Normalized Relative Compression and Finite-Context Models in DNA Similarity

Diogo M. Marto NºMec: 108298
DETI, University of Aveiro
Aveiro, Portugal
diogo.marto@ua.pt

Diogo Pinto NºMec: 110341
DETI, University of Aveiro
Aveiro, Portugal
pintodiogo@ua.pt

Miguel Vieira, NºMec: 85095
DETI, University of Aveiro
Aveiro, Portugal
miguelmvieira@ua.pt

Ilker Atik, NºMec: 123947
DETI, University of Aveiro
Aveiro, Portugal
ilker@ua.pt

I. Introduction

This work aims to implement Normalized Relative Compression ($\mathcal{NRC}$) by making use of the Finite Context Models (FCM) that we've implemented in the previous work and doing experiments revealing information content of a set of sequences relative to a reference sequence i.e. Meta.

$\mathcal{NRC}$ is a measure used to quantify the information content of one object (a sequence) relative to another. $\mathcal{NRC}$ evaluates the fraction of a target object's information that cannot be derived from another reference object. In this work the reference object is Meta and the target objects are the sequences from the database.

We are aiming to find out which sequences from the database have lower $\mathcal{NRC}$ values, in other words, which sequences can be derived from the Meta at the best/requiring a minimum number of bits. A supporting aim is to find optimal parameters that will led lower $\mathcal{NRC}$ values. The sequences that can be derived from the reference better are those that contain similarity with the Meta in terms of contexts. The $\mathcal{NRC}$ measure can also be used to find which sequences are more compressible using the Meta as a model.

$\mathcal{NRC}$ computation is based on calculating the total bits required to represent a target sequence relative to a reference. For this calculation we are using information obtained from Meta and then for each sequence accumulating bits per symbol to find the total bits required to represent the sequence.

II. Implementation

Since this work is a continuation of the previous work on FCM, the computation of bits per symbol remains the same, with one difference; for this work, we are first learning the entire sequence of Meta, saving it as a frequency table, and then performing the $\mathcal{NRC}$ computation on target sequence rather than learning a sequence online and calculating bits per symbol as the sequence is processed as we did previously. The implementation steps can be summarized as below:

Initial phase

- Read database and meta, preprocess/filter the contents
- Extract sequences from the database, separate content and sequence names
- Create frequency table from Meta based on given context length (k parameter)

NRC calculation

- Process retrieved sequences from the database
- Calculate $\mathcal{NRC}$ for each using a finite context model to estimate the numbers of bits to compress a sequence using the mode trained on Meta sequence according to (1).

$$\mathcal{NRC}(x\|y) = \frac{C(x\|y)}{|x|\log_2(\mathcal{A})} \tag{1}$$

where $C(x\|y)$ represents the number of bits needed to lossless compress x given exclusively a model trained with y, $|x|$ the size of sequence x, and $\log_2(\mathcal{A})$ the alphabet of sequence x.

- The computation is parallelized using threads

Retrieving the best sequences

- Sort $\mathcal{NRC}$ results in ascending order and get n best sequences
- We are interested in lower $\mathcal{NRC}$ values since a lower value indicates the sequence is more similar to Meta and it requires fewer bits to represent

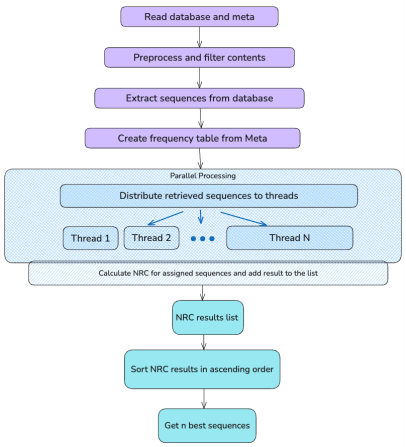

Fig. 1: Implementation strategy.

A. Data Pre-processing

- Removing irrelevant characters (non-ATCG) and newlines from both Meta and database sequences to ensure only valid symbols are processed
- Splitting a database into individual sequences using @ as delimiter and then separating sequence name and content using the first newline

B. Frequency Table Construction

- Frequency table is built by sequentially processing Meta content using a sliding window of length context width (k parameter) and recording context and succeeding symbols while shifting one symbol on each step
- The data structure used for storing frequencies of contexts and their subsequent symbols is identical to that of FCM i.e. based on *HashMap<String, CharCounts>*
- The frequency table is utilized in the calculation of $\mathcal{NRC}$

C. NRC Calculation

- For each sequence from the database, we're calculating the total bits required to represent it using probabilities derived from Meta's frequency table
- We are applying Laplace smoothing using parameter α.
- A lower value indicates a sequence has a similar content compared to the reference in Meta
- After computing all NRCs we sorted them to obtain the best sequences The best sequences are the ones with the lowest $\mathcal{NRC}$ values

D. Multithreading

- After creating the frequency table from Meta, the computation of $\mathcal{NRC}$ for each sequence can be done in parallel, this is possible because the frequency table is immutable i.e. not modified after creating it using Meta content, therefore we can share the same table among many processes easily
- We implemented a multithreaded approach to run the $\mathcal{NRC}$ calculations in parallel/in a non-blocking fashion, as each calculation is isolated they can be computed and then added to the list of results
- Each sequence result is added to the list as they're completed, and after completion of calculating NRCs of all sequences, the best sequences can be obtained by sorting the list of results

III. Results

To evaluate and compare the performance and accuracy of our Normalized Relative Compression ($\mathcal{NRC}$) implementations, we conducted a comprehensive set of tests across all six programming languages. For each implementation, the program was executed using a wide range of parameters to assess its behavior under various settings.

A. Parameters for tests

- α values:

A total of 12 different alpha values were tested: [1, 0.25, 0.06, 0.015, 0.004, 0.001, 0.00025, 0.00006, 0.00001, 0.0000025, 0.0000006, 0.0000001].

- Context lengths (k-values):

A total of 19 context lengths were tested: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

This resulted in a total of 228 parameter combinations per implementation.

For each (k, α) pair, the following data was collected:

- The top 20 best sequences from the reference database (with the lowest $\mathcal{NRC}$ scores).
- The individual $\mathcal{NRC}$ score and name of each of these top 20 sequences.
- The mean $\mathcal{NRC}$ score of the top 20 sequences, representing how well, on average, the best-matching sequences were compressed.
- The standard deviation of the $\mathcal{NRC}$ scores across the top 20 sequences, indicating the variability or consistency of similarity.
- The execution time required to process the combination of (k, α).
- The memory usage during the computation.

Each test was executed using a consistent pipeline, and the results were systematically saved in language-specific JSON files. This organization facilitated cross-language comparisons and enabled automated analysis of the outcomes. With all the data collected, several analyses were conducted to better understand the performance and behavior of $\mathcal{NRC}$:

- Heatmaps were generated to visualize how the mean and standard deviation of the $\mathcal{NRC}$ scores vary across combinations of k and α, helping to identify optimal settings.
- The evolution of $\mathcal{NRC}$ scores across different context sizes was plotted to evaluate how compression effectiveness changes with increasing memory of the model.
- A progressive bit analysis, tracking how many bits accumulated over the sequence length, to assess encoding efficiency and compression stability.
- A mapping analysis was carried out to understand where the best-matching fragments from the database aligned in the metagenomic reference (Meta). This helped identify regions of high similarity in the sample.
- A dimension reduction of the distance matrix between sequences where the distance is $\mathcal{NRC}$ as a way to visualize how the lower scores of $\mathcal{NRC}$ to Meta sample are correlated.

For all analyses, we used the outputs from the Java implementation. Although the execution time and memory consumption vary across programming languages, the $\mathcal{NRC}$ scores and relative results are consistent between all implementations.

B. Programming languages

To evaluate the performance of the Normalized Relative Compression ($\mathcal{NRC}$) algorithm across different programming languages, we developed implementations in Python, Java, C++, C, Rust, and Zig. The goal was to explore how the choice of language and underlying features such as memory

management, concurrency, and native performance affect both execution time and memory usage.

All implementations follow the same algorithmic logic, but the Zig implementation is the only one that does not use multithreading, unlike the others which parallelize the computation where applicable.

All benchmarks were run on an MSI GF63 Thin laptop equipped with an Intel i7 10th Gen CPU and an NVIDIA GeForce RTX 2060 Max-Q, with 16GB of RAM. All tests were conducted using Linux via WSL (Windows Subsystem for Linux). The test involved computing $\mathcal{NRC}$ values for all sequences in the reference database across multiple α values and context widths ($k = 6, 10, 15, 20$).

To account for variability in the measurements, we performed multiple runs for each combination of alpha and context width (k). For each configuration, we collected the execution time and memory usage and computed both the mean and the standard deviation across all runs.

These values were then visualized using bar plots with error bars shown in Fig. 2 and Fig. 3. The error bars represent the standard deviation, providing a clear picture of the consistency and variability of each implementation's performance. Bars with larger error ranges indicate less consistent performance across runs, while shorter bars reflect more stable behavior.

a) *Processing Results:*

As shown in the figure below, Python was consistently the slowest implementation by a large margin. On the other end, C, Rust, and Zig offered the best performance, with C being the fastest overall in most configurations — except when $k = 6$, where Rust slightly outperformed the others.

Java and C++ performed similarly, being significantly faster than Python but slower than C, Rust, and Zig. This can likely be attributed to the overhead of the JVM and the use of higher-level abstractions, despite both languages supporting multi-threading and being compiled.

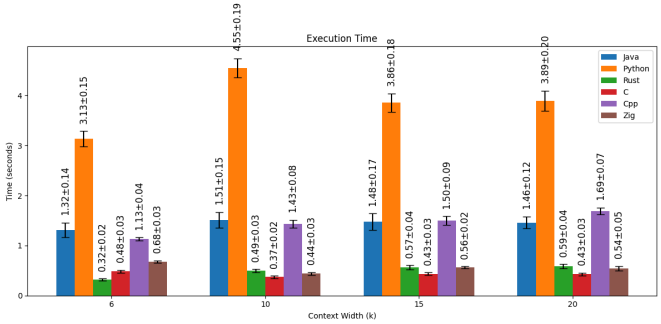

Fig. 2: Execution Time by Implementation for k 6, 10, 15 and 20.

b) *Memory Usage:*

Regarding memory usage, Java consumed the most memory by a significant margin. This is expected given the garbage-collected nature of the JVM and internal object representations. The remaining five implementations — Python, C++, C, Rust, and Zig — had comparable memory usage, with C again being the most memory-efficient across all tested configurations.

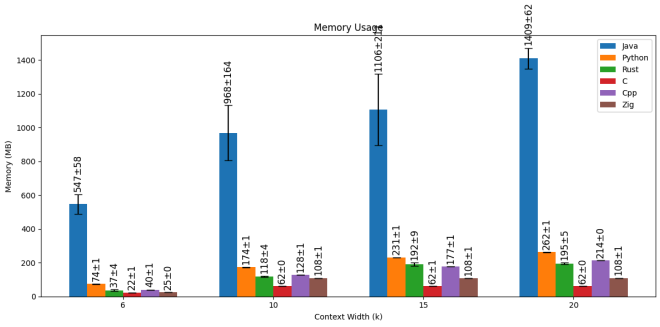

Fig. 3: Memory Usage by Implementation.

C. Context length (k) and Smoothing parameter (alpha)

To identify the optimal combination of context length k and smoothing parameter α, we performed an analysis based on heatmaps generated from the results.

Due to the computational cost and for better visualization, we restricted the analysis to a subset of values:

Selected α values:

- [1, 0.25, 0.06, 0.015, 0.004, 0.001, 0.00025].

Selected context sizes (k):

- [6, 7, 8, 9, 10, 11, 12, 13, 14, 15].

The first heatmap displays the mean $\mathcal{NRC}$ score of the top 20 sequences for each (k, alpha) combination. Lower mean values indicate higher similarity between the reference sequence (Meta) and the best-matching database sequences, meaning better compression and higher inferred similarity.

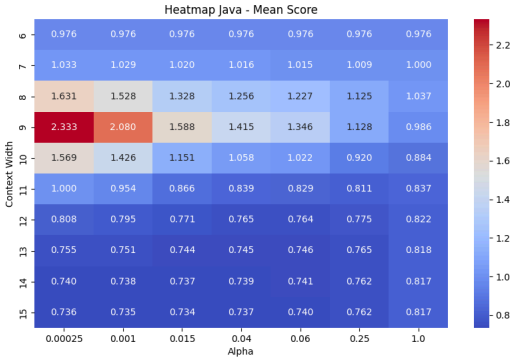

Fig. 4: Best Combination of k and α values - Mean Score.

From this heatmap, we observed a clear pattern:

- The best combination was found to be $k = 15$ and $\alpha = 0.015$, yielding the lowest mean score overall.
- In general, increasing the context length k tends to decrease the mean $\mathcal{NRC}$ score, which suggests better model precision and more effective compression as the memory of the model grows
- However, a noteworthy exception occurs at k = 6, which surprisingly also produces relatively good results.
  - From k = 6 to k = 9, the average score worsens slightly. An interesting aspect of the standard deviation heatmap is the notable peak observed at $k = 9$. This value corresponds to the highest variability

in scores across different sequences. However, this also implies that the model built from the meta sequence is not yet well aligned with the structure of the meta itself. In other words, the compression model, at this specific context length, is not effective at compressing its own source — the meta sequence — which results in inconsistent scores. If the model cannot represent the meta efficiently, it is unlikely to generalize well to other sequences. This explains why we observe such a high dispersion of results at this point: the model is still in a transitional state between underfitting (too generic) and overfitting (too specific), and therefore fails to consistently represent the patterns within the meta.

- From k = 10 onwards, the mean score begins to improve consistently again, ultimately achieving the best performance at k = 15.

This behavior suggests that while deeper context generally improves model performance, very small context sizes (e.g., k = 6) can also capture useful patterns, possibly due to higher generalization.

That said, this improvement with larger values of k comes with a trade-off: as the context increases, the model becomes more specific in its predictions. While this leads to more precise matching for clearly distinguishable sequences, it also makes generalization much harder when multiple sequences are highly similar.

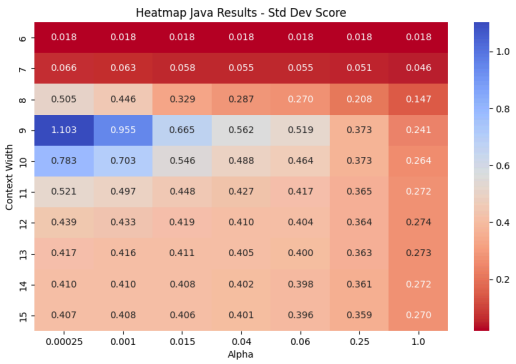

Fig. 5: Best Combination of k and α values - STD Dev Score.

The interpretation of the standard deviation heatmap is more nuanced. In this case, a higher standard deviation among the top 20 $\mathcal{NRC}$ scores is desirable, as it indicates a clearer distinction between sequences that are present (or highly similar) in the Meta sample and those that are not.

- The highest standard deviation values were observed around k = 9, as well as its neighboring values (k = 8 and k = 10), particularly for lower alpha values.
- These high-variance zones suggest that the model is more capable of distinguishing between matching and non-matching sequences, making it easier to identify true positives in the metagenomic analysis.
- In contrast, very low standard deviation values (as seen with k = 6) indicate that the scores among the top 20 sequences are all very similar—suggesting poor discriminative power and less confidence in identifying whether a sequence is truly present in the "Meta" sample.
- After reaching the peak at k = 9, the standard deviation begins to slightly decrease again around k = 11, then stabilizes across higher values of k, showing consistently moderate variation.

In summary, the analysis of the standard deviation heatmap highlights that although $k = 9$ shows the greatest distinction between sequences, it is not suitable due to its poor mean performance. Instead, the region from $k = 11$ and beyond with lower alpha values offers the most promising balance between compression quality and discriminative capacity.

Together, these heatmaps help identify not only the best-performing parameter combinations but also provide insight into the trade-off between precision and consistency when choosing the context length and smoothing parameter.

D. NRC score evolution across k

To better understand how the context length (k) affects the model's ability to distinguish whether a given sequence exists in the meta sequence, we analyzed the score variation of selected sequences across different values of k. Specifically, we considered all sequences that appeared at least once in the top 5 (based on their $\mathcal{NRC}$ score, where lower values indicate better matches) for $\alpha = 1$. This subset of sequences allows us to track score evolution and identify patterns of distinction as k increases.

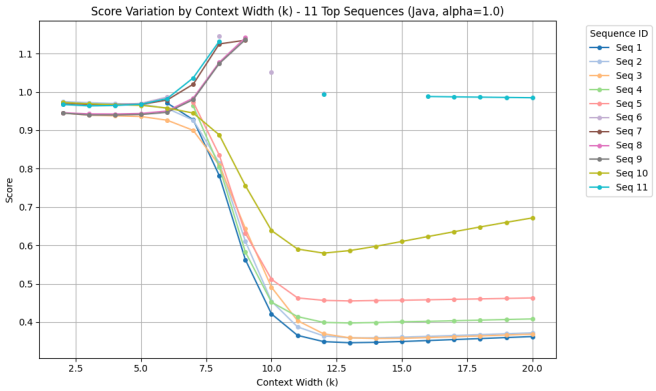

Fig. 6: $\mathcal{NRC}$ score Variation by k for the 11 top sequences with α 1.

Fig. 7: Legend for Score Variation by k for Fig. 6.

Fig. 6 illustrates how the score changes for these sequences as k increases from 2 to 20. Each colored line represents one of the selected sequences. We observe that, for values of k below 8, all sequence scores are clustered very closely together, indicating that it is difficult to distinguish between them. This supports the findings from the standard deviation heatmap, which showed that low values of k (especially $k = 6$) had a very low standard deviation—meaning little score variability and, thus, low distinguishability among sequences.

However, a distinct turning point is observed around $k = 9$, where the score curves begin to diverge significantly. This aligns with the earlier analysis where $k = 9$ was found to correspond to the highest standard deviation, meaning the score variability was at its peak. Such variability is crucial, as it enables the identification of which sequences are likely present in the meta. Beyond $k = 11$, the scores begin to stabilize, and the standard deviation decreases, although it remains higher than for very small k values. This suggests a trade-off between stability and discriminative power.

Another interesting observation from the plot is the appearance of gaps in some curves. These gaps indicate that the corresponding sequences no longer appear in the top 20 for those specific values of k, reflecting a performance drop for those combinations.

Furthermore, a more detailed inspection of the individual sequences reveals several interesting behaviors. For instance, sequence 1 (NC_001348.1 – Human herpesvirus 3) consistently maintains one of the lowest scores across almost all values of k beyond 8, indicating a strong likelihood of its presence in the meta sequence. Similarly, sequences 2, 3, 4, and 5 also demonstrate stable or improving scores as k increases, which reinforces their potential relevance. In particular, sequence 4 (Super ISS Si1240) shows a noteworthy pattern: for low values of k (specifically $k \leq 6$), it does not appear in the top 5. However, from $k = 7$ onwards, it consistently remains among the top sequences, with a remarkably low and stable score. This transition suggests that the model needs sufficient contextual depth to reliably capture the signal of this sequence within the meta, highlighting the importance of considering slightly higher values of k to avoid missing relevant matches.

On the other hand, some sequences exhibit a notable performance decline as context deepens. Sequence 7 (gi|1013949526 – Iris yellow spot virus), sequence 8 (gi|12084983 – Yaba-like disease virus), and sequence 9 (gi|363539767 – Megavirus chilensis) are two clear examples—while they appear in the top 5 for lower values of k, they drop out after $k = 8$ and 9, suggesting that they initially benefited from shallow context similarity, but were filtered out as deeper patterns were considered likely meaning that they aren't very similar to the Meta sample.

An important outlier is sequence 10 (gi|49169782 – Human Coronavirus NL63), which exhibits a rather unique behavior. After $k = 11$, while most other sequences stabilize or slightly improve (the cases of sequences 1, 2, 3, 4, and 5), the score of sequence 10 increases steadily. This divergence from the general trend highlights an anomalous score progression, raising questions about its alignment dynamics and suggesting that it becomes increasingly dissimilar to the meta as context grows. This peculiar case will be further explored in the subsequent sections Reverse Meta and Bits Estimation Progression.

Sequence 6 (gi|1008861384 – Tobacco leaf curl Yunnan virus satellite DNA beta clone 17) shows minimal relevance, appearing in the top 20 only twice throughout the whole k range—specifically at $k = 8$ and $k = 10$. This sporadic presence likely results from chance at these particular configurations rather than any consistent match with the meta sequence.

Lastly, sequence 11 (gi|66396483 – Ageratum yellow vein China virus-associated DNA beta) presents an oscillating behavior. It disappears from the top 20 after $k = 8$, suggesting a weak alignment as context deepens, but re-emerges after $k = 15$, with scores that remain relatively near to one. Despite this late return, its overall performance remains less convincing compared to the consistently strong sequences, reinforcing the likelihood that it is not a true match.

Taken together, this detailed breakdown confirms that increasing context width improves discriminative power up to a point, and helps identify sequences that genuinely align with the meta, while also filtering out false positives. It also highlights the importance of tracking score evolution across k values—not just absolute scores at isolated points—when interpreting results.

E. Bits Estimation Progression

To understand how compression evolves along the sequence, we analyzed the bits per character estimated along the sequence of each target sequence. This progression reveals how informative or surprising each symbol is for the model learned from the sample (Meta). By tracking the number of bits required to encode each character as the sequence is processed, we can visualize patterns of compressibility, identify regions of high similarity, and detect which parts of the DNA sequences, if any, were included in the sample Meta.

This analysis was performed using the frequency table built from the reference sequence and applying it to each sequence in the database. For each symbol in a target sequence (after the initial k-mer), we computed the corresponding symbol's probability and converted it into bits using the base 2 logarithm. The resulting bit values were collected and plotted sequentially, reflecting the bit footprint across the length of the sequence.

By generating bit progression plots for the top 20 sequences with k=15 and alpha=0.015, we can see that the top 6 (as expected from the $\mathcal{NRC}$ scores) show high similarity with Meta. These figures apply a moving average to smooth out mutation-induced spikes (with a window size adjusted to the length of each sequence).

The progressions can be generally divided into four types:

1- No similarity with Meta. As seen in Fig. 8, the bit progression remains mostly constant at a value around 2, indicating no learned information from the Meta.

2- High similarity with Meta. As shown in Fig. 9, Fig. 10, and Fig. 11, the bit progression remains consistently close to 0, suggesting the entire sequence is included in the Meta.

3- Zoned similarity. Fig. 12 demonstrates a Bit progression with regions near 0 and others with higher values, indicating partial similarity or inclusion.

4- Moderate similarity Fig. 13, the progression fluctuates around 1 — not entirely included in Meta, but sharing some degree of similarity.

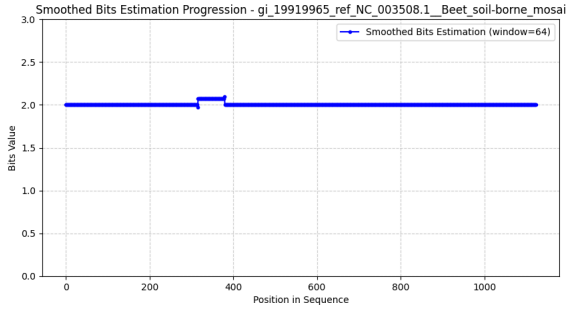

Fig. 8: Progression of bit estimation of meta across "beet soil" sample.

Fig. 8 demonstrates a mostly constant value of bit estimation, but a low similarity between Meta and the chosen sample, since the model does have information about that sequence.

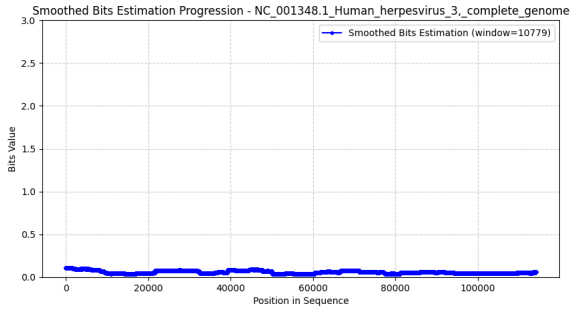

Fig. 9: Progression of bit estimation of meta across herpes virus sample.

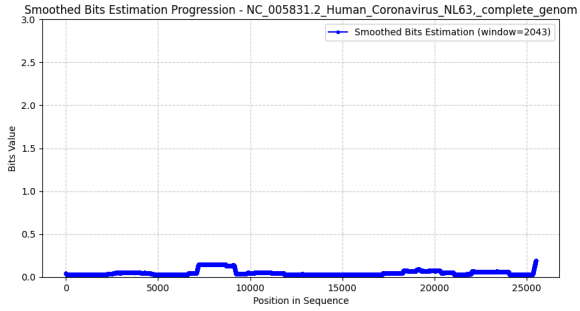

Fig. 10: Progression of bit estimation of meta across coronavirus sample.

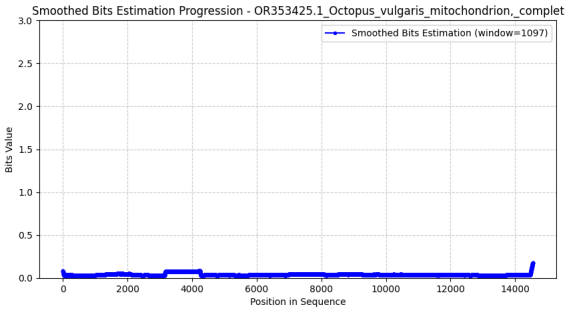

Fig. 11: Progression of bit estimation of meta across octopus mitochondrion sample.

In Fig. 9, Fig. 10, and Fig. 11, we can see that the samples with the top 3 scores also have progressions that indicate similarity across the entire sequence, meaning that those sequences have large parts that were included in the Meta sample, even if fragmented and unordered.

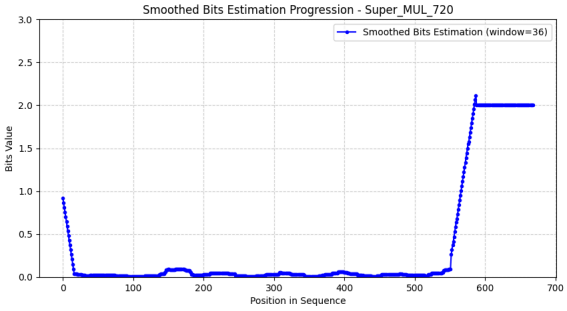

Fig. 12: Progression of bit estimation of meta across Super_MUL_720 sample.

In Fig. 12, the behavior is similar with samples with the top 3 scores of $\mathcal{NRC}$, staying very close to 0, but the end segments appear to have low similarity with meta, probably because although the sequence where included in meta the ending of the sequence may as been ignored or maybe if that parts of the sequence where included, they would be forced to be preceded or succeeded by samples from another source since we are talking about the start or the end of this one. The other "Super" sample (Super_ISS_Si1240) has a similar behavior.

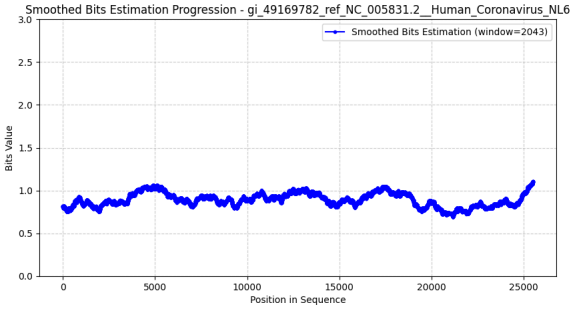

Fig. 13: Progression of bit estimation of meta across coronavirus 2nd sample.

In Fig. 13, we can observe that the behavior of the progression is not as similar to the previously analyzed cases. This indicates that the sample (gi|49169782 – Human Coronavirus NL63) exhibits some similarity with the Meta, but we cannot conclude that it is directly included in Meta. This observation is consistent with the fact that another sample of Human Coronavirus NL63 (NC_005831.2) was already identified as being included in Meta. We can see how both Coronavirus samples compare in Fig. 14 .

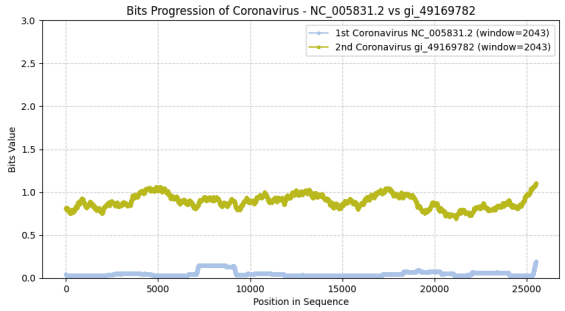

Fig. 14: Progression of bit estimation of meta across both Coronavirus samples.

It is expected for different strains or isolates of the same virus species to exhibit high genetic similarity. Therefore, this analyzed sample (gi|49169782 – Human coronavirus NL63) is not part of Meta itself but shares high similarity with another sample from the same viral species that is included in Meta.

In short:

1) These samples of DNA are included in meta, for the most part in their entirety:
- NC_001348.1 – Human herpesvirus 3
- NC_005831.2 – Human Coronavirus NL63
- OR353425.1 - Octopus vulgaris mitochondrion
- Super ISS Si1240
- Super MUL 720

2) The end of these sequences may have not been included in the Meta sample:
- Super ISS Si1240
- Super MUL 720

3) This annalizes allows for a distinction between samples that are included in meta from the ones that just manifest some degree of similarity, like in:
- gi|49169782 – Human coronavirus NL63

F. Reverse Meta and DB

If training the $\mathcal{NRC}$ model with the Meta sample getting the top 20 $\mathcal{NRC}$ scores and extracting the progression of bits along each selected sequence, allows us to identify which sequences are included in the meta (even if fragmented), and exactly which parts of the sequences are included. In theory, we can reverse the process and build a model with each sequence and analyze the Meta sample, and be able to see where that sequence or its portions are located in the Meta sample.

To that effect, we selected the top 6 sequences previously identified for k=15 and alpha=0.015, used each one to train a model (getting 6 individual models), and then applied it to the meta, extracting the $\mathcal{NRC}$ score, and the bits estimation progression, having the following $\mathcal{NRC}$ scores:

1- NC_001348.1 – Human herpesvirus 3 => meta
- 0.6710076813146131

2- NC_005831.2 – Human Coronavirus NL63 => meta
- 0.9325375014615953

3- OR353425.1 - Octopus vulgaris mitochondrion => meta
- 0.9617068534433778

4- Super ISS Si1240 => meta
- 0.9970488912380578

5- Super MUL 720 => meta
- 0.9985320172037934

6- gi|49169782 – Human coronavirus NL63 => meta
- 0.9610963851627308

As we can see these scores are worse, almost indistinguishable from random samples (that should give scores around 1), except for the model trained with herpesvirus that has a score more or less reasonable. This is because the models trained on the db samples capture only one genome while the Meta sample can contain multiple, meaning that the individual models are unfit to describe the Meta sample in its entirety. Due to this, we think that only looking at this $\mathcal{NRC}$ score doesn't wield much information so by looking at progressions we can extract more relevant information.

All these progressions have similar results, like in the Fig. 15, with a fallback value at 2, and sporadic and random positions of values that are either very close to 0 or very close to 6, but the most important observation is that there are very specific zones where the values are constantly at either close to 0 or 6.

The main difference between the progressions is the positions and the size of the zones.

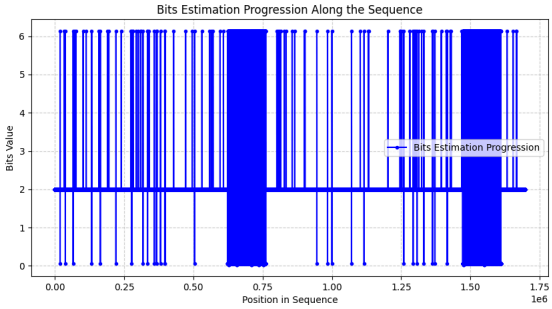

Fig. 15: Progression of bit estimation of Coronavirus across meta.

The reasons for these behaviors (for these settings of k=15 and alpha=0.015) are that the models when they don't have information about the current context give a value of 2, when they have information for the current context and the next symbol agrees with the model assumption it is given a value very close to 0, and in opposition when the next symbol to the context goes against the model assumption, it is given a value close to 6.

This indicates that:

1- the dense zones, indicate zones in the meta where this specific sample is located.

2- these zones don't have the samples sequentially in their entirety, they seem to be fragmented and possibly randomly unordered.

3- the sporadic and random spikes in values may be ignored for this analysis since they don't provide relevant information.

Knowing the above we can compile the different produced data in a unique dot graphic in Fig. 16, where is only plotted at each position in the meta sequence, the min value of the progressions (comparing each model trained with the different samples), and associating that point a color (following legend in Fig. 7), symbolizing the source of the value.

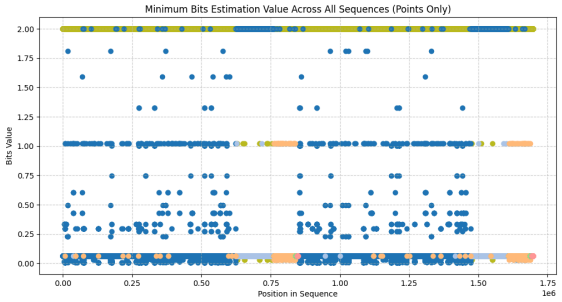

Fig. 16: Mapping of Meta sequence.

The resulting Fig. 16 as the resulting values mainly close to 0, with no significant gaps, meaning that at least one of the models during the entire sequence meta has some reasonable assumption of the symbol sequence, so there is no unknown region that was created without being considered for this annalizes.

There is a significant amount of values spread along the sequence with values close to 1 or 2, this is directly related to the frequency of mutation and/or fragmentation.

Considering only the values close to 0, the sequence is strongly zoned, since the best value comes mainly from the same source for a big part of the meta sequence until it flips the source and continues again with this new source for another big part of the sequence until again flips the source.

In Fig. 17, we determine graphically exactly these zones.

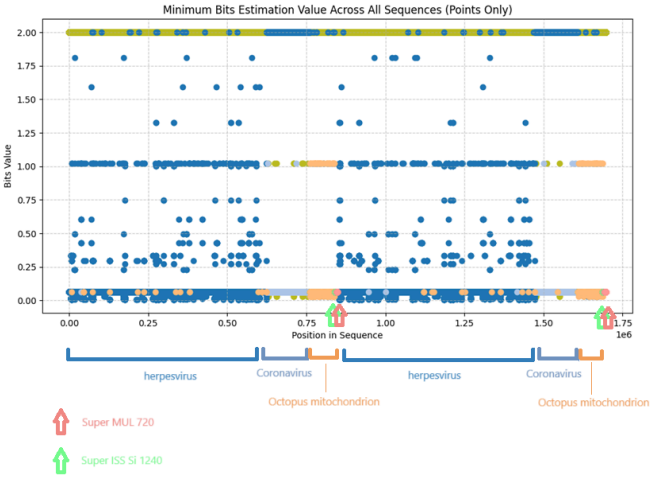

Fig. 17: Progression of bit estimation of meta across coronavirus 2nd sample.

As we can see, there seems to be a repetition of the zones after around the halfway point, so talking only about the first half.

The first zone is the biggest and occupies more than half of the sequence, it comes from the sample of herpesvirus (NC_001348.1 – Human herpesvirus 3), then we have a significant zone that occupies around half of the rest of the sequence with a source from coronavirus (NC_005831.2 – Human Coronavirus NL63), then we have a similar zone with similar size from Octopus mitochondrion (OR353425.1 - Octopus vulgaris mitochondrion) and at the end, we have 2 small zones with source of each "super" sample (Super MUL 720 and Super ISS Si1240).

The values from the 2nd sample of coronavirus (gi| 49169782 – Human coronavirus NL63) never outperform the outer sources reliably or consistently in either zone, the raw graphics of the progression show that they are very similar to the Fig. 15 (1st sample of coronavirus), with almost identical zones. So this just corroborates the belief that this 2nd sample does not relate directly with meta, it just has a high degree of similarity with the 1st sample of coronavirus (NC_005831.2 – Human Coronavirus NL63), and this one is the one that is included in meta.

G. *Relations of top samples*

To understand if the top selections were related in any way we conducted an experiment where we created a matrix D where each $D_{i,j}$ corresponds to the average of the $\mathcal{NRC}$ that model i gives to j and the $\mathcal{NRC}$ that model j gives to i (D is then symmetric). For our database of 239 samples that means that D as dimensions 239×239 so to visualize this information in a lower dimension, we performed t-SNE dimensionality reduction which should preserve the local structure of D.

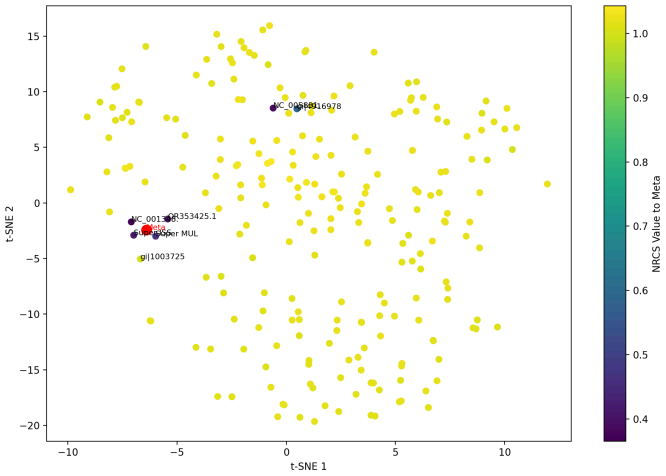

Fig. 18: Visualization of the first t-SNE components applied to the distance matrix with the Meta sample highlighted in red.

In Fig. 18 we can see that the top samples concerning the Meta sample model are close to each other in the plot this means that the $\mathcal{NRC}$ score between the top samples is close, although sequences "NC_005831.2 Human Coronavirus NL63, complete genome" and "gi|49169782|ref|NC_005831.2| Human Coronavirus NL63, complete genome" are not close to the other top samples. In general, we can see that some sample groups have low $\mathcal{NRC}$ scores between themselves.

IV. Conclusion

This project successfully implemented and analyzed the Normalized Relative Compression ($\mathcal{NRC}$) technique to identify similarities between sequences in a biological database and a metagenomic reference sample (Meta). By leveraging Finite Context Models (FCM), we were able to quantify how well a given sequence could be compressed using a model trained on the reference, providing a principled method for assessing sequence similarity.

Implementations were developed in six programming languages — Python, Java, C++, C, Rust, and Zig — offering a broad comparison in both computational performance and result consistency. Our results demonstrated that while $\mathcal{NRC}$ score outputs remained consistent across implementations, execution time and memory usage varied significantly.

A thorough experimental design was carried out by exploring a wide grid of parameters. These enabled a multi-faceted analysis to determine the best-performing configurations and gain insights into model behavior.

Overall, the $\mathcal{NRC}$ method combined with FCM proved to be an effective tool for metagenomic analysis, capable of revealing complex similarities in noisy and potentially fragmented biological datasets. The results not only confirm the viability of this approach but also open the door for further improvements and applications in bioinformatics and exobiology.

References