Neural Computing and Applications (2024) 36:18363–18376
https://doi.org/10.1007/s00521-024-10130-4

ORIGINAL ARTICLE

DNACoder: a CNN-LSTM attention-based network for genomic sequence data compression

K. S. Sheena[1] · Madhu S. Nair[1]

Received: 10 December 2023 / Accepted: 26 June 2024 / Published online: 25 July 2024

Abstract
Genomic sequencing has become increasingly prevalent, generating massive amounts of data and facing a significant challenge in long-term storage and transmission. A solution that reduces the storage and transfer requirements without compromising data integrity is needed. The effectiveness of neural networks has already been endorsed in tasks like image and speech compression. Adapting them to recognize the intricate patterns in genomic sequences could help to find more redundancies and reduce storage requirements. The proposed method, called DNACoder, leverages deep learning techniques to achieve significant compression ratios while preserving the essential information in genomic data and offers a high-performance compression for genomic sequences in any data format. The results of the experiments clearly demonstrate the effectiveness of the method and its potential applications in genomic data storage. Our proposed method improves compression by 21.1% on bits per base compared to existing compressors on the benchmarked dataset. By using a deep learning prediction model that is structured as a convolutional layer followed by an attention-based long short-term memory network, we propose a novel lossless and reference-free compression approach (DNACoder), which can also be utilized as a reference-based compressor. The experimental outcome on the tested data illustrates that the advocated compression algorithm's CNN-LSTM model makes generalizations effectively for genomic sequence data and outperforms the state-of-the-art methods.

Keywords Genomic sequence · DNA · Deep learning · Lossless compression · CNN · LSTM · Attention mechanism · Markov model

1 Introduction

Genomic data have been growing significantly due to the development of high-throughput sequencing technologies. It is an additional responsibility to figure out how to store the data effectively. For decades, several strategies have been proposed to compress genomic data.

The entire set of an organism's DNA (deoxyribonucleic acid), which makes up the genetic material in organisms and carries the genetic instructions for development, growth, and reproduction, is known as its genome. Adenine (A), guanine (G), cytosine (C), and thymine (T) are the four chemical building components that make up a nucleotide, which is the monomer of the polymer DNA [1]. Next-generation sequencing (NGS) is an evolving technology for DNA and RNA sequencing and variant/mutation detection [2, 3]. NGS can sequence thousands of genes or whole genomes in a short period of time. NGS data are being applied broadly as a valuable tool for clinical applications [4, 5]. While storage and computing power advancements have made it easier to handle genomic data, the need for compression techniques persists due to storage costs, data transfer efficiency, computational requirements, privacy concerns, and future scalability. Genomic data compression remains an essential tool to optimize the storage, processing, and sharing of this vast and complex information.

The profound effect of whole genomes in healthcare, such as personalized drug design, disease diagnosis, therapeutic decision, preventive medicine, and follow-up of patients, may soon necessitate the storage of DNA

✉ K. S. Sheena
sheenaks@cusat.ac.in; sheenasekhar@gmail.com

[1] Artificial Intelligence & Computer Vision Lab, Department of Computer Science, Cochin University of Science and Technology, Kochi, Kerala 682022, India

sequences of individuals in portable devices like smartphones and smart health cards that allow us to keep a copy of our essential health records on hand. Genomic data compression remains an effective technique for streamlining the processing, storage, and sharing of this enormous amount of data. The data-specific DNA sequence compression is thus having an increasing relevance. In addition to the advantages of data compression, the researchers also gain new insights from the informational comprehension of the genome that is made possible by compression.

We attempted to devise a novel approach that applies a deep learning model for learning the patterns in the sequence, and then predicts the next element in the sequence, incorporates appropriate entropy coding, and yields a compressed data stream that ensures a lossless decompression. Incorporating deep learning techniques for DNA sequence modeling enables the effective encoding and compression of DNA sequences based on the Markov property. Markov processes can be used to model and predict the next element in a DNA sequence by assuming that the probability of observing a particular base (A, T, C, or G) at a given position in the sequence depends only on the preceding few bases. By reducing the amount of storage space needed for huge DNA datasets, deep learning techniques make it simpler to maintain and analyze genetic data. However, challenges exist in finding a balance between compression efficiency and lossless retaining of the biological sequence.

With the exponential growth rate of genomic data, efficient compression techniques are essential for storing and transmitting DNA sequences. The proposed CNN-LSTM model-based compression algorithm can significantly lessen the storage required for DNA sequences, making it a more cost-effective and practical compression tool for large-scale genomic databases. Real-time applications involve transmitting compressed DNA sequences for genomic analysis, diagnosis, and designing personalized medicine. DNA sequence compression algorithms can facilitate the efficient transmission of genomic data over networks with limited bandwidth or high latency during genomic data sharing between research institutions. DNA sequence compression can expedite computational analyses of genomic data by reducing the time required for transferring large genomic datasets.

Deep learning models can capture the hidden features of DNA sequences and generate compressed representations that preserve essential information. In the context of DNA sequence compression, deep learning exhibits immense potential. By merging deep learning with conventional compression methods, compression efficiency can be enhanced. This approach enables the development of compression algorithms that can adapt to the unique characteristics of DNA sequences, leading to more effective compression methods compared to traditional approaches.

The major contributions of the work are outlined as follows:

- We propose a novel reference-free sequence compression algorithm (DNACoder) for genomic sequences by using a deep learning prediction model to compress the data.
- We have presented a novel architecture for the probability prediction model using a CNN-LSTM attention-based network.
- A novel error-handling technique that gives more redundancies is introduced with suitable entropy coding schemes.
- The performance of the proposed model is tested against different state-of-the-art genomic sequence compression techniques and evaluated using various compression metrics. In comparison to other methods, the proposed compression algorithm performs better in various quantitative metric values.
- The proposed compression algorithm based on the CNN-LSTM network can adapt to different data formats, providing scalable and efficient solutions for managing and analyzing emerging genomic data modalities.

2 Related work

Reference-based (vertical) and reference-free (horizontal) approaches have been used to address the lossless compression of genomic sequences [6]. The horizontal genomic compression techniques aim to exploit the similarities and the redundancies present within the sequence to achieve compression. The only thing that the horizontal mode depends on is the target sequence, and the compression of each genomic data file is carried out independently. The reference sequence-based techniques benefit from the high similarity of DNA sequences from sources that are similar, such as individuals from the same species. When a specific sequence is chosen as the reference sequence, the other sequences can be compared against it using a number of different techniques to determine the differences, which can then be stored as a stand-in for non-reference sequences. The amount of storage is drastically decreased because just the differences are kept on the compressed file.

The algorithms that are reported for genomic data compression include dictionary-based, statistical, and transform-based methods. The dictionary-based compression techniques aim to locate repeats or substrings with a high degree of repetition in the DNA sequence. These substrings are then used to create a dictionary, which is then compressed using a variety of encoding schemes to

reduce the proportion of highly repetitive substrings. While this approach can achieve high efficiency in compressing some DNA sequences, the length and frequency of the substrings in the dictionary have an impact on the compression rate.

The statistical compression techniques develop a probabilistic model from identifying and summarizing patterns in DNA sequences based upon the assumption that the occurrence of DNA bases in sequence reads follows a particular statistical model. Once the model has been established, it is used to predict the bases in the sequence. The model should be able to accurately predict the subsequent bases in the DNA sequence with high probability and then apply efficient coding schemes like arithmetic coding for further compression. The transform-based methods transform the genomic sequences into other forms to get more redundancies. The Burrows—Wheeler Transform (BWT)-based approach is an example of such a method. The BWT performs a permutation of the characters in the sequence, such that characters in lexically similar contexts will be near each other. S. Grumbach and F. Tahi et al. introduced biocompress [6], a compression technique for genomic sequences, thereby opening the research on the data-specific compression algorithm for genomic sequences. Biocompress is a lossless compression algorithm based on the regularities present in the DNA, like repeats and palindromes. A number of insightful methods have been proposed afterward.

A combination of binary and delta encoding is employed in the algorithm DELIMINATE [7]. The sequence data and the header are handled independently. Sequence data are compressed using this approach in two stages. The 7-Zip archiver [8] is used to further compress the files created during the compression process [8]. The FastA/MultiFASTA file format is supported. It can handle all possible characters in a FASTA file. MFCompress [9] algorithm compresses genomic sequence using a probabilistic model that complies with the Markov property. The header text is encoded using one finite-context model. A forward adaptive coding approach utilizing arithmetic coding and multiple competing finite-context models is employed to encode the DNA sequences. XM [10] is another compressor for genomic sequences using a context-based expert predictive model and arithmetic coding. In this research, a unique technique for biological sequence compression is presented that utilizes repetition within sequences as well as statistical features. It uses several 'experts' to predict the probability of the next symbol. The final distribution is obtained by combining expert probabilities. XM offers a greater compression ratio whereas takes a long computational time. Nucleotide Archival Format (NAF) [11] is a reference-free compression tool available for sequence files. In NAF, the sequences are concatenated and converted into 4-bit encoding, storing two nucleotides per byte, and facilitating fast encoding and decoding while accommodating all IUPAC nucleotide codes. All data streams are compressed using the zstd compressor [12]. Decompression involves decompressing individual streams and reassembling them into FASTA or FASTQ format.

Apart from the conventional compression methods for genomic data, CoGI [13] uses a new paradigm by transforming genomes into image representations. CoGI utilizes the inherent spatial structure and similarity patterns within genomes and maps the genome sequence onto a two-dimensional matrix as an image. Using this mapping, the authors exploit the existing image compression techniques to reduce the size of the genomic data. This method supports both reference-free and reference-based compression. The algorithm Genozip [14] delivers compression for prevalent genomic data formats in genomics research, namely FASTA, FASTQ, VCF, GVF, SAM/BAM/CRAM, PHYLIP, and 23andMe formats. It supports compression with or without a reference genome sequence. Genozip is a framework that can also be used for the quick creation and implementation of new compression algorithms for recognized or unrecognized genomic data types and file formats by offering a modular and extensible design.

Some compression schemes use neural networks for compressing genomic sequences; the compressor GeCo3 [15] utilizes a feed-forward artificial neural network that provides the mixing of multiple context models and substitution-tolerant context models. The mixing method is portable and requires the probabilities of the models as inputs. DeepZip [16] compresses a variety of synthetic data, text, and genomic datasets losslessly. It combines recurrent neural network predictors with an arithmetic coder. The recurrent neural network (RNN) is used as the model for the probability predictor block. In [17], a lossy compression approach uses a simple deep neural network (DNN) autoencoder for genomic sequences. The algorithm in [18] uses a deep neural network model to estimate the conditional probability of occurrence of the next base in the sequence, and further encoding is done using the arithmetic coding technique. Arithmetic coding is sensitive to errors or corruption in the encoded data. Even a single bit error in the compressed data, due to the errors in the probability estimation phase, can lead to a loss of the decoded information. Additionally, a novel lossless compression scheme using a CNN autoencoder [19] is contributed to this context. This paper introduces a scheme to address the loss incurred after the reconstruction of the data by autoencoder.

Constructing deep learning models for sequence data where the arrangement of elements holds significance, necessitates consideration of the sequential structure of the data. The performance of deep learning is indispensable in sequence prediction tasks across a wide range of sequence processing domains such as time series forecasting, natural

language processing (NLP), speech recognition, bioinformatics, and genomics [20–22]. Furthermore, deep learning has been shown to be beneficial for bioinformatics applications, including biological signal processing [23], disease prediction utilizing electronic health records and many more [24, 25]. The work [26] uses CNN and LSTM neural networks on biological data for predicting protein subcellular localization, protein secondary structure, and peptides binding to Major Histocompatibility Complex Class II molecules. The research work [27] discusses various deep learning technologies and some well-known deep learning frameworks specifically suitable for resolving omics problems. The CNN-LSTM attention network provides a robust method for DNA sequence prediction by utilizing CNNs for feature extraction, LSTMs for capturing long-term relationships, and attention mechanisms for focusing on relevant sequence regions. As deep learning can learn from data and find insights in data, it can play a crucial role in problem domains like the compression of DNA sequences.

3 Methods

The proposed lossless genomic data compression method (DNACoder) predicts the bases in the sequences from an initial sequence context using a trained deep learning model that uses CNN-LSTM layers with an attention mechanism.

A Markov process is a stochastic model that describes a sequence of events where the probability of transitioning from one state to another depends only on the current state and not on the previous history of states. The Markov models provide a simple way to model dependencies in the DNA sequence. An order-k Markov model gives the probability of a symbol in a position given k preceding symbols. The LSTM neural network is a type of artificial neural network designed for sequential data modeling, and it can be used to model Markov processes to some extent. The order of the Markov model and the amount of the training data will affect the accuracy of the predictions. A Markov chain is a discrete sequence of states that adheres to the Markov property, and each state is taken from a discrete state space (which may or may not be finite). An "order-k" Markov chain means that the probability of transitioning to the next state depends on the previous k states [28]. Then, the Markov property implies that

$$P(S_t|S_{t-1}S_{t-2}\ldots) = P(S_t|S_{t-1},\ldots,S_{t-k})$$

where $P(S_t)$ represents the conditional probability of transitioning from state S_{t-1} to state S_t.

DNACoder reads the sequence from a genomic data file and compresses it using the proposed deep learning model. The model consists of a CNN layer followed by an attention-based LSTM neural network that takes the input sequence and starts to predict the next bases in the sequence. A prediction map is set to catch a wrong prediction, and a correction list is prepared along with the prediction map. Appropriate entropy coding methods further compress the prediction map and correction list. At the receiving end, the same model is used to predict the sequence; then, the wrong predictions are corrected using the prediction map and the correction list.

The pipeline of the compression process is depicted in Fig. 1. The algorithm starts with reading and extracting genomic sequence data from a genomic data file. A pre-processing step is required for the sequence in order to be fed into a neural network.

The length of the genomic sequences varies between files. The genomic sequence in a FASTA file ideally has character codes A, C, G, and T; however, other codes may be present. The sequences will be encoded using the standard IUB/IUPAC nucleic acid codes [29]. Let G_o be a genomic sequence, $G_o = g_0 g_1 g_2 \ldots g_{N_o-1}$ where $g_i \in \{A, C, G, T, N, R, S, W, Y, M, B, K\}$ and $N_o = |G_o|$ be the length of the sequence.

3.1 Pre-processing

The first stage is to perform a pre-processing, which entails three steps:

- Remove all non-ATGC character codes.
- Ignore all line breaks.
- Convert all character codes to uppercase.

After pre-processing, the genomic sequence can be represented as $G = g_0 g_1 g_2 \ldots g_{N_g-1}$ where $g_i \in \{A, C, G, T\}$ and $N_g = |G|$ be the length of the sequence. The sequence gets converted into an array of character codes, which are then encoded to the integer values of {0, 1, 2, 3}. Using a sliding window approach, the continuous integer encoded DNA sequence is converted into two sets X and Y. X is a set of discrete subsequences, and Y is the corresponding target bases for each subsequence in X. The window length W determines the length of the generated subsequences,

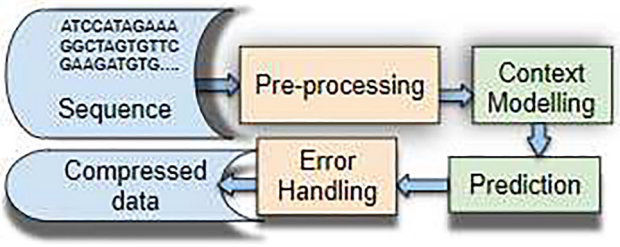

Fig. 1 The pipeline of the proposed compression process

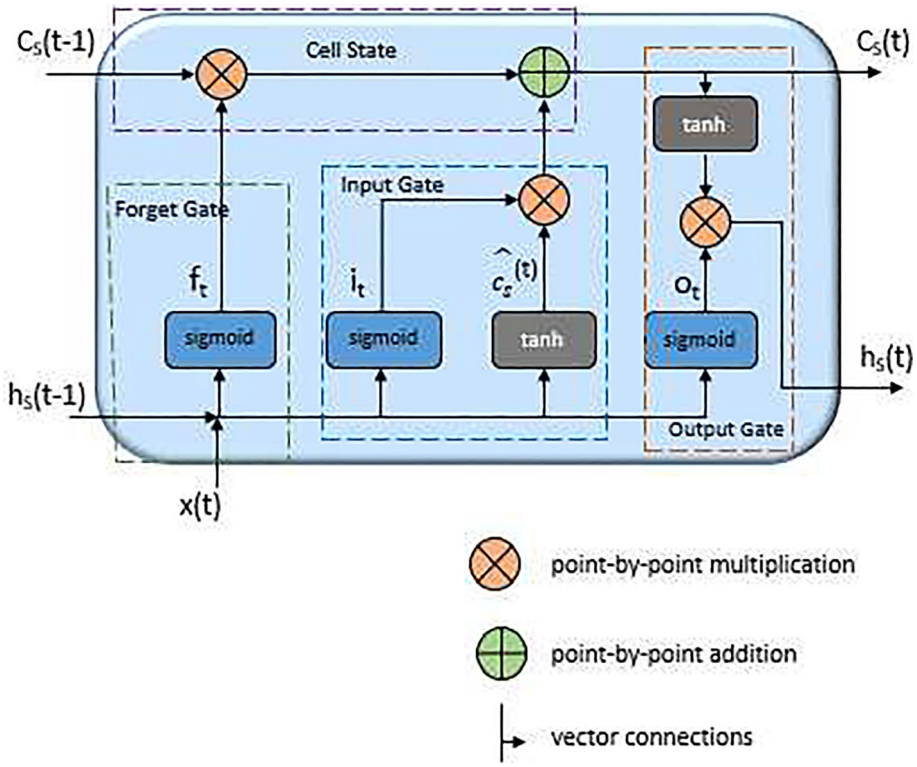

Fig. 2 The LSTM cell [31]

which decides the look-back period for the prediction of the target base.

For example, a sequence $\{g_0g_1g_2g_3g_4g_5 \ldots\}$ with a sliding window length $W = 3$ is converted to a set of overlapping sequences

$X = \{\{g_0g_1g_2\}\{g_1g_2g_3\}\{g_2g_3g_4\}\{g_3g_4g_5\} \ldots\}$ and a set of target bases

$Y = \{g_3, g_4, g_5, g_6 \ \ldots\}$.

The X and Y are then converted to the one-hot encoded form. Categorical values can be transformed into numerical values in a variety of ways, and each approach has its own trade-offs and impact on the feature set. We used one-hot encoding, for which the model performed better.

$0 \rightarrow [1, 0, 0, 0]$
$1 \rightarrow [0, 1, 0, 0]$
$2 \rightarrow [0, 0, 1, 0]$
$3 \rightarrow [0, 0, 0, 1]$

3.2 The LSTM cell

The LSTM cell in Fig. 2 is the fundamental block of the LSTM network [30], which contains a memory cell and three gates. The input gate i_t, the forget gate f_t, and the output gate o_t to control the flow of information in and out of the cell, and the cell state c_s allows LSTMs to maintain the long-term dependencies of the bases over long DNA sequences. The gates control the addition and removal of information from the cell state. An LSTM unit has a hidden state h_s; the hidden state and the input data are used to regulate the cell state c_s.

The LSTM cell accepts three inputs at each timestep t:

- $x(t)$: The input at time t.
- $h_s(t-1)$: The previous hidden state at time $t-1$.
- $c_s(t-1)$: The previous cell state at time $t-1$.

The following intermediate values are calculated by the LSTM cell:

- $f_t = \text{sigmoid}(W_f * [h_s(t-1), x(t)] + b_f)$. W_f: weight matrix between forget gate and input gate. b_f: bias vector w.r.t. W_f.
- $i(t) = \text{sigmoid}(W_i * [h_s(t-1), x(t)] + b_i)$.
  $\hat{c_s}(t) = \tanh(W_c * [h_s(t-1), x(t)] + b_c)$. W_i: weight matrix between input gate and output gate. W_c: weight matrix between cell state and network output. b_i: bias vector w.r.t. W_i. b_c: bias vector w.r.t. W_c. Updated cell state: $c_s(t) = f(t) * c_s(t) + i(t) * \hat{c_s}(t-1)$.
- $o(t) = \text{sigmoid}(W_o * [h_s(t-1), x(t)] + b_o)$.
  $h_s(t) = o(t) * \tanh(c(t))$. W_o: weight matrix of output gate. b_o: bias vector w.r.t. W_o.

3.3 Proposed CNN-LSTM sequence prediction model

Figure 3 portrays the proposed CNN-LSTM model architecture. The bases in the DNA sequences are integer

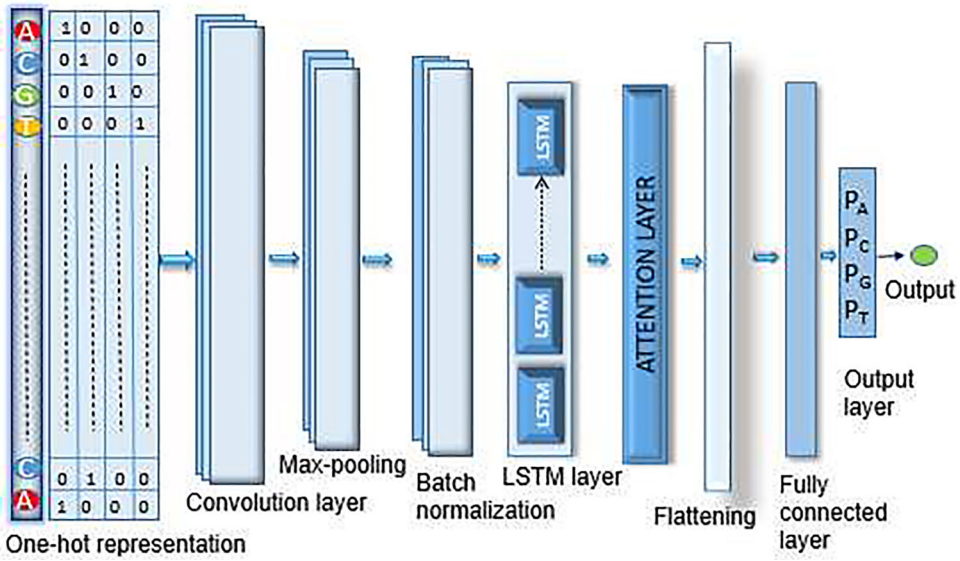

Fig. 3 The proposed model architecture

encoded and then are transformed into one-hot encoded representation. The model accepts the input sequence data in a 3-dimensional array.

The array's dimensions are *samples* × *windowlength*× *features*. where *samples* is the number of observations, window length W is the look-back period to make a prediction, and *features* indicate the number of features present in the input data.

The first layer is a one-dimensional convolutional layer with 32-filter feature detectors. The convolutional layer creates a feature map by performing the convolution operation between the input (or the previous feature map) vector and the feature extractor filter. A kernel size of 4 is applied to each feature detector. This CNN layer helps the model to learn local features. The first CNN layer's output is fed to the Max pooling layer, which helps to reduce the spatial dimensions of the feature maps while retaining the most essential information by selecting the maximum value within each pooling window. This allows the network to focus on the most salient features and become less sensitive to small variations in the input sequence, making the model more robust. To normalize the inputs to the layers in the network, a batch normalization layer is introduced before the LSTM layer. By normalizing the activations in each mini-batch, the Batch normalization layer helps to solve the difficulty in converging when the input distributions to each layer change during training. Next is the LSTM layer, which can capture long-term dependencies in the sequence. At each timestep t, the LSTM layer takes the input $x(t)$ and the previous hidden state $h_s(t-1)$ as inputs and computes the updated hidden state $h_s(t)$ and cell state $c_s(t)$. The output $h_s(t)$ of LSTM is then used as the input to the next layer.

An attention layer allows models to focus on specific parts of the input, enhancing their ability to comprehend intricate patterns. It enables the model to focus on different parts of the input sequence when generating an output. By including an attention mechanism, the LSTM layer is directed to focus on the most relevant parts of the input sequence, potentially improving the overall performance of the model. The attention layer receives the hidden states of the LSTM layer as inputs and are considered as vectors called Q(query), K(key), and V(value); the key is usually the same as the value [32, 33].

q_i: hidden vector for ith output base
k_j: hidden vector for jth input base
V_j: hidden vector for jth input base

The similarity score is used by the attention mechanism to align the input and output sequences. It is helpful to focus on the most essential information in the input sequence.

- Similarity Score Calculation: First, compute a similarity score between the query vectors (Q) and the key vectors (K). This score indicates how much attention should be paid to each element in the input sequence when making predictions. A commonly used similarity function is the dot product or scaled dot product. We have used the simple dot product similarity function as in Eq. 1.

$$\text{Score}(q_i, k_j) = q_i{}^T \cdot k_j \tag{1}$$

- Attention Weights: The generated scores are then subjected to a softmax function to produce attention weights

$$Attention(q_i, k_j) = \frac{\exp(\text{Score}(q_i, k_j))}{\sum_{j'=1}^{n} \exp(\text{Score}(q_i, k_{j'}))} \quad (2)$$

where n is the total number of key-value pairs in the sequence. The Eq. 2 computes the relative importance of each key vector for the given query vector.

- Weighted Sum: The attention weights are used to compute a weighted sum of the value vectors (V). This weighted sum becomes the output of the attention layer:

$$Attention\ Output(q_i) = \sum_{j=1}^{n} Attention(q_i, k_j) \cdot V_j \quad (3)$$

The attention output is a weighted combination of the value vectors, where the weights are determined by the attention scores.

In a sequence prediction model, this attention mechanism is often applied to capture different types of relationships and dependencies within the input sequence. The Attention Output enables the model to produce precise and context-aware predictions, such as the nth base and mth base carrying more information for generating the next base. A flattening layer is added to transform the multidimensional data into a one-dimensional vector, which prepares the data for further processing by the fully connected layers. The fully connected layer enables the network to capture high-level features and make predictions based on the learned representations. The final layer is also a fully connected layer with softmax activation that produces the network's output.

The categorical cross-entropy loss measures the dissimilarity between the predicted probability distribution over classes and the true probability distribution (one-hot encoded) of the target labels. The categorical cross-entropy loss is calculated using the predicted distribution $P(y\_pred)$ and the true distribution $P(y\_true)$.

The formula for calculating the loss $L(y\_true, y\_pred)$ for a single data point is:

$$L(y\_true, y\_pred) = -\sum (y\_true * log(y\_pred)) \quad (4)$$

It serves as the objective function for training the prediction model, encouraging the model to assign high probabilities to the correct base and low probabilities to incorrect ones. Based on the gradient of the loss function with respect to the weight parameters, backpropagation is used to repeatedly optimize the network parameters. An optimisation algorithm is required to minimize the loss function in Eq. 4; Adam Optimizer is used in this work.

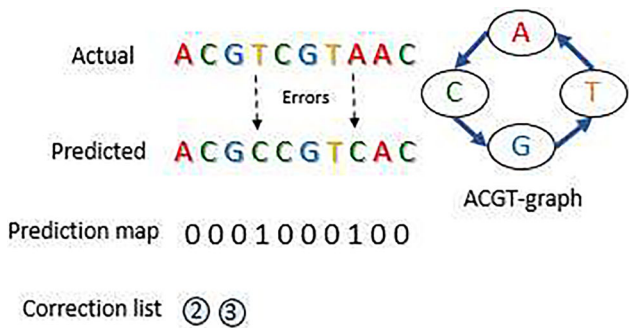

Fig. 4 The process of encoding

3.4 Genomic sequence data encoding

The process of encoding and decoding is described by the Algorithms 1 and 2.

1. Sender side: The proposed method assumes both the sender and the receiver have the same trained model to predict the bases in the sequence. In Algorithm 1, the input $X = \{x_1, x_2, \ldots, x_n\}$ can be viewed as a sliding window having a fixed window size W (Line 1); it advances one position after each prediction. In Line 5, the model predicts the next base in the DNA sequence considering previous W bases; it takes an initial sequence context to start with. A prediction map *PredMap* is maintained to keep track of wrong predictions; a 0 indicates a correct prediction, and a 1 indicates an error (Line 6–7). Whenever the prediction goes wrong, a correction factor d is computed as the path length obtained when traversing the $ACGT - graph$ from the predicted base node to the actual base node. This correction factor d is added to a list called correction list *CorrList* (Line 8–11). The process of encoding is illustrated with an example in Fig. 4. In the case of the genomic sequence, a loss is inadmissible; the *PredMap* and *CorrList* address completely the loss incurred during prediction. The entropy coding techniques further compress the *PredMap*, *CorrList*, and the initial sequence context. Each entropy coding involved in this algorithm is described in Sect. 3.4.1. Finally, the sequence compressor comprises the output as *PredMap*, *CorrList*, and *sequence_context*.
2. Receiver side: At the receiver side, using the associated decoders, each component of the received data is decompressed. The Model takes the initial sequence context $X = x_1, x_2, \ldots, x_n$ and generates the subsequent bases in the sequence. Algorithm 2 outlines the decoding process. For each prediction *Ypred*, the prediction map (*PredMap*) is examined for a 1 (Line 7); if a 1 is encountered, then the first element (correction factor) d of *CorrList* is removed (Line 8–9), and the *ACGT-graph* is traversed d distance from the node labeled as *Ypred* to get the actual base (Line 10). Each time the error occurs, the actual base is found and updated to the output Y as well as the sequence context window X (Line12), and this will prevent the error from propagating. After each prediction, the sequence context window is advanced by one position to include *Ypred* (Line 13).

Algorithm 1 Encoding

Input: G
Output: Prediction map ($PredMap$)**, Correction list** ($CorrList$)
$X \leftarrow \{x_1, x_2, \ldots, x_n\}$
$PredMap \leftarrow []$
$CorrList \leftarrow \{\}$
while $i \leq N_g$ **do**
  $Ypred \leftarrow Model(X)$ ▷ Proposed CNN-LSTM model
  if $Ypred = yActual$ **then**
    $PredMap[i] \leftarrow 0$
  else
    $PredMap[i] \leftarrow 1$
    $d \leftarrow Compute\_d(Ypred, yActual)$ ▷ Traverse the ACGT-graph
    $CorrList.add(d)$
  end if
  X is advanced to include $yActual$
end while

Algorithm 2 Decoding

Input: $Sequence\_Context\ X = \{x_1, x_2, \ldots, x_n\}$, $PredMap$, $CorrList$
Output: $Y = \{y_1, y_2, \ldots, y_{N_g}\}$
$k = 0$
$N_g \leftarrow len(PredMap)$
$X \leftarrow \{x_1, x_2, \ldots, x_n\}$
$Y = \{\}$
while $i \leq N_g$ **do**
  $Ypred \leftarrow Model(X)$ ▷ Proposed CNN-LSTM model
  if $PredMap[i] \neq 0$ **then** ▷ wrong prediction
    $d \leftarrow CorrList(k)$
    $k \leftarrow k + 1$
    $Ypred \leftarrow Traverse\_d(Ypred, d)$
  end if
  $Y.add(Ypred)$
  X is advanced to include $Ypred$
end while

3.4.1 Entropy coding

In this section, we describe the methods for compressing the correction list (*CorrList*) and the prediction map (*PredMap*) produced by the sequence encoding algorithm.

Encoding prediction map As the prediction map is a binary sparse vector, it can be compressed effectively using sparse array compression techniques. In the proposed method, the binary elements are packed to form an 8-bit integer value, and then the run-length encoding (RLE) is employed for further bit reduction. The RLE is a simple lossless data compression technique used to reduce the size of the data by encoding consecutive repeated characters or values as a single character followed by a count of how many times it repeats. It works well when applied to data with repetitive patterns or runs of the same value. It is remarkably efficient when dealing with data that contains long runs of repeated values and is commonly used in scenarios like binary image compression. The RLE introduces minimal overhead in terms of storage for encoding information (e.g., the count of repeated characters), making it efficient for compressing prediction map (*PredMap*) with long runs of zeros.

Encoding correction list The sequence encoder generates an ordered list of correction factors *CorrList* corresponding to each 1s present in the prediction map array. We have used a robust compressor *numcompress* [34]. The tool *numcompress* is based on the Google-encoded polyline format [35] and is used to compress and decompress any numerical series with a good compression ratio.

3.5 Decompression

At the receiver side, the compression procedure is conducted in reverse to complete the decompression in two phases. First, the correction list, the prediction map (*CorrList* and *PredMap*) and the initial sequence context are decompressed by corresponding decoders. In the second phase, the CNN-LSTM prediction model accepts the initial sequence context and regenerates the sequence as described in Sect. 3.4 and Algorithm 2. The output values are undergone post-processing (inverse of one-hot encoding) and are then transformed into an integer vector. Third, the integer vector is transcoded to the DNA character codes 'A,' 'C,' 'G,' and 'T,' and then to the DNA sequence; thereby, the initial sequence G is retrieved. Using the proposed method, the genomic sequence is finally deciphered without any information being lost.

4 Results

The proposed method can be used as a lossless reference-free genomic sequence compressor for the genomic sequence data in any file format, and it can be extended to compress multiple sequences in a reference-based approach. We make an effort to evaluate the proposed method's performance in comparison to three *state-of-the-art* reference-free compressors, NAF, XM, and GeCo3, as well as one general-purpose compressor, Gzip. We have tested our sequence compression method for the benchmarked dataset in FASTA format. Several quantitative metrics are commonly used to express the compression algorithm's performance. Researchers may use different performance metrics to evaluate genomic sequence compressors, such as compression ratio, compression speed, decompression speed, and memory usage. Direct comparisons of different compressors might be challenging since the evaluation metrics chosen can influence how a compressor's performance is perceived. Compressed size (CS) in bytes, compression time (CT), and decompression time (DT) in seconds are used to measure performance to ensure uniformity.

$$\text{Compression ratio (CR)} = \frac{\text{Size of the output stream}}{\text{Size of the input stream}} \tag{5}$$

Compression in bits per base is computed as

$$\text{CB} = \frac{\text{Compressed size in bits}}{\text{Total No of bases in the sequence}} \tag{6}$$

An output stream that is more extensive than the input stream indicates an expansion, and the compression ratio will be greater than 1. A compression ratio of 0.4 indicates that compressed output data occupies 40% of its initial size. When discussing the DNA compression techniques, it makes logical to use the term bpb (bits per base), which refers to the average number of bits needed to represent one base in the output stream using the Eq. 6 [36]. The genomic data is incredibly diverse, including DNA, RNA, and protein sequences, as well as various file formats such as FASTA, FASTQ, SAM/BAM, and VCF. Each compressor may perform differently on different data types and formats, making it difficult to establish a broadly applicable benchmark. A comparison between different genomic sequence compressors is challenging because of the file format supported and the alphabet considered. Since different genomic sequences have a different repetitive index and different sequence characteristics, various algorithms' performance seems to be biased on different datasets.

Table 1 DNACorpus

Seq. ID	Species name	Type
OrSa	*Oriza sativa*	Eukaryota, plant
HoSa	*Homo sapiens*	Eukaryota, animalia
GaGa	*Gallus gallus*	Eukaryota, animalia
DaRe	*Danio rerio*	Eukaryota, animalia
DrMe	*Drosophila miranda*	Eukaryota, animalia
EnIn	*Entamoeba invadens*	Eukaryota, amoebozoa
ScPo	*Schizosaccharomyces pombe*	Eukaryota, fungi
PlFa	*Plasmodium falciparum*	Eukaryota, protozoan
EsCo	*Escherichia coli*	Bacteria
HePy	*Helicobacter pylori*	Bacteria
AeCa	*Aeropyrum camini*	Archaea
HaHi	*Haloarcula hispanica*	Archaea
YeMi	*Yellowstone lake mimivirus*	Virus, mimivirus
BuEb	*Bundibugyo ebolavirus*	Virus
AgPh	*Aggregatibacter* phage	Virus, phage

4.1 Experimental results and analysis

In order to better comprehend our method, we performed the comparative analysis with the state-of-the-art methods using the benchmark dataset proposed in [37] as DNACorpus. The DNACorpus includes the genomic sequences of main domains and kingdoms. This dataset has

Table 2 Training sequences

Seq. ID	Species name	Type
OrSa11	*Oriza sativa*, chr.11	Eukaryota, plant
HoSa3	*Homo sapiens*, chr.3	Eukaryota, animalia
DaRe2	*Danio rerio*, chr.2	Eukaryota, animalia
SaCe	*Saccharomyces cerevisiae*	Eukaryota, fungi
SaEn	*Salmonella enterica*	Bacteria
AcMi	*Acanthamoeba polyphaga mimivirus*	Virus, mimivirus
ObLa	*Oberland virus*	Virus, partial genome
AcPh	*Acinetobacter* phage AM24	Virus, phage

Table 3 The compression results obtained for NAF, GeCo3, XM, Gzip, DNACoder for DNACorpus dataset

DNA sequences		NAF		GeCo3		XM		Gzip		DNACoder	
Seq. ID	Size (B)	CS (B)	CR	CS (B)	CR	CS (B)	CR	CS (B)	CR	CS (B)	CR
OrSa	43,262,523	9,992,929	0.231	8,745,124	0.202	8,470,212	0.196	12,247,694	0.283	7,095,054	0.164
HoSa	189,752,667	43,757,076	0.231	39,384,515	0.208	38,940,458	0.205	53,129,246	0.280	34,345,233	0.181
GaGa	148,532,294	37,297,848	0.251	34,875,638	0.235	33,879,211	0.228	42,244,285	0.284	21,537,183	0.145
DaRe	62,565,020	13,453,230	0.215	11,502,879	0.184	11,302,620	0.181	17,323,427	0.277	10,698,618	0.171
DrMe	32,181,429	8,231,322	0.256	7,717,519	0.240	7,538,662	0.234	9,214,348	0.286	5,406,480	0.168
EnIn	26,403,087	6,155,141	0.233	5,263,852	0.199	5,150,309	0.195	7,616,039	0.288	4,462,122	0.169
ScPo	10,652,155	2,715,812	0.255	2,631,926	0.247	2,524,147	0.237	3,095,627	0.291	1,853,475	0.174
PlFa	8,986,712	2,118,124	0.236	1,992,294	0.222	1,925,841	0.214	2,494,880	0.278	1,276,113	0.142
EsCo	4,641,652	1,205,862	0.260	1,142,948	0.246	1,110,092	0.239	1,341,960	0.289	789,081	0.170
HePy	1,667,825	414,259	0.248	380,539	0.228	384,071	0.230	472,589	0.283	231,828	0.139
AeCa	1,591,049	406,344	0.255	386,017	0.243	387,030	0.243	459,578	0.289	260,932	0.164
HaHi	3,890,005	971,459	0.250	910,858	0.234	913,346	0.235	1,115,264	0.287	704,091	0.181
YeMi	73,689	17,766	0.241	17,193	0.233	16,861	0.229	20,926	0.284	11,643	0.158
BuEb	18,940	4925	0.260	4803	0.254	4642	0.245	5762	0.304	3750	0.198
AgPh	43,970	11,284	0.257	10,977	0.250	10,711	0.244	12,961	0.295	8574	0.195
Total	534,263,017	126,753,381	0.237	114,967,082	0.215	112,558,213	0.211	150,794,586	0.282	88,684,176	0.166

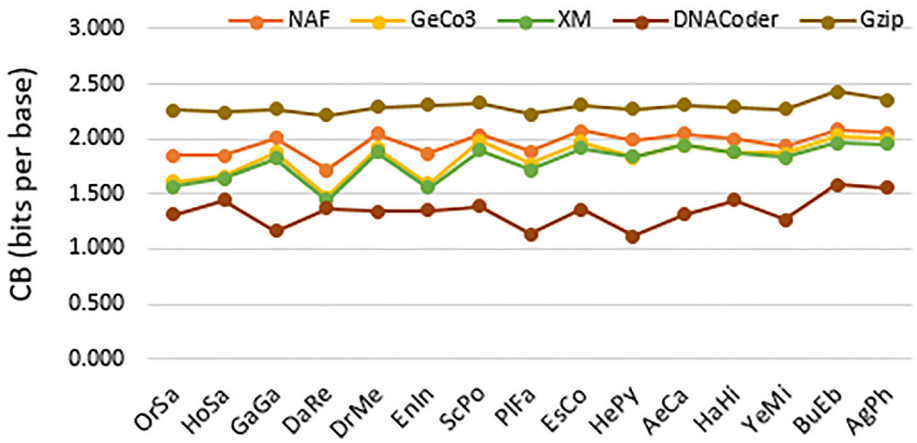

Fig. 5 Compression in bits per base (CB)

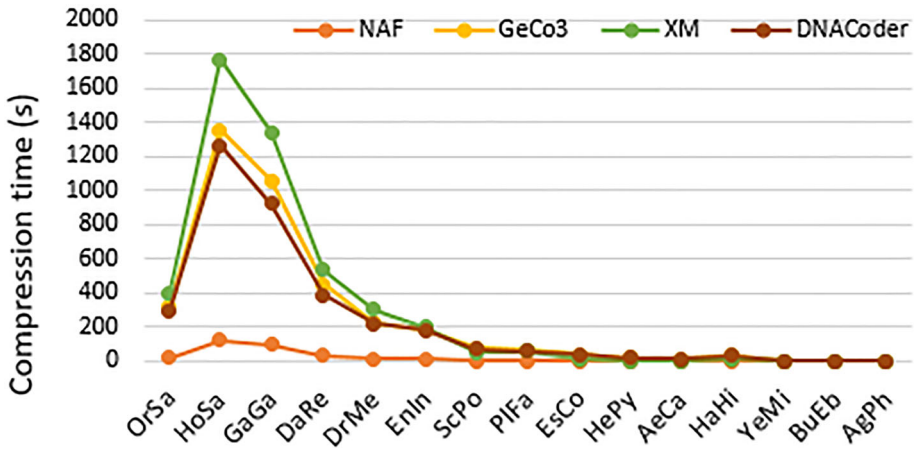

Fig. 6 Compression time obtained in seconds

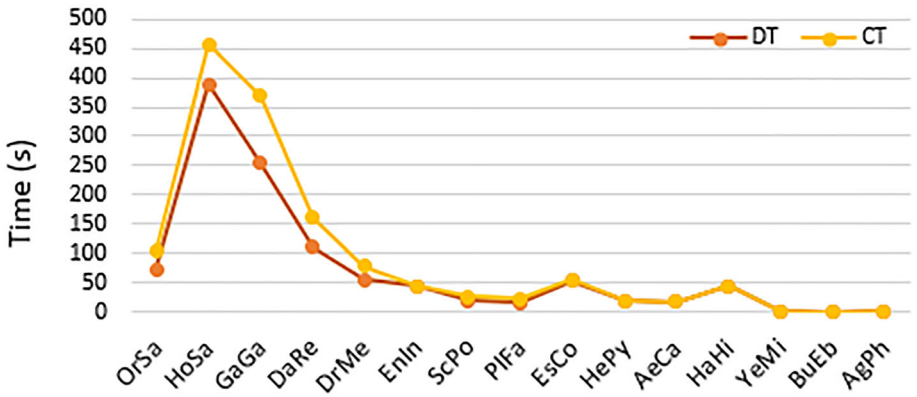

Fig. 7 Compression and decompression time obtained for DNACoder

been widely used in most other works in genomic sequence compression.

The DNACorpus includes the sequences as shown in Table 1. In Table 1, the Seq. ID represents a sequence file, and type defines the kingdom of the species from which the sequence has been extracted. The dataset includes chromosome 1 of Oriza sativa, chromosome 4 of reference human genome, chromosome 2 of Gallus Gallus, chromosome 3 of Danio rerio, chromosome2 of Drosophila miranda and genomes of other species as listed in Table 1.

We used input sequence length (look-back period W) with size 64, and the model was trained using the sequences from the training dataset mentioned in Table 2 for multiple epochs (maximum of 50); an accuracy of 92.16 % was achieved after 16 h of training. The sequences were used to train the model with a 0.2 split between training and validation data and tested with the sequences in DNACorpus.

The proposed method, DNACoder, has been implemented in Python using Tensorflow and Keras, and the experiments are run over a system with an NVIDIA A30 24 GB GPU @ 33 MHz and using a CUDA Version 12.0.

The training sequences, which are listed in Table 2, are used to train the model and Table 3 shows the comparison results for compressing DNACorpus. In all files, the number of character codes in the alphabet used is 4, and each file consists only of the sequence part. In Table 3, the size represents the number of bases in the sequence and is the same as that of size in bytes. The first eight sequences in DNACorpus are from the eukaryotic domain, and the rest are from the prokaryotic domain. Two bacterial sequences, two archaeal sequences, and three viral sequences are all part of the prokaryotic domain. The BuEb and AgPh viruses, in particular, are challenging to compress. To achieve better outcomes, separate training is carried out for each domain. Since each character code requires one byte of storage, DNA sequences are the same size as sequence length. It is obvious from the results in Table 3 that XM is the best among the four existing compressors; based on the compression ratio. XM produces a compressed output that occupies 21% (106.52 MB) of its original size (507.75MB). As the DNACoder's output contains only 16.6% of the input data size, the experimental findings demonstrate that it performs substantially better than XM. Figure 5 shows the evaluation of

compression in bits per base, and DNACoder offers the best compression performance as it needs 1.328 bits on average to represent a single base in the entire sequence. XM, which has CB = 1.678 and CR = 0.210, has better compression performance, but as can be seen in Fig. 6, it takes longer to compress data. Figure 6 shows that NAF gives the fastest compression. Compared with GeCo3 and XM, DNACoder achieves a better compression ratio while consuming the least time. The time measures of DNACoder are plotted in Fig. 7, which shows that the decompression process is quicker than the compression process.

Most sequence databases, such as DNA DataBank of Japan (DDBJ) and the National Center for Biotechnology Information (NCBI), use gzip to compress their data. Gzip gained popularity since it was open source, portable, sturdy, and offered considerable speed and compactness compared to other options available at the time. DNA-specific compression algorithms are designed to exploit patterns and redundancies specific to DNA sequences. As a result, they can often achieve higher compression ratios compared to general-purpose compressors like gzip, which may not fully exploit the unique characteristics of DNA sequence data. It is obvious from the results in Table 3 that all the DNA-specific compression algorithms outperform gzip; in fact, it is evident that a DNA-specific compression standard is required.

DNA sequences can vary significantly in length, from short segments to entire genomes. Deep learning models, particularly neural networks, typically work well with fixed-length input data. If the sequences in the genomic dataset have varying lengths, we need to preprocess them or use sequence-specific models. The choice of the neural network architecture is crucial. For DNA sequence compression, the deep learning model's architecture should be tailored to the specific characteristics of DNA sequences. The deep learning models can be adapted to DNA sequence compression tasks, but their success depends on careful consideration of data, model architecture, loss functions, and evaluation metrics. Performance can be achieved with a large and representative dataset, proper pre-processing, and an understanding of the unique characteristics of DNA sequences. The adoption of appropriate entropy coding techniques plays a crucial role in the overall performance of this compression method. Although RLE is efficient in compressing the binary sparse vector produced by the DNACoder, in situations where there are very short runs of repeated values, the encoding overhead can outweigh any potential compression benefits.

5 Conclusion

The human genome consists of approximately 3 billion base pairs of DNA. On average, any two humans share about 99.9% of their DNA sequence. This means that the genetic variation among individuals is relatively small, accounting for less than 0.1% of the genome. This high similarity helps to develop a generalized deep learning model that can predict the next element of the sequence with high probability.

The DNA sequences can be quite complex, and while Markov models provide a simple way to model dependencies, more advanced machine-learning techniques are often used for more accurate DNA sequence analysis and prediction tasks. Deep learning models have proven to be incredibly effective in a range of fields, including speech recognition, computer vision, and natural language processing. However, several factors and considerations will determine whether a deep learning model can effectively generalize and execute DNA sequence compression.

Deep learning models with extensive structures, like convolutional neural networks (CNNs), long short-term memory networks (LSTMs), or transformers, can be computationally intensive and demand considerable computational resources for tasks like training, hyperparameter fine-tuning and conducting ablation studies. This complexity can pose challenges for real-time or large-scale applications of DNA sequence compression, especially when dealing with vast genomic data.

Deep learning models trained on one dataset may not generalize well to unseen data or different types of DNA sequences. This lack of generalization can be a formidable challenge in the case of DNA compression, where it's essential to ensure that the compression algorithm performs well across diverse genomic sequences and organisms.

Incorporating biological insights into the deep learning model architecture design or training process could lead to more effective compression methods. Researchers can explore variations of CNN-LSTM architectures or integrate them with other types of neural networks for better performance. Further investigation into applying deep learning in DNA sequence compression is necessary for addressing the growing demands of genomic data analysis, enabling discoveries in genomics, and advancing applications in personalized medicine and healthcare. The future of deep learning-based DNA sequence compression is anticipated to benefit from algorithmic breakthroughs, domain-specific knowledge, and advances in neural network architectures.

In this work, our objective was to create a novel algorithm that maximizes compression performance while retrieving sequences without losing any information using a prediction-based deep learning model. Our new technique

can be used to handle the loss that occurred during DNA sequence prediction in an efficient manner, and our method produced results that were on par with those of the existing methods in the literature. Optimizations such as parallelization or hardware acceleration can further be explored to improve compression and decompression time. Even though our deep learning model performs well, improved models and outcomes may be achieved with alternate preprocessing and model architecture designs.

Acknowledgements The first author is thankful to the Cochin University of Science and Technology for the financial support through the University-SRF fellowship (File Ref.No.Ac.C3/2021 dated 27/02/2023).

Data availability statement Data supporting this study is included within the article and/or supporting materials.

Declarations

Conflict of interest We declare that all the authors have approved the manuscript and agree with the submission to this journal. There is no conflict of interest to declare.

References

1. Watson JD, Crick FH (1953) The structure of DNA. In: Cold spring harbor symposia on quantitative biology, vol 18. https://doi.org/10.1101/sqb.1953.018.01.020
2. Batley J, Edwards D (2009) Genome sequence data: management, storage, and visualization. Biotechniques 46:333–336. https://doi.org/10.2144/000113134
3. Church GM, Gilbert W (1984) Genomic sequencing. Proc Natl Acad Sci 81:1991–1995. https://doi.org/10.1073/pnas.81.7.1991
4. Slatko BE, Gardner AF, Ausubel FM (2018) Overview of next-generation sequencing technologies. Curr Protoc Mol Biol 122(1):59
5. Mardis ER (2017) DNA sequencing technologies: 2006–2016. Nat Protoc 12:213–218. https://doi.org/10.1038/nprot.2016.182
6. Grumbach S, Tahi F (1993) Compression of DNA sequences. In: [Proceedings] DCC'93: data compression conference, pp 340–350. https://doi.org/10.1109/DCC.1993.253115
7. Mohammed MH, Dutta A, Bose T, Chadaram S, Mande SS (2012) Deliminate a fast and efficient method for loss–less compression of genomic sequences: sequence analysis. Bioinformatics 28:2527–2529. https://doi.org/10.1093/bioinformatics/bts467
8. 7-Zip file archiver. https://www.7-zip.org
9. Pinho A, Pratas D (2013) MFcompress: a compression tool for FASTA and multi-FASTA data. Bioinformatics (Oxford, England). https://doi.org/10.1093/bioinformatics/btt594
10. Cao M, Dix T, Allison L, Mears C (2007) A simple statistical algorithm for biological sequence compression, pp 43–52. https://doi.org/10.1109/DCC.2007.7
11. Kryukov K, Ueda MT, Nakagawa S, Imanishi T (2019) Nucleotide archival format (NAF) enables efficient lossless reference-free compression of DNA sequences. Bioinformatics 35:3826–3828. https://doi.org/10.1093/bioinformatics/btz144
12. Zstandard: zstd. https://github.com/facebook/zstd
13. Xie X, Zhou S, Guan J (2015) COGI: towards compressing genomes as an image. IEEE/ACM Trans Comput Biol Bioinf 12:1275–1285. https://doi.org/10.1109/TCBB.2015.2430331
14. Wang R, Zang T, Wang Y (2019) Human mitochondrial genome compression using machine learning techniques. Hum Genom 13:2225–2230. https://doi.org/10.1186/s40246-019-0225-3
15. Silva M, Pratas D, Pinho AJ (2020) Efficient DNA sequence compression with neural networks. GigaScience. https://doi.org/10.1093/gigascience/giaa119
16. Goyal M, Tatwawadi K, Chandak S, Ochoa I (2019) Deepzip: lossless data compression using recurrent neural networks. In: 2019 data compression conference (DCC), pp 575–575. https://doi.org/10.1109/DCC.2019.00087
17. Absardi ZN, Javidan R (2019) A fast reference-free genome compression using deep neural networks. In: 2019 big data, knowledge and control systems engineering (BdKCSE), pp 1–7. https://doi.org/10.1109/BdKCSE48644.2019.9010661
18. Lan D, Tobler R, Souilmi Y, Llamas B (2021) Genozip: a universal extensible genomic data compressor. Bioinformatics 37(16):2225–2230. https://doi.org/10.1093/bioinformatics/btab102
19. Sheena KS, Nair MS (2024) GENCoder: a novel convolutional neural network based autoencoder for genomic sequence data compression. IEEE/ACM Trans Comput Biol Bioinform 21:405–415. https://doi.org/10.1109/TCBB.2024.3366240
20. Barzola-Monteses J, Gomez-Romero J, Espinoza-Andaluz M, Fajardo W (2022) Hydropower production prediction using artificial neural networks: an Ecuadorian application case. Neural Comput Appl 34(16):13253–13266
21. Uddin MZ, Dysthe KK, Følstad A, Brandtzaeg PB (2022) Deep learning for prediction of depressive symptoms in a large textual dataset. Neural Comput Appl 34(1):721–744
22. Jin Z, Yang Y, Liu Y (2020) Stock closing price prediction based on sentiment analysis and LSTM. Neural Comput Appl 32:9713–9729
23. Singhal V, Mathew J, Behera RK et al (2021) Detection of alcoholism using EEG signals and a CNN-LSTM-ATTN network. Comput Biol Med 138:104940
24. Choi Y-A, Park S-J, Jun J-A, Pyo C-S, Cho K-H, Lee H-S, Yu J-H (2021) Deep learning-based stroke disease prediction system using real-time bio signals. Sensors 21(13):4269
25. Mou H, Yu J (2021) CNN-LSTM prediction method for blood pressure based on pulse wave. Electronics 10(14):1664
26. Jurtz VI, Johansen AR, Nielsen M, Almagro Armenteros JJ, Nielsen H, Sønderby CK, Winther O, Sønderby SK (2017) An introduction to deep learning on biological sequence data: examples and solutions. Bioinformatics 33(22):3685–3690
27. Zhang Z, Zhao Y, Liao X, Shi W, Li K, Zou Q, Peng S (2019) Deep learning in omics: a survey and guideline. Brief Funct Genom 18(1):41–57
28. Brémaud P (2001) Markov chains: Gibbs fields, Monte Carlo simulation, and queues, vol 31. Springer, Berlin
29. Nomenclature committee of the international union of biochemistry (NC-IUB). nomenclature for incompletely specified bases in nucleic acid sequences, recommendations 1984. Eur J Biochem **150**:1–5 (1985) https://doi.org/10.1111/j.1432-1033.1985.tb08977.x
30. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
31. Understanding LSTM networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs
32. Luong M-T, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025

33. Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. CoRR https://doi.org/10.48550/arXiv.1409.0473arXiv:1409.0473
34. NCBI genome datasets. https://www.ncbi.nlm.nih.gov/data-hub/genome
35. numcompress. https://github.com/amit1rrr/numcompress
36. Salomon D (2006) Data compression: the complete reference. Springer, Boston
37. Pratas D, Pinho AJ (2019) A DNA sequence corpus for compression benchmark. In: Fdez-Riverola F, Mohamad MS, Rocha M, De Paz JF, González P (eds) Practical applications of computational biology and bioinformatics, 12th international conference. Springer, Cham, pp 208–215