Scientific Research Publishing

Open Journal of Applied Sciences, 2026, 16(11), 4737-4749
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Improving Arithmetic Coding by Reducing Interval Calculations for Text Data Compression

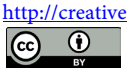
Namogo Silué[1,2,3*], Sié Ouattara[1,2,4], Mouhamadou Dosso[3], Alain Clément[5]

[1]Laboratory of Communication and Information Sciences and Technologies, Houphouët Boigny National Polytechnic Institute, Yamoussoukro, Côte d'Ivoire
[2]Houphouët Boigny National Polytechnic Institute, Yamoussoukro, Côte d'Ivoire
[3]Department of Mathematics and Computer Science, Doctoral School of Science, Technology, and Sustainable Agriculture, Félix Houphouët-Boigny University (UFHB), Abidjan, Côte d'Ivoire
[4]School of Geomatics and Land Use, Abidjan, Côte d'Ivoire
[5]LARIS, SFR MATHSTIC, University of Angers, Angers, France
Email: *siluenamogo@hotmail.fr, sie_ouat@yahoo.fr, mouhamadoudoss@yahoo.fr, alain.clement@univ-angers.fr

How to cite this paper: Silué, N., Ouattara, S., Dosso, M. and Clément, A. (2026) Improving Arithmetic Coding by Reducing Interval Calculations for Text Data Compression. *Open Journal of Applied Sciences*, **16**, 4737-4749.
https://doi.org/10.4236/ojapps.2026.1611262

Received: October 11, 2026
Accepted: November 22, 2026
Published: November 25, 2026

Copyright © 2026 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/



Open Access

Abstract

The steady increase in the volume of digital data has made it essential to develop high-performance compression techniques to reduce storage and bandwidth requirements during data transmission. Among lossless compression methods, arithmetic coding is widely recognized for its efficiency, especially in the compression of text data. Arithmetic coding offers excellent compression ratios for text data; however, its efficiency is often compromised by the computational cost associated with continuously updating the coding intervals. For this reason, in this paper we propose eliminating the intermediate interval calculations for each symbol in a sequence. The results show that the proposed technique for reducing interval calculations significantly improves the performance of arithmetic coding. By eliminating the intermediate steps that are usually required, it reduces the number of calculations performed for each symbol, thereby speeding up the compression process. This simplification does not result in any loss of performance; on the contrary, it enables compression rates higher than those achieved with traditional arithmetic coding. The compression ratios observed with the proposed method range from 86.45% to 96.87%, while those of traditional arithmetic coding range from 85.94% to 87.87%. These results highlight the effectiveness of the approach in terms of both execution time and data compression.

Keywords

Lossless Compression, Entropy, Arithmetic Coding, Interval Calculation

Reduction, Text Data Processing, Compression Performance

1. Introduction

Today, the rapid and dramatic development of digital technologies and the exponential growth of data have made it essential to optimize existing compression techniques [1] [2]. In this context, arithmetic coding has emerged as an effective method of lossless compression [3], widely used in various fields such as image, video, and text processing. By associating a single real number in the interval [0, 1[[4] with an entire message, it approaches Shannon's theoretical entropy [5] and offers optimal compression ratios [6]-[8], particularly for sources with an extended alphabet or highly unbalanced probabilities. However, despite its theoretical advantages, arithmetic coding has certain practical limitations [9] [10]. In particular, the calculation [3] and repeated updating of intervals during the encoding and decoding process result in high computational complexity [11] [12]. Thus, reducing the computational cost [13] of the intervals becomes a major challenge for improving the overall efficiency of this technique. Given this issue, it is reasonable to ask how to optimize arithmetic coding by reducing the computations associated with the intervals while maintaining its compression performance.

With this in mind, we will second analyze the theoretical foundations of arithmetic coding; in Section 3, we will examine the main optimization techniques proposed in the literature; then, in Section 4, we will present our contribution; and in Section 5, we will evaluate the performance of this proposed improvement to arithmetic coding. In Section 6, we will present the results, and in Section 7, we will discuss them. Finally, in Section 8, we will conclude our proposal and suggest directions for future work.

2. Theoretical Foundations of Arithmetic Coding

2.1. The Principle of Arithmetic Coding and Interval Recursion

The algorithm divides the initial interval [0, 1[into subintervals in proportion to the probability of each symbol. When a symbol is processed, the algorithm uses its interval as a new basis, which is then subdivided again for the next symbol. The various stages of the coding process are [14]:

- Let X be a sequence of m symbols such that $X = x_1 x_2 \cdots x_m$ which draws its values from a source $S = \{s_1, s_2, \cdots, s_n\}$,
- Calculate the probability associated with each symbol in source S; let's denote it as p_i = Probability ($S = s_i$).
- Assign a subinterval to each sk symbol [Ls_k, Hs_k[proportional to its probability, with $Hs_k - Ls_k = p_k$.
- Set the lower bound (L) of the working interval to 0 and the upper bound (H) to 1.

- As long as there is still a character left in the sequence to be encoded:
  - width = $H - L$
  - $L = L + \text{width} * Ls_k$
  - $H = L + \text{width} * Hs_k$
- Ultimately, any value in the interval $[L, H[$ uniquely represents the input sequence.

Generally, the lower bound (L) of the interval is chosen as the representative of the sequence, or the mean value may be chosen: $\frac{L+H}{2}$. The value chosen to represent the sequence to be encoded is called code(X). It is important to note that code(X) is represented by a number of bits equal to the value $[-\log_2(P(X))] + 1$.

For example [9]: **Table 1** presents the text "xxyxxzxzxx" compressed using arithmetic coding. The text consists of three letters, $A = \{x, y, z\}$, the alphabet. The probability distributions of this text are $P(x) = 0.7$, $P(y) = 0.1$, and $P(z) = 0.2$, as shown in **Table 1**.

Table 1. Frequencies of letters in "xxyxxzxzxx" text [9].

Symbol	x	y	z
Frequency	7	1	2

And **Figure 1** presents the determination of this alphabetic arithmetic coding label.

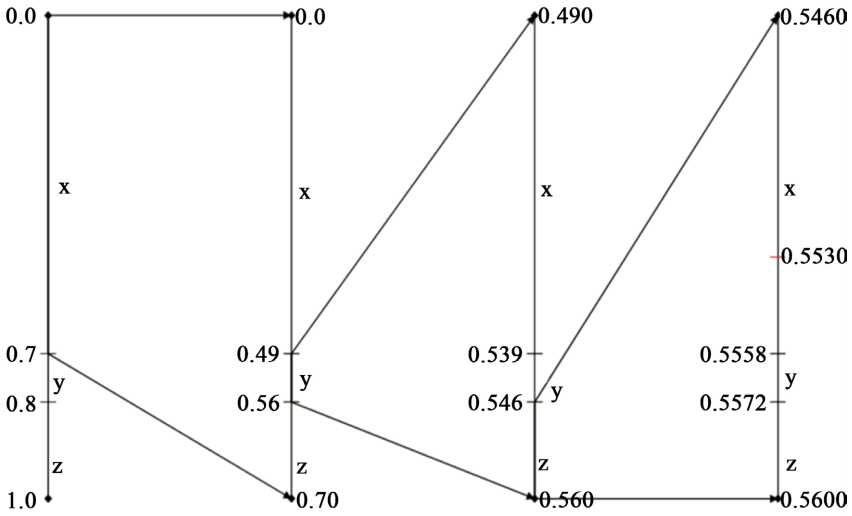


Figure 1. Determination of the alphabetic arithmetic coding label [9].

2.2. Theoretical Foundations and Entropy

Arithmetic coding is directly derived from information theory. The average length of the code generated for N symbols converges to the entropy $H(X)$ of the source. Shannon entropy defines the theoretical minimum number of bits required to encode a source:

$$H(X) = -\sum p(x)\log_2\left[p(x)\right]$$

Thus, with this probabilistic approach, frequent symbols are assigned wider intervals, which shortens the final code.

3. Key Optimization Techniques Proposed in the Literature

There are several approaches to reducing the computational effort involved in calculating intervals in arithmetic coding for text compression. A fundamental approach consists of replacing computationally expensive multiplications with shifts and additions. Chevion *et al*. [15] proposed this generic method, which allows each multiplication to be replaced by a simple binary shift operation and an addition. Theoretical analysis shows that the loss of efficiency is minimal because, for a binary alphabet, the increase in code length is at most 1.1%, resulting in an efficiency of 98.9% compared to exact arithmetic coding. For multivalued alphabets, the maximum loss reaches 1.91% in the worst case. Quasi-arithmetic coding, developed by Howard and Vitter [16], takes a different approach**:** Instead of dynamically calculating the subintervals at each step, all arithmetic operations are performed offline to generate coding tables. When applied to text compression in the Fast PPM (Prediction by Partial Matching) system, quasi-arithmetic coding achieves a compression throughput nearly twice that of PPMC, with a comparable compression ratio. Another approach involves improving probabilistic modeling, which reduces the number of binary decisions that the arithmetic encoder must process. Steinruecken *et al*. [17] proposed PPM-DP, an algorithm that replaces PPM**'**s classical escape mechanism with a generalized **"**blending**"** of predictions from all available contexts, controlled by hyperparameters (α and β). Although this approach slightly increases the computation time per symbol, it significantly improves the compression ratio for human text, outperforming all published variants of PPM (PPMII, PPMZ), CTW, BWT, and LZMA on the Canterbury and Calgary corpora. In the field of context-adaptive binary arithmetic coding (CABAC), Karwowski and Domański [18] proposed replacing the finite-state machines used for probability estimation with the context tree weighting (CTW) technique. This improvement allows for a more accurate calculation of conditional probabilities, which indirectly reduces redundancy and thus the number of computations required to achieve a given compression ratio. Asymmetric numbering systems (ANS), introduced as an alternative to arithmetic coding, combine the speed of Huffman coding with the near-entropy efficiency of arithmetic coding. Dube and Yokoo [19] developed a technique for rapidly constructing quasi-optimal symbol distributions for the tabled ANS (tANS) variant. Their iterative sorting method, based on stationary state probabilities, converges rapidly toward nearly optimal distributions. Hardware architectures for tANS have also been proposed, capable of encoding one symbol per clock cycle, achieving data rates of 146 to 290 megasymbols per second [20]. This performance far exceeds that of traditional arithmetic coding while maintaining comparable compression efficiency. These advances make it possible to consider deploying arithmetic coding in resource-constrained applications where it was pre-

viously impractical.

4. Presentation of Our Contribution

4.1. Principle of Improved Arithmetic Coding

We have decided to reduce the complexity of the calculations. This will reduce the encoding and decoding time. The modified arithmetic coding method we propose allows us to move directly from the first interval to the last interval without processing each intermediate step. It is designed using an approach based on cumulative probabilities and a technique for directly calculating the final interval. The idea is to directly calculate the final interval for the entire message without going through the intermediate steps typical of arithmetic coding. To do this, we use the cumulative probabilities of the symbols to determine the interval corresponding to the complete message.

4.1.1. Calculating Probabilities and Cumulative Values

- For each symbol in the message, we calculate its probability of occurrence.
- Next, we create a set of cumulative probabilities for each symbol, where each entry corresponds to the sum of the probabilities of all preceding symbols.

For example, for a message {*a*, *b*, *c*} with probabilities $P(a)$, $P(b)$, $P(c)$, the cumulative probabilities are:

$$C(a)=0,\ C(b)=P(a),\ C(c)=P(a)+P(b)$$

4.1.2. Direct Determination of the Final Interval

To encode an entire message $M=m_1m_2\cdots m_n$; that is, M is composed of the symbols $m_1,m_2,\cdots,m_n$, so we determine the final interval by multiplying the probabilities associated with each symbol and using the cumulative probabilities.

- Lower bound of the interval: L_M

$$\begin{aligned}L_M &= C(m_1)+P(m_1)\times C(m_2)+P(m_1)\times P(m_2)\times C(m_3)+\cdots\\ &\quad +P(m_1)\cdots P(m_{n-1})\times C(m_n)\end{aligned}$$

- Upper bound of the interval: H_M

$$H_M = L_M + P(m_1)\times P(m_2)\times\cdots\times P(m_n)$$

4.1.3. Direct Compression

The message is then represented by a number in the interval $[L_M, H_M[$. According to [15], the sequence's representative can be the lower bound (L) of the interval, or one can choose the mean value: $\frac{L+H}{2}$. And we will call code(X) the value chosen to represent the sequence to be encoded. Furthermore, the code (X) is represented by a number of bits equal to the value $[-\log_2(P(X))]+1$.

4.2. Implementation of the Proposed Algorithm

We implement the proposed algorithm using MATLAB.

4.2.1. Pseudo-Code for the Proposed Algorithm

Input: Message $M = m_1\ m_2 \ldots m_n$

Output: code(X), bits

1. Calculate the probabilities P(s) for each symbol s

2. Calculate the cumulative probabilities C(s):

C(first symbol) ← 0

For each subsequent symbol s:

C(s) ← C(previous symbol) + P (previous symbol)

3. Compute the lower bound L_M:

$L_M \leftarrow 0$

product ← 1

For i from 1 to n:

$L_M \leftarrow L_M + \text{product} \times C(m_i)$

$\text{product} \leftarrow \text{product} \times P(m_i)$

4. Calculate the upper bound H_M:

$H_M \leftarrow L_M + \text{product}$

5. Choose the representative code(X):

code(X) ← L_M // lower bound

// or

code(X) ← $(L_M + H_M) / 2$ // average value

6. Calculate the number of bits:

$P(X) \leftarrow \text{product}$ // $P(X) = P(m_1) \times P(m_2) \times \ldots \times P(m_n)$

$\text{bits} \leftarrow \text{floor}(-\log_2(P(X))) + 1$

7. Return code(X) and bits

4.2.2. Pseudo-Code for Decoding the Proposed Algorithm

Input: code(X), message length n, tables P and C

Output: Message $M = m_1\ m_2 \ldots m_n$

1. Initialize:

low ← 0

high ← 1

value ← code(X)

2. For i from 1 to n, do:

range ← high – low

pos ← (value – low) / range

// Find the symbol mi such that $C(m_i) \leq pos < C(m_i) + P(m_i)$

For each symbol s:

if $C(s) \leq pos < C(s) + P(s)$ then

$m_i \leftarrow s$

exit the loop

// Update the interval

$\text{high} \leftarrow \text{low} + \text{range} \times (C(m_i) + P(m_i))$

$\text{low} \leftarrow \text{low} + \text{range} \times C(m_i)$

// Add the decoded symbol

$M \leftarrow M + m_i$

3. Return the message M

5. Comparison of Conventional Arithmetic Coding and Improved Arithmetic Coding

For the evaluation, let's use the following metrics from [6]: compression ratio, average code length, and source entropy. And the following datasets:

Data 0**:** aaaaaaaa

Data 1: arararar

Data 2: abcabcabcabc

Data 3: abcdabcdabcdabcd

Let's consider variables such as the initial volume (IV), the compressed volume (CV), the compression ratio (CR) of our data, the average code length (L), and the source entropy (H).

- $\text{Compression ratio} = \dfrac{\text{Initial Volume} - \text{Compressed Volume}}{\text{Initial Volume}}$
- $L = \dfrac{\text{number of code bits}}{\text{number of initial characters}}$
- $H = \sum_{k=1}^{N} P_k \log_2 \dfrac{1}{P_k}$

6. Results

6.1. Comparison of the Two Methods Using Encoding

Table 2 and Table 3 summarise the applications of the metrics in Section 5 and show a comparison based on encoding.

6.2. Verification of the Proposed Method through Decoding

We choose the following sequences: "abab" and "abba." These must be assigned distinct representations and exact reconstructions. These strings share their first symbol and their symbol frequencies, which would allow us to verify that the method preserves the order of the symbols.

Table 2. Traditional arithmetic coding.

Traditional Arithmetic Coding						
Data \ Settings	Data Code	Initial Volume (bit)	Final Volume (bit)	Average Length code (bit/symbol)	Source Entropy (bit/symbol)	Compression ratio (%)
Data 0	aaaaaaaa	64	2	2	0	87.87
Data 1	arararar	64	9	4.5	1	85.94
Data 2	abcabcabcabc	96	21	7	1.583	78.12
Data 3	abcdabcdabcdabcd	128	21	5.25	2	83.59

Table 3. Improved arithmetic encoding.

Traditional Arithmetic Coding						
Settings / Data	Data Code	Initial Volume (bit)	Final Volume (bit)	Average Length code (bit/symbol)	Source Entropy (bit/symbol)	Compression ratio (%)
Data 0	aaaaaaaa	64	2	2	0	96.87
Data 1	arararar	64	4	2	1	93.75
Data 2	abcabcabcabc	96	13	4.33	1.583	86.45
Data 3	abcdabcdabcdabcd	128	9	2.25	2	92.96

The encoding of "abab" yields a code word equal to 0.34375, and that of "abba" yields a code word equal to 0.40625.

Starting with 0.34375 and applying the decoding procedure described above, we obtain exactly "abab." The same is true for 0.40625, which yields "abba." We can see that the decoding is reversible and the representations are distinct.

However, it is important to specify the complete compressed representation, which goes beyond the simple estimate $[-\log_2(P(X))] + 1$. It is the complete binary output that allows the decoder to reconstruct the message independently. This output consists of three concatenated parts: header, probability model, and arithmetic code.

➢ Header

- Number of distinct symbols (8-bit encoded)
- Exact length of message n (encoded as 16 bits or 32 bits)

➢ Probability model

For each character in the alphabet (in a fixed order, such as lexicographical order):

- The symbol itself (if the alphabet is not known in advance)
- Its full frequency

The decoder then recalculates:

$P(s) = \frac{\text{frequency}(s)}{n}$, $C(s)$ = cumulative sum of the previous probabilities

➢ Arithmetic code

- The value code(x) (the mean of the interval [L_M, H_M]) or the lower bound)
- Encoded as $k = \left[-\log_2\left(P(X)\right)\right] + 1$ bit

The decoder obtains the information as follows in **Table 4**:

Table 4. Information obtained by the decoder.

Information	Source in the compressed stream
Alphabet Σ	List of symbols transmitted in the model
Probability model	Frequency: calculation of $P(s)$ and $C(s)$
Message length n	Explicit field in the header
End of message	Determined the value of n

- Measuring the actual compressed size
  The total size (in bits) is:
  Compressed size = header size + probability model size + k where $k = \left[-\log_2\left(P\left(X\right)\right)\right] + 1$. We no longer rely solely on the estimate k. We measure the actual size of the complete binary stream.

❖ Example using the word: banana ($n = 6$)

- Alphabet = {a, b, e, n}: 4 symboles
- Frequencies: a (2), b (1), e (1), n (2)
- K = 11 bits (based on the exact probabilities)

6.3. Comparing the Two Methods over Time

We supplement the encoding and decoding codes for both methods with a built-in time-measurement function (similar to MATLAB's 'tic/toc') along with the standard deviation, which measures the dispersion of execution times around the mean. To do this, we use the following hardware and environment:

- Computer: ASUS (model TUF Gaming F15)
- Processor: Intel Core i7-11800H @ 2.30 GHz
- RAM: 16 GB DDR4
- Operating System: Windows 11 (64-bit)
- MATLAB Version: R 2023b
- Number of repetitions: 50 independent runs for each word and each method
- Measurement: total time, including probability estimation, cumulative sum calculation, and all preprocessing
  For a summary of the durations, see Table 5.

Table 5. Summary of times (Mean ± Standard deviation).

Sequences	Length	Method	Encoding Time (ms)	Decoding Time (ms)
Banana	6	Traditional	0.182 ± 0.021	0.215 ± 0.024
Banana	6	Enhanced	0.094 ± 0.012	0.187 ± 0.019
Commission	10	Traditional	0.247 ± 0.028	0.298 ± 0.031
Commission	10	Enhanced	0.113 ± 0.015	0.241 ± 0.022
Philosophy	11	Traditional	0.261 ± 0.029	0.312 ± 0.033
Philosophy	11	Enhanced	0.118 ± 0.014	0.253 ± 0.025
Computer Science	12	Traditional	0.279 ± 0.032	0.335 ± 0.036
Computer Science	12	Enhanced	0.124 ± 0.016	0.268 ± 0.027

We note that the proposed method reduces the time required for encoding (by approximately 45% to 55%) and also for decoding compared to the traditional method.

6.4. Strengthening the Evaluation of the Proposed Technique

To make general statements about text compression, we will expand the evaluation of our algorithmic contribution using a named reference corpus. We have chosen the Canterbury Corpus, and the selected (text) files are as follows (see Table 6):

Table 6. Selected (text) files from the canterbury corpus.

File	Description	Approximate size
Alice29.txt	Literary text	152 ko
Asyoulik.txt	Play (Shakespeare)	125 ko
Lcet10.txt	Technical text	426 ko
Plrabn12.txt	Poetry (Paradise Lost)	481 ko
Grammar.lsp	Lisp source code	3.7 ko

- Documented pretreatment

  For each file**:**

  1) Read the file in text mode

  2) Convert to a sequence of symbols (characters)

  3) Calculate the empirical 0th-order frequencies across the entire file

  4) Construct the $P(s)$ and $C(s)$ tables

  5) No other actions or transformations (no filtering, character removal, or normalization)

  This extension allows us to evaluate our proposal using real, diverse text data not just repetitive synthetic strings.

6.5. Explanation of the Numerical Reliability of the Proposed Technique

It must be noted that the LM direct calculation and the calculation of the product $P(x)=\prod P(m_i)$ are numerically unstable. In MATLAB double precision (double), the product rapidly underflows to 0 when the message length is a few dozen symbols, making the result unrecoverable. However, to manage finite precision and underflow, the MATLAB implementation uses arbitrary precision via the VPA (Variable Precision Arithmetic) function from the Symbolic Math Toolbox. This allows us to maintain a nonzero product and an interval [LM, HM[that can be used even for long messages. However, there is a possible alternative. This involves working in the logarithmic domain $\left(\sum \log P(m_i)\right)$ to completely avoid underflow while remaining in double precision. To do this, you must satisfy the supported input conditions (see Table 7).

We can see that numerical errors are detected. These checks ensure that the results can be recovered and that all numerical degradation is explicitly reported.

Table 7. Covered admission conditions.

Condition	Supported	Action
Strictly positive probabilities	Yes	No
Sum of probabilities = 1	Yes	Verified at input
Length $n \leq 30$ (double)	Yes	No
Length $n > 30$	Yes	Only with vpa
Zero probability	No	Reject with error
Empty message	No	Reject with error

7. Discussion of Improved Arithmetic Coding

The use of our technique for reducing interval calculations demonstrates an excellent balance between execution speed and compression ratio. Most importantly, this method allows us to bypass the traditional intermediate steps of arithmetic encoding, which saves time by avoiding successive calculations for each symbol. Furthermore, we observe that the compression ratios are significantly higher than in the conventional approach. They range from 86.45% to 96.87%, compared to 85.94% and 87.87% for traditional arithmetic coding.

8. Conclusions and Perspective

This work has demonstrated that it is possible to overcome the main technological barrier to arithmetic coding: its computational slowness. The implemented applications show that this approach offers a particularly advantageous trade-off between execution speed and compression efficiency. By eliminating the successive intermediate calculations associated with the processing of each symbol, the method simplifies the coding process while improving overall performance. The results also confirm a significant increase in compression ratios compared to the conventional method. Thus, the proposed technique constitutes a relevant and effective alternative for optimizing compression systems based on arithmetic coding.

- Future work could focus on evaluating this technique on more diverse and larger datasets to confirm its robustness and generalizability.
- It would also be interesting to explore its integration into real-time compression systems or resource-constrained environments, where reducing computation time is a major challenge.
- Finally, further research could focus on combining this method with more advanced probabilistic models or adaptive compression techniques in order to further improve compression performance and the system's overall efficiency.

Author Contributions

We present this taxonomy of contributor roles in order to recognise authors' in-

dividual contributions, minimise disputes regarding authorship, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Namogo Silué	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Sié Ouattara		✓				✓		✓	✓	✓	✓	✓		
Mouhamadou Dosso	✓		✓	✓			✓			✓	✓		✓	✓
Alain Clément	✓		✓	✓			✓			✓	✓	✓	✓	✓

C: Conceptualization; M: Methodology; So: Software; Vo: Validation; Fo: Formal analysis; I: Investigation; R: Resources; D: Data Curation; O: Writing—Original Draft; E: Writing—Review & Editing; Vi: Visualization; Su: Supervision; P: Project administration; Fu: Funding acquisition.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Hu, Y., Liu, Y. and Yang, Y. (2025) Adaptive Arithmetic Coding-Based Encoding Method toward High-Density DNA Storage. *Journal of Computational Biology*, **32**, 298-315. https://doi.org/10.1089/cmb.2024.0697

[2] Hosseini, M. (2025) A Survey of Data Compression Algorithms and Their Applications. arXiv: 2506.10000. https://doi.org/10.48550/arXiv.2506.10000

[3] Khairi, N.A., Jambek, A.B. and Ismail, R.C. (2019) Performance Evaluation of Arithmetic Coding Data Compression for Internet of Things Applications. *Indonesian Journal of Electrical Engineering and Computer Science*, **13**, 591-597. https://doi.org/10.11591/ijeecs.v13.i2.pp591-597

[4] Tsai, C.H. (2026) Revisiting Data Compression with Language Modeling. arXiv: 2601.02875v1.

[5] Duda, J. (2014) Asymmetric Numeral Systems: Entropy Coding Combining Speed of Human Coding with Compression Rate of Arithmetic Coding. arXiv: 1311.2540v2.

[6] Silué, N., Ouattara, S., Dosso, M. and Clément, A. (2024) Quantitative Comparative Study of the Performance of Lossless Compression Methods Based on a Text Data Model. *Open Journal of Applied Sciences*, **14**, 1944-1962. https://doi.org/10.4236/ojapps.2024.147127

[7] Townsend, J., Bird, T. and Barber, D. (2019) Practical Lossless Compression with Latent Variables Using Bits Back Coding. arXiv: 1901.04866v1.

[8] Hu, C. and Tang, J. (2026) A Model-Driven Lossless Compression Algorithm Resistant to Mismatch. arXiv: 2601.17684v1.

[9] Erdal, E. and Ergüzen, A. (2019) An Efficient Encoding Algorithm Using Local Path on Huffman Encoding Algorithm for Compression. *Applied Sciences*, **9**, Article 782. https://doi.org/10.3390/app9040782

[10] Zabolotnii, S., Yarmilko, A., Rozlomii, I. and Mysiura, Y. (2022) Applying the Arithmetic Compression Method in Digital Speech Data Processing. CEUR Workshop Proceedings, Vol-3624: Selected Papers of the X International Scientific Conference "Information Technology and Implementation" (IT&I-2023). *Conference Proceedings*, Kyiv, 20-21 November 2023, 170-179.

https://doi.org/10.55056/ceur-ws.org/vol-3624/paper_15.pdf

[11] Upadhyay, V., Kokila, J. and Vaish, A. (2025) Optimized Arithmetic Coding for Efficient Data Compression in the Resource-Constrained Internet of Things(IoT). arXiv: 2508.08840.

[12] Jayasankar, U., Thirumal, V. and Ponnurangam, D. (2021) A Survey on Data Compression Techniques: From the Perspective of Data Quality, Coding Schemes, Data Type and Applications. *Journal of King Saud University—Computer and Information Sciences*, **33**, 119-140. https://doi.org/10.1016/j.jksuci.2018.05.006

[13] Guo, Z., Fu, J., Feng, R. and Chen, Z. (2021) Accelerate Neural Image Compression with Channel-Adaptive Arithmetic Coding. 2021 *IEEE International Symposium on Circuits and Systems* (*ISCAS*), Daegu, 22-28 May 2021, 1-5. https://doi.org/10.1109/iscas51556.2021.9401277

[14] AbuSafiya, M. (2020) Text Compression Based on Letter's Prefix in the Word. *Computers, Materials & Continua*, **64**, 17-30. https://doi.org/10.32604/cmc.2020.09282

[15] Chevion, D., Karnin, E.D. and Walach, E. (1991) High Efficiency, Multiplication Free Approximation of Arithmetic Coding. *Proceedings of the IEEE Data Compression Conference, DCC* 1991, Snowbird, 8-11 April 1991, 43-52.

[16] Howard, P.G. and Vitter, J.S. (1993) Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding. [*Proceedings*] *DCC* 93: *Data Compression Conference*, Snowbird, March 30-1 April 1993, 98-107. https://doi.org/10.1109/dcc.1993.253140

[17] Steinruecken, C., Ghahramani, Z. and MacKay, D. (2015) Improving PPM with Dynamic Parameter Updates. 2015 *Data Compression Conference*, Snowbird, 7-9 April 2015, 193-202. https://doi.org/10.1109/dcc.2015.77

[18] Karwowski, D. and Domański, M. (2016) Context-Adaptive Binary Arithmetic Coding with Precise Probability Estimation and Complexity Scalability for High-Efficiency Video Coding. *Journal of Electronic Imaging*, **25**, 1-16. https://doi.org/10.1117/1.jei.25.1.013010

[19] Dubé, D. and Yokoo, H. (2019) Fast Construction of Almost Optimal Symbol Distributions for Asymmetric Numeral Systems. 2019 *IEEE International Symposium on Information Theory* (*ISIT*), Paris, 7-12 July 2019, 1682-1686. https://doi.org/10.1109/isit.2019.8849430

[20] Najmabadi, S.M., Wang, Z., Baroud, Y. and Simon, S. (2015). High Throughput Hardware Architectures for Asymmetric Numeral Systems Entropy Coding. 2015 9*th International Symposium on Image and Signal Processing and Analysis* (*ISPA*), Zagreb, 7-9 September 2015, 256-259. https://doi.org/10.1109/ispa.2015.7306068