Modeling Lossless Genome Compression in Viruses Using Context Mixing and Arithmetic Encoding

Srivarma Battini

School of Computing, Amrita Vishwa Vidyapeetham, Amritapuri, Kollam, 690525, Kerala, India.

.

Abstract

With the rapid growth of genomic data, efficient and lossless compression of genome sequences has become a critical challenge in bioinformatics. Traditional compression tools fail to exploit the inherent patterns in nucleotide sequences, leading to suboptimal storage efficiency. This project addresses the problem by applying statistical modeling techniques—specifically n-gram based context mixing combined with arithmetic encoding—to compress viral genome data effectively. The original sequences are then accurately reconstructed using arithmetic decoding, ensuring fully lossless decompression.

Keywords: Context Mixing, Arithmetic Encoding, n-gram Model, Lossless Compression, Probabilistic Modeling, Viral Genomes

1 Motivation

The rapid growth of genomic data in modern biology, virology, and medicine has created an urgent need for efficient storage and transmission of genome sequences. Viral genomes play a crucial role in areas such as disease surveillance, vaccine development, and genetic research. However, these sequences are usually stored in text formats like FASTA, which are not designed for space or speed. As a result, large amounts of genomic data can quickly overwhelm storage systems and create challenges for sharing and processing, especially in environments with limited resources.

General-purpose compression tools like ZIP and GZIP treat genome sequences as plain text, overlooking the biological structure and redundancy in nucleotide patterns. As a consequence, they provide only limited compression performance. There is a growing demand for specialized, lossless compression methods that take advantage of domain-specific patterns to achieve better results. In this context, statistical modeling approaches such as n-gram analysis can learn the dependencies between nucleotides, allowing for improved prediction and compression.

This project is driven by the potential of combining these statistical models with encoding techniques like context mixing and arithmetic encoding. These methods not only improve the compression ratio by adapting to short- and long-range dependencies in genome data but also ensure perfect reconstruction of the original sequence. Developing efficient and reliable techniques is essential for enabling scalable genome storage, fast transmission, and accurate bioinformatics analysis in today's data-driven world.

2 Introduction

This project introduces a lossless genome compression method that uses n-gram-based statistical modeling, context mixing, and arithmetic encoding. The system is built to efficiently compress DNA sequences by modeling their biological structure with several n-gram models, like bigrams (2-gram) and 4-grams. These models predict the next nucleotide based on the preceding characters, capturing both short- and long-range dependencies.

To improve accuracy, predictions from different models are combined through context mixing. The resulting probability distributions go into an arithmetic encoder, which compresses the full sequence into a compact decimal representation. You can later decompress the compressed data perfectly with an arithmetic decoder and the same context models, ensuring lossless reconstruction.

The method is tested using four real-world viral genome datasets: SARS-CoV-2 (Coronavirus), Zika Virus, Dengue Virus, and Human Immunodeficiency Virus (HIV). These datasets vary in length and sequence composition, offering a wide range for evaluating compression performance.

To measure effectiveness, both theoretical (entropy-based) and physical (file-size based) compression ratios are calculated. The results confirm the method's ability to achieve high compression efficiency while ensuring perfect data recovery, highlighting its usefulness for scalable genome storage and processing in modern bioinformatics workflows.

3 Methodology

The proposed method for genome compression uses statistical modeling and entropy coding to achieve efficient and lossless compression of genomic sequences. The overall process includes three main stages: training n-gram models, applying context mixing to combine predictions, and using arithmetic encoding to create a compact binary representation. Decompression occurs with the same probabilistic framework, ensuring perfect reconstruction of the original sequence.

3.1 Context-Based Probability Modeling

DNA sequences are composed of four nucleotides: A, C, G, and T. Their repetitive patterns make them suitable for context-based prediction. This project uses two n-gram models:

- **2-gram model** (bigram): predicts the next nucleotide based on the previous one.
- **4-gram model**: uses the preceding three nucleotides as context.

Each model scans the genome sequence to count symbol frequencies based on context, which are then normalized into probability distributions. For unseen contexts, a uniform distribution over A, C, G, and T is used.

3.1.1 Context Mixing Strategy

To enhance prediction accuracy, the outputs of the 2-gram and 4-gram models are combined using context mixing. This technique averages the probability distributions generated by each model to produce a more reliable estimate for the next nucleotide. By combining predictions from both short and long contexts, context mixing captures both local and broader sequence patterns. This approach improves robustness, especially in regions where one of the models might be less reliable due to limited training data or rare contexts.

3.1.2 Arithmetic Encoding Process

Using mixed probabilities, the sequence is compressed with arithmetic encoding. This method represents the entire sequence as a subinterval of the range $[0, 1)$. It narrows the range for each character based on its predicted probability. The final encoded value is a single floating-point number that uniquely identifies the input sequence.

This step greatly reduces storage size by giving shorter codes to more likely sequences, as predicted by the context-mixed model. The final decimal is then converted to binary form and saved as a '.bin' file.

3.1.3 Arithmetic Decoding and Lossless Verification

To ensure that the compression is lossless, the encoded value is decoded using the same mixed probability model. The decoding process reconstructs the original sequence symbol by symbol by reversing the arithmetic encoding. The decoded output is then compared to the original sequence to ensure zero mismatches.

In all datasets tested, including SARS-CoV-2, Zika, Dengue, and HIV genomes, the decompressed sequences matched the originals exactly. This confirms the method's reliability.

3.2 Summary of Core Methods

Table 1 summarizes the core components of the proposed compression system, including statistical modeling, context mixing, and entropy coding. Each method contributes to the overall accuracy and efficiency of the pipeline.

Table 1 Core Methods Used in the Genome Compression Workflow

S.No	Method	Description
1	**N-gram Modeling**	Learns statistical patterns in DNA sequences by analyzing how often a nucleotide follows a given context of length n. This project uses 2-gram and 4-gram models.
2	**Context Mixing**	Combines predictions from multiple n-gram models (bigram and 4-gram) by averaging their probability outputs to improve robustness and accuracy.
3	**Arithmetic Encoding**	Encodes the DNA sequence as a single fractional number in the range $[0, 1)$, using the predicted probabilities to narrow the interval after each symbol.
4	**Arithmetic Decoding**	Reverses the encoding process using the same probability model to recover the original sequence character-by-character — ensuring lossless reconstruction.
5	`get_mixed_probs(i)`	Returns the mixed probability distribution at position i by extracting contexts of length 2 and 4, and combining predictions from both models using the context mixer. Used in both encoding and decoding.

4 Results

The proposed method was evaluated on four viral genome datasets to assess compression accuracy and efficiency. Each dataset varies in size and sequence complexity, providing a diverse testbed for generalization.

Table 2 Lossless Compression Results on Viral Genome Datasets

Dataset	No. of Characters	Mismatches	Theoretical Ratio	Physical Ratio
Coronavirus (SARS-CoV-2)	29,903	0	0.2537	0.2494
Zika Virus	10,794	0	0.2560	0.2515
Dengue Virus	10,735	0	0.2523	0.2477
HIV	9,181	0	0.2473	0.2416

As shown in Table 2, all datasets were compressed and decompressed with zero mismatches, confirming that there is no data loss after decompression for these methods. The physical compression ratios closely match the theoretical limits based on entropy, demonstrating the efficiency and accuracy of the context mixing and arithmetic encoding approach across different viral genome lengths.

5 Equations

This section presents the core mathematical formulations used in the proposed genome compression method, including n-gram probability estimation, context mixing, arithmetic encoding, entropy, and compression ratio.

5.1 N-gram Model Probability Estimation

The n-gram model estimates the probability of the next nucleotide based on its preceding $n - 1$ characters. The conditional probability of symbol x_i given its context is defined as:

$$P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1}) = \frac{\text{count}(x_{i-(n-1)}, \ldots, x_i)}{\text{count}(x_{i-(n-1)}, \ldots, x_{i-1})} \tag{1}$$

This formula is used for both 2-gram and 4-gram models in the system. If a specific context is not observed during training, a uniform distribution is used.

5.2 Context Mixing Formula

To improve prediction accuracy, two probability models (2-gram and 4-gram) are combined through context mixing:

$$P_{\text{mixed}}(b) = \frac{1}{2}P_{\text{2-gram}}(b) + \frac{1}{2}P_{\text{4-gram}}(b) \tag{2}$$

This approach balances short- and long-context predictions, improving the overall model robustness.

5.3 Arithmetic Encoding Range Update

Arithmetic encoding compresses a sequence by iteratively narrowing a range $[low, high)$ based on the predicted probabilities of symbols. For each symbol x_i, the range is updated as follows:

$$\begin{aligned} \text{range} &= \text{high}_i - \text{low}_i \\ \text{high}_{i+1} &= \text{low}_i + \text{range} \times C_{\text{upper}}(x_i) \\ \text{low}_{i+1} &= \text{low}_i + \text{range} \times C_{\text{lower}}(x_i) \end{aligned} \tag{3}$$

Here, $C_{\text{lower}}(x_i)$ and $C_{\text{upper}}(x_i)$ represent the cumulative probabilities for the current symbol in the sorted alphabet.

5.4 Entropy of the Sequence

Entropy provides a theoretical lower bound on the number of bits required to encode a sequence. It is computed using the predicted probabilities:

$$H = -\sum_{i=1}^{n} \log_2 P(x_i \mid \text{context}) \tag{4}$$

Where $P(x_i \mid \text{context})$ is the mixed probability of nucleotide x_i given its preceding context.

5.5 Compression Ratio

The compression ratio indicates how much the data was reduced. It is defined as:

$$\text{Compression Ratio} = \frac{\text{Compressed Size (bits)}}{\text{Original Size (bits)}} \tag{5}$$

A lower ratio implies better compression efficiency. This is reported both for theoretical (entropy-based) and actual (file-based) compression.

6 Algorithms, Program Codes

This section includes key Python implementations used in the genome compression project. Each component contributes to the compression pipeline: from probability modeling to encoding and decoding.

6.1 N-gram Model for Probability Estimation

Listing 1 N-gram model for context-based prediction

```
class NGramModel:
    def __init__(self, n):
        self.n = n
        self.context_counts = defaultdict(lambda: defaultdict(int))

    def train(self, sequence):
        for i in range(len(sequence) - self.n):
            context = sequence[i:i+self.n]
            next_char = sequence[i+self.n]
            self.context_counts[context][next_char] += 1

    def predict(self, context):
        if context not in self.context_counts:
            return {ch: 0.25 for ch in 'ACGT'}
        counts = self.context_counts[context]
        total = sum(counts.values())
        return {ch: counts.get(ch, 0) / total for ch in 'ACGT'}
```

6.2 Context Mixing of N-gram Models

Listing 2 Context mixing for combining multiple models

```
class ContextMixer:
    def __init__(self, models):
        self.models = models

    def mix(self, contexts):
        final_probs = {ch: 0 for ch in 'ACGT'}
        for i, model in enumerate(self.models):
            probs = model.predict(contexts[i])
            for ch in 'ACGT':
                final_probs[ch] += probs[ch]
        return {ch: final_probs[ch] / len(self.models) for ch in 'ACGT'}
```

6.3 Mixed Probability Fetching Function

Listing 3 Returns the mixed probability at position i

```
def get_mixed_probs(i):
    context2 = sequence[i:i+2]
    context4 = sequence[i:i+4]
    return mixer.mix([context2, context4])
```

6.4 Arithmetic Encoding

Listing 4 Arithmetic encoding using mixed probabilities

```
def arithmetic_encode(sequence, probs_fn):
    low, high = Decimal(0), Decimal(1)
    for i, ch in enumerate(sequence):
        probs = probs_fn(i)
        total = Decimal(0)
        for base in 'ACGT':
            range_width = high - low
            if base == ch:
                high = low + range_width * (total + Decimal(probs[base]))
                low = low + range_width * total
                break
            total += Decimal(probs[base])
    return (low + high) / 2
```

6.5 Arithmetic Decoding

Listing 5 Arithmetic decoding to recover the original sequence

```
def arithmetic_decode(encoded_value, probs_fn, sequence_length):
    result = ""
    value = Decimal(encoded_value)
    low = Decimal(0)
    high = Decimal(1)
    for i in range(sequence_length):
        probs = probs_fn(i)
        total = Decimal(0)
        for base in 'ACGT':
            prev_total = total
            total += Decimal(probs[base])
            range_width = high - low
            curr_low = low + range_width * prev_total
            curr_high = low + range_width * total
            if curr_low <= value < curr_high:
                result += base
                low, high = curr_low, curr_high
                break
    return result
```

7 Conclusion

This project introduces a lossless genome compression method that combines context-based statistical modeling with arithmetic encoding. By using both 2-gram and 4-gram models and integrating their predictions through context mixing, the system captures both short-range and long-range dependencies within genomic sequences. These mixed probabilities allow for accurate and compact encoding through arithmetic coding, which can then be reversed using the same model to ensure perfect reconstruction.

The method was tested on four real-world viral genome datasets: SARS-CoV-2, Zika, Dengue, and HIV. For all datasets, the compression was significant, and the decompressed output matched the original sequences with no mismatches. Both theoretical (entropy-based) and actual (file-based) compression ratios confirmed the model's effectiveness and reliability.

The results show that combining probabilistic modeling with entropy coding provides a scalable and strong solution for genome compression. This approach is particularly useful for bioinformatics workflows where efficient, lossless DNA storage and transmission are crucial.

Future improvements may involve adaptive weighting for context mixing, compression of unclear nucleotide codes, and scaling the method for larger genomic datasets across different species.

References

[1] Cevallos-Valdiviezo, H., Dávila-Montehermoso, C., Abad-Murillo, B., Zambrano, C., and Tapia, J. E. Lossless genome data compression using V-gram. In: *International Youth Conference on Electronics, Telecommunications and Information Technologies (IYCE)*, Springer, 2022, pp. 162–168.

[2] Mahoney, M. Adaptive context mixing for lossless data compression. Technical Report, Florida Institute of Technology, 2005.

[3] Langdon, G. G. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2), 135–149, 1984.

[4] National Center for Biotechnology Information (NCBI). (1983). Human immunodeficiency virus 1, complete genome. GenBank: `NC_001802.1`. https://www.ncbi.nlm.nih.gov/nuccore/NC_001802.1

[5] National Center for Biotechnology Information (NCBI). (2009). Zika virus, complete genome. GenBank: `NC_012532.1`. https://www.ncbi.nlm.nih.gov/nuccore/NC_012532.1

[6] National Center for Biotechnology Information (NCBI). (2020). Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome. GenBank: `NC_045512.2`. https://www.ncbi.nlm.nih.gov/nuccore/NC_045512.2

[7] National Center for Biotechnology Information (NCBI). (1989). Dengue virus 1, complete genome. GenBank: `NC_001477.1`. https://www.ncbi.nlm.nih.gov/nuccore/NC_001477.1