

Vector embeddings by sequence similarity and context for improved compression, similarity search, clustering, organization, and manipulation of cDNA libraries

Daniel H. Um^{a,*}, David A. Knowles^{a,b,c,d}, Gail E. Kaiser^a

^a Department of Computer Science, Columbia University, New York, NY, USA

^b Department of Systems Biology, Columbia University, New York, NY, USA

^c The Data Science Institute, Columbia University, New York, NY, USA

^d New York Genome Center, New York, NY, USA

ARTICLE INFO

Keywords:

Clustering
Compression
Similarity Search
Natural Language Processing
Bioinformatics
Vector Embeddings

ABSTRACT

This paper demonstrates the utility of organized numerical representations of genes in research involving flat string gene formats (i.e., FASTA/FASTQ⁵). By assigning a unique vector embedding to each short sequence, it is possible to more efficiently cluster and improve upon compression performance for the string representations of cDNA libraries. Furthermore, by studying alternative coordinate vector embeddings trained on the context of codon triplets, we can demonstrate clustering based on amino acid properties. Employing this sequence embedding method to encode barcodes and cDNA sequences, we can improve the time complexity of similarity searches. By pairing vector embeddings with an algorithm that determines the vector proximity in Euclidean space, this approach enables quicker and more flexible sequence searches.

1. Introduction

As DNA sequencing costs decline in line with Moore's Law, parallels between computer science and synthetic biology are emerging, leading to the formation of a new multidisciplinary field. DNA's four base pairs can be likened to a cell's binary code of 1s and 0s, with the ribosome acting as the compiler that translates this genetic code into proteins. Additionally, technologies like CRISPR-Cas9 and Illumina sequencing equip cells with read-and-write capabilities, opening new avenues for genetic manipulation and analysis.

Recent research in synthetic biology has centered on developing tools to program cells; however, genetic data management tools have not advanced similarly. There is currently no clear consensus on how genetic data should be organized, stored, and accessed by researchers.

Recent research in synthetic biology has focused on developing tools to program cells, yet the development of genetic data management tools has lagged, resulting in no clear guidelines for how researchers should organize, store, and access genetic data (Bagheri et al., 2019). The absence of centralized genetic research software tools (Van der Auwera et al., 2013; Chen et al., 2017; Di Tommaso et al., 2017; Faksri et al., 2016; Roguski et al., 2018; Shelton et al., 2015; Shen et al., 2016) often results in

significant delays—averaging around one and a half months—in reproducing bioinformatic research findings.

This study addresses the practical challenges in bioinformatics and computational biology by exploring improved methods for managing and manipulating genetic data using vector embeddings. Current industry practices for working with genomic data files are cumbersome and inefficient, for instance, single-cell RNA-seq data is often stored in AWS S3 Drive folders organized by run date, then read into Python files line by line with preprocessing steps including stripping and additional checks. Additionally, genes are usually managed as flat-file entries rather than as structured objects, which exacerbates inefficiencies that could be alleviated by adopting software database methods and principles (Hospital et al., 2016). Further, leveraging sequence and context vector embeddings (Mikolov et al., 2013) offers significant potential for improving genetic data management and handling.

There have been several findings that provide key insights on improving data management. The Childhood Cancer Data Lab in Philadelphia developed an ensemble method for dimensionality reduction, incorporating techniques such as principal component analysis and non-negative matrix factorization (Way et al., 2020). Tang et al. demonstrated the benefit of similarity-based clustering prior to compression,

* Correspondence to: Mudd Building, 500 W 120th St, New York, NY 10027, USA.

E-mail addresses: dh2102@columbia.edu (D.H. Um), dak2173@columbia.edu (D.A. Knowles), kaiser@cs.columbia.edu (G.E. Kaiser).

while Minicom introduced an innovative compression method (Tang and Li, 2021). Du et al. presented a vector-embedding method for distributed gene co-expression, utilizing transcriptome-wide gene co-expression data to predict gene-gene interaction based solely on gene names (Guerrero-Sanchez et al., 2020). This paper introduces a novel vector embedding approach tailored for cDNA libraries, eliminating the need for gene classification (Du et al., 2021).

The findings from these four key literature review papers will be elaborated in the Methods section, explaining how they guided this investigation. The literature review establishes the foundation for the thesis question of improving genetic data management and aligns with the study's objective of enhancing efficiency through vector embeddings.

2. Material and methods

2.1. Overview

For this investigation, context embeddings were trained on short-sequence Uniprot aggregate FASTA files with 3D structures, Swiss-Prot Reviewed Binding Site, and human alternative splicing data. Clustered compression (Kowalski and Grabowski, 2020) and similarity search (Flot, 2010) speed comparisons were performed on three FASTA files of approximately 20 MB in size constructed from experimental data and titled *sandmouse.fa*, *moriarty-neg.fa*, and *pathogen.fa*. Once the vector-embedding scripts were written and tested using these files, cDNA data from 10x Genomics were downloaded and used for the remainder of the investigation, specifically the 1k PBMCs from a Healthy Donor (v2 Chemistry) dataset.

The methodology for exploring DNA base pair representation was based on the idea that clustering similar sequences by their vector-embedding representation could enhance the performance of tasks reliant on semantic ordering. This includes similarity searches (Huber et al., 2018), ranking, recommendation systems, deduplication, record matching, and anomaly detection. For data pre-processing, file readers were designed to efficiently handle multiple types of FASTA files, including edge cases like a mix of lowercase and uppercase strings (e.g., "AtCG") and FASTA files with "none" reads or reads of vastly different lengths.

Functions were written to encode and decode DNA strings into unique integers (i.e., using 16^{index} multiplied by a one-hot-coded base in bytes to encode and a modulus operator to peel layers back). To encode, each DNA base (A, T, C, G) is first converted to a unique binary code (e.g., A = 00, T = 01, C = 10, G = 11). These binary codes are then treated as digits in a base-16 (hexadecimal) numeral system. In this system, each position in the DNA sequence contributes to the integer value by being multiplied by 16 raised to the power of its position index. This method ensures that each DNA sequence is represented by a unique integer.

Decoding reverses this process by using the modulus operator to extract each base's code starting from the least significant digit and reconstructing the original sequence. Additionally, a modified version of k-means clustering was used, where similarity was determined based on two-mer frequency, with the original indices preserved. The clustered sequences were converted into byte arrays, where the sequence boundaries were marked by the original indices. This structure enables efficient conversion back to the original FASTA/FASTQ format when necessary.

An improved vector embedding based on context was also implemented (Nti-Addae et al., 2019). In this approach, triplets of nucleotides (amino acid codons) were abstracted as words in the context of the polypeptide sentence. In Fig. 1, the center word is CUU at position t , and the window size is two, spanning from w_{t-2} to w_{t+2} .

Amino acids were mapped to one-hot-encoded vectors, which were then fed into the model as inputs. Linear neurons, serving as a lookup table, constituted the hidden layer. The weights in the hidden layer were optimized using gradient descent, with new weights backpropagated to

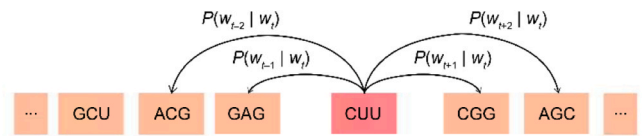


Fig. 1. Skip-gram model showing center codon highlighted in red and a window size of two.

refine the hidden layer further. The output layer was optimized using softmax to support cross-entropy between the output layer to the target layer, representing the current amino acid sequence in its context. The extracted weight matrices from the hidden layer were used to generate the 2D context embeddings for each amino acid.

The sequence embedding, context embedding, barcode, original file index, codon information, gene information, and t-SNE data were centralized into a four-table relational database (Gimadiev et al., 2021; Kozanitis and Patterson, 2016; Zhang et al., 2018; Williams and Tkachenko, 2014) to facilitate easy reproduction of results and access for future experimentation (Carver et al., 2008). To demonstrate the ease of data manipulation, a t-SNE plot color-coded by vector-embedding values was produced—this plot, though not typically produced, serves as a proof of concept to demonstrate the ease of joining two tables through a primary key (i.e., the barcode) for rapid visualization.

Following the creation of vector-embedding algorithms and data organization, sequences were clustered based on the distances between their sequence embeddings. The hypothesis tested with this clustering method was that certain compression algorithms, specifically those that reduce size by consolidating redundancies (Al Yami and Huang, 2019; Chandak et al., 2019; Friis et al., 2016; Kryukov et al., 2020; Liu et al., 2019; Luo et al., 2018; Rajarajeswari and Apparao, 2011; Shen et al., 2016), could be improved by compressing clusters of similar sequences. Another application of the vector embeddings was for similarity searches. Tools from FAISS, Facebook Research's library for efficient similarity search and clustering of dense vectors, were utilized to perform speed benchmarks. For these tests, it was necessary to allocate each integer vector embedding into a $nx64$ embedding float32 array, and map cDNA barcodes to a FAISS index. IndexFlatL2 was used as the distance metric (Jegou et al., 2018).

Below are the summary statistics for the FASTA files used in this study, which highlight the key metrics for each dataset:

The "3D_Structure_1_200_seq_length_17552.fasta" file contains 17,552 sequences with an average length of 129 base pairs and a mean GC content of 9.036383 %. This file is crucial for studying the structural context of sequences in three dimensions.

The "alternative_splicing_human_10541.fasta" file has 10,541 sequences with an average length of 679 base pairs and a mean GC content of 8.868443 %. This dataset is valuable for analyzing sequences involved in alternative splicing events in humans.

The "binding_site_28052.fasta" file includes 28,052 sequences with an average length of 150 base pairs and the highest mean GC content at 9.658848 %. This file examines binding site sequences.

The variation in sequence length and GC content across these datasets underscores the effectiveness of the vector embedding and clustering algorithms. By encoding DNA sequences into unique integer values and clustering them based on vector embeddings, we achieve significant improvements in data handling efficiency, redundancy reduction, and similarity search speed. This approach offers significant potential for advancing genetic data management and analysis in genomics and computational biology.

These three files were selected because they provide non-proprietary exon data encoding three distinct sequences with diverse functions, which are likely to exhibit strong patterns. Specifically, we selected a binding site sequence of length 28,052 entries, a 3D protein-coding sequence with a length of 17,552 entries (including polar, nonpolar, and other types of amino acids), and a human alternative splicing

sequence with a length of 10,541 entries. All of these are publicly available on Mendeley Data: [binding site](#), [3D protein](#), and [alternative splicing sequence](#).

2.2. Sequence embedding

The first type of vector embedding was constructed using the following approach. Characters A, T, C, and G were mapped to unique integer representations. Different mappings were tested, including:

1. One-hot-encoded integer representation. Here, T, C, and G were represented as integers 1000, 0100, 0010, and 0000.
2. Standard mapping: In this method, A, T, C, and G were mapped to 0, 1, 2, and 3, respectively.
3. Binary representations: This method used the binary representations of the characters themselves. Although the specific code was written in Python, pseudocode is presented here for general understanding.

Algorithm 1. Mappings for vector embeddings

```

SET bits from [2,4,8];
SET base_size TO 2^bits // base_size is used for encoding the sequences in Algorithm 2

if encoding EQUALS onehotencoding then
  SET onehotencoding...mapping TO
  "A": 8, "C": 4, "G": 2, "T": 1, "N": 15, "E": 0;
  SET onehotencoding...mapping TO
  8: "A", 4: "C", 2: "G", 1: "T", 15: "N", 0: "E";
end
else if encoding EQUALS standard then
  SET standard...mapping TO
  "A": 0, "C": 1, "G": 2, "T": 3, "N": 4;
  SET standard...mapping TO
  0: "A", 1: "C", 2: "G", 3: "T", 4: "N";
end
else
  SET noencoding...mapping TO
  No change from FASTA and FASTQ char dtypes
end

```

After the characters were mapped to their integer representations, strings were encoded into unique numerical forms using [Algorithm 2](#). This algorithm multiplies the integer mapping of each nucleotide base by an encoder, which was previously defined in the mapping function (with three bits used for the encoder values: 2, 4, and 8).

The numerical representation of each string was then generated, ensuring that each string had a unique numeric identifier. The proximity of these numerical representations indicated the similarity between the corresponding nucleotide sequences.

TensorFlow was used to define the graph structure within the `define\_graph` function in the [GitHub](#) code, which involved setting up the computational graph for training the model.

Algorithm 2. Algorithm for Encode Sequence as Unique Integer Embedding

```

Data: sequence: string
Result: integer embedding
SET val TO 0;
for i, base IN enumerate(sequence) do
  val := val + encoding...mapping[base] * (base_size^i);
end
// see Algorithm 1 for base_size definition
RETURN val

```

Algorithm 2 converts a sequence into a numerical value using a positional numeral system with a specified base size (base_size). In this system, each character's contribution to the final value is weighted exponentially according to its position, giving characters in earlier positions a greater influence on the encoded value. Consequently, variations in early positions result in more significant changes in the

numerical representation, which in turn has a substantial impact on clustering and similarity calculations.

The rationale behind this approach is that differences in early positional should have a dominant effect on similarity scores and clustering outcomes, reflecting their greater importance in distinguishing sequences. By scaling the significance of positional differences, the encoded integers support more meaningful clustering and similarity assessments.

After each string was embedded into its unique numerical representations, it was converted into a byte array to facilitate compression and search processes. [Algorithm 3](#) then divided the unique integer representing each string into smaller chunks, which were converted into bytes.

Algorithm 3. Algorithm for encoding cluster as vector embeddings

```

Data: cluster: list of strings
Result: bytearray
SET ec TO [];
SET N TO len(cluster);
SET indices TO [];
SET lengths TO [];
SET byte_array TO N.to_bytes(4, "little") // Function to_bytes converts an integer to byte array
for sequence, index IN cluster do
  SET length TO floor((len(sequence) + self._s * self._bits - 1) / 8 / self._bits);
  SET es TO self.encode_sequence(sequence).to_bytes(length, "little")
  // Function to_bytes converts an integer to byte array
  ec.append(es);
  indices.append(index);
  lengths.append(length);
end
for index IN indices do
  byte_array := byte_array + index.to_bytes(4, "little") // Function to_bytes converts an integer to byte array
end
for length IN lengths do
  byte_array := byte_array + length.to_bytes(1, "little") // Function to_bytes converts an integer to byte array
end
for es IN ec do
  byte_array += es;
end
RETURN byte_array

```

It is important to note that this step results in a slight increase in size. However, the performance gains achieved from compressing clustered byte arrays based on sequence similarity outweigh this temporary increase.

To reconstruct the original FASTA or FASTQ file from the compressed byte array, we needed to store indices as delimiters between subarrays. This requirement was the main reason for the slight increase in size between the numerical integer representations of the strings and the byte array. Since the sequences in the FASTA or FASTQ file were reordered to optimize clustering and maximize compression, this step was essential. While this reordering was not relevant for a cDNA library where barcodes are used, it is crucial for plain FASTA files, where maintaining the sequence order is important for concatenating short reads into larger genes.

Ultimately, constructing these vector embeddings for gene sequences as byte arrays was essential for both compression and similarity search, as discussed in more detail later in this paper.

2.3. Clustering

Before delving into the compression process, it is important to explain the theory behind the proposed method for improving the compression ratio and to note previous efforts to enhance the industry standard for genetic data compression.

[Mansouri et al. \(2020\)](#) introduced a novel binary method of encoding genes, in which genetic information is stored in four files. This method uses a recursive decision step to encode the most frequent base pair (A, T, C, or G) as one, while the others are encoded as zeros. The resulting file is then compressed. According to their study, the DNA-SBE system “outperforms state-of-the-art compressors and proves its efficiency in terms of special conditions imposed on compression data.” The paper

provides useful graphics and pseudocode that allow the reader to comprehend the methodology and replicate the study.

A study by Tang et al., referenced in the introduction, identified several useful unsupervised algorithms that could improve biological representations by leveraging characteristics from various compression models (Tang and Li, 2021). The ensemble method they outlined achieved more effective compression and provided a more refined approach to dimensionality reduction, particularly with non-negative matrix factorization (NMF), compared to traditional techniques such as principal component analysis (PCA). Furthermore, Tang et al. proposed that in unsupervised compression of short-read databases, clustering based on vector embeddings rooted in sequence similarity can losslessly improve certain compression methods.

Our clustering method builds on this concept with a modified version of vector-embedding k-means clustering that clusters granularly into groups of three before forming larger chunks. This modularity is advantageous because certain compression algorithms perform better with larger clusters, while others are more effective with tightly clustered, highly similar sequences (Nti-Addae et al., 2019). The common thread among compression methods that benefit from this clustering method is their focus on reducing file size by addressing redundancies.

In addition to the algorithm detailed below, our method involves selecting features with the highest variance to inform the grouping of processes for compression. This approach ensures that the compression process is optimized for the specific characteristics of the data being compressed.

Algorithm 4. Clustering

```

Data: V: array, l: Optional[float] = 0.3
Result: array
if len(V) < 3 then
    RETURN [V]
end
else
    SET c1, c2 TO kmeans(V, 2);
    if len(c1) EQUALS 0 then
        RETURN [c2]
    end
    if len(c2) EQUALS 0 then
        RETURN [c1]
    end
    if len(c1) + len(c2) ≥ 4 then
        if len(c1) EQUALS 1 and len(c2) EQUALS 1 then
            SET ca, cb TO c1, c2;
            SET cbmin TO mindist(ca[0], cb);
            if dist(ca[0], cbmin) < 1 * dist(mindist(cbmin, cb[cb ≠ cbmin])) then
                SET ca TO np.array([ca[0], cbmin]);
                SET cb TO cb[cb ≠ cbmin];
                RETURN cluster(cb, l) + [ca];
            end
        end
    end
    else
        RETURN cluster(c1, l) + cluster(c2, l);
    end
end
else
    RETURN [c1, c2]
end
end

```

One of the key strengths of this methodology is its ability to learn codon sequences in a way similar to how words are learned in sentences. By identifying patterns in codon usage, we can better understand the roles amino acids play in proteins with various folding patterns, reverse-engineer their functions in organisms, and gain deeper insights into natural proteins.

Another strength of this methodology is that if the hypothesis regarding vector embedding and context learning for codons is validated, it could greatly enhance the efficiency of protein sequence searches and enable better compression of cDNA files. This would be especially valuable for archiving purposes in libraries, where smaller file sizes are required.

3. Results

3.1. Context learning

To extract meaning from the vector embeddings of genes, we can draw parallels with natural language processing. In this context, we can treat codons as words and sequences as sentences. To convert raw sequences into codons, nucleotide triplets were mapped to their corresponding one-letter amino acid codes. For each short sequence, three frames were tested, and the frame with the fewest non-assignments was chosen.

$$L(\theta) = \prod_{k=1}^T \prod_{-m \leq j \leq m} \Pr(w_{k+j} | w_k, \theta)$$

Further, the loss function was defined as the minimization of a negative likelihood. In the following equation, m represents the window size, which in this case is set to 2.

As shown in Fig. 2, the one-hot-encoded version of the codon was initially fed in as the input. In the hidden layer, the weights are given initial determination, followed by gradient descent and cross-entropy calculations with the target to update the weights in the hidden layer, as illustrated in the diagram above.

Although this issue is uncommon, particularly with smaller datasets, larger training sets might lead to more frequent encounters with non-mapping amino acids that do not follow a discernible pattern, acting as a catch-all. In our scripts, we exclude non-mapping amino acids from the overall table and the graph. However, if these non-mapping amino acids are plotted, this inclusion moves the result further from the remaining amino acids as the training date increases, causing the remaining amino acids to cluster more closely together.

One way around this would be to exclude any short read containing a non-mapping amino acid; however, this approach risks excluding valuable data due to single nucleotide misreads. Alternatively, we could exclude non-mapping data from the model itself, but this would exclude training on non-mapped amino acids and their subsequent amino acids, depending on the window size used.

For a rotating 3D version of these figures, please run this [script](#) (note: matplotlib is required). Furthermore, animations of Figs. 3–5 are linked [here](#); however, it should be noted that these were created using animation software,

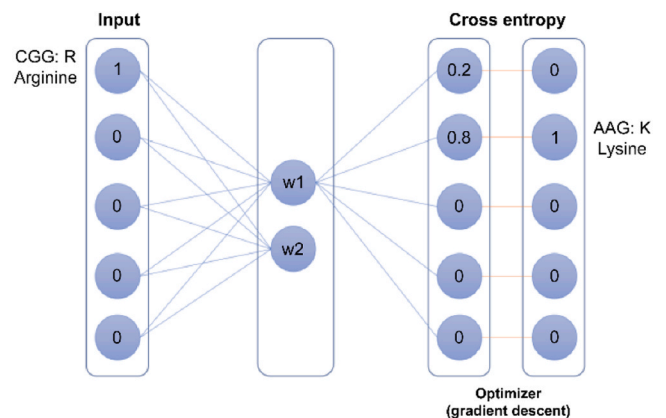


Fig. 2. The model for learning weights in codon context begins with an input layer consisting of the one-hot-encoded codons. These encoded vectors are fed into the hidden layer, where initial weights are determined. The hidden layer then processes the codon contexts by performing initial computations. To optimize these weights, gradient descent is applied to minimize the loss function. The model uses cross-entropy loss, comparing the predicted and actual codon contexts, to iteratively update the hidden layer's weights. This process ensures the model develops meaningful representations of codon contexts. Finally, the output layer utilizes the optimized weights to predict the next codon in the sequence, using the softmax function to produce a probability distribution over the possible codons.

and the locations are approximations.

As noted in the Methods section, amino acids were first mapped to one-hot-encoded vectors and used as input. A linear neuron served as a lookup table within the hidden layer, where weights were optimized using gradient descent, and updated weights were backpropagated through the hidden layer. The output layer was optimized using the softmax function.

As can be seen in Figs. 3 to 5, patterns of clustering appear to emerge between amino acids with similar properties. Notably, the 3D structures and the binding site datasets exhibit similar embeddings, where certain amino acids (such as proline, valine, alanine, arginine, and tyrosine) tend to cluster together, as do methionine, cysteine, glycine, tryptophan, phenylalanine, and glutamine. In contrast, the alternative splicing dataset produces substantially different embeddings, likely because the sequences in this dataset are entropically less optimal by definition.

3.2. Clustered compression - bits per base, encoding method, clustering algorithm

This section demonstrates how bits per base, encoding method, and compression algorithm all influence the effectiveness of clustered compression. As shown in Fig. 7, to examine a variety of bits per base options (i.e., 2, 4, and 6), three encoding methods (one-hot-encoding, standard encoding, and no encoding), and five compression algorithms (lzma, zlib and gzip, bz2, and zpac). These permutations were applied to three tester files, pathogen.fa, moriarty-neg.fa, and sandmous.fa. For more information on the aforementioned parameters, such as bits per base, encoding, and compressors, refer to the Methods section.

The best performance was achieved with 8 bits per base, using standard or no encoding, and applying lzma on pathogen.fa. Among the different parameters tested, bits per base had the largest effect on performance, with 4 bits per base showing the largest variance, based on the coupled compressor. It is also noteworthy that one-hot-encoding performed nearly as well as standard encoding, despite requiring additional index information, where both were clustered by sequence similarity.

Initially, lzma appeared to perform the best, while zpac was notably less effective. However, when comparing gzip and lzma, it's important to note the trade-off between speed and compression ratio. Although lzma generally offers better performance, it takes around 10 % longer to compress compared to gzip, which is the current industry standard. For this reason, a researcher requiring rapid compression and decompression might decide to stick to gzip, whereas someone working with large datasets for long-term storage might choose lzma. A comprehensive table with 90 rows detailing the complete compression experiment can be found in the appendix.

The primary compression finding is detailed in Section 3.3, where we compare the performance of the same compression algorithm, with and without clustering, to test the hypothesis that clustering enhances the utilization of redundancies by grouping similar sequences.

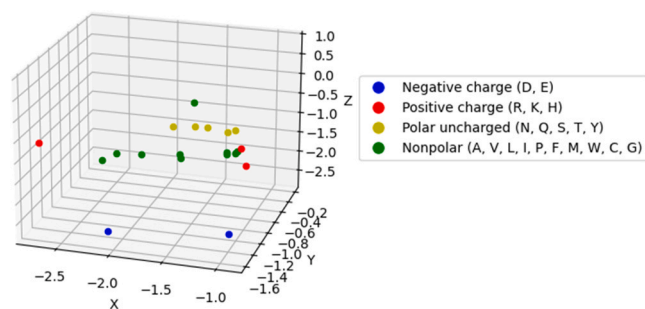


Fig. 3. Context embeddings trained on short-sequence Uniprot FASTA files with 3D structures – N = 17,552.

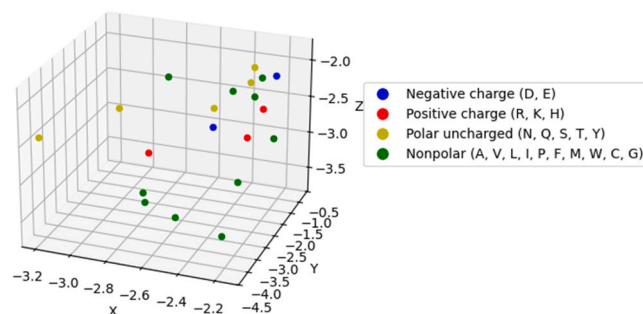


Fig. 4. Context embeddings trained on short-sequence Uniprot FASTA files for Swiss-prot reviewed binding site – N = 28,052.

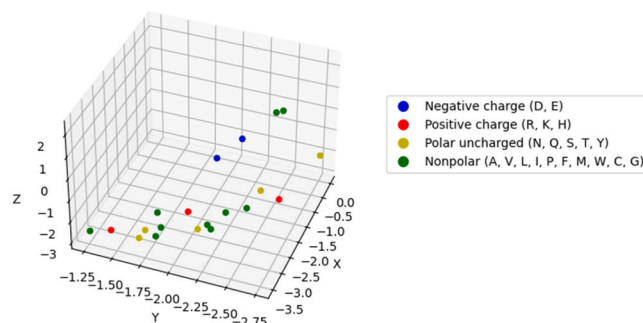


Fig. 5. Context embeddings trained on short-sequence Uniprot FASTA files for human alternative splicing – n = 10,541.

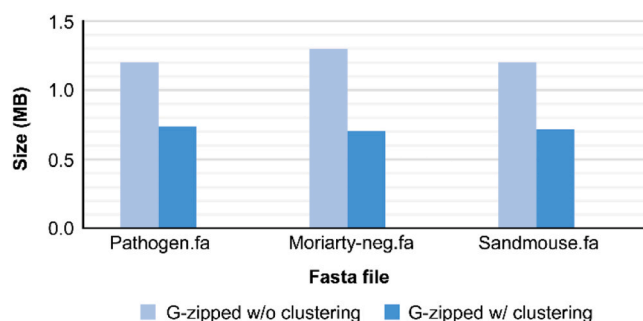


Fig. 6. Bar chart showing compression improvements for the FASTA files pathogen.fa, moriarty-neg.fa, and sandmous.fa, comparing gzip without clustering and size after gzip with clustering. Note that the original one-hot encoded FASTA file sizes were all ~24.3 MB for all three files.

3.3. Clustered compression - zipped with vs without clustering

The initial compression experiments were performed on three FASTA files – pathogen.fa, moriarty-neg.fa, and sandmous.fa – each approximately 20 MB in size. For the preliminary tests, gzip was used as a compression algorithm due to its integration with Python and widespread use for FASTA files (Pinho and Pratas, 2014).

As can be seen in Fig. 6 above, clustering improved the compression performance by about 50 %. Notably, moriarty-neg.fa showed the best-clustered compression, but it had the worst non-clustered performance. This disparity demonstrates that compression effectiveness depends on the sequence similarity within each cluster and the inherent clusterability of the original dataset.

The cluster compression method involved grouping sequences into clusters based on sequence similarity, using a sequence alignment tool, such as Clustal Omega. Each cluster was then compressed individually using gzip. This approach leverages the redundancy within each cluster

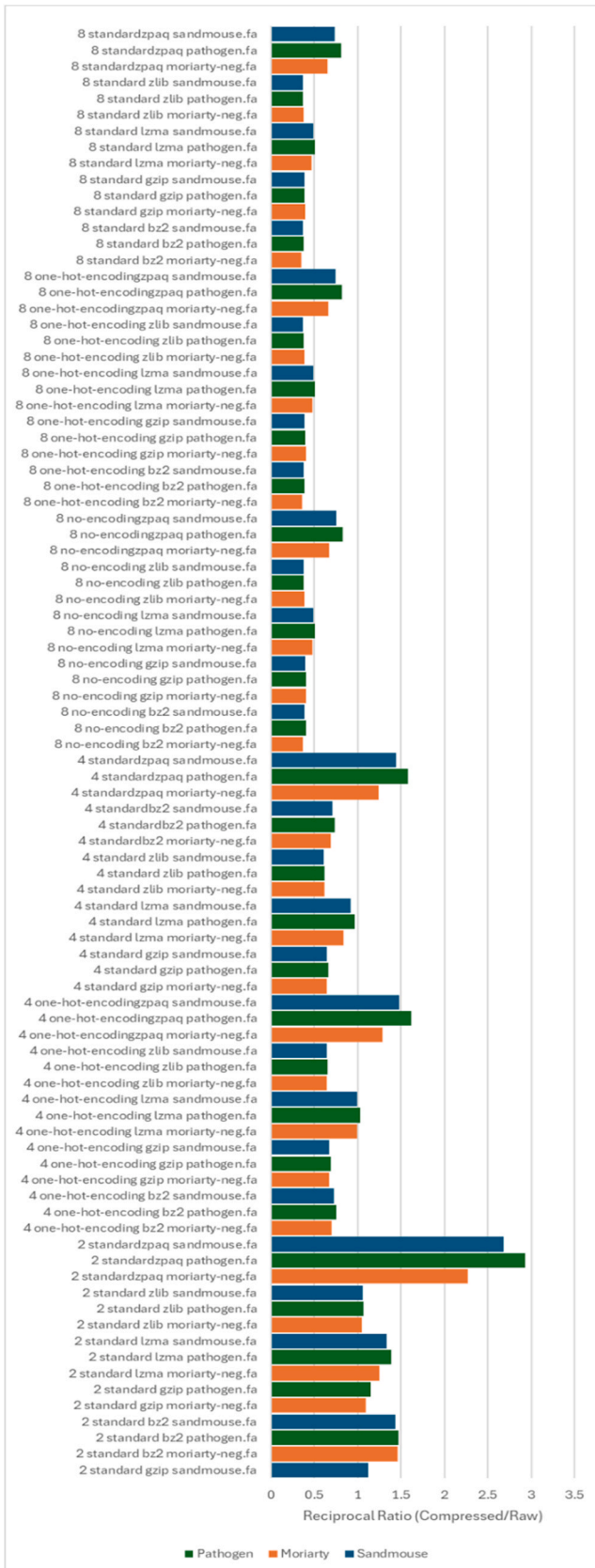


Fig. 7. Reciprocal compression ratio (compressed/uncompressed¹¹) performance vs. bits per base, encoding, compressor on the three FASTA files pathogen.fa, moriarty-neg.fa, and sandmouse.fa. Anything below 1 indicates successful compression.

to achieve better compression ratios.

3.3.1. Data restoration accuracy and impact on subsequent analysis

The clustered compression method retains the ability to fully and accurately restore the original data. After decompression, sequences can be accurately reassembled into their original clusters, ensuring data integrity. However, any errors occurring during compression or decompression could potentially result in loss of sequence information or misalignment of clusters. Such issues could impact subsequent analyses that rely on precise sequence data, such as phylogenetic studies or mutation tracking. Thus, ensuring the reliability of the compression and decompression processes is essential.

3.4. Similarity search and speed comparisons

In this section, we compare the speed of similarity between DNA sequences using vector embeddings versus transposition-based string similarity searches. Vector embeddings were used to index our set of vectors x_i in dimension d . To search for the closest indices, including the query itself, the method computes argmin_i to identify the nearest neighbors. Unlike string similarity searches, which handle one sequence at a given time (i.e., batch processing), vector-embedding similarity searches can process several sequences simultaneously. This approach can specify the specific ratio between precision and speed. Additionally, vector embedding searches can perform the maximum inner product search (i.e., $\text{argmax}_i(x_i, x_j)$) instead of the minimum Euclidean search. The FAISS package from Facebook research enhances this process by supporting radius-based queries (range searches) and storing indices on disk rather than as RAM. The following equation provides how vector-embedding similarity searches are performed using FAISS with argmin Euclidean distance (L^2).

$$j = \text{argmin}_i \|x - x_i\|$$

Jellyfish is a library used for performing transposition-based string similarity searches. The following equation outlines Jaro similarity with Jellyfish.

$$\text{sim}_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3} \left(\frac{\text{matches}}{\text{str}_1} + \frac{\text{matches}}{\text{str}_2} - \frac{\text{matches} - \text{transpositions}}{\text{matches}} \right) & \text{otherwise} \end{cases}$$

For the purposes of this experiment, the Jaro distance was used because it focuses on a measure of characters in common, making it a more direct comparison to vector embedding similarity searches. Other methods in the Jellyfish package, such as Levenshtein distance (which counts the number of edits needed to convert one string into the other) and Hamming distance (which counts the number of mismatched characters), have additional complexities that are less relevant to this comparison.

This allows for a more direct comparison between vector-embedding string similarity search and the Jaro distance, as Jaro distance measures similarity based on character matches, rather than penalizing mismatches or edit operations.

As can be seen in the results of this experiment in Table 1 and Fig. 8, although the Jellyfish string similarity search performs well on smaller datasets, it does not scale efficiently with larger numbers of sequences compared to FAISS. Jellyfish's performance deteriorates exponentially with increasing dataset size while vector-embedding similarity search with FAISS maintains relatively consistent time complexity.

FAISS's effectiveness can be explained by the batch processing of vector indices, unlike the pairwise similarity search performed by Jellyfish. The ability of FAISS to handle batches of vectors rather than individual comparisons allows it to scale effectively with large datasets. Using trained embeddings, the FAISS approach leverages the high-dimensional representation of sequences to quickly and accurately identify similar sequences, benefiting from the computational

Table 1
Summary stats for Fasta files:.

File	Number of Sequences	Mean Length	Mean GC Content
3D_Structure_1_200_seq_length_17552.fasta	17,552	129.45	9.04
alternative_splicing_human_10541.fasta	10,541	679.63	8.87
binding_site_28052.fasta	28,052	150.10	9.66

*GC-content refers to the percentage of guanine (G) and cytosine (C) bases in a DNA or RNA molecule, representing the proportion of these bases among the total nitrogenous bases (including adenine and thymine in DNA, and adenine and uracil in RNA).

Table 2
Speed comparison between vector-based and string-based similarity scoring.

Sequences	Vector-based (s)	String-based (s)
10	0.2105	0.0042
100	0.0762	0.0278
1000	0.0968	0.1308
10000	0.0181	1.2905
100000	0.0213	12.8318

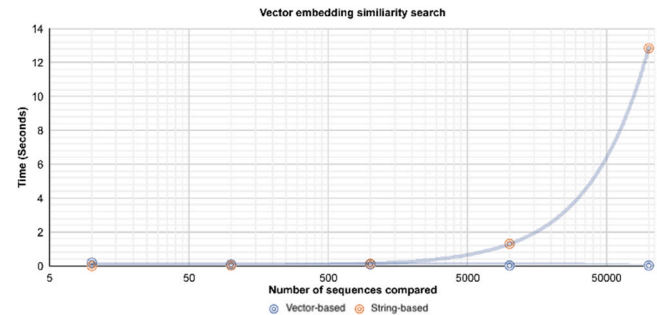


Fig. 8. Speed comparison plot between vector-based and string-based similarity scoring.

efficiencies of vector operations over traditional string comparison methods. Consequently, the FAISS offers significant advantages in processing large datasets with minimal degradation in performance.

3.5. Summary of key findings and strengths

- **Context learning:** By applying vector embeddings to genes, inspired by natural language processing, we effectively clustered amino acids with similar properties. However, embeddings differed notably in alternative splicing datasets due to their higher entropy. This is supported by the data from the three images particularly for nonpolar amino acids, 65 % of the nonpolar amino acids exhibited linear separability in three dimensions.
- **Clustered compression:** Implementing compression with clustering improved performance by approximately 50 %, with lzma performing the best among tested algorithms. The bits per base significantly influenced outcomes, with 4 bits per base showing the most variation. The analysis compared file sizes after gzip compression, with and without clustering, using a paired t-test. The null hypothesis (H_0) posited that clustering would have no significant effect on compression efficiency, meaning there would be no significant difference in file sizes between the two methods. The results showed a mean file size reduction of 0.67 MB with clustering, a t-statistic of 20.00, and a p-value of 0.0025, with a 95 % confidence interval from 0.52 MB to 0.81 MB. These findings indicate a statistically significant improvement in compression efficiency with clustering, leading to

the rejection of the null hypothesis. Consequently, clustering before gzip compression is confirmed to significantly enhance file size reduction, particularly beneficial for large datasets. However, given the small sample size, the t-test may not fully capture the variability, and practical significance should be considered alongside statistical significance. Work has shown below:

The mean difference in file sizes between gzip without clustering and with clustering was calculated using the formula:

$$\text{MeanDifference} = \frac{\sum (\text{FileSizeWithoutClustering} - \text{FileSizeWithClustering})}{N}$$

Plugging in the values:

$$\begin{aligned} \text{MeanDifference} &= \frac{(1.2 - 0.6) + (1.5 - 0.8) + (1.4 - 0.7)}{3} \\ &= \frac{0.6 + 0.7 + 0.7}{3} = 0.67\text{MB} \end{aligned}$$

The paired t-test statistic (t) is calculated as:

$$t = \frac{\text{MeanDifference}}{\text{StandardErroroftheDifference}}$$

In this case, the t-statistic was calculated to be 20.00, with a corresponding p-value of 0.0025. This indicates a highly significant difference between the two conditions.

A 95 % confidence interval for the mean difference was constructed as:

$$\text{CI} = \text{MeanDifference} \pm (t_{\alpha/2} \times \text{StandardError})$$

The 95 % confidence interval for the mean difference is (0.52 MB, 0.81 MB).

- **Similarity search and speed comparisons:** Vector-embedding similarity search demonstrated superior scalability over string-based similarity search, maintaining relatively constant time complexity as dataset sizes increased. The vector-based method significantly outperforms the string-based method for smaller sequence sizes, showing improvements of 50x and 2.7x for 10 and 100 sequences, respectively. However, for 1000 sequences, the string-based method is faster. As the sequence size increases, the vector-based method demonstrates substantial gains, achieving a 71x speed-up for 10,000 sequences and an impressive 602x speed-up for 100,000 sequences.

4. Limitations

While this study demonstrates the advantages of using vector embeddings for gene sequence representation, several limitations must be acknowledged:

1. **Theoretical Justification:** While empirical results show improvements in clustering and compression performance, a more rigorous theoretical foundation is necessary to justify their effectiveness, particularly in the context of biological binding patterns. Given the translational nature of this paper, the theoretical insights are better understood from the natural language processing standpoint:

Both Continuous Bag of Words (CBOW) and skip-gram are methods for learning vector representations of words in a corpus. CBOW uses the vector-sum of a word's neighbors to predict the word's vector itself, while skip-gram focuses on having a word's vector close to its neighbors.

For CBOW, the training objective is to maximize the log probability:

$$\sum_{i \in C} \ln \Pr(w_i | w_j : j \in N + i)$$

using the vector sum:

$$v := \sum_{j \in N+i} v_{w_j}$$

and a dot-product-softmax model:

$$\Pr(w | w_j : j \in N + i) := \frac{e^{v_w \cdot v}}{\sum_{w \in V} e^{v_w \cdot v}}$$

For skip-gram, the objective is to maximize the log-probability of the context words given the target word:

$$\sum_{i \in C} \ln \Pr(w_j : j \in N + i | w_i)$$

follow a similar probability model.

While the models offer a mathematical basis that can potentially extend to clustering and compression in biological contexts, the theoretical explanation remains rooted in linguistic theory. The study still lacks a comprehensive theoretical justification specific to biology and genetics.

2. **Positional Dependency in Base Size Encoding:** A limitation of this study is the positional dependency of the base size encoding in [Algorithm 2](#), which can affect the consistency and reliability of the embeddings. The essence of [Algorithm 2](#) involves converting sequences into numerical values based on base size, with the position and type of bases significantly influencing the final encoded value. This leads to varying degrees of impact on clustering and sequence similarity calculations, particularly for sequences where certain positions are more critical than others.
3. **Clarity vs. Information Density in Figures:** A limitation of this study is the balance between clarity and information density in the figures. While the primary focus is to highlight how amino acids with similar properties cluster together when training on sequences (analogous to words in sentences), the figures struggle to convey this clearly. Attempts to use vector graphics for more detailed insights resulted in overly dense and messy figures, which detracted from the primary point.
4. **Accuracy Comparisons:** A limitation of this study is the lack of accuracy comparisons through fuzzy searches or inexact matches, which would be helpful in understanding the performance of vector-based similarity searches. Speed was prioritized due to its primary advantage in embedding-based search models. Although preliminary experiments showed that accuracy performance was not significantly different, the focus on near-exact searches limited the number of variables tracked.
5. **Reliance on FAISS:** The study heavily relies on Facebook Research's FAISS algorithm for efficient similarity searches, which poses a potential limitation. Any changes or limitations within the FAISS library could directly impact the effectiveness of the proposed method. Exploring alternative algorithms or discussing potential fallbacks would enhance the robustness of the proposed approach.
6. **Generalizability of Results:** The generalizability of the results is limited by the datasets used in this study. Although diverse, these data may not encompass the full range of variability seen in real-world genetic data. Further testing on a broader array of datasets is necessary to confirm the applicability of the findings across different contexts.
7. **FASTA-Specific Compression:** Although gzip is a common compression algorithm for FASTA files and is widely used in the industry (e.g., by companies such as Fluent BioSciences), considering FASTA-specific algorithms such as MFCCompress ([Pinho et al., 2014](#)) in a side-by-side compression could have been informative. However, the

focus of this study was on comparing the performance of the same compression algorithm with and without clustering to test the hypothesis that clustering improves redundancy utilization.

Addressing these limitations in future research will help validate and refine the proposed methods, ensuring their robustness and applicability across different contexts.

5. Conclusions

This study highlights the advantages of using ordered numerical representations for genes in research that employs flat string formats like FASTA, FASTQ, and cDNA libraries. By assigning a unique vector embedding to each short sequence, we can achieve more efficient clustering and increased compression performance. Furthermore, applying this sequence embedding approach to encode barcodes and cDNA sequences enables the creation of a relational database. Integration with Facebook Research's FAISS algorithm facilitates rapid and modular sequence similarity searches that surpass industry-standard methods. Additionally, by learning alternative coordinate vector embeddings tailored to codon triplets' contexts, we can effectively group sequences based on their amino acid characteristics.

Future research could involve leveraging these trained vector embeddings for sequence prediction tasks. For instance, employing recurrent neural networks (RNNs) or transformers trained on these embeddings could enable accurate prediction of gene expression patterns or functional protein domains from DNA sequences. This predictive capability holds promise for advancing precision medicine by aiding in the identification of genetic markers for diseases and potential therapeutic targets. Moreover, expanding the application of vector embeddings to predict evolutionary trajectories or environmental adaptations of organisms based on genomic sequences could deepen our understanding of biodiversity and ecosystem dynamics.

Ethics approval statement

This research did not contain any studies involving animal or human participants, nor did it take place in any private or protected areas. No specific permissions were required for corresponding locations.

Funding

Prof. Kaiser's work is funded in part by DARPA/NIWC-Pacific N66001-21-C-4018 and in part by NSF CCF-1815494, CNS-2247370, and CCF-231305. Prof. Knowles' work is partially supported and funded by the Department of Systems Biology, Department of Computer Science, and the Data Science Institute at Columbia University and the New York Genome Center. Daniel Um declares that he has no known competing financial interests or personal relationships that could have appeared to influence his contribution to this paper.

CRediT authorship contribution statement

Daniel Um: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Gail E. Kaiser:** Writing – review & editing, Supervision. **David A. Knowles:** Writing – review & editing, Supervision.

Declaration of generative AI and AI-assisted technologies in the writing process

No generative AI was used in the writing of this manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial

interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

I am grateful to Shouvik Mani and Leslie Ramos for reviewing the manuscript for scientific soundness, Jincheng Xu for proofreading the machine learning theory, Jennifer Choi-Um for sentence fluency and language edits, Samantha Um for her assistance with journal formatting, and Laura Makin for both proofreading and converting the equations from LaTeX to MathType.

GitHub repository

danielum16/cDNA-Vector-Embeddings: Repository for the paper "Vector Embeddings by Sequence Similarity and Context for Improved Compression, Similarity Search, Clustering, Organization, and Manipulation of cDNA Libraries." (github.com)

Mendeley data

Context embeddings trained on short sequence FASTA files with binding site on Mendeley Data are associated with "Fig. 4: Context embeddings trained on short-sequence FASTA files with binding site – N = 28,052" for "Vector Embeddings by Sequence Similarity and Context for Improved Compression, Similarity Search, Clustering, Organization, and Manipulation of cDNA Libraries" by Daniel H. Um et al.

Context embeddings trained on short sequence FASTA files for human alternative splicing on Mendeley Data are associated with "Fig. 5: Context embeddings trained on short-sequence FASTA files for human alternative splicing – n = 10,541" for "Vector Embeddings by Sequence Similarity and Context for Improved Compression, Similarity Search, Clustering, Organization, and Manipulation of cDNA Libraries" by Daniel H. Um et al.

Context embeddings trained on short sequence FASTA files with 3D structures on Mendeley Data are associated with "Fig. 3: Context embeddings trained on short-sequence FASTA files with 3D structures – N = 17552" for "Vector Embeddings by Sequence Similarity and Context for Improved Compression, Similarity Search, Clustering, Organization, and Manipulation of cDNA Libraries" by Daniel H. Um et al.

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at [doi:10.1016/j.compbiolchem.2024.108251](https://doi.org/10.1016/j.compbiolchem.2024.108251).

Data availability statement

The datasets for this investigation are available in the Mendeley Data repository, itemized below. Additional data not in the aforementioned repository are available from the corresponding author upon request.

References

- Al Yami, S., Huang, C.H., 2019. LFASTQC: a lossless non-reference-based FASTQ compressor. PLOS ONE 14, e0224806. <https://doi.org/10.1371/journal.pone.0224806>.
- Bagheri, H., Muppirala, U., Masonbrink, R.E., Severin, A.J., Rajan, H., 2019. Shared data science infrastructure for genomics data. BMC Bioinforma. 20, 436. <https://doi.org/10.1186/s12859-019-2967-2>.
- Carver, T., Berriman, M., Tivey, A., Patel, C., Böhme, U., Barrell, B.G., Parkhill, J., Rajandream, M.A., 2008. Artemis and ACT: viewing, annotating and comparing sequences stored in a relational database. Bioinformatics 24, 2672–2676. <https://doi.org/10.1093/bioinformatics/btn529>.
- Chandak, S., Tatwawadi, K., Ochoa, I., Hernaez, M., Weissman, T., 2019. Spring: A next-generation compressor for FASTQ data. Bioinformatics 35, 2674–2676. <https://doi.org/10.1093/bioinformatics/bty1015>.
- Chen, S., Huang, T., Zhou, Y., Han, Y., Xu, M., Gu, J., 2017. AfterQC: automatic filtering, trimming, error removing and quality control for FASTQ data. BMC Bioinforma. 18, 80. <https://doi.org/10.1186/s12859-017-1469-3>.
- Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E., Notredame, C., 2017. Nextflow enables reproducible computational workflows. Nat. Biotechnol. 35, 316–319. <https://doi.org/10.1038/nbt.3820>.
- Du, L., Liu, Q., Fan, Z., Tang, J., Zhang, X., Price, M., Yue, B., Zhao, K., 2021. Pyfastx: A robust python package for fast random access to sequences from plain and gzipped FASTA/Q files. Brief. Bioinform. 22, bbaa368. <https://doi.org/10.1093/bib/bbaa368>.
- Faksri, K., Tan, J.H., Chaiprasert, A., Teo, Y.Y., Ong, R.T., 2016. Bioinformatics tools and databases for whole genome sequence analysis of mycobacterium tuberculosis. Infect. Genet. Evol. 45, 359–368. <https://doi.org/10.1016/j.meegid.2016.09.013>.
- Flot, J.F., 2010. Seqphase: A web tool for interconverting phase input/output files and FASTA sequence alignments. Mol. Ecol. Resour. 10, 162–166. <https://doi.org/10.1111/j.1755-0998.2009.02732.x>.
- Friis, S.L., Buchard, A., Rockenbauer, E., Børsting, C., Morling, N., 2016. Introduction of the python script Strinings for analysis of STR regions in FASTQ or BAM files and expansion of the Danish STR sequence database to 11 Strs. Forensic Sci. Int. Genet. 21, 68–75. <https://doi.org/10.1016/j.fsigen.2015.12.006>.
- Gimadiev, T., Nugmanov, R., Batyrshin, D., Madzhidov, T., Maeda, S., Sidorov, P., Varnek, A., 2021. Combined graph/relational database management system for calculated chemical reaction pathway data. J. Chem. Inf. Model. 61, 554–559. <https://doi.org/10.1021/acs.jcim.0c01280>.
- Guerrero-Sanchez, V.M., Maldonado-Alconada, A.M., Sánchez-Lucas, R., Rey, M.D., 2020. Specific protein database creation from transcriptomics data in nonmodel species: holm oak (Quercus ilex L.). In: Jorriñ-Novó, J., Valledor, L., Castillejo, M., Rey, M.D. (Eds.), Plant Proteomics. Methods in Molecular Biology, 2139. Humana, New York, NY, pp. 57–68. https://doi.org/10.1007/978-1-0716-0528-8_4.
- Hospital, A., Andrio, P., Cugnasco, C., Codo, L., Becerra, Y., Dans, P.D., Battistini, F., Torres, J., Goni, R., Orozco, M., Gelpi, J.L., 2016. BIGNASim: A NoSQL database structure and analysis portal for nucleic acids simulation data. Nucleic Acids Res. 44, D272–D278. <https://doi.org/10.1093/nar/gkv1301>.
- Huber, N., Parson, W., Dür, A., 2018. Next generation database search algorithm for forensic mitogenome analyses. Forensic Sci. Int. Genet. 37, 204–214. <https://doi.org/10.1016/j.fsigen.2018.09.001>.
- Jegou, H., Douze, M., Johnson, J., 2018. Faiss: A library for efficient similarity search. Engineering at meta. (<https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/>). (accessed 13 March 2023).
- Kowalski, T., Grabowski, S., 2020. Engineering the compression of sequencing reads. bioRxiv 2, 2020–2005. <https://doi.org/10.1101/2020.05.01.071720>.
- Kozanitis, C., Patterson, D.A., 2016. GenAp: a distributed SQL interface for genomic data, 4 February BMC Bioinforma. 17, 63. <https://doi.org/10.1186/s12859-016-0904-1>.
- Kryukov, K., Ueda, M.T., Nakagawa, S., Imanishi, T., 2020. Sequence compression benchmark (SCB) database-A comprehensive evaluation of reference-free compressors for FASTA-formatted sequences. GigaScience 9, gaa072. <https://doi.org/10.1093/gigascience/giaa072>.
- Liu, Y., Yu, Z., Dinger, M.E., Li, J., 2019. Index suffix-prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression. Bioinformatics 35, 2066–2074. <https://doi.org/10.1093/bioinformatics/bty936>.
- Luo, Q., Guo, C., Zhang, Y.J., Cai, Y., Liu, G., 2018. Algorithms designed for compressed-gene-data transformation among gene banks with different references. BMC Bioinforma. 19, 230. <https://doi.org/10.1186/s12859-018-2230-2>.
- Mansouri, D., Yuan, X., Saidani, A., 2020. A new lossless DNA compression algorithm based on a single-block encoding scheme. Algorithms 13, 99. <https://doi.org/10.3390/a13040099>.
- Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. CORR [abs]/1301.3781. <https://doi.org/10.48550/arXiv.1301.3781>.
- Nti-Addae, Y., Matthews, D., Ulat, V.J., Syed, R., Sempéré, G., Pétel, A., Renner, J., Larmande, P., Guignon, V., Jones, E., Robbins, K., 2019. Benchmarking database systems for genomic selection implementation. Database (Oxf.) 2019. <https://doi.org/10.1093/database/baz096>.
- Pinho, A.J., Pratas, D., 2014. MFCompress: a compression tool for FASTA and multi-FASTA data. Bioinformatics 30, 117–118. <https://doi.org/10.1093/bioinformatics/btt594>.
- Rajarajeswari, P., Apparao, A., 2011. DNABIT compress – genome compression algorithm. Bioinformation 5, 350–360. <https://doi.org/10.6026/97320630005350>.
- Roguski, L., Ochoa, I., Hernaez, M., Deorowicz, S., 2018. FaStore: A space-saving solution for raw sequencing data. Bioinformatics 34, 2748–2756. <https://doi.org/10.1093/bioinformatics/bty205>.
- Shelton, J.M., Coleman, M.C., Herndon, N., Lu, N., Lam, E.T., Anantharaman, T., Sheth, P., Brown, S.J., 2015. Tools and pipelines for BioNano data: molecule assembly pipeline and FASTA super scaffolding tool. BMC Genom. 16, 734. <https://doi.org/10.1186/s12864-015-1911-8>.
- Shen, W., Le, S., Li, Y., Hu, F., 2016. SeqKit: A cross-platform and ultrafast toolkit for FASTA/Q file manipulation. PLOS ONE 11, e0163962. <https://doi.org/10.1371/journal.pone.0163962>.
- Tang, T., Li, J., 2021. Transformation of FASTA files into feature vectors for unsupervised compression of short reads databases. J. Bioinform. Comput. Biol. 19, 2050048. <https://doi.org/10.1142/S0219720020500481>.
- Van der Auwera, G.A., Carneiro, M.O., Hartl, C., Poplin, R., Del Angel, G., Levy-Moonshine, A., Jordan, T., Shakir, K., Roazen, D., Thibault, J., Banks, E., Garimella, K.V., Altshuler, D., Gabriel, S., DePristo, M.A., 2013. From FastQ data to high confidence variant calls: the genome analysis toolkit best practices pipeline,

- 11.10.1–11.10.33 Curr. Protoc. Bioinforma. 43. <https://doi.org/10.1002/0471250953.bi1110s43>.
- Way, G.P., Zietz, M., Rubinetti, V., Himmelstein, D.S., Greene, C.S., 2020. Compressing gene expression data using multiple latent space dimensionalities learns complementary biological representations. *Genome Biol.* 21, 109. <https://doi.org/10.1186/s13059-020-02021-3>.
- Williams, A., Tkachenko, V., 2014. The Royal Society of Chemistry and the delivery of chemistry data repositories for the community. *J. Comput. Aided Mol. Des.* 28, 1023–1030. <https://doi.org/10.1007/s10822-014-9784-5>.
- Zhang, T., Miao, J., Han, N., Qiang, Y., Zhang, W., 2018. MPD: a pathogen genome and metagenome database. *Database (Oxf.)* 2018, bay055. <https://doi.org/10.1093/database/bay055>.