**Journal homepage:** https://itce.vn.ua/en

**Information Technologies and Computer Engineering** **Vol. 23, No. 2. 2026**

**Article's History:** Received: 28.01.2026 Revised: 21.05.2026
Accepted: 25.06.2026 Published: 03.08.2026

ISSN 1999-9941; e-ISSN 2078-6387
UDC 004.8:004.7:681.3
DOI: 10.31649/vitce/2.2026.178

# Data compression algorithms in modern web technologies: An analysis of Gzip, Brotli and Zstandard methods for optimising content delivery

**Dmytro Lutsak***
PhD in Technical Sciences
King Danylo University
76018, 35 Yevhena Konovaltsia Str., Ivano-Frankivsk, Ukraine
https://orcid.org/0000-0001-9496-3542

**Abstract.** The study aimed to provide a theoretical justification for and a comparative evaluation of the lossless compression algorithms Gzip, Brotli and Zstandard in modern web technologies for optimising content delivery. The methodological framework encompassed a theoretical generalisation of the principles of lossless compression, an analysis of technical specifications and software implementations, the construction of a comparative matrix, structural and functional modelling of the web pipeline, and a formalised assessment of the algorithms' suitability for different types of web resources. As a result, the study established that the efficiency of web content compression was determined not only by the properties of a specific algorithm, but also by the structure of the resource, the level of data redundancy, the response formation mode under the Hypertext Transfer Protocol, content encoding support, CPU overhead, memory requirements, and the capabilities of the client-server infrastructure. The study demonstrated that Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript and JavaScript Object Notation (JSON) are suitable for lossless compression due to the repetitiveness of syntactic constructs, service characters, tags, selectors, code fragments, keys and nested structures. A comparative analysis identified the functional profiles of the algorithms: Gzip – a basic, compatible solution for static and dynamic responses over the Hypertext Transfer Protocol; Brotli – an option for pre-compressed static text resources; Zstandard – a solution for high-speed, streaming and scalable scenarios, provided there is infrastructure support. The structural-functional model of the web compression pipeline were used for the selection of a method to be presented as a sequence of technical decisions: "resource type → generation mode → structural repeatability → encoding support → latency and resource requirements → algorithm selection → transmission of the compressed stream → client-side decompression". The practical significance of the results lies in the ability of web developers, system architects and performance optimisation specialists to use the proposed scheme to select Gzip, Brotli or Zstandard according to the resource type, compression mode and capabilities of the web infrastructure

**Keywords:** compression; entropy coding; resource type; encoding support; algorithm selection

## Introduction

The research relevance is determined by the fact that optimising the delivery of web content remains one of the key factors in improving the performance of web applications, reducing the load on network infrastructure, and enhancing the user experience, particularly given the growing volumes of text, script, multimedia and service data. In web technologies, compression not only serves the auxiliary function of reducing file size but is also part of a broader performance system that combines redundancy coding, entropy-based data representation, dictionary-based methods, caching, resource minimisation, and algorithm selection based on content type, network speed, and the computational capabilities of the client and server environments. Under these conditions, comparing Gzip, Brotli and Zstandard is significant not merely as a simple comparison of compression ratios, but as a theoretical and applied task of evaluating the trade-offs between the degree of compression, encoding and decoding speeds, memory usage, response latency, and the algorithm's suitability for various web content delivery scenarios.

In a study by I. Shvedov (2024), the optimisation of web resources is examined through a combination of

**Suggested Citation:**
Lutsak, D. (2026). Data compression algorithms in modern web technologies: An analysis of Gzip, Brotli and Zstandard methods for optimising content delivery. *Information Technologies and Computer Engineering*, 23(2), 178-190. doi: 10.31649/vitce/2.2026.178.

*Corresponding author (*dmytrolutsak@outlook.com*)

Copyright © The Author(s). This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/)

compression, caching and JavaScript minimisation. Compression is presented as a component of comprehensive optimisation aimed at reducing the volume of transmitted data and shortening page loading times. A. Fesenko & R. Hapon (2024) analysed lossless data compression algorithms and hash functions used in off-the-shelf software solutions. Their analysis focused on the functional capabilities of the algorithms, performance, integrity control and suitability for application software development. C. Jaranilla & J. Choi (2023) systematised the requirements and trade-offs of applying compression methods in key-value stores, specifically the relationship between compression ratio, access speed, memory consumption, and system performance. These parameters are closely related to web compression, as the choice of algorithm in web infrastructure also depends on the balance between the size of the compressed resource, processing time and server load. L. Zheng *et al.* (2022) investigated the optimisation of Zstandard based on the parallel, concurrent use of multiple hash tables. The paper emphasises the role of match search organisation, parallelisation, and hashing structure in improving the algorithm's performance. S. Karandikar *et al.* (2023) examined the hardware-system aspect of compression and decompression in hyperscale computing environments. Their approach emphasises throughput, power consumption, and computational resource utilisation, which is crucial for scalable server-side content delivery. X. Chen *et al.* (2024) proposed a cross-domain benchmark for lossless compression of floating-point numerical data. This study justifies the need for a unified set of criteria for comparing algorithms, correlating with the evaluation of Gzip, Brotli and Zstandard in terms of compression ratio, compression speed, decompression speed, implementation complexity and suitability for different types of web content. B. Wollmer *et al.* (2022b) analysed cross-entity delta encoding in web compression as a method of using inter-entity similarity to improve compression efficiency. Within this approach, web data is viewed as a collection of resources with repetitive structural elements, and the potential for compression is linked not only to individual files but also to the repetition of patterns across a set of pages or objects. M. Karahan *et al.* (2025) compared the performance of compression algorithms for log data in the context of regulatory requirements for information storage. The study considered not only the size of the compressed output but also speed, reliability and compliance with the operational constraints of the information system. In the work by B. Wollmer *et al.* (2022a), Compaz is presented as a tool for evaluating approaches to web compression based on user scenario data from real websites. Compression is viewed as a process linked to resource allocation, structural repetition, and the sequence of web content transmission. E. Maltsev & O. Muliarevych (2024) investigated serialisation formats from the perspective of spatial efficiency in data transmission within distributed systems. A comparison of JavaScript Object Notation (JSON) with alternative formats revealed that the size of a structured message depends on the data representation method, utility elements and structural repetition.

A review of the cited sources provided grounds for identifying a research niche: existing works either examined individual web optimisation techniques, analysed compression in other types of information systems, or focused on specialised algorithms for specific datasets. At the same time, there was a lack of systematic integration of the theoretical explanation of entropy and dictionary coding with a comparative assessment of Gzip, Brotli and Zstandard specifically in the context of web content transmission optimisation. There remained a need for a formalised comparative framework that would take into account the compression ratio, computational complexity, compression and decompression speeds, compatibility with web infrastructure, content types, and practical limitations of server-side implementation. This need determined the logic of the subsequent analysis, in which compression was viewed as an algorithmic pipeline for selecting a compression method based on resource characteristics, network conditions and performance requirements.

The study aimed to determine the functional capabilities and limitations of Gzip, Brotli and Zstandard in the process of optimising web content transmission, considering algorithmic principles, resource costs and the technical conditions of client-server interaction. The main objectives of the study were: to reveal the theoretical foundations of lossless data compression in web technologies through a combination of dictionary-based repetition search, entropy coding and statistical redundancy representation; to conduct a comparative assessment of Gzip, Brotli and Zstandard in terms of compression ratio, compression and decompression speeds, memory usage and computational complexity; to develop an algorithmic framework for selecting a compression method to optimise web content delivery, based on resource type, file size, update frequency, latency requirements and the suitability of pre-compression or dynamic compression.

## Materials and Methods

This study was a review-based theoretical and analytical investigation. The research was grounded in standard and protocol specifications for the web transmission of compressed content, documentation of algorithms and software implementations, as well as practical sources relating to browser, server and client-side processing of compressed streams. The first group included HTTP semantics RFC 9110 (Fielding *et al.*, 2022), DEFLATE RFC 1951 (Deutsch, 1996a), GZIP RFC 1952 (Deutsch, 1996b), Brotli RFC 7932 (Alakuijala & Szabadka, 2016), Zstandard RFC 8878 (Collet & Kucherawy, 2021) and Compression dictionary transport RFC 9842 (Meenan & Weiss, 2025). These were used to analyse the protocol conditions for transmitting compressed content, the principles of dictionary and entropy coding, the container organisation of Gzip, the Brotli format, the specifics of Zstandard and the role of pre-agreed dictionaries. The second group comprises the WHATWG (n.d.),

Free Software Foundation (n.d.) documentation, the zlib 1.3.1 Manual (Gailly & Adler, 2024), the Brotli compression format (Google, n.d.a), and Zstandard (Meta Platforms, n.d.). These materials were used to summarise the software implementation of the algorithms, their compression modes, streaming processing, dictionary mechanisms, and technical limitations. The third group comprised the Zlib documentation in Node.js v26.2.0 (n.d.), CURLOPT_ACCEPT_ENCODING explained (Curl, n.d.), Lighthouse (Google, n.d.b; n.d.c) and MDN Web Docs (n.d.). They have been used to analyse server-side compression of responses using the Hypertext Transfer Protocol, encoding coordination between client and server, browser-side streaming, and the instrumental assessment of web performance.

The method of theoretical analysis was applied to generalise the principles of lossless compression and to develop a compression model for text-based web content. Gzip is considered as an algorithm combining dictionary-based repetition detection and entropy coding; Brotli – as a method with a static dictionary, context modelling and enhanced efficiency for repetitive text structures; Zstandard is considered as an algorithm with flexible compression levels, fast decompression, dictionary support and suitability for streaming scenarios. In the developed model, dictionary-based mechanisms are presented as a method for identifying repetitive sequences in Hypertext Markup Language, Cascading Style Sheets, JavaScript, JavaScript Object Notation and other web resources, whilst entropy coding is presented as a mechanism for further reducing information redundancy.

A comparative analysis method was used to create an evaluation matrix for Gzip, Brotli and Zstandard. The algorithms were compared in terms of compression ratio, compression and decompression speed, RAM usage, support for streaming mode, the ability to use a dictionary, compatibility with content encoding via the Hypertext Transfer Protocol, suitability for static resources, dynamic responses and high-load systems, as well as ease of implementation. A ranked qualitative scale was used for evaluation: 3 – high level of compliance with the criterion; 2 – average or conditional level; 1 – limited level. Intermediate values of 1-2 or 2-3 indicated a contextual assessment dependent on the compression mode, resource type, algorithm settings, encoding support, and web infrastructure conditions.

The structural-functional modelling method was applied to describe the web compression pipeline as a sequence of operations: determining the resource type, assessing the size and repetitiveness of the data, selecting an algorithm, performing pre-compression or dynamic compression, transmitting the response with the appropriate content encoding, client-side decompression, and evaluating the impact on page performance. To formalise the comparison, a generalised compression algorithm evaluation vector (1) was used:

$$C_i = \left[R_i, T_{c_i}, T_{d_i}, M_i, S_i, W_i\right], \quad (1)$$

where $C_i$ – algorithm evaluation profile $i$; $R_i$ – compression level; $T_{c_i}$ – time or the relative complexity of compression; $T_{d_i}$ – duration or relative complexity of decompression; $M_i$ – memory requirements; $S_i$ – level of compatibility with the web infrastructure; $W_i$ – suitability for a specific type of web content. This formalisation served as an analytical framework for comparing algorithms based on identical parameters. The compression ratio was interpreted as the ratio of the original resource size to the size after compression, whilst computational complexity was assessed in terms of the time, memory and processor resources required for encoding and decoding the stream.

The critical comparative generalisation method was applied to identify the advantages, limitations and use cases of Gzip, Brotli and Zstandard in web transmission. The algorithms were compared according to the following criteria: the degree of compression of text-based web resources, compression speed, decompression speed, RAM usage, support for streaming, the ability to use a dictionary, compatibility with content encoding via the Hypertext Transfer Protocol, suitability for static resources, dynamic responses and high-load systems, as well as ease of implementation into web infrastructure. Based on this, an algorithmic scheme for selecting a compression method has been developed, built on a comparison of the resource type, its update frequency, file size, latency requirements, availability of pre-compression, support for the corresponding encoding by the browser or client, and the expected reduction in the volume of transmitted data. In the scheme, Gzip is considered as a solution for the combined compression of static and dynamic responses, Brotli for pre-compressed static text resources, and Zstandard for high-speed, streaming, and scalable scenarios, provided that the web infrastructure supports the relevant encoding.

## Results

**The main algorithmic principles of lossless compression of web content.** An analysis of the principles of lossless web content compression revealed that the effectiveness of Gzip, Brotli and Zstandard in modern web technologies was determined not only by the algorithmic properties of each method, but also by the structure of the text-based resources transmitted between the server and the client. The basis of compression was the reduction of information redundancy through the identification of repetitive sequences, their dictionary representation and subsequent entropy encoding. For web content, this logic is relevant, as Hypertext Markup Language, Cascading Style Sheets, JavaScript and JavaScript Object Notation contain repetitive syntactic constructs, service characters, tags, selectors, keys, parameter names, code snippets and structured data blocks. It has been established that Gzip served as the baseline compatible web compression algorithm, as it relied on a combination of LZ77 dictionary encoding and Huffman coding and was integrated into a large part of the server-side, client-side and intermediate web infrastructure. Brotli exhibited a different application

profile: its effectiveness for static text resources was linked to the use of a dictionary-contextual approach, which better suited the repetitive structure of HTML, CSS, JavaScript and JSON. Zstandard was defined as an algorithm with a flexible compression profile, for which scalable compression levels, high processing speed, support for dictionary modes and suitability for streaming scenarios were essential. Thus, the difference between the three methods lay not in the absolute superiority of any single algorithm, but in the varying degrees of compatibility between the resource structure, the compression mode, and the requirements for web transmission performance. The generalised logic of lossless compression of text-based web content is shown in Figure 1.

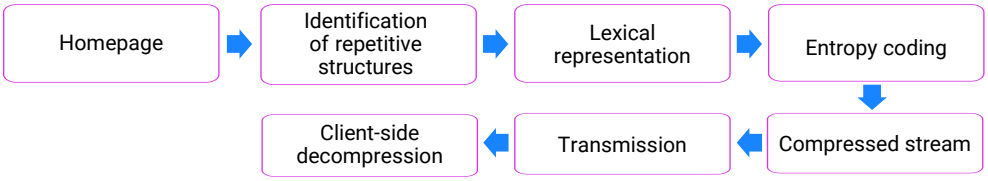


**Figure 1.** Lossless compression model for text-based web content

**Source:** compiled by the author based on P. Deutsch (1996a; 1996b), J. Alakuijala & Z. Szabadka (2016), Y. Collet & M. Kucherawy (2021), P. Meenan & Y. Weiss (2025)

The model shown in Figure 1 demonstrates that lossless compression of web content is based on the sequential identification of repetitive fragments, their representation in a dictionary, and the subsequent entropy encoding of the residual redundancy. This approach reduces not the content completeness of the resource, but the redundancy of its encoding, whilst preserving the ability to fully reconstruct the data on the client side. For text-oriented web resources, this relates to the repetition of tags, attributes, selectors, utility constructs, keys, delimiters, variable names, parameters and nested structures in Hypertext Markup Language, Cascading Style Sheets, JavaScript and JavaScript Object Notation. Consequently, the choice of compression algorithm must take into account not only the expected reduction in file size, but also compression time, decompression speed, compatibility with content encoding via the Hypertext Transfer Protocol, the possibility of pre-processing resources, and support for the algorithm by a specific web infrastructure. In summary, the model presented clarifies that Gzip, Brotli and Zstandard should be compared not as universally superior or inferior algorithms, but as solutions with varying suitability for the resource structure, response generation mode and technical conditions of web transmission. Therefore, further comparison should be based on a set of criteria related to content type, data redundancy, processing costs, and infrastructure support for compression.

**Comparative analysis of lossless web content compression algorithms.** A comparison of Gzip, Brotli and Zstandard using a single set of criteria revealed that these algorithms did not form a simple hierarchy of performance, in which one method would be universally superior to the others across all web transmission scenarios. Their performance depended on the balance between compression ratio, compression speed, decompression speed, RAM requirements, support for streaming mode, the ability to use dictionaries, compatibility with HTTP content encoding, and the type of resource being transferred. Consequently, Gzip emerged as the baseline solution, Brotli as the algorithm for efficient compression of static text resources, and Zstandard as the flexible algorithm for scenarios where processing speed, scalability, and the ability to adjust compression levels were relevant. The summarised results of the comparison of algorithms according to the defined criteria are presented in Table 1.

**Table 1.** Comparison table of evaluation criteria for Gzip, Brotli and Zstandard

| Evaluation criteria | Gzip | Brotli | Zstandard | Analytical significance of the criterion for web transmission |
|---|---|---|---|---|
| Degree of compression of text-based web resources | 2 | 3 | 2-3 | Identifies potential reductions in the size of HTML, CSS, JavaScript and JSON files without any loss of data |
| Compression speed | 3 | 1-2 | 3 | Determines the algorithm's suitability for dynamic HTTP responses and resources generated during a request |
| Decompression rate | 3 | 3 | 3 | Affects the speed at which resources are restored on the client side after receiving a compressed stream |
| RAM usage | 3 | 2 | 2-3 | Assess the scalability of compression under a high volume of concurrent requests |
| Support for batch processing | 3 | 2 | 3 | Determines the algorithm's suitability for large responses, streaming data and incremental content delivery |
| Option to use a dictionary | 1-2 | 3 | 3 | Affects the compression efficiency of repetitive structures, page templates and similar API responses |
| Compatibility with HTTP content encoding | 3 | 3 | -2 | Determines the practical feasibility of using the algorithm in a real-world client-server transmission |

Table 1. Continued

| Evaluation criteria | Gzip | Brotli | Zstandard | Analytical significance of the criterion for web transmission |
|---|---|---|---|---|
| Suitability for static resources | 2 | 3 | 2-3 | Shows the algorithm's performance for files that can be compressed at the time of the request |
| Suitability for dynamic responses | 3 | 1-2 | 3 | Determines the suitability of using the algorithm for server-generated real-time responses |
| Suitability for high-load systems | 3 | 2 | 3 | Describes the balance between processing speed, server load and scalability |
| Ease of implementation | 3 | 2-3 | 1-2 | Determines the technical complexity of integrating the algorithm into the existing web infrastructure |
| General application profile | 3 | 3 | 2-3 | Summarises the practical role of the algorithm in optimising the delivery of web content |

**Note:** 3 – high level of compliance; 2 – moderate or limited level of compliance; 1 – low level of compliance

**Source:** compiled by the author based on T.T. Fielding *et al.* (2022), Free Software Foundation (n.d.), J.-L. Gailly & M. Adler (2024), Google (n.d.a), Meta Platforms (n.d.)

An analysis of Table 1 showed that Gzip scores highest in terms of compatibility, compression speed, decompression speed, ease of implementation and suitability for dynamic responses. Brotli is notable for its compression ratio for text-based web resources, suitability for static files and the ability to utilise dictionary-based mechanisms. Zstandard demonstrates strong performance in terms of compression speed, decompression speed, support for streaming, dictionary modes, and suitability for high-load systems; however, its practical application depends on support for the relevant encoding by server, client, and intermediate environments. As a result, the comparison matrix clarified the functional distinctions between the three algorithms. Gzip served as the baseline compatible solution for a wide range of web scenarios; Brotli was the most suitable for pre-compressed static text resources; Zstandard had advantages in speed, streaming and scalable scenarios, provided there was infrastructure support. This comparison confirmed that the choice of compression algorithm should be determined not by a single metric, but by a set of criteria related to the type of resource, its generation mode, latency requirements, and the capabilities of the web infrastructure.

**The functional organisation of compression in the web data transmission circuit.** An analysis of the web compression pipeline revealed that the effectiveness of Gzip, Brotli and Zstandard was determined not by the algorithms' individual properties, but by their position in the sequence of server-side processing, protocol negotiation and client-side data reconstruction. Throughout the full web content transmission cycle, compression acted as an intermediate transformation of the HTTP response, which had to balance the degree of data reduction against Central Processing Unit (CPU) costs, RAM usage, channel bandwidth, content-encoding support and the acceptable latency budget. Consequently, an algorithm with a higher compression ratio did not always yield the best result for web transmission if its use increased runtime delay or created a risk of incompatibility on the client side. As a result, it was found that the key distinction lay between two processing modes: pre-compressed static resources and dynamic response compression during the request. For pre-compressed assets, the primary technical parameter was the maximum gain in file size, as the encoding overhead was shifted outside the processing of the current HTTP request. For runtime compression, the priority shifted to encoding speed, decoding stability, low CPU overhead and predictable memory footprint. This explained the different functional roles of the algorithms: Brotli was better suited to static HTML, CSS, JavaScript and JSON resources; Gzip remained valuable as a compatible fallback option for a wide range of responses; Zstandard demonstrated its advantages where the infrastructure supported high-speed or streaming compression. The technical logic of the web compression pipeline also showed that the decision on which algorithm to choose was formed at the intersection of three loops. The server-side loop determined the resource type, its update frequency, response size, structural repeatability, cacheability and compression mode. The protocol loop ensured coordination of content-encoding between the client and the server. The client loop determined the ability of the browser or other receiving environment to perform decompression without compromising page rendering speed. A generalised sequence of these functional blocks is shown in Figure 2.

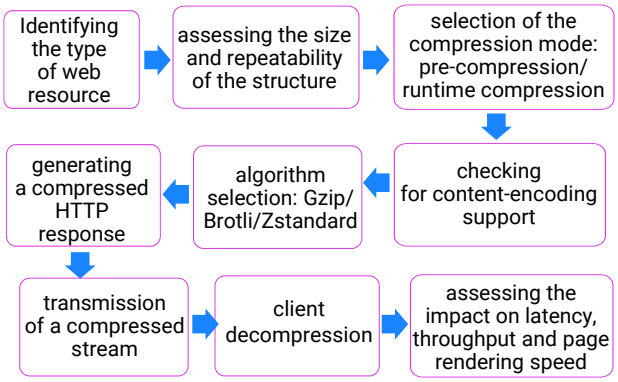


**Figure 2.** Structural-functional model of a web compression pipeline

**Source:** compiled by the author based on R.T. Fielding *et al.* (2022), WHATWG (n.d.), Node.js v26.2.0 (n.d.), Curl (n.d.), MDN Web Docs (n.d.)

The model presented in Figure 2 shows that compression in the web pipeline combines server-side resource generation, protocol-level encoding negotiation, and client-side data reconstruction. At the server level, the following factors are taken into account: content type, response size, update frequency, cacheability, structural redundancy, and the CPU time required for encoding. At the protocol level, the key factor is content encoding negotiation, which ensures the correct transmission of the compressed stream between the server and the client. At the client level, the speed of decompression, the reconstruction of the original resource, and the impact on page performance are significant. Thus, the structural-functional model clarified that optimising web content transmission was not simply a matter of selecting the algorithm with the highest compression ratio. A rational choice depended on the resource generation mode, response size, data redundancy, caching capabilities, acceptable CPU overhead, memory footprint, latency budget, content-encoding support, and client decompression speed requirements. The combination of these parameters determined whether it was appropriate to use Gzip as a universal, compatible solution, Brotli as a tool for pre-compressed static resources, or Zstandard as a flexible algorithm for high-speed and streaming scenarios.

**A comparative analysis of web compression algorithms based on evaluation metrics.** A formalised evaluation of Gzip, Brotli and Zstandard based on the vector defined in formula (1) showed that the profile of a compression algorithm in web transmission was not determined by a single metric, but by the interplay of compression ratio, compression overhead, decompression speed, resource acceptability, compatibility with web infrastructure, and suitability for a specific content type. Following this logic, Gzip, Brotli and Zstandard formed different functional application profiles: Gzip – a profile for compatible runtime compression; Brotli – a profile for pre-compression of static text resources; Zstandard – a profile for high-speed and streaming processing provided infrastructure support is available. The results of the formalised comparison also showed that improving a single parameter did not automatically give the algorithm an advantage in all web scenarios. A higher compression ratio was only of practical significance when the compression overhead did not increase response latency. Fast decompression was critical for the client-side loop; however, its effect diminished in the absence of support for the corresponding encoding in the server-side or intermediate environment. Consequently, the formalised profile specified not the general "efficiency" of the algorithm, but the limits of its rational application in specific web content delivery modes. A generalised evaluation of the algorithms based on the components of the vector is presented in Table 2. An analysis of Table 2 showed that a formalised evaluation based on the $C_i$ vector clarifies the relationship between the compression ratio, compression complexity, decompression speed, memory requirements, compatibility with web infrastructure, and the algorithm's suitability for a specific type of web content. Gzip performs best in terms of compatibility, predictability of compression during requests, and fast client-side resource restoration. Brotli is notable in terms of reducing the size of text-oriented resources in pre-compression mode. Zstandard should be considered an algorithm with a flexible balance between compression level, processing speed, throughput, and resource consumption; however, its application depends on support for the relevant encoding by server, client, and intermediate environments. Thus, the $C_i$ vector can be used to compare algorithms against specific web transmission conditions rather than being ranked by a single metric. In summary, a profile-based comparison using the $C_i$ vector shows that Gzip, Brotli and Zstandard should not be evaluated based on a single dominant parameter, but rather through a combination of compression ratio, compression and decompression speeds, memory requirements, compatibility with web infrastructure and the type of web content. This approach clarifies the limits of the rational application of each algorithm: some scenarios require, above all, compatibility and predictable processing during the request, whilst others require a higher level of pre-compression or high-speed streaming processing.

**Table 2.** A formalised evaluation of Gzip, Brotli and Zstandard based on the components of the $C_i$ vector

| Evaluation component | Gzip | Brotli | Zstandard |
|---|---|---|---|
| $R_i$ – compression rate | Basic compression of HTTP text responses; slower than Brotli for static resources | Most significant reduction in HTML, CSS, JavaScript and JSON following pre-compression | Varies depending on the compression level; strikes a balance between file size and speed |
| $T_{ci}$ – time or complexity of compression | Suitable for runtime compression | Costs are rising at a rapid pace | Suitable for high-speed compression |
| $T_{di}$ – duration or complexity of decompression | Rapid client recovery | Suitable for restoring pre-compressed assets | Designed for rapid restoration of flow |
| $M_i$ – memory requirements | Estimated server requirements | Depends on the compression level | Depends on the mode and the dictionary |
| $S_i$ – compatibility with web infrastructure | Most established support | Supported by modern web environments | Depends on whether the infrastructure supports encoding |
| $W_i$ – suitability for a particular type of web content | Static and dynamic HTTP responses | Static text-based resources | Streaming, high-speed and scalable scenarios |

Table 2. Continued

| Evaluation component | Gzip | Brotli | Zstandard |
|---|---|---|---|
| General application profile | Basic, cross-platform algorithm suitable for a wide range of web scenarios | Algorithm for pre-compressed resources prioritising size reduction | Flexible algorithm for supported scenarios with throughput requirements |

**Source:** compiled by the authors based on P. Deutsch (1996a; 1996b), J. Alakuijala & Z. Szabadka (2016), Y. Collet & M. Kucherawy (2021), J.-L. Gailly & M. Adler (2024), Google (n.d.a), Meta Platforms (n.d.)

**Algorithmic formalisation of the selection of a web content compression method.** A critical analysis of the comparison results showed that Gzip, Brotli and Zstandard did not form a universal ranking of performance across all web transmission scenarios. The choice of algorithm depended on a combination of technical conditions: the type of resource, its generation mode, update frequency, response size, level of structural redundancy, support for the relevant encoding, acceptable latency budget, CPU overhead and the expected reduction in data transfer volume. Under this logic, the most effective algorithm was not the one that provided maximum compression in an isolated case, but the one whose profile matched the specific web content delivery mode. For static text-based resources that could be compressed before the request, prioritising Brotli was technically justified. Its advantage was evident when the compression overhead was shifted to the file pre-processing stage and did not increase the processing time of the current HTTP request. For dynamic responses or scenarios where maximum compatibility with clients and intermediate components remained critical, Gzip remained the rational choice. For scenarios requiring fast compression, streaming, scalability and rapid stream resumption, Zstandard could offer an advantage, but only provided that the relevant encoding was supported by the server, client and intermediate environments. The obtained results can be used to present the choice of compression method as a sequence of technical decisions, in which each subsequent step refines the conditions for applying the algorithm. First, the type of resource and the nature of its generation were determined; next, its size, repetitiveness and the possibility of prior compression were assessed; after that, encoding support, acceptable latency and server load requirements were considered. The generalised logic for selecting Gzip, Brotli or Zstandard is shown in Figure 3.

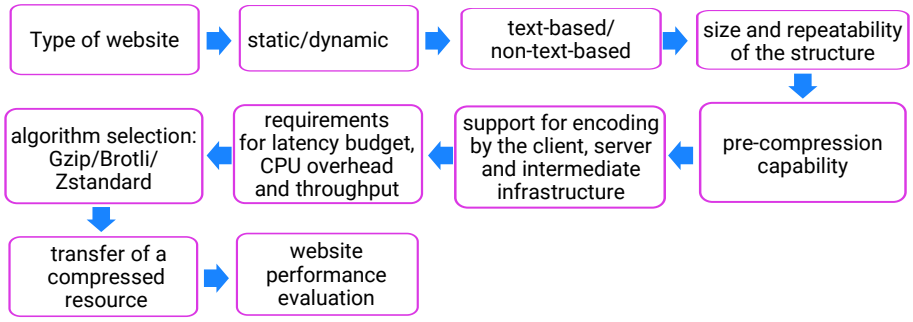


**Figure 3.** Algorithmic flowchart for selecting Gzip, Brotli or Zstandard to optimise web content delivery

**Source:** compiled by the author based on R.T. Fielding *et al.* (2022), P. Meenan & Y. Weiss (2025), WHATWG (n.d.), Node.js v26.2.0 (n.d.), Curl (n.d.), Google (n.d.b; n.d.c), MDN Web Docs (n.d.)

The diagram shown in Figure 3 illustrates the conditional branching logic for selecting a compression algorithm based on the resource type, its generation mode, acceptable latency, and support for the relevant encoding within the web infrastructure. For static text resources that can be pre-processed, Brotli is the recommended choice; for dynamic responses generated on-demand, Gzip is recommended; and for streaming or high-load scenarios, Zstandard is recommended, provided there is technical support within the client-server chain. Thus, the scheme does not rank the algorithms by absolute superiority, but defines the conditions for their rational application in accordance with the constraints of web transmission. Generalised scenarios for the application of Gzip, Brotli and Zstandard in web transmission are presented in Table 3.

**Table 3.** Recommended scenarios for using Gzip, Brotli and Zstandard in web transmission

| Webcast script | Conditions of use | Priority algorithm | Alternative algorithm | Analytical justification for the choice | Key restrictions |
|---|---|---|---|---|---|
| Static HTML, CSS and JavaScript resources | Resources rarely changed, had a text-based structure and could be prepared in advance | Brotli | Gzip | Pre-compression made it possible to use a higher compression ratio without increasing the duration of the current request | Encoding overhead was undesirable for high-level runtime compression |

Table 3. Continued

| Webcast script | Conditions of use | Priority algorithm | Alternative algorithm | Analytical justification for the choice | Key restrictions |
|---|---|---|---|---|---|
| JSON API responses | Data contained duplicate keys, nested structures and a consistent format | Brotli/ Gzip | Zstandard | Brotli was suitable for cached or repetitive responses; Gzip remained stable for dynamic API responses | Choice depended on the frequency of updates, the size of the response and the client's support |
| Dynamic pages | Response was generated on the fly and required minimal additional processing time | Gzip | Zstandard | Gzip provided predictable runtime compression and the lowest risk of incompatibility | Lower compression ratio compared to Brotli for static text resources |
| Live answers | Data was transmitted gradually, with throughput and stream recovery speed being the critical factors | Zstandard | Gzip | Zstandard performed better in scenarios involving fast compression, fast decompression and streaming | Practical implementation depended on the infrastructure's support for encoding |
| Reusable structured resources | Resources had similar templates, shared sections or recurring structures | Brotli/ Zstandard | Gzip | Lexical mechanisms improved the efficiency of transmitting such resources | Proper implementation of lexical or pre-compression was required |
| Scenarios involving server load limiting | Priorities were to keep CPU overhead and memory footprint under control and to ensure stable processing of large numbers of requests | Gzip/ Zstandard | Brotli | Gzip ensured stability and compatibility; Zstandard was the preferred choice for high-speed mode | At high levels, Brotli could increase compression overhead |

**Source:** compiled by the author

An analysis of Table 3 showed that the choice of compression algorithm depends on the technical configuration: the type of resource, how it is generated, the possibility of pre-processing, the acceptable latency, and support for the relevant encoding. For static text resources, Brotli is the preferred choice if compression is performed before the request; for dynamic pages and parts of application programming interface responses, Gzip remains the most suitable option due to the predictability of processing during the request; for streaming and high-load scenarios, Zstandard may be used provided there is infrastructure support. Repetitive structured resources and responses in JavaScript object notation occupy an intermediate position, as the choice of algorithm depends on caching, update frequency, response size and latency requirements. Thus, Table 3 clarifies that the algorithm is not determined solely by the data format: the technical context of web content transmission and processing is decisive. Thus, the results obtained confirmed that a rational choice between Gzip, Brotli or Zstandard should be based on a combination of compression ratio, computational cost, decompression speed, memory requirements, resource generation mode, compatibility with web infrastructure, and the expected impact on content delivery performance.

## Discussion

The results obtained showed that the effectiveness of Gzip, Brotli and Zstandard in web transmission was determined not by any single algorithmic property, but by the interplay between resource structure, compression mode, infrastructure compatibility and computational costs. This approach was consistent with the findings of J. Hu *et al.* (2024), where lossless compression of information messages was linked to the pre-transformation of data structures and improved coding efficiency through the specialised representation of byte sequences. In interpretation, this corresponded to the established role of the repetitiveness of HTML, CSS, JavaScript and JSON, since in both cases the compression performance was determined not only by the choice of algorithm, but also by the extent to which the data structure was suitable for compact representation without loss of information. The assertion regarding the continued relevance of classical lossless algorithms in modern information systems was consistent with the review by D.M. Matusiak (2025), in which classical approaches to compression were considered the foundation of many modern solutions, despite the emergence of new algorithmic and neural network models. This confirmed the conclusion that Gzip retained its status as a fundamental compatible solution not because of its maximum compression ratio, but due to established infrastructure support, the predictability of runtime compression, and its suitability for a wide range of HTTP responses. The advantage of Brotli for text and structured resources was consistent with the findings of M.Y. Satriyawibawa *et al.* (2024), where Brotli was used as a component to improve data delivery efficiency in combination with other encoding procedures. This was consistent with the conclusion reached regarding the suitability of Brotli for pre-compressed HTML, CSS, JavaScript and JSON resources, where the cost of compression could be deferred beyond the duration of the current request. At the same time, the results showed that a universal approach to compression did not yield equally effective results for heterogeneous data types. This finding was consistent with the study by V. Alevizos *et al.* (2025), in which a logarithmic compression method for data with high amplitude variability was designed to adapt the representation scheme to the specific characteristics of the input array. The findings regarding the role of compression and decompression speeds were consistent with the study

by H. Zhang *et al.* (2025), where the optimisation of Snappy was linked to the refinement of encoding strategies for efficient hardware implementation. In this work, the algorithm's performance was assessed in terms of its ability to ensure fast processing under resource constraints, which corresponded to the role of runtime compression, memory footprint and CPU overhead established in the results. A review and benchmark of neural network-based universal compressors by H. Sun *et al.* (2025) showed that modern approaches to lossless compression were increasingly being evaluated on various data sources, incorporating multi-criteria performance. This approach was consistent with the comparative matrix produced, in which Gzip, Brotli and Zstandard were compared not based on a single metric, but on a combination of parameters: compression ratio, compression speed, decompression speed, memory requirements, streaming capabilities and infrastructure compatibility. The theoretical link between language sequence modelling and compression, presented by G. Delétang *et al.* (2024), reinforced the interpretation of text-oriented web resources as objects with high structural and statistical predictability. This approach was conceptually consistent with the established pattern whereby the highest compression potential was achieved where the structure of the resource was regular and repetitive.

Further developments in neural network-enhanced lossless text compression methods are reflected in the study by S.S. Narashiman & N. Chandrachoodan (2024), where text compression was improved through the use of models capable of predicting data structure. This approach confirmed the significance of sequence predictability for reducing redundancy, but at the same time highlighted the difference in the results obtained: for web transmission, the priority remained not on complex neural network mechanisms, but on algorithms compatible with the existing client-server infrastructure. An empirical comparison of hybrid text compression methods, conducted by M. Sahu & J. Panda (2025), showed that the effectiveness of the algorithms depended on the language, character encoding and the structure of the text array. This was consistent with the conclusion that the type of resource and the nature of its internal organisation determined the suitability of the algorithm to no lesser extent than the technical compression model. B. Carpentieri (2025) considered data compression concerning time constraints, where the choice of algorithm depends not only on the degree of compression achieved, but also on the permissible processing time. This approach is consistent with the conclusion reached in this article that the practical feasibility of Zstandard cannot be determined solely by its technical characteristics or potential compression ratio. In the paper by G. Senniappan & R. Lourdusamy (2025), the evolution of approaches from LZ77 to Zstandard was presented as a transition from classical dictionary-based mechanisms to more flexible and efficient compression schemes. This corresponded to the established functional distinction: Gzip remained an infrastructure-stable representative of the classical approach, Brotli enhanced the efficiency of text compression through dictionary-contextual mechanisms, and Zstandard demonstrated advantages where speed, streaming and scalability were required. The advantages of Zstandard in high-speed compression and decompression scenarios were further corroborated by the findings of J. Jeba Johannah *et al.* (2026), where Zstandard's high performance was examined in the context of cache memory operations. This confirmed the conclusion that Zstandard formed a distinct application profile focused on throughput, rapid stream recovery and compression level control. A comparative study of Zstandard, zlib and LZ4, conducted by A.P. Maulidina *et al.* (2023), showed that lossless compression algorithms had different profiles in terms of speed and compression ratio. This pattern correlated with the resulting criteria matrix, in which Gzip, Brotli and Zstandard were not ranked as universally better or worse, but were distributed according to scenarios of rational application. The use of Brotli in secure multi-cloud storage systems, as described by A.J.M. Kumar *et al.* (2023), confirmed that this algorithm can be an effective component of more complex data transmission and storage systems. This approach confirmed the findings regarding the suitability of Brotli for pre-compressed text-oriented resources, where the priority was to reduce the data volume prior to the request.

Research on the compression of large language models has also confirmed the general principle of profile-based redundancy reduction. In the context of microservice architecture, F. Atasoy *et al.* (2024) examined compression as a factor influencing the speed of communication between services and the resource costs of data transfer. Comparison results showed that a higher compression ratio does not guarantee a better practical outcome if the time taken for compression or decompression increases the overall processing cost. Text compression in the paper by E. Öztürk (2024) was linked to the selection of an algorithm based on the properties of the input text, rather than a universal ranking of methods. The data obtained demonstrated that the compression result depends on the text structure, the repetitiveness of fragments, and the algorithm's usage scenario. The results of X. Li *et al.* (2024) regarding Logshrink showed that effective log compression was achieved through the use of commonality and variability in structured records. This directly correlated with the established finding regarding JSON API responses and repetitive structured resources as intermediate scenarios for algorithm selection. In the paper by S. Zhang *et al.* (2024), the compression of Hypertext Transfer Protocol messages is considered in the context of edge computing networks, where not only the size of the transmitted data is relevant, but also the stability of processing on user devices and intermediate nodes. This approach directly supports the conclusion of this paper that the choice between Gzip, Brotli or Zstandard should incorporate the message type, latency, bandwidth and the conditions of client-server interaction. In the study by Z. Lin *et al.* (2024), the conversion of compression formats in content delivery networks is associated

with an increased load on bandwidth and intermediate infrastructure. This is consistent with the findings regarding Zstandard presented in the article: even a technically efficient algorithm does not deliver the expected effect without support for the corresponding encoding by the server, client and intermediate environments. The Tracezip method, proposed by Z. Chen *et al.* (2025), demonstrated that specialised trace compression in distributed systems can improve the efficiency of transmitting and storing service data by exploiting their structural redundancy. This was consistent with findings regarding repetitive structured resources in web environments, for which Brotli or Zstandard dictionary-based mechanisms could improve efficiency provided that pre-compression or dictionary-based compression was correctly implemented.

A summary of the comparison with relevant sources showed that the results obtained supported the current trend towards a shift from single-parameter compression evaluation to multi-criteria profiling of algorithms. In studies devoted to classical, modern, neural network, hardware-oriented and specialised compression methods, efficiency was most often linked not to a single metric, but to a balance between the degree of data reduction, processing speed, resource consumption, data type and implementation conditions. In this context, the results regarding Gzip, Brotli and Zstandard clarified the applied web-oriented aspect of this pattern: Gzip served as a stable, compatible algorithm, Brotli provided an advantage for pre-compressed text resources, and Zstandard formed the profile of a high-speed, streaming solution provided there was infrastructure support. Thus, the discussion confirmed that the optimisation of web content delivery should be based on a tailored choice of algorithm in accordance with the resource structure, compression mode, and technical constraints of the client-server environment.

## Conclusions

As part of the study, a theoretical and analytical framework has been developed for evaluating the lossless compression algorithms Gzip, Brotli and Zstandard for optimising web content delivery, based on a comparison of compression ratio, computational complexity, decompression speed, resource requirements, compatibility with web infrastructure, and suitability for different types of resources. The study demonstrated that text-oriented web resources such as HTML, CSS, JavaScript and JSON are highly suitable for lossless compression due to the repetitiveness of syntactic constructs, service characters, tags, selectors, keys, parameters, code fragments and nested structures; whilst compression efficiency was determined not only by the algorithm but also by the resource generation mode, the possibility of pre-processing, support for content-encoding, and acceptable server and client processing costs. Based on a comparative matrix, it was established that Gzip formed the profile of a baseline compatible solution for a wide range of static and dynamic HTTP responses, as it provided predictable runtime compression, fast client-side resource retrieval and the lowest risk of infrastructure incompatibility. Brotli emerged as the algorithm of choice for pre-compressed static text resources, where the priority was to minimise the transfer volume without factoring encoding costs into the current request time. Zstandard should be viewed as a flexible algorithm for high-speed, streaming and scalable scenarios where throughput, fast decompression, compression level control and managed resource consumption are key considerations, provided that the web infrastructure supports the relevant encoding. The web compression pipeline is defined as a sequence of technical decisions: “resource type → formation mode → structural repeatability → encoding support → latency and resource requirements → algorithm selection → transmission of the compressed stream → client-side decompression”, where each stage defines the boundaries of the method’s rational application. The study established that there was no single best-fit algorithm for all web transmission scenarios: the choice between Gzip, Brotli or Zstandard should be determined by the resource profile, compression mode, expected size reduction, CPU overhead, memory requirements, and the capabilities of the server, client and intermediate environments. Further research should address empirical testing of the proposed scheme for various types of web resources, recording compression ratio, latency, throughput and CPU overhead.

## Acknowledgements

None.

## Funding

None.

## Conflict of Interest

None.

## References

[1] Alakuijala, J., & Szabadka, Z. (2016). *Brotli compressed data format (RFC 7932)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc7932.

[2] Alevizos, V., Yue, Z., Edralin, S., Xu, C., Gerolimos, N., & Papakostas, G.A. (2025). A logarithmic compression method for magnitude-rich data: The LPPIE approach. *Technologies*, 13(7), article number 278. doi: 10.3390/technologies13070278.

[3] Atasoy, F., Akkaya, A., Kocakır, N., & Karademir, Ö. (2024). Performance analysis of compression algorithms on matrix data: Data transfer optimization in microservices architectures. *Turkish Journal of Nature and Science*, 1, 49-60. doi: 10.46810/tdfd.1425959.

[4] Carpentieri, B. (2025). Data compression with a time limit. *Algorithms*, 18(3), article number 135. doi: 10.3390/a18030135.

[5] Chen, X., Tian, J., Beaver, I., Freeman, C., Yan, Y., Wang, J., & Tao, D. (2024). FCBench: Cross-domain benchmarking of lossless compression for floating-point data. *Proceedings of the VLDB Endowment*, 17(6), 1418-1431. doi: 10.14778/3648160.3648180.
[6] Chen, Z., Pu, J., & Zheng, Z. (2025). TraceZip: Efficient distributed tracing via trace compression. *Proceedings of the ACM on Software Engineering*, 2, 411-433. doi: 10.1145/3728888.
[7] Collet, Y., & Kucherawy, M. (2021). *Zstandard compression and the "application/zstd" media type (RFC 8878)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc8878.
[8] Curl. (n.d.). *CURLOPT_ACCEPT_ENCODING explained*. Retrieved from https://surl.li/sbusvv.
[9] Delétang, G., *et al*. (2024). Language modeling is compression. In *International conference on learning representations 2024 (ICLR 2024)* (pp. 14165-14181). Vienna: Messe Wien Exhibition and Congress Center.
[10] Deutsch, P. (1996a). *DEFLATE compressed data format specification version 1.3 (RFC 1951)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc1951.
[11] Deutsch, P. (1996b). *GZIP file format specification version 4.3 (RFC 1952)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc1952.
[12] Fesenko, A., & Hapon, R. (2024). Comparison of compression algorithms and hash functions of ready-made software solutions in the context of creating your own application. *Electronic Professional Scientific Journal "Cybersecurity: Education, Science, Technique"*, 3(23), 284-309. doi: 10.28925/2663-4023.2024.23.284309.
[13] Fielding, R.T., Nottingham, M., & Reschke, J. (2022). *HTTP semantics (RFC 9110)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc9110.
[14] Free Software Foundation. (n.d.). *GNU Gzip: General file (de)compression*. Retrieved from https://www.gnu.org/software/gzip/manual/gzip.html.
[15] Gailly, J.-L., & Adler, M. (2024). *zlib 1.3.1 manual*. Retrieved from https://www.zlib.net/manual.html.
[16] Google. (n.d.a). *Brotli*. Retrieved from https://github.com/google/brotli.
[17] Google. (n.d.b). *Introduction to Lighthouse*. Retrieved from https://developer.chrome.com/docs/lighthouse/overview.
[18] Google. (n.d.c). *Lighthouse: Optimize your website*. Retrieved from https://developer.chrome.com/docs/devtools/lighthouse.
[19] Hu, J., Gao, Y., Jin, Q., Zhao, G., & Lu, H. (2024). An efficient lossless compression algorithm for maritime safety information using byte encoding network. *Journal of Marine Science and Engineering*, 12(7), article number 1075. doi: 10.3390/jmse12071075.
[20] Jaranilla, C., & Choi, J. (2023). Requirements and trade-offs of compression techniques in key-value stores: A survey. *Electronics*, 12(20), article number 4280. doi: 10.3390/electronics12204280.
[21] Jeba Johannah, J., Chandra Sekaran, S., & Harish, S. (2026). High performance Zstandard compression and decompression for L cache memory. In *2026 international conference on innovative computing, intelligent communication and smart electrical systems (ICSES)* (pp. 1-6). Chennai: Institute of Electrical and Electronics Engineers. doi: 10.1109/ICSES66558.2026.11478819.
[22] Karahan, M., Erdal, E., & Ergüzen, A. (2025). Performance comparison of compression algorithms for log data in compliance with Turkish Law No. 5651. *Journal of Computer & Electrical and Electronics Engineering Sciences*, 3(2), 54-60. doi: 10.51271/jceees-0029.
[23] Karandikar, S., *et al*. (2023). CDPU: Co-designing compression and decompression processing units for hyperscale systems. In *ISCA '23: Proceedings of the 50th annual international symposium on computer architecture* (pp. 1-17). New York: Association for Computing Machinery. doi: 10.1145/3579371.3589074.
[24] Kumar, A.J.M., Columbus, C.C., George, B.E., & Christopher, S.C. (2023). *Ensuring secure and efficient multi-cloud storage with Brotli compression and hierarchical data protection*. doi: 10.21203/rs.3.rs-3221497/v1.
[25] Li, X., Zhang, H., Le, V.H., & Chen, P. (2024). Logshrink: Effective log compression by leveraging commonality and variability of log data. In *ICSE '24: Proceedings of the 46th IEEE/ACM international conference on software engineering* (pp. 1-12). New York: Association for Computing Machinery. doi: 10.1145/3597503.3608129.
[26] Lin, Z., Lin, Z., Liu, X., Ying, Z., & Chen, C. (2024). Unveiling the bandwidth nightmare: CDN compression format conversion attacks. In *2024 2nd international conference on big data and privacy computing (BDPC)* (pp. 97-106). Macau: Institute of Electrical and Electronics Engineers. doi: 10.1109/BDPC59998.2024.10649043.
[27] Maltsev, E., & Muliarevych, O. (2024). Beyond JSON: Evaluating serialization formats for space-efficient communication. *Advances in Cyber-Physical Systems*, 9(1), 9-15. doi: 10.23939/acps2024.01.009.
[28] Matusiak, D.M. (2025). A review of classical lossless data compression algorithms and their contemporary relevance. *Bulletin of the Military University of Technology*, 74(2), 31-40. doi: 10.5604/01.3001.0055.3096.
[29] Maulidina, A.P., Wijaya, R.A., Mazel, K., & Astriani, M.S. (2023). Comparative study of data compression algorithms: Zstandard, zlib & lz4. In A. Mirzazadeh, Z. Molamohamadi, B. Erdebilli, E.B. Tirkolaee & G.-W. Weber (Eds.), *Proceedings of the second international conference "Science, engineering management and information technology"* (pp. 394-406). Cham: Springer. doi: 10.1007/978-3-031-72284-4_24.

[30] MDN Web Docs. (n.d.). *Compression streams API*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Compression_Streams_API.
[31] Meenan, P., & Weiss, Y. (2025). *Compression dictionary transport (RFC 9842)*. Retrieved from https://datatracker.ietf.org/doc/html/rfc9842.
[32] Meta Platforms. (n.d.). *Zstandard*. Retrieved from https://facebook.github.io/zstd/.
[33] Narashiman, S.S., & Chandrachoodan, N. (2024). *Alphazip: Neural network-enhanced lossless text compression*. doi: 10.48550/arXiv.2409.15046.
[34] Node.js v26.2.0. (n.d.). Retrieved from https://nodejs.org/api/zlib.html.
[35] Öztürk, E. (2024). XCompress: LLM assisted Python-based text compression toolkit. *SoftwareX*, 27, article number 101847. doi: 10.1016/j.softx.2024.101847.
[36] Sahu, M., & Panda, J. (2025). *Performance evaluation of efficient hybrid compression methods for devanagiri-encoded Hindi text using lossless algorithms*. doi: 10.48550/arXiv.2504.20747.
[37] Satriyawibawa, M.Y., Andono, P.N., Soong, L.W., & Kiat, N.P. (2024). LSB-2 steganography with Brotli compression and base64 encoding for improving data embedding capacity. *Synchronous: Journal and Research in Informatics Engineering*, 8(2), 878-884. doi: 10.33395/sinkron.v8i2.13264.
[38] Senniappan, G., & Lourdusamy, R. (2025). Bridging the gap between traditional and contemporary compression algorithms: Analytical insights from LZ77 to Zstandard. In *2025 7th international conference on innovative data communication technologies and application (ICIDCA)* (pp. 1374-1379). Coimbatore: Institute of Electrical and Electronics Engineers. doi: 10.1109/ICIDCA66325.2025.11280361.
[39] Shvedov, I. (2024). Comparison of optimization techniques: Resource compression, caching, javascript minification. *Herald of Khmelnytskyi National University. Technical Sciences*, 343(6(1)), 372-379. doi: 10.31891/2307-5732-2024-343-6-55.
[40] Sun, H., Ma, H., Ling, F., Xie, H., Sun, Y., Yi, L., Yan, M., Zhong, C., Liu, X., & Wang, G. (2025). A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data. *Frontiers of Computer Science*, 19, article number 197360. doi: 10.1007/s11704-024-40300-5.
[41] WHATWG. (n.d.). *Compression standard*. Retrieved from https://compression.spec.whatwg.org/.
[42] Wollmer, B., Wingerath, W., Ferrlein, S., Gessert, F., & Ritter, N. (2022a). Compaz: Exploring the potentials of shared dictionary compression on the web. In T. Di Noia, I.-Y. Ko, M. Schedl & C. Ardito (Eds.), *Proceedings of the 22nd international conference "Web engineering"* (pp. 473-476). Cham: Springer. doi: 10.1007/978-3-031-09917-5_38.
[43] Wollmer, B., Wingerath, W., Ferrlein, S., Panse, F., Gessert, F., & Ritter, N. (2022b). The case for cross-entity delta encoding in web compression. In T. Di Noia, I.-Y. Ko, M. Schedl & C. Ardito (Eds.), *Proceedings of the 22nd international conference "Web engineering"* (pp. 177-185). Cham: Springer. doi: 10.1007/978-3-031-09917-5_12.
[44] Zhang, H., Li, C., Xue, M., Zhao, B., & Bao, J. (2025). Optimized snappy compression with enhanced encoding strategies for efficient FPGA implementation. *Electronics*, 14(15), article number 2987. doi: 10.3390/electronics14152987.
[45] Zhang, S., Cao, J., Zhang, M., Xu, M., Han, W., Zhai, Y., & Zhang, Z. (2024). POSTER: Co4U: Efficient and robust HTTP message compression for edge computing networks. In *ACM SIGCOMM Posters and Demos '24: Proceedings of the ACM SIGCOMM 2024 conference: Posters and Demos* (pp. 81-82). New York: Association for Computing Machinery. doi: 10.1145/3672202.3673753.
[46] Zheng, L., Wu, Y., Zhu, M., Ge, W., & Yang, J. (2022). Design and optimization of Zstandard algorithm based on concurrent streaming of multiple hash tables. In *2nd international conference on laser, optics and optoelectronic technology (LOPET 2022)* (pp. 571-576). Bellingham: International Society for Optics and Photonics. doi: 10.1117/12.2649516.

# Алгоритми стиснення даних у сучасних веб-технологіях: аналіз методів Gzip, Brotli та Zstandard для оптимізації передачі контенту

**Дмитро Луцак**
Кандидат технічних наук
Університет короля Данила
76018, вул. Євгена Коновальця, 35, м. Івано-Франківськ, Україна
https://orcid.org/0000-0001-9496-3542

**Анотація.** Метою дослідження було теоретично обґрунтувати та порівняльно оцінити алгоритми безвтратного стиснення Gzip, Brotli та Zstandard у сучасних веб-технологіях для оптимізації передавання контенту. Методологічна база охоплювала теоретичне узагальнення принципів безвтратної компресії, аналіз технічних специфікацій і програмних реалізацій, побудову порівняльної матриці, структурно-функціональне моделювання вебконвеєра та формалізовану оцінку придатності алгоритмів до різних типів вебресурсів. У результаті встановлено, що ефективність стиснення веб-контенту визначалася не лише властивостями конкретного алгоритму, а й структурою ресурсу, рівнем повторюваності даних, режимом формування відповіді за протоколом передавання гіпертексту, підтримкою кодування вмісту, витратами центрального процесора, вимогами до пам'яті та можливостями клієнт-серверної інфраструктури. Показано, що мова розмітки гіпертексту, каскадні таблиці стилів, JavaScript і об'єктна нотація JavaScript є придатними до безвтратної компресії завдяки повторюваності синтаксичних конструкцій, службових символів, тегів, селекторів, фрагментів коду, ключів і вкладених структур. Порівняльний аналіз дав змогу визначити функціональні профілі алгоритмів: Gzip – базове сумісне рішення для статичних і динамічних відповідей за протоколом передавання гіпертексту; Brotli – варіант для попередньо стиснених статичних текстових ресурсів; Zstandard – рішення для швидкісних, потокових і масштабованих сценаріїв за умови інфраструктурної підтримки. Структурно-функціональна модель вебконвеєра компресії дала змогу подати вибір методу як послідовність технічних рішень «тип ресурсу → режим формування → повторюваність структури → підтримка encoding → вимоги до затримки й ресурсів → вибір алгоритму → передавання стисненого потоку → клієнтська декомпресія». Практична значимість результатів полягає у можливості використання запропонованої схеми веброзробниками, системними архітекторами та фахівцями з оптимізації продуктивності для вибору Gzip, Brotli або Zstandard відповідно до типу ресурсу, режиму компресії та можливостей вебінфраструктури

**Ключові слова:** компресія; ентропійне кодування; тип ресурсу; підтримка encoding; вибір алгоритму