Arithmetic N-gram: an efficient data compression technique

Ali Hassan[1] · Sadaf Javed[1] · Sajjad Hussain[1] · Rizwan Ahmad[1] · Shams Qazi[1]

Received: 3 April 2023 / Accepted: 5 February 2024
Published online: 13 March 2024

Abstract
Due to the increase in the growth of data in this era of the digital world and limited resources, there is a need for more efficient data compression techniques for storing and transmitting data. Data compression can significantly reduce the amount of storage space and transmission time to store and transmit given data. More specifically, text compression has got more attention for effectively managing and processing data due to the increased use of the internet, digital devices, data transfer, etc. Over the years, various algorithms have been used for text compression such as Huffman coding, Lempel-Ziv-Welch (LZW) coding, arithmetic coding, etc. However, these methods have a limited compression ratio specifically for data storage applications where a considerable amount of data must be compressed to use storage resources efficiently. They consider individual characters to compress data. It can be more advantageous to consider words or sequences of words rather than individual characters to get a better compression ratio. Compressing individual characters results in a sizeable compressed representation due to their less repetition and structure in the data. In this paper, we proposed the ArthNgram model, in which the N-gram language model coupled with arithmetic coding is used to compress data more efficiently for data storage applications. The performance of the proposed model is evaluated based on compression ratio and compression speed. Results show that the proposed model performs better than traditional techniques.

Keywords Data compression · Huffman · LZW · N-gram · Arithmetic coding · Compression ratio · Large language models

1 Introduction

With the significant development of digital technology, according to a statistics report [1], the number of Internet of Things (IoT) devices worldwide is forecast to triple from 9.7 billion in 2020 to more than 29 billion in 2030. IoT devices such as sensors, actuators, smart mobiles, appliances, or machines are programmed to transmit data over the internet for specific applications. Some IoT devices continuously monitor environmental factors and send critical information to the control center every second. Such IoT devices require high data rates, low latency, and space-efficient storage of this big data. Similarly, big data has gained attention in many fields. In the medical field, researchers use big data to find disease risk factors and symptoms to assist doctors in diagnosing diseases and medical disorders. Different companies use big data to enhance operations and boost customer service. In artificial intelligence (AI), learning models rely on big data to effectively learn future trends. However, the exponential growth of data from different sources, such as social

✉ Rizwan Ahmad, rizwan.ahmad@seecs.edu.pk; Ali Hassan, ahassan.phdee21seecs@seecs.edu.pk; Sadaf Javed, sjaved.phdee22seecs@seecs.edu.pk; Sajjad Hussain, sajjad.hussain2@seecs.edu.pk; Shams Qazi, shams.qazi@seecs.edu.pk | [1]School of Electrical Engineering and Computer Sciences (SEECS), National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan.

media, smart sensors, IoT devices, mobile phones, medical devices, and online transactions, makes processing, storing, and transferring big data challenging.

In this era of the digital world and limited resources, there is a need for efficient data compression techniques for the storage and transmission of data. Compression results in efficient usage of available storage and transmission bandwidth [2]. There are two main categories of data compression techniques: lossy compression and lossless compression [3, 4]. Lossy compression techniques compress the data with loss of certain information which can not be recovered. Effective compression can be achieved using lossy techniques but these techniques are only limited to the application where certain loss is acceptable such as video conferencing, stream media, multimedia, etc. On the other hand, lossless techniques compress the data while preserving all the information, which means original data can be reconstructed. Lossless techniques are used for medical image compression, text compression, etc., where losing details is unacceptable.

In literature, most of the compression techniques are based on compressing individual letters of each character. Such compression techniques may not be effective in some cases where data is composed of repeated words or sequences of words. In such cases, compression based on words or a sequence of words can be more efficient and significantly reduce the space required for data storage applications such as digital libraries, archives, datasets for learning, sensor networks, etc. In addition, compression plays a significant role in compressing information in large language models (LLMs) like ChatGPT, PaLM2, and LLaMA (LLM Meta AI) for reducing storage and memory requirements. These models are mostly accessed over a cloud-based system. Transmitting large amounts of data over a network makes transmission slow and requires large bandwidth. The efficient compression can significantly enhance the transmission process and reduce transmission time to improve user experience. It can enable us to deploy these large models without massive hardware investment [5]. In this paper, we proposed the ArthNgram model, in which the N-gram language model coupled with arithmetic coding is used to compress data more efficiently for data storage applications. In this model, the N-gram language model is used to find the probability distribution and sequence of words. Then arithmetic encoding is used to compress data based on that distribution. The contributions of this work are as follows:

- Proposed the ArthNgram model that integrates the N-gram language model and arithmetic coding.
- Evaluated proposed ArthNgram model for $N = 1, 2, 3, 4, 5$ based on performance metrics: compression ratio, time, complexity, and speed.
- Performed the comparative analysis of the proposed model with traditional compression techniques: Huffman coding, LZW coding, and arithmetic coding.

The rest of this paper is structured as follows. In Sect. 2, traditional compression techniques are discussed. In Sect. 3, the proposed model is discussed. Section 4 provides the performance metrics. Results are discussed in Sect. 5. Finally, in Sect. 6, the proposed paper is concluded.

2 Related work

Since the use of the internet and the growing number of IoT devices, digital storage data systems, text file transfer, and embedded systems have significantly increased, text compression approaches have drawn more attention to managing and leveraging this data effectively. It is used in various applications, including data storage and transmission, digital archives, and cloud computing. It helps to reduce the storage space required for text data, improve transmission efficiency over slow or limited bandwidth networks, reduce the cost of storing and transmitting large amounts of text data, and allow quick and effortless transfer of large text files between devices. Text compression is a type of data compression in which text is compressed based on lossless compression techniques. The methods and technology for text compression are always being researched to be made better. Different lossless compression techniques have been used for text compression, such as Huffman, Shannon-Fano coding, arithmetic coding, run-length, Lempel-Ziv-Welch (LZW) coding, etc [6]. In [7], authors presented an alternative run-length approach based on Burrows-Wheeler Transform (BWT) and arithmetic coding for text compression. Their approach converts the large text file into smaller text files, with each containing the same number of characters. Each small file is transformed and compressed using BWT and run-length coding. Then, further compression is achieved using arithmetic coding. In [8], a text file in 8 different languages is compressed using compression techniques (LZW coding, Arithmetic Coding) to observe the effect of different languages on compression ratio. It is noted that the compression ratio varies with text language. Another lossless approach is proposed based on Huffman coding for text compression. This approach is also implemented on FPGA for validating the working of the proposed algorithm [9]. In [10], run-length coding and delta coding-based compression technique is proposed for compressing medical data with improved compression ratio. Authors in [11] proposed a technique based on Huffman coding in order to improve the compression ratio and to enhance image compression's quality. In [12], a hybrid approach based on LZW and Huffman coding is proposed where data is first compressed by Huffman coding and then by LZW. It is observed that the proposed hybrid approach provides better compression than these two techniques individually. In [13], a dictionary-based technique is proposed for text compression based on the quaternary Huffman technique. Huffman with quaternary tree architecture is used instead of binary tree architecture to improve traversing time and compression ratio. In [14], n-gram dictionaries up to 5 gs are used to compress text in the Vietnamese language. The authors in [15] present a compression technique based on Shannon-Fano and Huffman algorithm to compress large files, more specifically in mobile devices. The dynamic text compression algorithm is proposed based on Huffman and LZW using n-gram dictionaries up to 2-grams [16]. In [17], a graph compression technique is proposed based on Huffman coding, pattern detecting, and matching principles to handle big data issues on resource-constrained IoT devices. A survey is conducted in [18] on deep learning-based models for compression. Authors in [19] provide a comparative study of Huffman and LZW techniques on 12 different test files of variable size. Compression plays a significant role in the practical deployment of LLMs by reducing their size and computational requirements. Authors in [20] proposed the activation-aware quantization method for LLMs compression. In [21], a structural pruning method is proposed for LLMs compression which compresses LLMs while maintaining their linguistics capabilities.

3 Compression techniques

This section gives a brief overview of different well-known compression techniques such as Huffman coding, LZW coding, arithmetic coding, etc.

3.1 Huffman coding

Huffman coding is a lossless compression technique that is based on the frequency of occurrence of symbols. It gives a variable length codeword to each symbol. It assigns shorter codewords to the symbols with a high frequency of occurrence and longer codewords to symbols with a low frequency of occurrence. It uses the prefix rule in order to ensure that the codeword assigned to any symbol is not the prefix of another symbol. The main steps of Huffman coding are following:

1. Arrange the probabilities of symbols in descending order and consider them as the nodes
2. Repeat the below steps until all nodes form a single tree

   (a) Select two nodes with the smallest probability
   (b) Merge them to form a new node whose probability is sum of these two nodes.

3. Traverse tree to get codewords for each symbol

The time complexity of Huffman coding is $O(nlogn)$ where n represents the number of unique symbols in data to be compressed.

3.2 LZW coding

LZW is a dictionary-based lossless compression technique. It is used in many applications such as Win-zip, GIF, 7zip, etc. LZW replaces the character's strings with a single code. The idea of this technique depends on reappearing patterns. It does not involve the analysis of incoming data. During the encoding process, it constructs the indexed dictionary in order to compress the data. LZW coding involves the following steps:

1. Initialize by creating an indexed dictionary with the single-character entries
2. Read each character of the input data one at a time into a buffer.
3. Check whether the buffer exists in the dictionary or not

4. If it exists, add a subsequent character to the buffer and repeat step 3.
5. If it does not exist, add it to the dictionary as a new entry and get the buffer's index in the dictionary as an output
6. Add the next subsequent character to the buffer and repeat from step 2 until the end of input data

The time complexity of LZW coding is $O(n)$. This is because it constructs a dictionary while character-by-character processing of the input data containing n characters.

3.3 Arithmetic coding

Arithmetic encoding is a technique used for lossless data compression which encodes data to be transmitted into a string of 1 s and 0 s based on probabilities on the number line between 0 and 1 [22, 23]. It is used in various applications such as image compression, text compression, data transmission, etc. As the size of the data to be encoded increases, the interval on the real line between 0 and 1 becomes smaller and the corresponding binary code string of that interval grows. It involves the following steps:

1. Map the input data over the range [0, 1] based on frequency of occurrence of symbols.
2. Divide the current range into sub-ranges based on the probability of each symbol
3. Select the sub-range associated with the next symbol to be encoded and consider it as a new current range
4. Repeat step 2 and 3 until end of symbols

The time complexity of this technique is $O(mn)$, where m is the total number of characters in data and n is the length of data.

To summarize, we have presented three benchmark techniques that will be used for comparison of our proposed scheme.

4 Proposed model

This section presents the proposed model for text compression. The motivation for the proposed model comes from the fact that modern systems have repetition of data and it can be more advantageous to consider words or sequences of words rather than individual characters to get a better compression ratio. Figure 1 shows the flow chart of the proposed model where we integrated the N-gram language model with arithmetic coding to compress text data effectively. In the proposed Model, first, we constructed the N-grams and calculated their probability distributions using the N-gram language model. Then, data is encoded using arithmetic coding based on these probability distributions.

4.1 N-gram model and N-gram construction

N-gram refers to a sequence of N words that is present in the text. The N-gram language model is used to predict the probabilities of N-grams in a given text. It is used in many applications such as spelling error detection and correction, text compression, language identification, etc. In order to generate N-grams from given text data considering ($N = 1, 2, 3, 4, 5$) respectively, the following steps are considered:

- split the text into tokens/smaller units with window size and added enough blank spaces before and after the token.
- Scanned all the tokens to generate all possible N-grams for ($N = 1, 2, 3, \dots 5$) and set a counter to each N-gram to get the frequencies of each unigram, bigram, trigram, four-gram, and five-gram.
- Calculated the probability of each N-gram based on their frequencies using the N-gram probability distribution model.

We considered N-gram based compression model for ($N = 1, 2, 3, 4, 5$) which differ in their probability distributions.

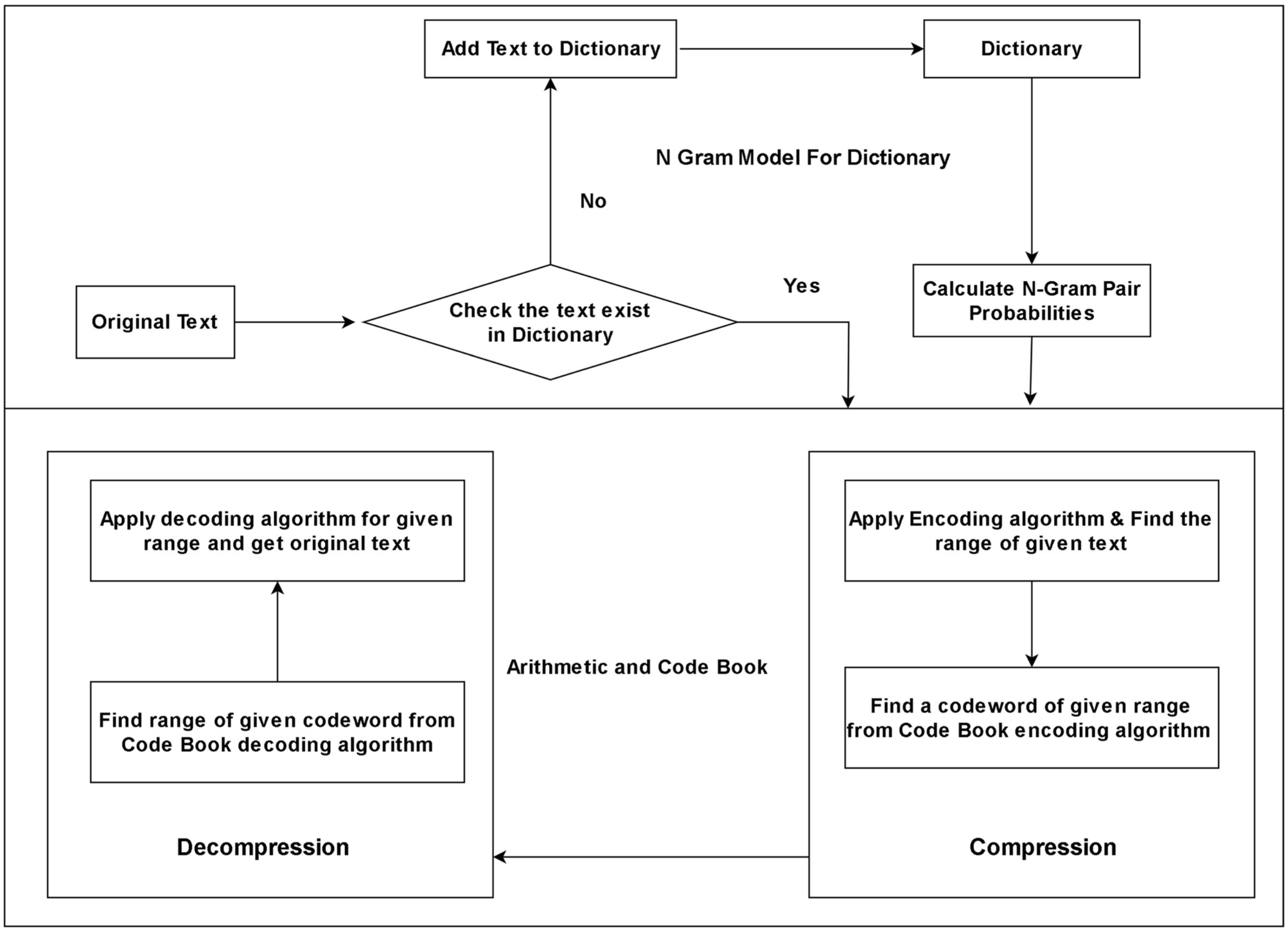
Add Text to Dictionary
Dictionary
N Gram Model For Dictionary
No
Yes
Original Text
Check the text exist in Dictionary
Calculate N-Gram Pair Probabilities
Apply decoding algorithm for given range and get original text
Find range of given codeword from Code Book decoding algorithm
Decompression
Arithmetic and Code Book
Apply Encoding algorithm & Find the range of given text
Find a codeword of given range from Code Book encoding algorithm
Compression

Fig. 1 Flow chart of proposed compression technique

4.2 Aithmetic N-gram (ArthNgram)

ArthNgram model couples the N-gram language model with arithmetic coding to compress data more efficiently. In this work, the ArthNgram model is considered for five different N-gram cases: Arithmetic Unigram (Arth1 g), Arithmetic Bigram (Arth2 g), Arithmetic Trigram (Arth3 g), Arithmetic Four-gram (Arth4 g), and Arithmetic Five-gram (Arth5 g). Figure 2

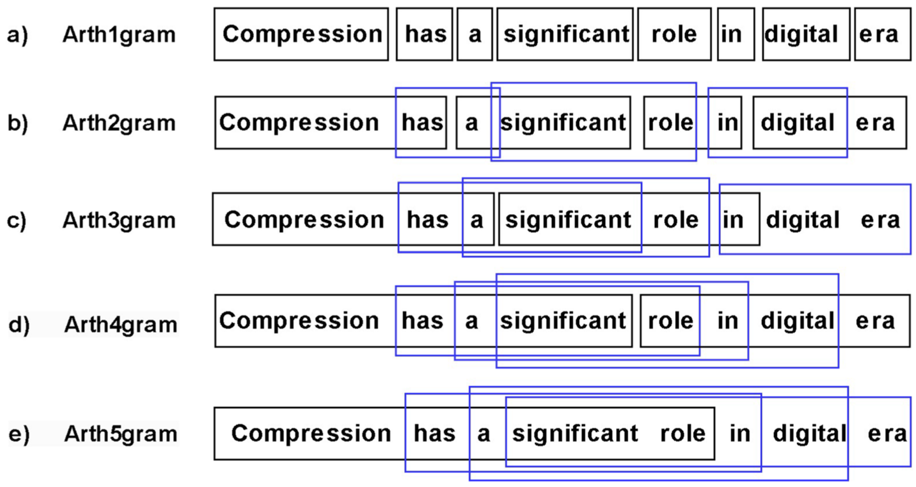
a) Arth1gram
Compression has a significant role in digital era
b) Arth2gram
Compression has a significant role in digital era
c) Arth3gram
Compression has a significant role in digital era
d) Arth4gram
Compression has a significant role in digital era
e) Arth5gram
Compression has a significant role in digital era

Fig. 2 ArthNgram Sequences

shows the sequence of words considered for five different N-gram cases. In an arithmetic unigram case, unigram-based arithmetic coding is used to compress the data. Each token in a unigram model is taken into account independently from every other token and the probability of each token depends on how frequently it appears in the data. In this case, the probability of a certain word sequence does not depend on its co-occurrence with the previous words. So, the probability of certain words sequence containing n unigrams can be calculated as follows:

$$P_r(w_1, w_2, ..., w_n) \approx \prod_{k=1}^{n} P_r(w_k) \tag{1}$$

In the case of other ArthNgram cases, the probability of a word in word sequence depends on its co-occurrence with the $(N-1)$ previous words in the data where $N = 1, 2, 3, 4, 5$. So, the generalized expression for the probability of certain word sequence containing n N-grams (bigrams, trigrams, four-grams, and five-grams) can be expressed as follows:

$$\begin{gathered} P_r(w_1, w_2, ..., w_n) \approx \prod_{i}^{K-1} P_r(w_i \mid w_{i-1}) \\ \prod_{k=m}^{n} P_r(w_k \mid w_{k-1,k-2,...,k-i}) \end{gathered} \tag{2}$$

Where $i = 1, 2, .., (k-1)$, $m = 2, 3, 4, 5$, and $w_0 = 1$.

4.3 N-gram dictionary

N-gram dictionary contained all the N-grams (unigram, bigram, trigram, 4-gram, and 5-gram) of the data with their probability distributions. It is assumed to be available on both ends: compression and decompression.

4.4 ArithNgram encoding

The ArithNgram Encoding uses the N-gram probabilities obtained from the dictionary to compress the text data using an arithmetic encoding algorithm. The first step is to find the interval lie in [0,1) for data to be compressed. For that, it works as follows:

1. We started with the current interval [l,h) as [0,1).
2. For each N-gram in the file, two steps are taken.

    (a) For each possible N-gram, We split the current interval [l,h) into sub-intervals based on their probabilities.
    (b) The sub-interval corresponds to the next actual N-gram and is selected as a new interval.

In the above step, we only compute the interval corresponding to the actual N-gram g_i using their cumulative probabilities as:

$$P_{cur} = \sum_{k=1}^{i-1} p_k(g) \tag{3}$$

$$P_{next} = P_{cur} + \sum_{k=1}^{i-1} p_k(g) \tag{4}$$

The new sub-interval is obtained as $(l + P_{cur}(h-l), l + P_{next}(h-l))$. To specify the end of the file, we assumed the file's length to be known. This encoding process is summarized in Algorithm 1, where *seq* represents the input text and *vocab* consists of N-grams, their probabilities, and min_r and max_r that represent the minimum and maximum value of the interval/ range of each n-gram. The *vocab* is assumed to be available on both encoding and decoding ends.

After getting the interval lies on the number line, a binary code string is generated using a codebook encoding algorithm for given data based on that interval.

Algorithm 1 Encoding Algorithm

```
input  : $seq$, $vcab$
output: $min_r$, $max_r$
Initialize: $min_r = 0$, $max_r = 1$
for $i = 1$ to $N_w$ do
    range=$max_r - min_r$;
    update the high value with $[low + range * Vcab(word(high))]$;
    update the low value with $[low + range * Vcab(word(low))]$;
end
```

4.5 Codebook

The codebook is used to convert the obtained interval from arithmetic encoding to binary string and vice versa. The working of the codebook is described in Algorithm 2 and 3. A codebook Encoding Algorithm 2 is used to encode the interval obtained from Algorithm 1 into a binary string. In this algorithm, *min*$_r$ and *max*$_r$ represent the minimum and maximum value of the internal, and *mp* represents the midpoint. An empty *codeword* is defined to store the binary string. Every time when *min*$_r$ is less the *mp*, we append 0 in the *codeword*. The obtained *codeword* is considered as the binary string for given data.

Algorithm 2 Codebook Encoding Algorithm

```
input  : $min_r$, $max_r$
output: codeword
Initialize: $codeword = []$, $mp = 0.5$, $low = 0$, $high = 1$, $n = 1$,
  $mid = 0.5$
while $True$ do
    if $min_r \leq mp$ then
        append 0 in codeword;
        update the high value with $mp$;
        update $mp$ with $(\frac{md}{2^n} + low)$;
    else
        append 1 in codeword;
        update the low value with $mp$;
        update $mp$ with $(\frac{md}{2^n} + low)$;
    end
    if $low \geq min_r \&\& high \geq max_r$ then
    else
        $n = n + 1$
    end
end
```

4.6 ArithNgram decoding

The ArithNgram decoding block reverses the encoding process. It converts an encoded binary string back into its original input sequence. The main steps of this block are as follows:

1. Convert binary string to range using a codebook decoding algorithm
2. Repeat the following steps until the end of the encoded data is reached:
   (a) Determine the n-gram symbol corresponding to the current range.
   (b) Update the range based on the probability of the n-gram symbol.
   (c) Read the next bits of the encoded data and update the value accordingly.
3. Output the original data.

The whole decoding process is summarized in Algorithm 3 and . In Algorithm 3, *codeword* represents the binary string for encoded data, *mp* is the midpoint, and min_r and max_r represent the minimum and maximum value of the interval respectively. Every time when it reads a bit from codeword, it updates *mp* with $\frac{mid}{2} + min_r$ until the last bit. When a 0 bit in the *codeword* is encountered, it updates max_r with *mp* and when a 1 bit in the *codeword* is encountered, it updates min_r with *mp*. The obtained value of min_r and max_r is considered as an interval that is used in the decoding algorithm to get back the original data. In decoding Algorithm 4, min_r represents the minimum value of interval obtained from the codebook decoding algorithm and N_w represents the number of words in encoded data. Every time N-grams are stored in *seq* based on min_r and min_r is updated until the end of N_w is reached. The obtained *seq* is the original data.

Algorithm 3 Codebook Decoding Algorithm

```
input  : codeword
output: $min_r$, $max_r$
Initialize: $mp = 0.5$, $mid = 0.5$, $min_r = 0$, $max_r = 1$, $n = 1$
for $i = 1$ to $length(codeword)$ do
    if $codeword(1, i) = 0$ then
        update the $max_r$ with $mp$;
        update $mp$ with $(\frac{mid}{2^n} + min_r)$;
    else
        update the $min_r$with $mp$;
        update $mp$ with $(\frac{mid}{2^n} + min_r)$;
    end
    $n = n + 1$
end
```

Algorithm 4 Decoding Algorithm

```
input  : $min_r$, $vcab$, $N_w$
output: $seq$, $max_r$
Initialize: $min_r = 0$, $max_r = 1$, $Temp = 0$
for $i = 1$ to $N_w$ do
    Find the N-grams in Vcab whose range = $min_r$;
    range =$Vcab(word(high))$-$Vcab(word(high))$;
    update the $temp$ with $min_r - Vcab(word(low))$ ;
    update the $min_r$ with $\frac{temp}{range}$
end
```

The time complexity of the proposed model is the sum of the N-gram language model and arithmetic coding. The complexity of the N-gram model is $O(kn)$ where k represents N-gram size and n represents the length of input data. The overall complexity can be $O(kn) + O(mn)$.

5 Results

This section displays and explains simulation results. All the simulation work is done in MATLAB. For performance analysis, we considered the following performance metrics: number of bits, pre-processing time, compression/decompression time, compression ratio, and compression/decompression Speed. In order to validate the performance of the proposed model, we compared it with some other techniques including Huffman, Arithmetic, and LZW, based on performance metrics. We performed the result analysis for 5 different text files of size 8192, 13904, 32400, 40992, 98560 bits respectively. Figure 3 show the frequency of occurrence of unigram, bigram, trigram, four-gram, and five-gram sequences of different test files respectively.

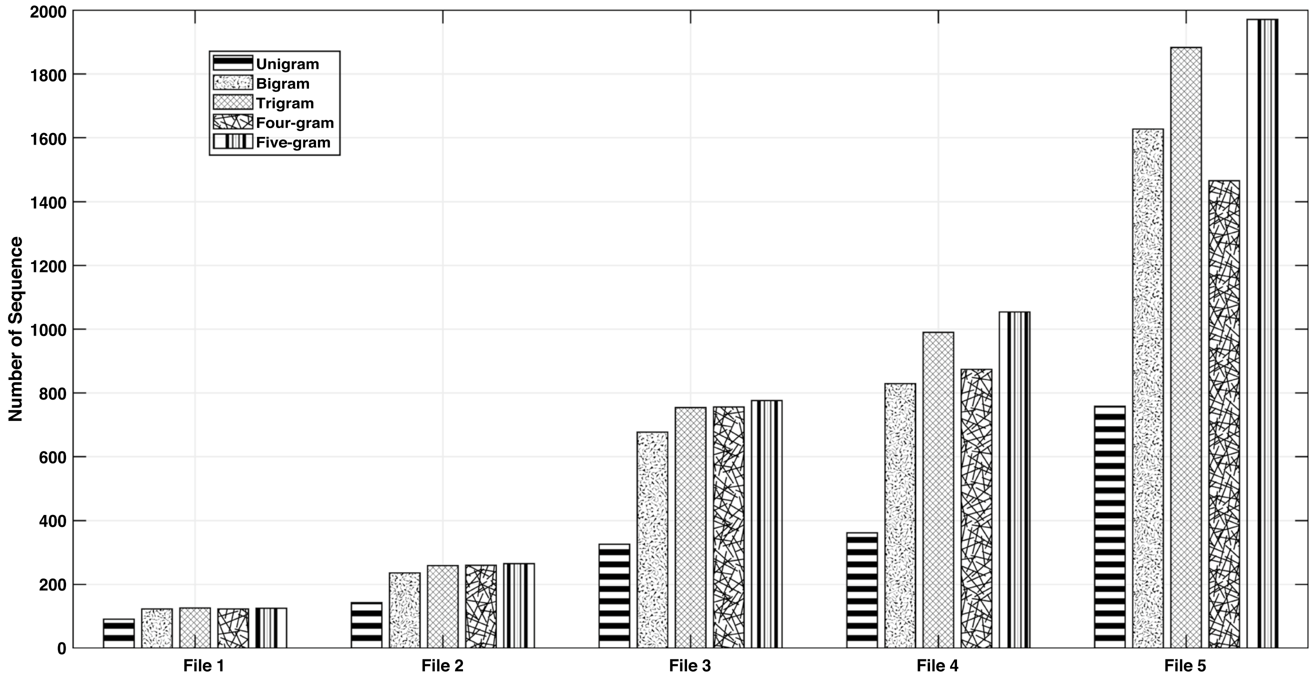

Fig. 3 Number of sequences in different test file

5.1 Compression time

The compression time is the amount of time taken by the model to compress the data. Figure 4 shows the preprocessing time taken by different schemes for different test files. Arth5 g takes the highest time among other schemes because the increased number of words in the N-gram sequence requires more processing time to meet the computational requirements. Figure 5 and 6 show the amount of time taken by all test files for the compression and decompression process. It

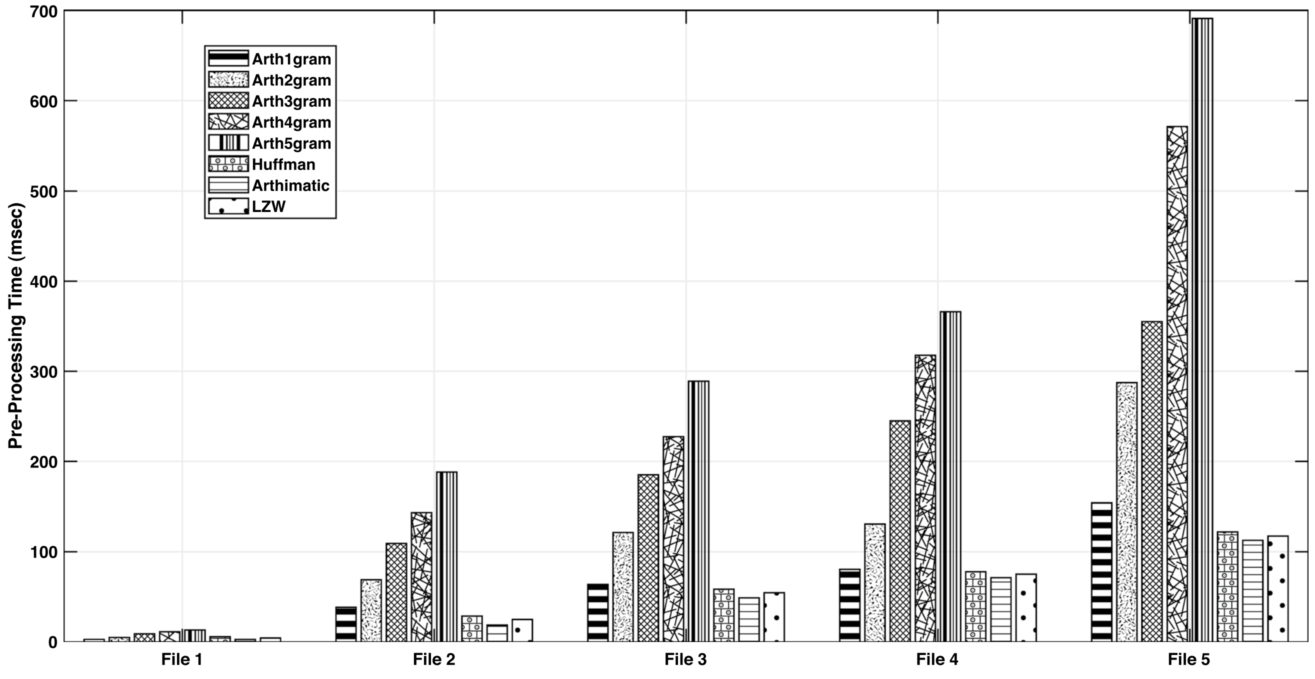

Fig. 4 Preprocessing Time

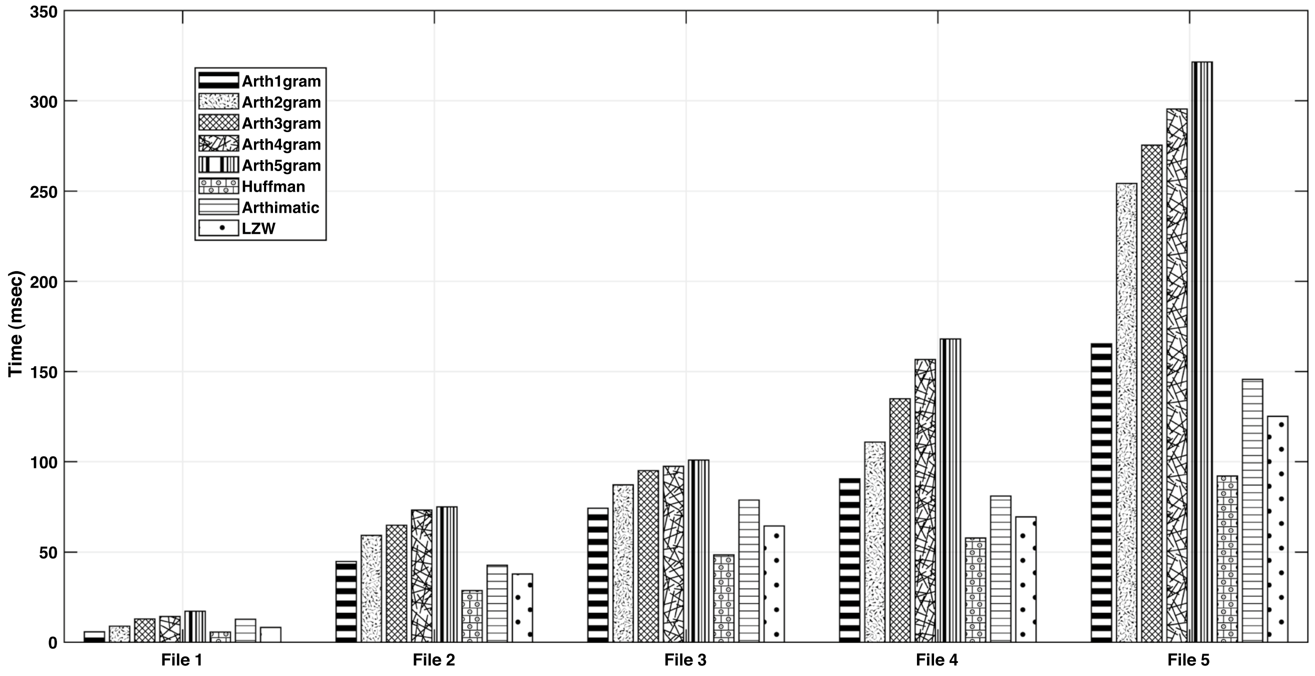

Fig. 5 Comparison of different compression schemes in terms of Compression Time

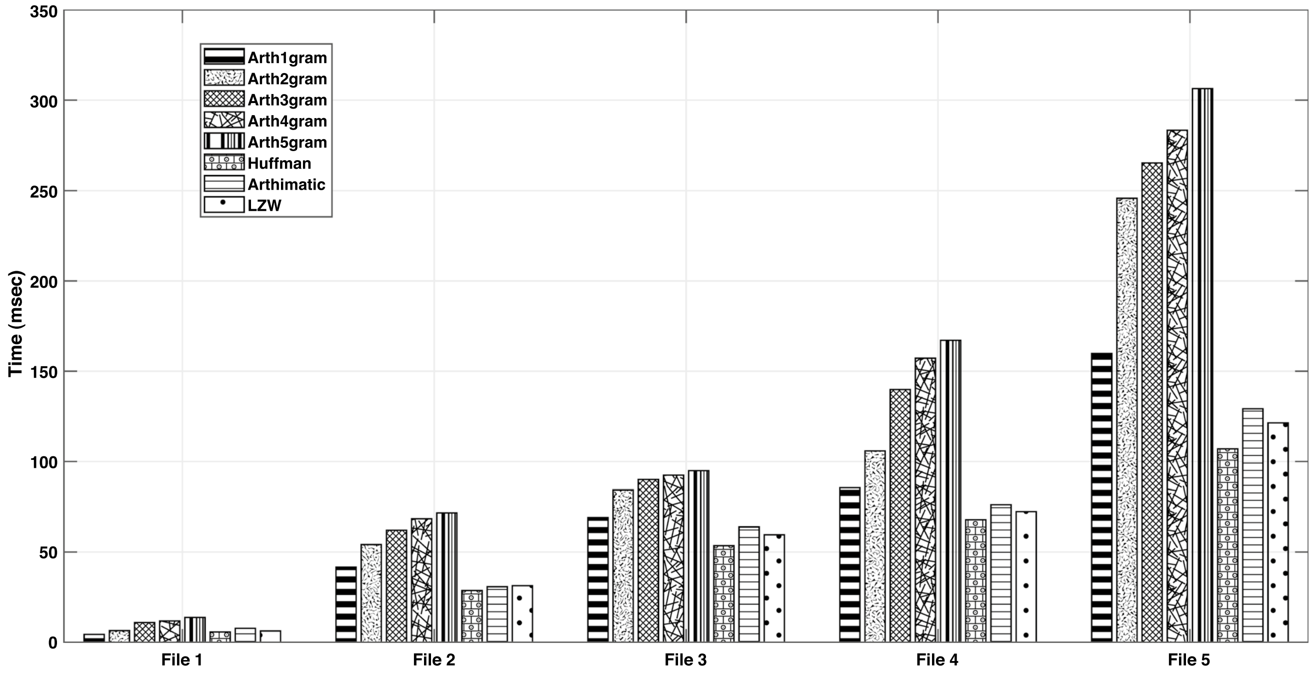

Fig. 6 Comparison of different compression schemes in terms of Decompression Time

can be seen that Arth5garm takes the highest amounts of time for the compression and decompression process. Huffman takes the lowest amount of time in all test files.

5.2 Compression ratio

Compression ratio (C_r) is mostly used as a performance metric to evaluate the performance of the proposed algorithm. The value of C_r depends on the text file to be compressed. Its value can vary for different algorithms. Even the same algorithm can have different values of C_r for different text files. It is calculated by

$$C_r = \frac{\text{Uncompressed file size}}{\text{Compressed file size}} \tag{5}$$

Where Uncompressed file size is the size of the input file and Compressed file size is the size of the compressed file.

Figure 7 shows the compression ratio of test files. It can be observed that the compression ratio of the proposed Arth1 g, Arth2 g, arth3 g, arth4 g, and Arth5 g is more than benchmark techniques which infers the significant impact of N-grams on compression. As the higher value of the compression ratio denotes a more efficient compression strategy since it reduces the size of the compressed file in comparison to the uncompressed file.

5.3 Space requirement

Space requirement depends on compression ratio. The higher value of the compression ratio denotes a more efficient compression strategy since it reduces the size of the compressed file in comparison to the uncompressed file. This infers that the higher compression ratio results in less space required for storing data. Figure 8 shows the size of test files after compression. The original size of 5 different test files is 8.192*Kb*, 13.904*Kb*, 32.400*Kb*, 40.992*Kb*, 98.560*Kb* respectively. It can be seen that arth3 g is performing best than others. On average it reduces 65% size as compared to traditional techniques which significantly reduces the space requirement for storage purposes.

5.4 Compression speed

Compression speed (s_{com}) measures how quickly data can be compressed into a smaller size. It is measured in the number of megabits per second (Mbits/sec). It is calculated as

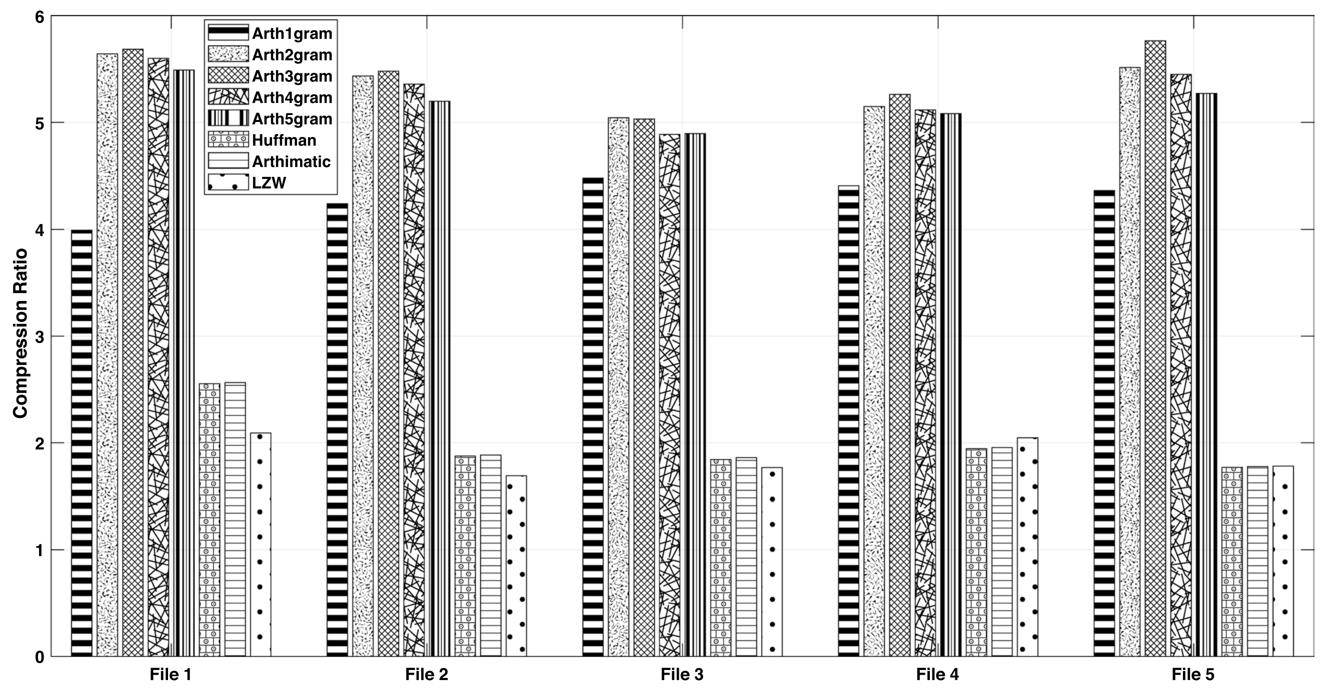

Fig. 7 Comparison of different compression schemes in terms of Compression Ratio

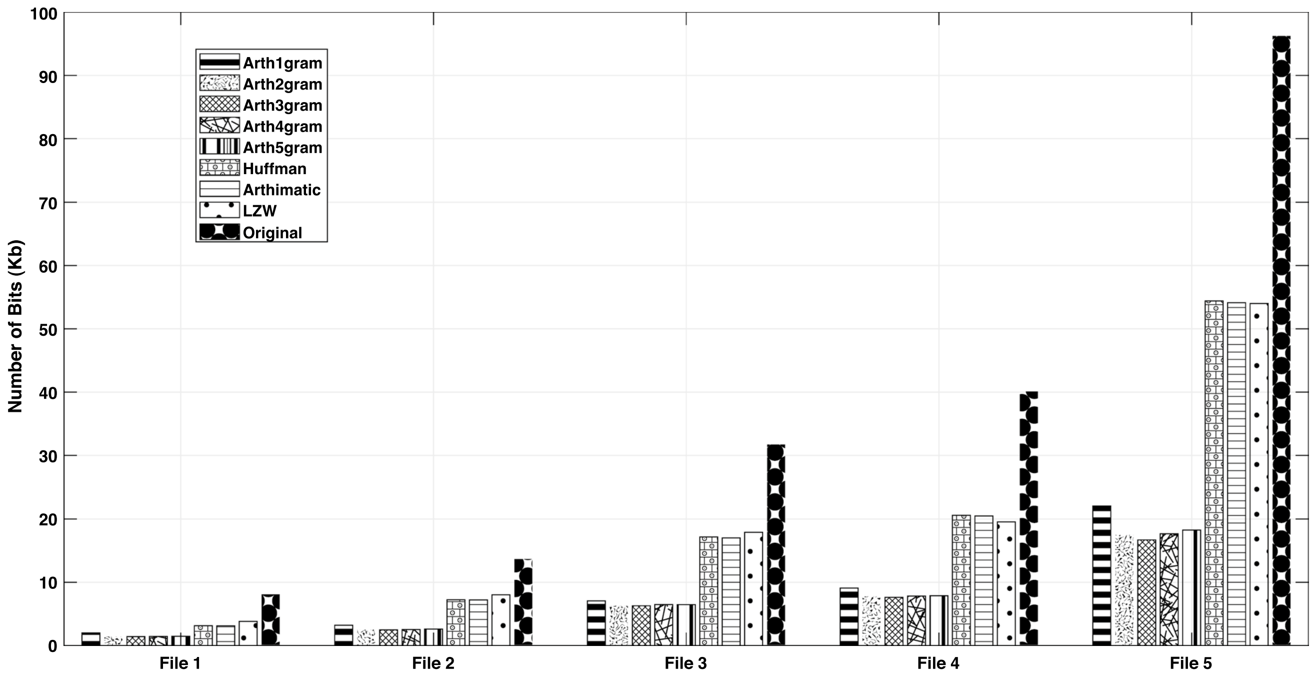

Fig. 8 Comparison of different compression schemes in terms of compressed File Size

$$s_{com} = \frac{\text{Uncompressed file size}}{\text{compression time}} \tag{6}$$

Similarly decompression speed (s_{decom}) measures how quickly compressed data can be converted into original data. It is measured in *Mbits/s*. It is calculated as

$$s_{decom} = \frac{\text{Uncompressed file size}}{\text{decompression time}} \tag{7}$$

Figure 9 and 10 show the compression and decompression speed of test files respectively. It can be noted that Arth1 g and convention techniques (Huffman, Arithmetic, LZW) have a higher compression speed. But Arth2 g, arth3 g, arth4 g, and Arth5 g have a lower compression speed because higher N-gram based compression results in increased complexity that leads to a lower compression speed as it requires more processing time to process larger N-grams.

From the above results, it is observed that Arth1 g has low performance in terms of both compression ratio and high compression speed compared to other ArthNgram techniques. Because in the case of Arth1 g, single words are considered that leads to a large pool of words to be compressed which results in a very low value of interval/range for the whole data to be compressed. That low value is converted into a longer binary string. However, in the case of other ArthNgram techniques, the sequence of words is considered. This sequence of words leads to a small pool of words to be compressed which results in a high value of range for the whole data to be compressed. That high value is converted into a smaller binary string compared to Arth1 g which makes ArthNgram more efficient in terms of compression ratio. Furthermore, in the case of ArthNgarm, N-grams ($N = 2, 3, 4, 5$) calculation for the probability of word sequences requires more time than Arth1 g. It is important to note that compression speed and compression time are inversely related. Therefore, Arth1 g has a higher compression ratio than conventional techniques and a higher compression speed than other ArthNgram techniques. Thus, it is concluded that, in general, selecting a larger value of N for N-gram based compression will provide a better compression ratio but slower compression speed, whereas selecting a smaller value of N will provide a higher compression speed but low compression ratio. The ideal value of N depends on the particular application and its specifications.

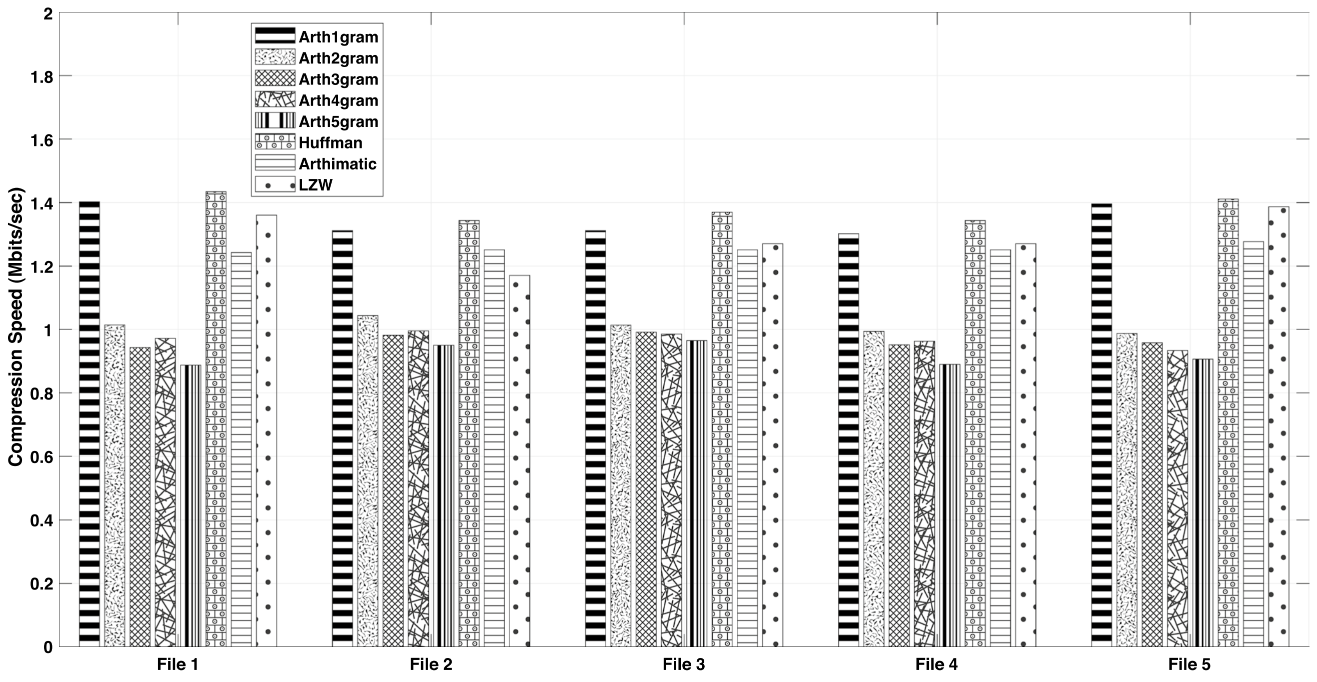

Fig. 9 Comparison of different compression schemes in terms of Compression Speed

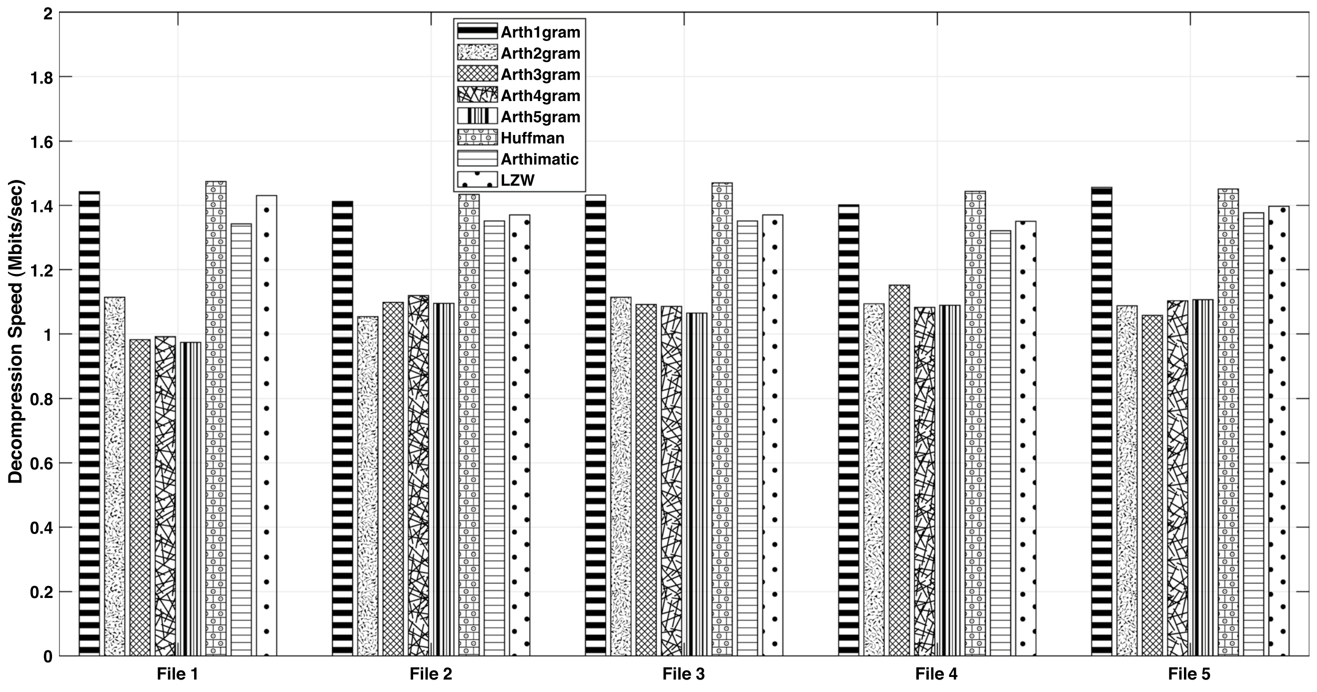

Fig. 10 Comparison of different compression schemes in terms of Decompression Speed

6 Conclusion

Text compression involves the process of reducing the size of text data in order to increase processing speed, decrease transmission time, and conserve storage space. In this paper, we integrate the N-gram language model with arithmetic coding. The N-gram model captures the structure and patterns of the text data effectively and efficiently in order to model the probability distribution of data. Then text data is encoded using arithmetic coding based on this obtained probability distribution which results in a high compression ratio. The simulation results show the effectiveness of the proposed model. It is observed that this model significantly increased the compression ratio.

Author contributions AH, SJ and SH presented the concept. AH, SJ, SH and RA wrote the main manuscript text. AH and SJ prepared all the figures and performed all the simulations. SH, SQ and RA helped with the performance comparison. RA, SH and SQ provided supervision. All authors reviewed the manuscript.

Funding Not applicable.

Data availability Not applicable.

Declarations

Ethics approval and consent to participate Not applicable.

Competing interests The authors have no conflicts of interest to declare relevant to the content of this work.

References

1. Statista. iot-connected-devices-worldwide. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. Accessed 16 Feb 2023.
2. Jayasankar U, Thirumal V, Ponnurangam D. A survey on data compression techniques: from the perspective of data quality, coding schemes, data type and applications. J King Saud Univ Comput Inform Sci. 2021;33(2):119–40.
3. Gupta A, Nigam S. A review on different types of lossless data compression techniques 2021.
4. Hussain AJ, Al-Fayadh A, Radi N. Image compression techniques: a survey in lossless and lossy algorithms. Neurocomputing. 2018;300:44–69.
5. Zhu X, Li J, Liu Y, Ma C, Wang W. A survey on model compression for large language models. arXiv preprint arXiv:2308.07633 2023.
6. Shanmugasundaram S, Lourdusamy R. A comparative study of text compression algorithms. Int J Wisdom Based Comput. 2011;1(3):68–76.
7. Rahman MA, Hamada M, Rahman MA. In *2021 IEEE 14th international symposium on embedded multicore/many-core systems-on-chip (MCSoC)*2021;287–291
8. Ignatoski M, Lerga J, Stanković L, Daković M. Comparison of entropy and dictionary based text compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian. Mathematics. 2020;8(7):1059.
9. Hameed M, Khmag A, Zaman F, Ramli AR. A new lossless method of Huffman coding for text data compression and decompression process with fpga implementation. J Eng Appl Sci. 2016;100(3):402–7.
10. Banerjee S, Singh GK. A new real-time lossless data compression algorithm for ECG and PPG signals. Biomed Signal Process Contr. 2023;79:104–27.
11. Otair M, Abualigah L, Qawaqzeh MK. Improved near-lossless technique using the huffman coding for enhancing the quality of image compression. Multimed Tools Appl. 2022;81(20):28509–29.
12. Shrividhiya G, Srujana KS, Kashyap SN, Gururaj C. In *2021 international conference on emerging smart computing and informatics (ESCI)* 2021;234–237
13. Habib A, Islam MJ, Rahman MS. A dictionary-based text compression technique using quaternary code. Iran J Comput Sci. 2020;3(3):127–36.
14. Nguyen VH, Nguyen HT, Duong HN, Snasel V. n-gram-based text compression. Computational intelligence and neuroscience 2016;2016
15. Mantoro T, Ayu MA, Anggraini Y. In *2017 International Conference on Computing, Engineering, and Design (ICCED)* (IEEE), 2017;1–5
16. Aburomman FTA. Dynamic with dictionary technique for arabic text compression. Int J Comput Appl. 2016;975:8887.
17. Chatterjee A, Shah RJ, Hasan KS. In: 2018 IEEE International conference on big data (big data) 2018;5137–5141
18. Gupta M, Agrawal P. Compression of deep learning models for text: a survey. ACM Trans Knowl Discov Data (TKDD). 2022;16(4):1–55.
19. Fauzan MN, Alif M, Prianto C. Comparison of Huffman algorithm and Lempel Ziv Welch algorithm in text file compression. IT J Res Develop. 2023;7(2):155–69.
20. Lin J, Tang J, Tang H, Yang S, Dang X, Han S. Awq: activation-aware weight quantization for llm compression and acceleration. arXiv preprint arXiv:2306.00978 2023
21. Ma X, Fang G, Wang X. Llm-pruner: on the structural pruning of large language models. arXiv preprint arXiv:2305.11627 2023.
22. Langdon GG. An introduction to arithmetic coding. IBM J Res Develop. 1984;28(2):135–49.
23. Kotha HD, Tummanapally M, Upadhyay VK. In J Phys Conf Ser. 2019;1228:012007.