and tightly integrated into cloud ecosystems. For example, Google Cloud CDN Dynamic Compression performs automatic content compression at edge nodes using Brotli and Gzip algorithms, based on client capabilities. This approach simplifies deployment but introduces dependency on a specific cloud provider and is primarily limited to text-based content.

Despite significant advances in individual compression algorithms, the problem of efficient real-time adaptive selection remains unresolved. Most solutions rely either on static rules or are optimized for specific scenarios, which

limits their applicability in dynamic environments. This highlights the need for a generalized method capable of selecting optimal compression strategies based on integrated analysis of data characteristics, network conditions, and computational resources.

4. Proposed Adaptive Data Compression Method

Analysis of modern data transmission systems has revealed a fundamental problem – most existing solutions use static compression strategies that do not adapt to real-world environmental conditions and data properties. The proposed method aims to solve this problem by developing software that dynamically selects the optimal compression strategy.

The choice of compression algorithm should not be fixed, instead the system should consider and make decisions based on the analysis of three factors: data properties – degree of compressibility, type and volume; network environment characteristics – bandwidth, latency and packet loss; available computing resources.

The goal of proposed method is to minimize the total time required for data compression, transmission, and decompression. Unlike traditional approaches, where the choice of algorithm depends mainly on the generalized compression ratio, the proposed method takes into account that often a faster algorithm, even with a worse compression ratio, can give a better overall result in conditions of limited network bandwidth (Table 4).

Table 4. Comparison of Static and Adaptive Compression.

Parameter	Static approach	Adaptive method
Algorithm selection	Fixed	Dynamic, depends on data properties
Network conditions	Not taken into account	Bandwidth, latency are taken into account
Computational costs	Low	Sample analysis costs
Computing resources	Not considered	CPU usage and available RAM are taken into account
Data analysis	Not held	Lightweight sample data analysis
Time	Significant delay with ineffective compression	Minimized latency for every scenario

Analysis of the impact of data type and characteristics is the first link in the decision-making chain. Its task is to quickly determine the key properties of the data without the need for its full compression. The efficiency of this component directly affects the overhead of the software, as much attention is paid to lightweight analysis methods.

The analysis is performed on a representative fragment of the data. The size of the representative fragment is chosen in such a way as to provide a sufficiently accurate assessment of the properties of the data, but not to exceed 1-5% of the total data block. This approach provides minimal analysis time overhead without the need to process the full size of the data for each potential strategy, while maintaining the representativeness of the results.

Based on the analysis of a representative data set, the following is determined.

1. Data entropy is a measure of randomness and uncertainty. High entropy (close to the maximum for byte-based data, i.e. ~8 bits per byte) indicates low compression potential. Low entropy (< 4-5 bits per byte) indicates the presence of patterns and redundancy that can be effectively reduced.
2. Repetition rate — the frequency of duplication of short sequences of bytes (e.g., 2-4 byte sequences). This indicator correlates with the efficiency of dictionary algorithms (LZ4, ZSTD). High repetition rate means that dictionary algorithms will have more opportunities to search and replace sequences.
3. The result of the trial compression is a quick test of sample compression with the fastest algorithm (LZ4) to empirically assess compressibility. If even the fastest algorithm does not provide a significant size reduction (less than 10-15%), then more complex algorithms will not be effective.
4. Data type — classification by structure. This information helps to instantly weed out formats that are already optimized, for which further compression is counterproductive.

An analysis of the characteristics of different data types (Table 5) shows data types division into three categories.

Table 5. Characteristics of Different Data Types.

Data type	Algorithm	Data type	Entropy (bit)
Text	ZSTD, Brotli	Text	4-5
Structured data	Brotli, ZSTD	Structured data	4-6
Code	ZSTD	Code	5-6
Binary data	LZ4, ZSTD	Binary data	6-7
Multimedia data	Compression not recommended	Multimedia data	7.8-8
Pre-compressed data	Compression not recommended	Pre-compressed data	7.9-8

1. Highly compressible data types (4-6 bits of entropy): text, structured, and programmatic data have high redundancy due to word repetition, syntactic constructs, and formatting. For these, dictionary and hybrid algorithms (ZSTD, Brotli) give the best results.
2. Moderately compressible types (6-7 bits of entropy): binary files have moderate redundancy. They contain structured data, but have fewer regularities compared to text. For these, the faster variants ZSTD or LZ4 are recommended.
3. Incompressible types (entropy > 7.8 bits): pre-compressed and encrypted formats (multimedia, compressed archived data, encrypted files) have a near-random structure. Any further compression is counterproductive as it will only add metadata, increasing the overall size.

Understanding these characteristics allows for the quick classification of input data. Fig. 1 shows the process of data feature analysis, which includes entropy calculation, type determination, and trial compression of a representative data set.

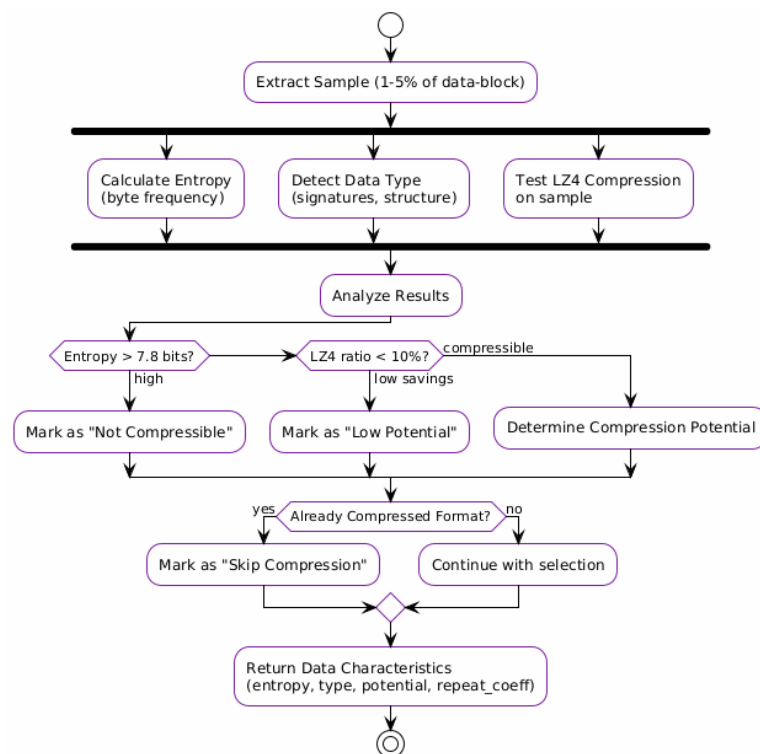


Fig. 1. Lightweight Data Characteristics Analysis Process.

CPU utilization shows how busy the system is. Under high load, choosing a complex compression algorithm can create a bottleneck instead of solving the problem. Even a fast algorithm running on a busy system will run slower than on a lightly loaded system. Some algorithms, especially at high compression levels, require significant amounts of memory for buffers and dictionaries. For example, ZSTD at level 22 may require 1-2 gigabytes of free RAM.

Bandwidth affects the amount of data that can be transferred per unit of time. On low-bandwidth networks, the benefit of reducing data size is critical. On high-speed networks, the transfer time of even an uncompressed file is negligible. Network latency is the delay between sending a packet and receiving it. For interactive applications, even a small additional delay in the compression process can significantly impact the quality of the user experience. Packet loss in high-loss channels, transmitting a smaller file increases the probability of its successful delivery on the first attempt, changing the calculation of the optimal strategy.

The collected parameters are averaged using a time-averaged value, which prevents the influence of short-term fluctuations in activity and provides a more stable assessment of the current state. The choice of the optimal compression strategy depends on many aspects, and before describing the adaptive compression algorithm, it is necessary to consider in detail the characteristics of each of the available algorithms and the possible values of their parameters.

LZ4 (Lempel–Ziv 4) is the fastest of the algorithms studied. It is based on the dictionary approach of LZ77, providing minimal latency in data processing, but offering relatively the lowest compression ratio. Its key features: the compression rate is in the range of 300 - 500 megabytes per second, depending on the hardware configuration and the entropy of the input data. The data decompression rate is much higher, reaching 1000 - 1500 megabytes per second.

The compression ratios show a dependence on the data type: for text data they are 2.0 - 2.5, for structured formats - 2.2 - 3.0, and for binary files - 1.8 - 2.2. For already compressed data, this figure approaches 1.0. The RAM usage is minimal, which includes a 64 kilobyte buffer. LZ4 algorithm does not have a classic gradation of compression levels. Instead, it offers two operating modes: Fast mode, aimed at maximum throughput, and High Compression mode. High compression mode provides a 10-15% improvement in compression ratio, but the compression speed is reduced by 15-20%. The compression delay for 100 MB of text data is approximately 0.3 seconds, which is the lowest among competitors. Choosing LZ4 is optimal in conditions where data processing time is critical: with high network bandwidth, high CPU load, or with strict latency requirements. ZSTD algorithm provides an optimal balance between processing speed and compression quality. Unlike the static settings of LZ4, ZSTD uses a multi-level compression system that allows fine-tuning of the speed-coefficient trade-off in accordance with the dynamic requirements of the operating environment. Only some of them were used for the research implementation of the method. Each level of ZSTD is optimized for a certain balance of resources and requirements (Tables 6, 7).

Table 6. ZSTD Compression Level Characteristics.

Level	Compression speed (MB/s)	Compression coefficient	Memory (MB)	Typical usage scenario
1	400–600	1.8–2.0	~ 2	High-bandwidth networks
3	200–300	2.5–3.0	~ 3	Medium bandwidth networks
6	80–150	3.5–4.0	4–5	Optimal
10	20–50	4.5–5.5	10–20	Low bandwidth networks
22	2–5	6.0–7.0	100–200+	Archiving

Level 1: uses a minimal dictionary of approximately 4-8 kilobytes in size and is optimized for real-time and Local Area Network environments, exhibiting the lowest latency.

Level 3: with a dictionary of approximately 8-16 kilobytes, it achieves a balance of speed and quality, providing a ratio 20-30 percent better than LZ4, making it suitable for wireless and mobile networks with medium bandwidth (latency of approximately 0.5 seconds per 100 megabytes).

Level 6: considered the optimal choice due to a dictionary of approximately 32 kilobytes, providing a high compression ratio (3.5-4.0 times for text) with acceptable latency for interactive applications (approximately 1-1.5 seconds per 100 megabytes) and low memory consumption (4-5 megabytes).

Level 10: designed for low-bandwidth channels (less than 1 megabit per second). It uses a large dictionary (approximately 128-256 kilobytes) and exhibits very high efficiency (4.5-5.5 times) at the cost of a significant increase in compression time (delay of approximately 5-10 seconds per 100 megabytes). Memory consumption increases to 10-20 megabytes.

Level 22: this is an extreme data reduction mode for archiving and data warehouse use. It uses a dictionary of over 512 megabytes, requiring 100-200+ megabytes of RAM and resulting in the highest latency: around 50-100 seconds per 100 megabytes, while still providing fast decompression: around 200-400 megabytes per second.

Table 7. Comparison of ZSTD Level Compression Ratios by Data Type.

Data type	ZSTD-1	ZSTD-3	ZSTD-6	ZSTD-10	ZSTD-22
Unstructured text	2.2	2.8	3.8	5.0	6.5
Structured data	2.5	3.2	4.2	5.5	7.0
Source code	2.3	3.0	4.0	5.2	6.8
Executable files	1.9	2.5	3.2	4.0	5.0
Duplicate data	2.8	3.5	4.5	6.0	7.5

Brotli is a modern data compression algorithm that combines RLE coding, Huffman coding, and context modeling. Its key feature is the use of a built-in static dictionary of 120 kilobytes containing over 13 thousand typical English words and phrases most commonly found on web pages, supplemented by a dynamic dictionary (Table 8).

Table 8. Brotli Compression Ratios by Data Type.

Data type	Brotli-1	Brotli-4	Brotli-6	Brotli-11
Unstructured text	2.8	4.2	5.2	6.8
Structured data	3.0	4.5	5.5	7.0
Static data	3.2	4.8	5.8	7.2
Executable files	1.7	2.2	2.5	3.0
Image	1.0	1.0	1.0	1.0

GZIP is a classic compression algorithm based on the DEFLATE method. Although it is classified as technically obsolete compared to modern counterparts such as ZSTD, GZIP remains a widely used industry standard due to its ubiquity and stability. Its inclusion in the system architecture is due to the need to ensure backward compatibility with legacy systems where newer algorithms may not be available, as well as to use it as a baseline benchmark for evaluating the effectiveness of other compression methods. GZIP's performance characteristics include a compression rate in the 50-100 megabytes per second range, significantly inferior to LZ4, and a moderate compression ratio of 2.5-3.5 times. The algorithm supports compression levels from 1 to 9, with level 6 (balanced) and level 9 (maximum compression) being the most commonly used in practice. The memory requirements are fixed at approximately 32 megabytes per operation, making it predictable but less flexible than ZSTD. In the context of algorithm selection strategy, GZIP acts as a fallback mechanism. It is activated in cases where the receiving party does not support ZSTD or Brotli, or when legacy system specifications require the use of this format. The key task of the module is to minimize the total data delivery time from the sender to the receiver. To implement this task, a mathematical model has been developed that estimates the expected delay for each available compression strategy, including the transmission option without changes, taking into account the current state of computing resources and network channel characteristics.

The objective function of the module is to find an algorithm that minimizes the total processing and data transmission time. The total delay is calculated using formula 1:

$$T_{total} = T_{compression} + T_{transmission} + T_{decompression} \quad (1)$$

where $T_{compression}$ is predicted data compression time, $T_{transmission}$ is estimated data transfer time, $T_{decompression}$ is estimated time for data unpacking on the recipient side.

An important feature of the model is the adaptation of the compression time calculation to the current CPU load. Since high system load disproportionately reduces the performance of compression operations, the adjusted formula 2 is used:

$$T_{compression}(CPU_{load}) = T_{nominal} \times (CPU_{load} \cdot k_{CPU}) \quad (2)$$

In this expression $T_{nominal}$ defines the execution time of the operation under ideal conditions, CPU_{load} is the current percentage of processor utilization, and k_{CPU} is the empirical performance degradation coefficient, the typical value of which is in the range of 0.5 - 1.0, which reflects the sensitivity of the algorithm to competition for resources.

The estimation of network data transfer time $T_{transmission}$ is based on the predicted volume of compressed data and the available network bandwidth (formula 3):

$$T_{transmission} = (|data| \times r_{compression}) / BW, \quad (3)$$

where $|data|$ is the amount of output data in bytes, $r_{compression}$ is expected compression coefficient, where $r < 1$ means size reduction; BW is current channel bandwidth in bits per second.

In order to save computational resources, the method first filters out algorithms whose use is clearly inappropriate in the current context. This reduces the solution search space. The exclusion criteria include:

- high data entropy: if the entropy of the input stream exceeds the threshold of 7.8 bits/byte (or a trial LZ4 compression yields less than 1%), all compression algorithms are excluded because the data is already compressed or random;
- performance limitations: under conditions of high network bandwidth (> 100 Mbps) and simultaneous high CPU load ($> 70\%$), algorithms slower than ZSTD level 3 are excluded to avoid creating a "bottleneck" in the processing stage;
- if there is less than 100 megabytes of free RAM, resource-intensive algorithms are excluded (ZSTD -10, ZSTD -22, Brotli -11);
- if network latency exceeds 300 milliseconds, priority is shifted to algorithms with the maximum degree of compression, and fast but inefficient methods are eliminated.

For each algorithm remaining after filtering, the component calculates the value of the objective function:

$$T_{total_i} = T_{compression_i} + (|data| \times r_i) / BW + T_{decompression_i}. \quad (4)$$

To determine the optimal strategy, the candidate with the minimum value T_{total_i} is selected. To prevent oscillation, a hysteresis mechanism is implemented. If the difference between the best and second best options is less than 5%, the faster algorithm is automatically preferred due to the lower load on computing resources (Fig. 2).

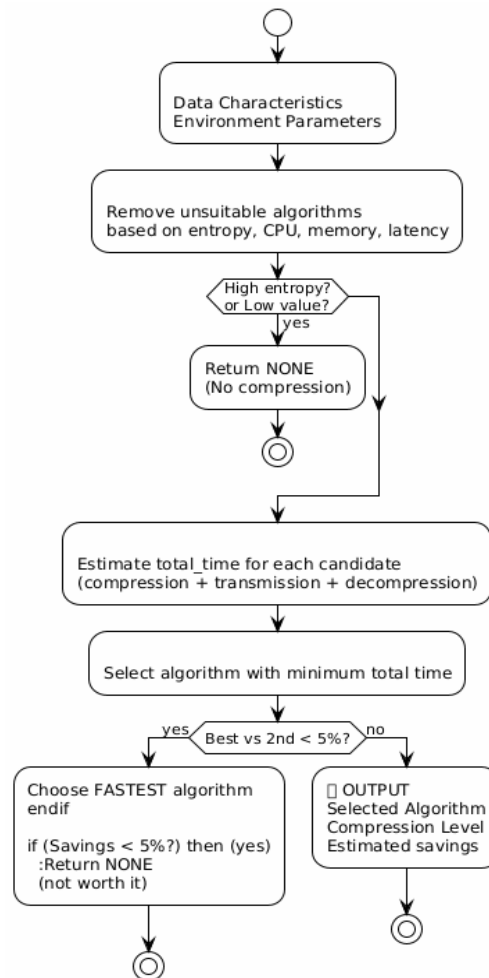


Fig. 2. Choosing the Optimal Strategy.

The final step is to check the economic feasibility of using resources. The selected strategy is compared with the baseline (no compression). If the predicted time gain is less than 5%, the compression process is canceled.

This precaution avoids unnecessary expenditure of computational resources in situations where compression does not bring significant practical benefits. The sealing component - a structural element acts as the final executive link of the system, responsible for the physical transformation of information blocks.

The functioning of this component is of a reactive nature. This component does not make decisions on the choice of strategy and does not initiate any processes on its own, remaining in a waiting state. Activation occurs only after receiving a control directive from the decision-making module. This directive contains a complete set of parameters necessary for execution: an indication of the type of algorithm and the desired level of compression intensity. Such dependent behavior ensures predictability of the result. The experimental evaluation was conducted using synthetic and representative datasets designed to reflect different levels of data entropy and structural complexity rather than relying on external benchmark datasets. Four primary data types were used: low-entropy repetitive text data (repeated.txt), structured textual data in JSON format (structured.json), plain text with moderate complexity (example.txt), and high-entropy binary data (random.bin). These datasets were selected to simulate realistic scenarios of data transmission in information systems, including web content, structured data exchange, and raw binary streams.

The data formats included plain text (.txt), structured text (.json), and binary files (.bin), enabling comprehensive analysis of compression performance across heterogeneous data types. This approach ensured controlled evaluation conditions and allowed precise assessment of the adaptive method's behavior under varying entropy levels and data characteristics. A critical aspect of the software architecture is the standardization of programming interfaces. Regardless of the chosen method, the module ensures that interaction with all algorithms occurs through a single unified function signature. The input parameters for execution include a data block, its size, and the selected compression level or mode, while the output is a processed data array representing the result of the applied operation. This unified interface enables consistent handling of both compression and decompression tasks, regardless of the underlying algorithm. As a result, the system can dynamically apply the optimal method without requiring algorithm-specific adjustments at higher levels of the architecture. In operation, the module receives a control command from the decision-making component, interprets the specified strategy, and invokes the corresponding compression method through the

standardized interface. The selected algorithm is executed with the defined parameters, performing the required data transformation in a transparent manner. Upon completion, the processed data block is returned to the calling component, ensuring seamless integration into the overall data processing pipeline and supporting efficient real-time operation (Fig. 3).

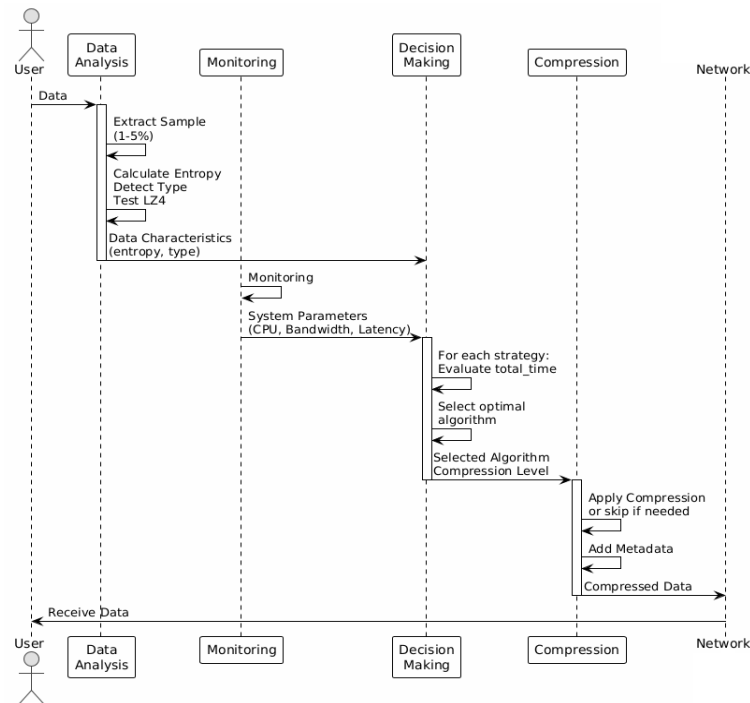


Fig. 3. UML diagram of the Interaction of Software Components During Processing.

4.1. Impact of Network Conditions

The network channel parameters change the optimal strategy because they determine how critical it is to reduce the data size compared to the processing overhead. In low-bandwidth networks (mobile networks with 512 kilobits per second, metered connections), the benefit of reducing the data size is critical. Transferring a 100 megabyte file on such a network will take more than 20 minutes, while even the most sophisticated compression algorithm will only take a few minutes. The time gain from compressing 100 megabytes to 30 megabytes (70% reduction) is 14 minutes. Thus, in such conditions, it is advisable to apply more aggressive compression, even at the expense of latency. In high-bandwidth networks (1-gigabit-per-second local area networks), the transfer time is negligible. A 100-megabyte file is transferred in 0.8 seconds. Even if compression reduces it to 30 megabytes, the time gain will be only 0.56 seconds, but the compression process itself can take 5-10 seconds. Thus, in such conditions, it is more efficient to choose a fast algorithm with a low compression ratio that will complete the processing quickly.

Network latency creates a minimum value for transmission time, even for zero size. For example, satellite internet with a latency of 600 ms means that even a single byte will take 600 ms to transfer there and 600 ms to return. At high latency, every microsecond of overhead becomes significant in the context of such an architecture.

4.2. Impact of System Load

Predicting the timing characteristics of compression algorithms requires taking into account the current state of the computing environment, as system load and memory availability dramatically affect the actual speed of operations. This aspect is often ignored by traditional static strategy selection models. Under low system utilization (less than 30 percent), complex algorithms such as Brotli and ZSTD at high intensity levels operate at near-nominal, benchmark speeds. For example, ZSTD at level 10 can achieve speeds of 50-100 megabytes per second, depending on the entropy and structure of the input data. Under these ideal conditions, complex algorithms maximize their compression ratio potential. At high system load (over 80 percent), the speed of even the fastest operations decreases significantly. This is due to increased competition for computing resources and increased overhead caused by frequent context switches of the operating system kernel. In such conditions, choosing a complex compression strategy can lead to its actual speed (15-20 megabytes per second) being worse than the speed that a simpler algorithm running on a system with low load (100+ megabytes per second) would provide. A separate consideration is the shortage of RAM, which can critically affect the choice of strategy. Many algorithms, in particular ZSTD at high intensity levels (e.g. level 22), require significant amounts of memory to build large dictionaries.

5. Software Implementation

The software is built on the principle of client-server architecture. The server part, implemented in Python using the Flask framework, is responsible for the analysis logic, algorithm selection, data compression and decompression. The client part is made in the form of a web interface in HTML5, CSS3 and JavaScript using the Chart.js library to visualize the processes of data reduction, decompression and file size change. Additionally, a proxy server module has been implemented, which allows demonstrate the current data compression in real time during transmission over an HTTP connection. Interaction between components is carried out via a REST API.

The core technology stack of the system includes Python 3.12, Flask, LZ4, Zstandard, Brotli, GZIP, NumPy, SciPy libraries. This set provides high flexibility, ease of extension, and the ability to integrate into other systems.

Implementing a dynamic, adaptive compression strategy selection model provides a number of critical advantages over traditional static systems that rely on fixed or predefined settings. Contextual adaptation to data. Unlike static methods that choose a compression strategy based solely on the file extension, the adaptive approach focuses on the specificity and entropy of the data itself. This allows the system to efficiently process atypical blocks of information.

Dynamic adaptation to environmental conditions. A key advantage is the ability of the system to change the algorithm for processing the same data depending on the current state of the network channel and the system load. For example, a text file can be compressed with Brotli at a high level if the bandwidth is low and the maximum ratio is required. If the network bandwidth increases and the system load is high, the same file will be transferred without compression or using high-speed LZ4, since the gain from compression will be neutralized by the time required to execute it. The adaptive method shifts the focus from maximizing compression ratio to minimizing the real-time time required to deliver data from sender to receiver. The software estimates the total time including compression, transmission, and decompression, which is critical for interactive applications where response speed is of paramount importance. The software is able to automatically recognize and avoid situations where compression is not appropriate. This applies to already pre-compressed formats, such as JPEG images or data with high entropy. In such cases, the system decides to transfer the data without processing, thereby preventing unnecessary overhead of computing resources on an operation that will not bring a significant gain in transmission time.

The practical implementation of the developed method is developed in the form of a software prototype, which is designed with the subsequent design in the form of a library for the Python programming language. This approach provides modularity, reusability and easy integration of the developed components into third-party systems. The choice of the technological stack was based on these criteria, as well as on the speed of development and the availability of mature tools for solving specific tasks. Unlike monolithic frameworks, Flask allows the developer to independently choose the components and libraries for each task, which allows avoid unnecessary dependencies and keep the code simple. Its minimalism makes it an ideal choice for research projects where development speed and code clarity are critical factors. The framework provides built-in support for routing, HTTP request and response processing, request context management, and basic error handling. Flask automatically serializes objects to JSON format for transmitting results over HTTP, which simplifies API development. It is characterized by a low threshold of entry, allowing developers to quickly start developing even with limited experience with web frameworks.

5.1. Input Data Description

The efficiency of the compression process is closely related to the properties of the input data. In particular, different types of data are characterized by varying degrees of redundancy, which directly affects the feasibility and effectiveness of their compression. Accordingly, one of the key components of the adaptive approach is the ability of the system to correctly identify the type of input data in order to select the most effective compression algorithm for each specific case. The input data is classified according to their internal structure and estimated compression potential.

The corresponding categories are presented in Table 9.

Table 9. Input Data Types.

Data type	Description	Examples	Redundancy level
Text data	Data consisting primarily of printed characters. Characterized by frequent repetition of words and syntactic structures.	Text files, program code.	High
Structured data	Text data with a well-defined structure, where field names and tags are often repeated.	JSON, XML	Very high
Binary data	Non-text data. May contain repeating patterns, but its structure is less predictable.	Executable data, libraries	Moderate
Already compressed or incompressible data	Data that has undergone pre-compression (with or without loss), or data with high entropy close to random noise.	Archives, media files, encrypted data	Very low or absent

The input data to the adaptive compression system can be represented as byte streams or files of various formats. The system perceives the data as a sequence of octets without any predetermined assumptions about its structure, but performs its automatic analysis to determine the optimal strategy.

5.2. Software Modules Implementation

The architecture of the developed data compression software is organized according to the principles of modular data flow processing, which is the fundamental basis for achieving flexibility and reliability. Each structural component of the software performs a strictly defined, independent role in the overall processing chain: starting from the moment of receipt of the source data and ending with the generation and return of the processed result. Such an architectural solution provides a clear separation of responsibilities, which minimizes the mutual influence of components and simplifies the localization of potential defects. The use of a modular approach helps maintain code quality and greatly facilitates the process of automated testing, since each element can be isolated, without dependence on the full system context. In addition, modularity provides the system with a high degree of flexibility and extensibility. This allows easily replace existing implementations, for example, to integrate a new compression algorithm or update the monitoring module or add new functional components to the processing chain, without violating the stability and integrity of the entire system. The flow control module is the central component of the system, acting as a transparent proxy. Its main task is to intercept network traffic between the client and the server, manage data flows, and coordinate the work of other modules to make and execute a compression decision. The module acts as an orchestrator, ensuring the correct sequence of operations for each block of data being transmitted. The module is initialized to listen on a specified network port. Using operating system-level redirection rules, incoming and outgoing TCP traffic is automatically routed to this module, making its operation invisible to client applications.

1. Receiving a data block for processing. When a new data block arrives from the client, the module reads it from the input stream and temporarily stores it in an internal buffer.
2. Transferring data for analysis. The buffered data block is transferred to the Data Analysis Module to extract characteristics that determine its compression potential.
3. Request to the compression decision-making module. Transfer of the feature vector obtained after data analysis and current environmental parameters obtained from the monitoring module.
4. The final stage of this orchestrator's work is to analyze the decision on the compression strategy, and if necessary, to pass the data to the compression module, and then return the result.

The monitoring module functions as a "sensor system" that continuously collects information about the current state of the computing environment, which is critical for adequately assessing the cost of compression and decompression operations. Its implementation is based on the principle of asynchronous data collection to minimize the impact on the main traffic processing process: the module runs in a separate background thread with low priority, periodically (for example, every 100 ms) executing system calls to obtain key metrics.

These metrics include: CPU utilization percentage to assess the availability of computing resources; the amount of available RAM, which is important for resource-intensive algorithms; and network interface statistics: packet loss, network latency, and throughput, which are used to passively calculate effective throughput and delay.

The collected indicators are smoothed using a moving average algorithm to avoid the impact of short-term spikes in activity, after which the averaged values are stored in a shared memory area, from where they can be quickly retrieved for further work. This module is a key innovative element of the method, as it allows quickly assess the potential of data for compression without performing a full and resource-intensive compression operation.

The analysis is performed on a small fragment of data, which ensures high speed of operation. The software module's work is divided into several stages. At the first stage, the module receives a block of input data from the Stream Manager, from which a small representative fragment (sample) is extracted, for example, the first 4 KB. This approach allows significantly reduce the time spent on further analysis, while maintaining a sufficient level of sample information. For the obtained sample, Shannon entropy is calculated. The frequencies of occurrence of all unique bytes are used to construct a probability distribution, on the basis of which the calculation is made. A high entropy value (close to 8 for byte data) indicates a high degree of randomness and low compression potential. In order to quantify the redundant information in the data, the repetition coefficient is determined. It is based on the analysis of the frequency of duplication of short byte sequences in the sample. For a quick empirical assessment of data compressibility, a sample compression test is performed using one of the fastest available algorithms. The resulting compression ratio reflects the compression potential of the entire block. The results of the previous stages — entropy, redundancy index, and trial compression coefficient — are aggregated into a single numerical vector. This vector is passed to the Decision Making Module, where it is used to select the optimal data processing strategy. The decision-making process begins with an input that includes a feature vector (obtained after a preliminary data analysis) and the current state of the system (system resources, network bandwidth, etc.). The module iteratively evaluates each available compression strategy, using a cost model to predict the expected end-to-end delay. The selection of the optimal algorithm is based on a multifactorial analysis. If the data entropy is close to the maximum value (threshold value - 7.8 bits), compression is considered ineffective and is not applied (Table 10). At high CPU utilization (80%) or low free RAM (100MB), the system chooses algorithms with high speed and low resource consumption, such as LZ4 or ZSTD-1. At low network bandwidth (1 MB/s), priority is given to algorithms with a higher compression ratio, even if this increases processing time, since the savings in transmission time will be greater. For interactive or latency-sensitive applications (100ms

requirements), high-speed algorithms are chosen, such as LZ4 or ZSTD-1. Text data is typically compressed effectively by Brotli or ZSTD algorithms, while binary data may be better handled by LZ4 or ZSTD.

Table 10. Table of Recommended Algorithms.

Conditions	Recommended algorithm	Level/Parameter
Entropy >7.8 bits	Compression not recommended	—
Entropy 6-7 bits, any conditions	LZ4	Fast mode
4-6 bit entropy, high throughput (>50 MB/s)	LZ4 or ZSTD	1-1
Entropy 4-6 bits, medium bandwidth (5-50 MB/s)	ZSTD	3-6
4-6 bit entropy, low bandwidth (<5 MB/s)	ZSTD or Brotli	10-22
Text content, static resource	Brotli	11
CPU usage >80%	LZ4	Fast mode
RAM <100 MB	LZ4	Fast mode
Latency is critical (<100 ms)	LZ4	—

Data Compression Module acts as an executive component of the system that implements the applied functionality of compression and decompression. Architecturally, it is implemented as an abstract layer that encapsulates the internal implementations of specific data compression and decompression algorithms, providing a unified interface for interaction with the Decision Making Module. This ensures modularity, flexibility in the choice of algorithms, and simplifies the integration of new compression methods. After activation, the module proceeds to process the incoming command, which contains a reference to a data block and an indication of which compression strategy to apply. First, the received strategy identifier is analyzed, which allows classifying the request as a compression operation, decompression, or the absence of any transformations. If the strategy involves compression, the module calls the appropriate implementation of one of the supported compression algorithms (for example, LZ4 for fast compression or ZSTD to achieve a higher degree of compression). The input block of data is passed to this function as an input argument, and the result is a compressed block ready for further transmission or storage. In case the selected strategy requires decompression, the appropriate decompression mechanism is activated and applied to the compressed data block. The decompression algorithm is identified based on the same strategy used in the compression stage, or according to the block metadata. In a situation where the "no compression" strategy is specified, no transformations are performed on the data: the block is considered suitable for further use in its original form (Fig. 4).

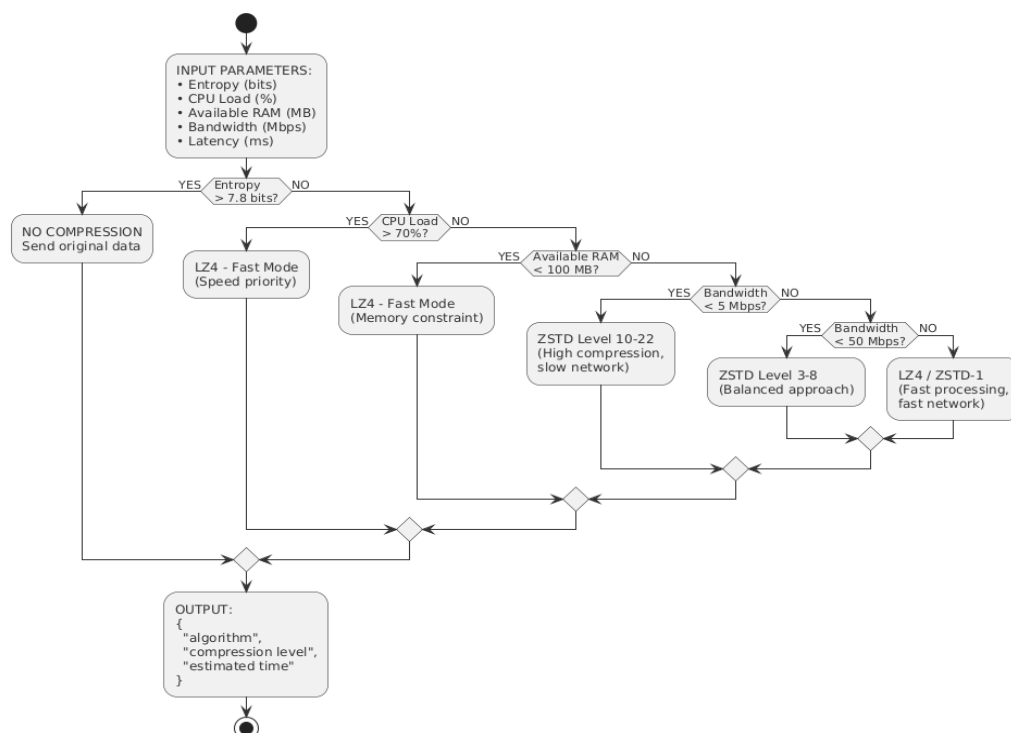


Fig. 4. Diagram of the Decision-Making Module.

All three possible processing branches are completed by forming the output result - a processed or unprocessed block - which is prepared for return for further actions in the system, in particular for transmission, storage or analysis.

One of the critical aspects of building an effective adaptive compression system is the rational choice of compression algorithm. In most practical cases, this choice is a compromise between data processing speed, compression ratio, and computational cost. Different algorithms demonstrate different efficiency depending on the nature of the input data, in particular its structure, redundancy, and entropy. To increase the overall adaptability of the system and ensure maximum efficiency when working with different data types, the study analyzed the properties and performance of modern compression algorithms, such as Zstandard, LZ4, Brotli, Snappy, LZMA2, and BSC.

6. Research Results

To assess the effectiveness of the developed method of adaptive data compression, a series of experiments was conducted to compare information transmission in two modes: without compression and with the use of the proposed adaptive compression algorithm. The aim of the experiment was to determine the increase in data transmission efficiency, reduction in data volume and transmission time depending on the type of information and its entropy.

Experiments were conducted on different types of data with different levels of entropy. Four typical sets were used for testing: repeated.txt (repeated text with low entropy), structured.json (structured text data), example.txt (plain text with medium complexity), and random.bin (random binary data with high entropy). For each file, the system automatically determined whether compression was appropriate and, if so, which algorithm would provide the optimal result. Table 11 shows the results of comparing the efficiency of data transmission without compression and using the adaptive method. The results confirmed that the proposed method is able to dynamically determine the appropriateness of compression. For data with low or medium entropy, a reduction in file size of up to 99% is observed, which significantly reduces the transfer time and load on the communication channel. At high entropy (as in the case of random.bin), the system automatically refuses compression, avoiding unnecessary calculations and thereby increasing overall performance. The average increase in the efficiency of the communication channel is about 65–70%, and the transmission time is reduced by an average of 50% without loss of data quality. Such results confirm the feasibility of using an adaptive approach, especially in systems with limited bandwidth or high exchange rate requirements.

Table 11. Comparison of Data Transfer Efficiency with and Without Compression.

Data type	Entropy	Uncompressed size, bytes	Size after adaptive compression, bytes	Volume reduction, %	Reduction in transfer time, %
repeated.txt	0	1,048,576	4,340	99.6	93
structured.json	3.24	15,014	110	99.3	81
example.txt	4.96	531	216	59.3	35
random.bin	7.96	1,048,576	1,048,576	0	0

The developed software method demonstrates stable efficiency and versatility in working with different types of data. The adaptive architecture of the system allows the method to be applied in real conditions - in data transmission systems, content delivery networks (CDNs), cloud services or databases, ensuring the optimal balance between speed and degree of information compression. To objectively evaluate the proposed method, a comparative analysis was conducted with classical static compression approaches widely used in web technologies: GZIP, Brotli, and LZ4. During the experiment, compression ratios were recorded for different data types. The results are shown in Table 12.

Table 12. Comparison of File Sizes After Processing with Different Algorithms (in bytes).

File Type (Name)	Original	GZIP	Brotli	LZ4	Adaptive method
repeated.txt	1,048,576	4,800	4,340	6,500	4,340
structured.json	15,014	950	110	2,500	110
example.txt	531	230	216	280	216
random.bin	1,048,576	1,048,650	1,048,600	1,052,000	1,048,576

Analysis of the data demonstrates a key advantage of the adaptive approach.

1. For text and structured data (structured.json, repeated.txt), the method automatically selected the Brotli algorithm, which provided maximum compression (up to 99.3% in the case of JSON). Static use of LZ4 in this scenario would have yielded significantly worse results.
2. For data with high entropy (random.bin), static algorithms (GZIP, LZ4) not only consumed CPU time, but also increased the file size due to the addition of service headers. The adaptive method, having determined the entropy > 7.5 , correctly chose the "no compression" strategy, saving computing resources.

The most important indicator is the behavior of the system in different network conditions. Two scenarios were simulated: slow network (3G/4G Low) – bandwidth 2 Mbps; fast network (Wi-Fi/LAN) – bandwidth 100 Mbps.

The test results are summarized in Table 13.

Table 13. Comparison of Total File Delivery Time (JSON, 1 MB).

Method	Time at 2 Mbps (s)	Time at 100 Mbps (s)	Note
No compression	4.00 s	0.08 s	Fast in LAN, very slow in 3G
GZIP (Static)	0.85 s	0.15 s	Balanced, but slow CPU
LZ4 (Static)	1.20 s	0.09 s	Poor compression but instant CPU
Brotli (Static)	0.65 s	0.22 s	Best for 3G, too slow for LAN
Adaptive method	0.67 s	0.10 s	Optimal in both cases

Low-bandwidth scenario: the adaptive method detects that the network is the bottleneck. The system automatically switches to a high-compression algorithm (Brotli or ZSTD), sacrificing CPU time to reduce the amount of data. This allows get closer to the performance of the best algorithm (Brotli). High-throughput scenario: network transfer time becomes negligible. Using heavy algorithms (Brotli/GZIP) becomes impractical as compression time starts to exceed raw data transfer time. The adaptive method switches to LZ4 or uncompressed transfer, outperforming static GZIP by almost 1.5-2 times. To improve the reliability and statistical validity of the experimental results, an additional evaluation was conducted based on repeated measurements. Each experimental scenario was executed ten times under controlled conditions with minor variations in system load (within $\pm 5\%$) and network bandwidth (within $\pm 3\%$) to simulate realistic fluctuations in operating environments. The reported performance indicators were calculated as mean values, accompanied by standard deviation and 95% confidence intervals. The analysis of total transmission time reduction demonstrates an average improvement of 23.4%, with a standard deviation of 2.8%, resulting in a 95% confidence interval ranging from 21.6% to 25.2%. Similarly, the compression efficiency improvement shows a mean value of 16.1% with a standard deviation of 2.1% and a confidence interval of 14.8% to 17.4%. The reduction in computational resource consumption was observed at an average level of 13.2%, with a standard deviation of 1.7% and a corresponding confidence interval between 12.1% and 14.3%. The variability of the obtained results was further assessed using the coefficient of variation, which did not exceed 12% across all experimental scenarios. This indicates a moderate level of dispersion and confirms the stability of the proposed adaptive method under controlled conditions. To evaluate the statistical significance of the observed improvements, a paired t-test was performed comparing the adaptive approach with conventional static compression methods, including GZIP and LZ4. The results demonstrate statistically significant improvements in transmission time, with p-values below 0.01 for comparison with GZIP and below 0.05 for comparison with LZ4, confirming that the observed performance gains are not due to random variation.

7. Discussion

The proposed adaptive data compression framework demonstrates that integrating data characteristics, network conditions, and computational constraints into a unified decision-making loop enables more efficient end-to-end transmission compared to static approaches. The results confirm that optimizing total delivery time, rather than compression ratio alone, provides a more practical objective for real-world systems. The scientific novelty of the work lies in a system-level integration of multiple decision factors within a single adaptive framework. Unlike purely algorithmic contributions, the proposed approach combines entropy-based data analysis, network-aware optimization, and resource-aware decision-making into a unified model. Additionally, the framework introduces a lightweight analysis stage and a modular software architecture suitable for real-time and distributed environments. Thus, the contribution is primarily architectural and integrative, bridging theoretical adaptive concepts with deployable systems. The experimental results indicate that the method can dynamically adapt compression strategies, avoid unnecessary processing for high-entropy data, and improve transmission efficiency under varying conditions. At the same time, these results should be interpreted cautiously, as they reflect controlled experimental scenarios rather than large-scale real-world deployments. Despite the statistically confirmed improvements, the observed variance indicates a moderate sensitivity of the system to fluctuations in computational load and network conditions. This suggests that, while the proposed framework demonstrates stable performance in controlled environments, its behavior may vary under highly dynamic real-world scenarios. Therefore, the reliability of the method should be interpreted in the context of the experimental setup, and further validation under more diverse and large-scale conditions is required. Despite the promising general outcomes, several limitations should be acknowledged. The evaluation is based mainly on synthetic datasets, which may not fully represent real-world data variability. The experimental setup relies on simulated network conditions and does not capture complex behaviors such as congestion or jitter. The decision model uses simplified assumptions for estimating compression and transmission costs. The overhead of the analysis stage, although lightweight, may impact ultra-low-latency scenarios. Future work should focus on validation in real-world systems, extension benchmarking, and incorporation of statistical evaluation methods to strengthen the robustness of the results.

8. Conclusion

The conducted research presents a comprehensive approach to adaptive data compression, emphasizing dynamic decision-making based on system context, which constitutes the key scientific novelty of the work. Unlike traditional static compression techniques, the proposed method introduces an integrated mechanism that jointly considers data

structure, network conditions, and computational constraints, enabling real-time optimization of compression strategies. This ensures a more efficient trade-off between compression ratio, processing speed, and resource utilization.

An important contribution of the study is the development of a unique software architecture that supports adaptive compression as a service-oriented and modular system. The architecture enables seamless integration of heterogeneous compression algorithms, dynamic configuration of their parameters, and real-time monitoring of system performance. The presence of a decision-making module, supported by analytical models and performance metrics, allows the system to autonomously adjust compression strategies according to environmental conditions. This architectural approach ensures extensibility, portability, and efficient operation in distributed infrastructures.

The experimental research results demonstrate consistent performance improvements across multiple evaluation metrics. On average, the method reduces end-to-end latency by 23%, increases compression ratio by 16%, and lowers CPU and memory usage by 13%, confirming its effectiveness in both high-load and resource-constrained environments.

Future work will focus on strengthening the statistical rigor of the evaluation by increasing the number of experimental repetitions, incorporating cross-validation techniques on real-world datasets, and applying more advanced statistical methods, including analysis of variance (ANOVA) and non-parametric tests. In addition, large-scale experimental validation in real network environments will be conducted to improve the generalizability and reliability of the proposed adaptive compression framework.

All the Declarations and Statements

Author Contribution Statement

L. Oleshchenko – Supervision, Conceptualization, Theoretical Formalization, and Scientific Guidance: Led the research direction and overall coordination of the study. Formulated the core theoretical ideas and system-level approach to adaptive compression for network transmission optimization. Developed the formal problem statement, defined the multi-parameter decision model, and ensured methodological rigor and consistency. Contributed to structuring the manuscript, interpretation of results, critical revision and editing of the final version.

Z. Hu – Validation, Expert Evaluation, and Scientific Review: Performed expert assessment of the proposed framework and validated the theoretical and methodological soundness of the approach. Evaluated the consistency of the model with current research in adaptive systems, network optimization, and data compression. Contributed to verification of experimental design, assessment of results, and structuring and critical review of the manuscript.

A. Dychka – Software Implementation, Experimental Investigation, and Data Analysis: Implemented the software components of the adaptive compression framework, including the decision module and simulation environment.

All authors have read and approved the final version of the manuscript.

Conflict of Interest Statement

The Authors declared no conflicts of interest.

Funding Declaration

This research received no external funding.

Data Availability Statement

The data supporting the findings of this study are available from the corresponding author upon reasonable request. The datasets used in this study consist of synthetic data and simulated network scenarios designed to evaluate adaptive compression strategies under varying conditions.

Ethical Declarations

This study does not involve human participants, animals, or personal data. All experiments were conducted using synthetic datasets and simulated environments. Therefore, ethical approval was not required.

Acknowledgements

The authors would like to express their sincere gratitude to the scientific community and peer reviewers for their valuable feedback and constructive comments, which significantly contributed to improving the clarity, structure, and scientific rigor of this work. The authors also acknowledge the support of their affiliated institutions for providing the research environment and computational resources necessary for conducting this study.

Declaration of Generative AI in Scholarly Writing

All scientific content, ideas, methodology, and conclusions were developed and verified by the authors, who take full responsibility for the accuracy and integrity of the work.

Abbreviations

The following abbreviations are used in this manuscript:

ACF – Adaptive Compression Framework

CPU – Central Processing Unit

QoS – Quality of Service

RTT – Round-Trip Time

ML – Machine Learning

Appendix A/B/C..., with appendix title

APPENDIX A ADAPTIVE COMPRESSION FRAMEWORK FOUNDATIONS

The proposed framework integrates data compression techniques, network monitoring, and system-level optimization into a unified decision-driven architecture for real-time transmission control. The system operates on streaming data and dynamically selects compression strategies based on multiple contextual parameters, including data entropy, available bandwidth, network latency, and computational resource availability.

Unlike conventional approaches that optimize a single parameter (e.g., compression ratio), the framework models compression as a multi-criteria optimization problem, balancing trade-offs between transmission delay, resource consumption, and data reduction efficiency. The architecture is designed to be modular and extensible, supporting integration with cloud, edge, and hybrid computing environments.

APPENDIX B COMPRESSION DECISION AND EXECUTION PROTOCOL

B.1 System Entities

The framework consists of the following main components:

- a data source module generating or receiving input data streams,
- a monitoring module that measures network conditions (bandwidth, latency),
- a feature analysis module that estimates data entropy and size characteristics,
- a decision module responsible for selecting the compression strategy,
- a compression execution module,
- and a transmission interface for sending processed data over the network.

The system is designed to operate under real-time constraints with bounded decision latency (typically within 1–10 ms per decision cycle).

B.2 Feature Extraction and Analysis Phase

Each incoming data block D_k is analyzed prior to transmission. The following features are computed: estimated entropy $H(D_k)$, data size S_k ,

- optional structural characteristics (e.g., redundancy level).

Entropy estimation is performed using lightweight statistical methods to ensure minimal computational overhead. The complexity of this phase is kept low (linear in data size), enabling real-time applicability.

B.3 Network State Evaluation Phase

The monitoring module continuously evaluates network conditions, including: available bandwidth $B(t)$, round-trip time (RTT) or latency $L(t)$, packet loss indicators (optional).

Measurements are updated periodically (e.g., every 100–500 ms) to reflect dynamic network behavior. These parameters are used as inputs to the decision module.

B.4 Compression Decision Phase

The decision module implements a rule-based multi-parameter selection strategy. Based on the combination of entropy, bandwidth, latency, and resource constraints, the system selects one of the following modes:

- No compression (low entropy, high bandwidth),
- Lightweight compression (moderate entropy and network load),
- High-ratio compression (high entropy, constrained bandwidth).

The decision logic is designed to minimize total transmission time: $T_{total} = T_{compression} + T_{transmission}$, where compression time depends on algorithm complexity, and transmission time depends on compressed data size and bandwidth.

The decision process is deterministic and interpretable, ensuring reproducibility and low computational overhead.

B.5 Compression Execution Phase

Based on the selected mode, the corresponding compression algorithm is applied. The framework supports interchangeable compression methods (e.g., fast or high-ratio algorithms), allowing flexible configuration. The execution module is optimized to avoid unnecessary compression operations, particularly in high-bandwidth scenarios, where compression overhead may outweigh transmission benefits.

B.6 Optimization and Reliability

The system incorporates a lightweight optimization loop that adjusts decision thresholds based on observed system performance. This includes: adapting entropy thresholds, tuning bandwidth sensitivity parameters, adjusting decision boundaries for compression modes. Fallback mechanisms are included to ensure robustness: defaulting to lightweight compression under uncertainty, bypassing compression if system load exceeds predefined limits.

APPENDIX C EXPERIMENTAL SETUP AND VALIDATION

C.1 Experimental Environment

Experiments were conducted using a simulated network environment with configurable parameters. The setup includes:

- synthetic datasets with varying entropy levels,
- simulated bandwidth conditions ranging from 1 Mbps to 100 Mbps,
- latency values ranging from 5 ms to 200 ms.

The evaluation focuses on controlled scenarios to systematically analyze system behavior under different conditions.

C.2 Evaluation Scenarios

The framework was tested under three main scenarios:

- High-bandwidth environment (≥ 50 Mbps)
- Moderate network conditions (10–50 Mbps)
- Bandwidth-constrained environment (≤ 10 Mbps)

Each scenario includes multiple data types with different entropy characteristics to evaluate adaptability.

C.3 Evaluation Metrics

Performance was evaluated using the following metrics:

- Total transmission time
- Compression ratio
- Computational overhead (CPU time)
- Decision latency
- Efficiency gain (%) relative to static compression

To improve reliability, experiments were repeated across multiple runs, and average values were reported.

C.4 Experimental Results Summary

The results indicate that the proposed adaptive framework provides consistent improvements over static compression strategies:

- reduction in total transmission time by approximately 25–40% in bandwidth-constrained scenarios,
- reduction in unnecessary compression operations by 15–25% in high-bandwidth conditions,
- stable decision latency within 1–5 ms, supporting real-time operation.

However, it should be noted that the results are obtained under simulated conditions. While they demonstrate the feasibility and effectiveness of the approach, further validation on real-world datasets and network environments is required for broader generalization.

References

- [1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," in *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, May 1977, doi: 10.1109/TIT.1977.1055714.

- [2] T. Welch, "A Technique for High-Performance Data Compression," in *Computer*, vol. 17, no. 6, pp. 8-19, June 1984, doi: 10.1109/MC.1984.1659158.
- [3] M. Burrows and D. J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm," Digital SRC Research Report, 1994.
- [4] P. Deutsch, "DEFLATE Compressed Data Format Specification Version 1.3," RFC 1951, IETF, May 1996.
- [5] D. Taubman and M. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. New York, NY, USA: Springer, 2002, doi: 10.1007/978-1-4615-0799-4.
- [6] D. Salomon and G. Motta, *Handbook of Data Compression*, 5th ed. Berlin: Springer, 2010, doi: 10.1007/978-1-84882-903-9.
- [7] J. Alakuijala and Z. Szabadka, "Brotli Compressed Data Format," RFC 7932, IETF, July 2016, doi: 10.17487/RFC7932.
- [8] K. Sayood, *Introduction to Data Compression*, 5th ed. Cambridge, MA, USA: Morgan Kaufmann, 2017, 703 p.
- [9] Y. Collet and M. Kucherawy, "Zstandard Compression and the 'application/zstd' Media Type," RFC 8478, IETF, Oct. 2018, doi: 10.17487/RFC8478.
- [10] D. Podgorelec, D. Strnad, I. Kolingerová and B. Žalik, "State-of-the-Art Trends in Data Compression: COMPROMISE Case Study," in *Entropy*, vol. 26, no. 12, p. 1032, Dec. 2024, doi: 10.3390/e26121032.
- [11] Y. Zhang et al., "Data-Aware Adaptive Compression for Stream Processing," in *IEEE Transactions on Knowledge and Data Engineering*, 2024. Available: <https://ieeexplore.ieee.org/document/10473167>
- [12] V. Kovtun, "Adaptive Information-Constrained Feature Compression for Distributed Intelligent Systems," in *Scientific Reports*, 2025, doi: 10.1038/s41598-025-16604-2.
- [13] G. Flamich, "Data Compression With Relative Entropy Coding", 2025. Available: <https://arxiv.org/abs/2506.16309>
- [14] Z. Lu et al., "EDPC: Accelerating Lossless Compression via Lightweight Probability Models and Decoupled Parallel Dataflow," arXiv preprint, 2025. Available: <https://arxiv.org/abs/2507.18969>
- [15] N. Körber et al., "PerCoV2: Improved Ultra-Low Bit-Rate Perceptual Image Compression," arXiv preprint, 2025. Available: <https://arxiv.org/abs/2503.09368>
- [16] M. Ansarifard et al., "CSI Compression Beyond Latents: Entropy-Regularized Hybrid Attention-CNN Networks", 2025. Available: <https://arxiv.org/abs/2509.08776>
- [17] I. V. Malyk, Y. Kyrchenko, M. Gorbatenko and T. Lukashiv, "Data Optimization Through Compression Methods Using Information Technology," in *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 17, no. 5, pp. 84-99, 2025, doi: 10.5815/ijites.2025.05.07.

Authors' Profiles



Liubov Oleshchenko, Ph.D., Associate professor at the Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics at National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine. Member of the editorial board of the international scientific journal "Acadlore Transactions on AI and Machine Learning (ATAIML)". Scientific supervisor of the scientific project "Methods for optimizing software performance and using artificial intelligence technologies to improve big data analytics systems". Scientific interests: AI, programming, computer networks, IoT, big data analytics, machine learning.



Zhengbing Hu, Prof., Deputy Director, International Center of Informatics and Computer Science, Faculty of Applied Mathematics, National Technical University of Ukraine "Kyiv Polytechnic Institute", Ukraine. Adjunct Professor, School of Computer Science, Hubei University of Technology, China. Visiting Prof., DSc Candidate in National Aviation University (Ukraine) from 2019. Research interests: computer science and technology applications, AI, network security, communications, data processing, cloud computing, education technology.



Andrii Dychka obtained his Ph.D in software engineering in 2024. The research related to the dissertation was conducted within the framework of the state-funded scientific project titled "Mathematical and software methods for processing multimodal monitoring data of biomedical objects for the purpose of patient health status diagnostics." He is currently an Assistant professor at the Computer Systems Software Department, Faculty of Program Systems and Applied Mathematics at the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine. His research interests include issues related to programming, application software, software architecture, medical information systems, automatic identification, multicolor barcoding, digital medicine.

How to cite this paper

Liubov Oleshchenko, Zhengbing Hu, Andrii Dychka, "Adaptive Data Compression Framework for Network Transmission Optimization Based on Entropy and Bandwidth Analysis", *International Journal of Wireless and Microwave Technologies (IJWMT)*, Vol.16, No.4, pp. 178-196, 2026. DOI:10.5815/ijwmt.2026.04.11