Master Project

Point cloud compression for DNA based storage

Author:
Romain Graux

Supervisors:
Prof. Dr. Touradj Ebrahimi
Davi Lazzarotto

Chapter 1

Introduction

Data is an important part of the society nowadays, it is responsible for the growth of the economy, the growth of the society, the growth of the world. The rise of what is known as Big Data will Facilitate things like newscasting (real-time forecasting of events), the development of inferential software that assesses project outcomes to date patterns, and the creation of advanced algorithms for correlations that enable a new understanding of the world. Overall, the rise of big data is hugely positive for society in all aspects. The other question presented negative bias, including the energy consumption to produce, process and store them. The Independent reported in 2016 that data centers will consume three times as much energy as they are currently using over the next decade. [1] Furthemore, the data production is growing exponentially while the storage hardware is created from finite resources.

It becomes naturally important to find more eco-friendly and capable ways to treat these data. In this field, a new paradigm has emerged: DNA-based storage. The genetic material DNA has garnered considerable interest as a medium for digital information storage because its density, durability and energy efficiency are superior to those of existing silicon-based storage media. Despite these advantages, DNA has not yet become a widespread information storage medium because the cost and the complexity of chemically synthesizing DNA are still prohibitively high. That is why researchers are still working on the development of a cheaper and more efficient DNA-based storage medium.

In this new era, it is important to explore new ways to store all types of information. It is therefore interesting to create new algorithms that are specifically made to encode any type of information directly into an ACGT code.

With the rising of autonomous vehicles based on Lidar sensors and the integration of Lidar cameras into smartphones, more and more 3 dimensional data is being produced which was not the case before. These data are rather heavy to store in their raw format. They are therefore a good example of data that should be stored in a new type of medium that is more efficient.

In this document we will explore how it would be possible to mix these two ideas to produce a single algorithm capable of storing these point clouds into a DNA-based medium.

Chapter 2

Problem definition

In this work, we will try to build an end-to-end point cloud compression model for quatarnary based entropy coding from publicly accessible existing implementations.

We will first discover the state of the art of learning based compression models for point cloud that leads to the best bitrates for various point clouds. This will be done by comparing the compression models of the state of the art that can be easily adapted to the needs of the DNA based coding. It will serve as a backbone that knows how to turn a point cloud into a smaller latent representation that contains only the information that is necessary to reconstruct the point cloud. From this backbone, we will adapt the binary entropy coding to a quaternary based entropy coding.

We will study the state of the art of DNA based storage and how we can store our sequences of **A**, **C**, **G** and **T** as efficiently as possible to minimise the cost of storing those datas while trying to build the most resistant sequences regarding the biological constraints and errors introduced.

To help us in building the strongest compression models for DNA based compression, we will study the state of the art of DNA based compression and how we can use the state of the art to build the most resistant compression models. We will also use a DNA storage simulator to study how well our compression model can produce the most resistant sequences and recover them to reconstruct the point clouds as faithful as the original one.

Chapter 3

State of the art

3.1 DNA storage

As we produce more data every year, it is becoming essential to store it as efficiently as possible. The current solution is to rely on electronic devices, organised in large data centers with an important energy consumption. For instance, the energy consumed by data centres in the European Union was of 76.8 TWh in 2018, which corresponds to 2.7% of the total electricity demand for that year. Predictions estimate both the absolute number and relative share to increase to respectively 98,52 TWh and 3.21% by 2030. As we can observe on Figure 1, it is not about to stop.

For both ecological and economical reasons, it is becoming increasingly interesting to find more efficient alternatives, such as DNA. Our genetic code packs billions of gigabytes into a single gram. A mere milligram of the molecule could encode the complete text of every book in the Library of Congress and have plenty of room to spare. [2]

Unfortunately, the present information retrieval latency and the high cost of the DNA sequencer among other instruments hinder a wide-spread application of DNA as storage. Currently, one cannot expect to replace a standard USB flash drive with a DNA-based variant without major changes to the experience. [2] Driven by the field's potential applications, promising developments are enabling cheaper, faster and smaller technology.

Furthermore, the current disadvantages with DNA-based storage are not relevant for all applications. For instance, DNA based storage is convenient for long term media preservation archives (so called cold media storage) which are infrequently accessed and thus do not need low information retrieval latency. [3]

3.1.1 DNA

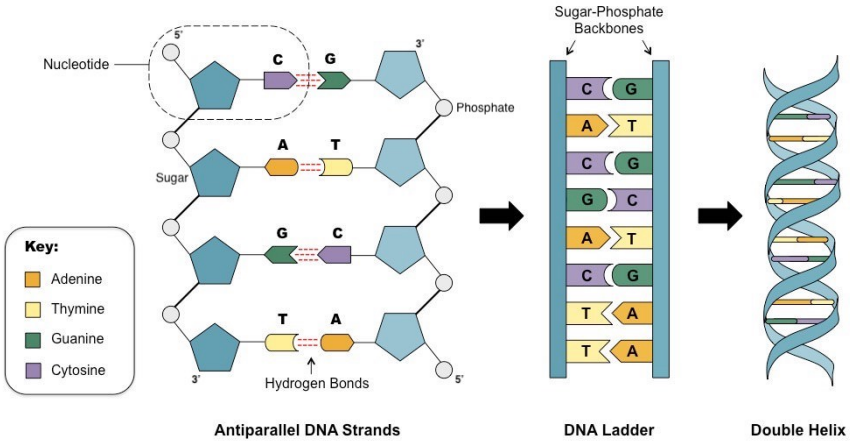


Figure 3.1: DNA structure

Desoxyribonucleic acid (DNA) is a chemical molecule composed of two polynucleotide chains that coil around each other to form a double helix. DNA is the primary information carrier in living organisms and as such, it is the building block of all life. It holds the essential

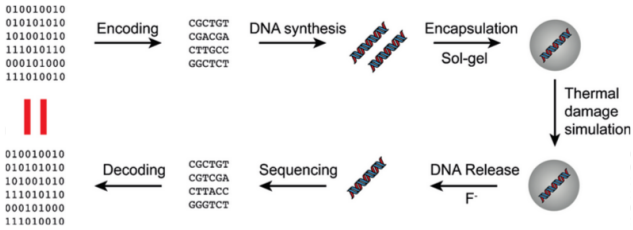


Figure 3.2: DNA storage pipeline

instructions for the development, maintenance and reproduction of all known organisms and many viruses.

The two polynucleotides chains are composed of monomeric units called nucleotides, as described on Figure 3.1. Each nucleotide is built from one of the four nitrogen bases (adenine [A], cytosine [C], guanine [G] and thymine [T]), a sugar called deoxyribose and a phosphate group. The nucleotides are linked to one another in a chain by a covalent bond between the sugar of one nucleotide and the phosphate group of the next, resulting in a alternating sugar-phosphate backbone. The nucleotides of the two separate chains are joined together by an hydrogen bond between their nitrogen bases according to base pairing rules (A-T, G-C), creating the aformentioned double helix. The nitrogen bases are divided into two groups, the pyrimidine bases (C and T) and the purine bases (A and G).

Both strands of DNA store the same information, in complementary pairs. The two strands of DNA run in opposite directions and are thus anti-parallel. The direction is determined by the 5' to 3' or 3' to 5' direction.

3.1.2 Usage

DNA is a powerful tool for storing information for biological systems so it can also find its place in the synthetic DNA storage. DNA storage can be used for applications that require for example:

- **High information density**: In 2017, scientists at Columbia University and the New York Genome Center published a method which allows perfect retrieval of information from a density of 215 petabytes per gram of DNA [4];
- **Data longevity**: Information stored in DNA can last for decades or centuries compared to conventional media which are replaced every 3-5 years [5];
- **Low energy consumption**: DNA does not use energy to store information, energy is only used to prevent the degradation of the DNA;
- **Ease of replication**: Thanks to the Polymerase Chain Reaction, DNA can be replicated easily [6].

3.1.3 End-to-end DNA storage

The way to store binary information into DNA based medium storage and then retrieve it back is called end-to-end DNA storage. It has several steps as seen on Figure 3.2.

- **Encoding**: Transform any kind of information into a nucleotide stream (A, C, G or T);
- **Synthesis**: Synthesise the nucleotide stream into an actual DNA molecule;
- **Encapsulation**: Encapsulate the DNA molecule into a DNA medium for long term storage;
- **Thermal damage**: The DNA medium can be damaged by thermal damage, this can cause the DNA to break or be damaged during the storage process;
- **Release**: The DNA molecules can be released from the medium, it gives us the ability to retrieve the information back.

- **Sequencing**: The DNA molecules are sequenced on a computer, this allows us to digitally retrieve the information back as DNA sequences.
- **Decoding**: The DNA sequences are decoded back into the original information.

3.1.4 Constraints

Unfortunately, nucleic acids have biological constraints and cannot be assembled in any order like it is the case for binary digits. The DNA strands have to be created in a way that the double helix binds well together and is not immediately desctructed. We must therefore respect the biological constraints to build strong strands that can last for a certain period of time.

In this part, we are going to go through some of the constraints that we have to respect to build a DNA strand and be able to recover it during sequencing time. Unfortunately, the list of constraints is not exhaustive and in reality, each arrangement of nucleic acids has an impact on the strength of a strand. As a consequence, we can only simulate the longevity of a strand thanks to past discoveries documented in literature but not strictly respect all the biological constraints.

All constraints could be reduced to limitations regarding GC content, long strands of a single nucleotide (so-called homopolymers), several repeated subsequences in a strand and motifs with biological relevance. In the next sections, we are going to divide the constraints into each step of the storage process, the explanation for each constraint comes directly from [7].

Synthesis

To synthesise synthetic DNA, *in silico* designed constructs have to be split into smaller fragments [usually 200–3000 base pairs (bp)] [8]. The fragments are then split into several oligonucleotides (so-called oligos) [usually 40-200 bp] that are individually synthesised. Once synthesised, all oligos are merged back together with either ligase or polymerase-based methods. One of the constraints on the GC content comes from the fact that depending on the synthesis method and the overall GC content of a fragment, the GC content of each oligo has to be within a specific range. In oligos with high GC content, neighboring guanines tend to form an increased amount of hydrogen bonds, leading to inter/intra-strand folding [9]. To assemble oligos into larger fragments, the melting temperature (and thus the GC content) should only deviate slightly between oligos. To adhere to this restriction, the designed DNA fragments should be homogenous with respect to GC content. Homopolymers further increase the synthesis complexity, leading to fragments that are only synthesizable by using modified oligos and more sophisticated assembly methods, resulting in increased synthesis costs.

PCR: Polymerase Chain Reaction

The amplification of DNA using polymerase chain reaction (PCR) is indispensable for biological science. From DNA synthesis over cloning to DNA sequencing, PCR is used in a wide range of applications. One important factor of a successful PCR is the base composition of the amplification substrate. High melting temperatures due to high GC content of the DNA fragments hinder the separation of strands during the denaturation phase of the PCR. This reduces the yield of the PCR process, since the polymerase cannot efficiently synthesize the growing strand in the presence of previously existing hydrogen bonds. Stretches of repetitive DNA or high GC content can lead to the formation of secondary structures, hindering the elongation of the growing strand. Repetitive regions, as well as homopolymers, can also lead to polymerase slippage, a process in which polymerase briefly loses the connection to the template strand and reconnects at a different position with similar nucleotides content [10] (a nice explanation can be seen here).

Storage

Further restrictions on the composition of the DNA construct are due to the cloning process: the GC content should be close to the GC percentage of the host genome and motifs used for the cloning process have to be avoided during the design of the DNA construct.

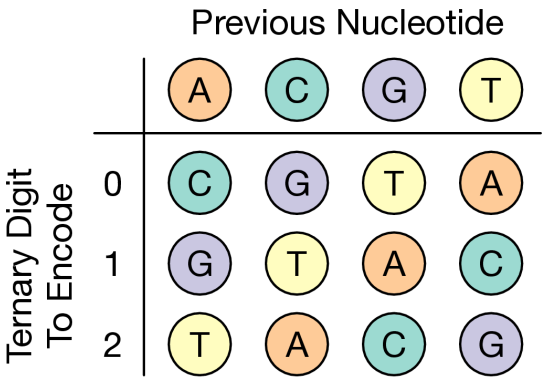


Figure 3.3: Encoding of the trit produced by a 3-ary Huffman tree for each symbol

Sequencing

The base composition of a DNA fragment is also an important factor for the successful retrieval of genetic information using DNA sequencing technologies. Illumina sequencing, Oxford Nanopore and PacBio sequencing technologies are biased toward DNA with an intermediate GC content, leading to reduced coverage of regions with strongly deviating GC content [11]. Illumina and Nanopore sequencers also show an increased error rate in the presence of homopolymers [11]. Depending on the sequencing method used, the resulting data show increased substitution rates for specific DNA patterns: for PacBio data, common substitution patterns are CG $\rightarrow$ CA and CG $\rightarrow$ TG, Nanopore data contain an increased amount of TAG $\rightarrow$ TGG and TAC $\rightarrow$ TGC substitutions [12] and a common substitution pattern in Illumina data is GGG $\rightarrow$ GGT [13].

3.1.5 MESA: Mosla Error Simulator

In order to simulate without going through an expensive and long process that is DNA synthesis and sequencing, we are going to use a simulator that takes into account a large majority of biological constraints. This simulator has been introduced [7] in March 2020 and is a web application for the assessment of DNA fragments in terms of guanine-cytosine (GC) content, homopolymer occurrences and length, repeating subsequences and undesirable sequence motifs. Furthermore, MESA contains a mutation simulator, using either the error probabilities of the assessment calculation, literature-based or user-defined error rates and error spectra. MESA is fully customizable using an easy-to-use web interface. All functionalities of MESA is also contained in a REST API, enabling the incorporation of MESA evaluations into custom workflows for high-throughput evaluation and error simulation of DNA.

As we have seen in the previous section, DNA has a lot of constraints during the synthesis, storage, PCR and sequencing steps. With this simulator it is now possible to have a feedback on the strength of a particular DNA strand and thus help us move towards the best DNA coding to ensure good information retrieval in the end.

3.2 DNA coding

3.2.1 Goldman coding

In 2013, Goldman and al. [14] proposed an algorithm to encode digital data into binary while respecting the main constraint of DNA sequencing (homopolymers). The technique indicates to encode data first in a 3-ary Huffman tree so that each byte can be encoded into the digits $0, 1, 2$ giving smaller representations to more frequent data and longer ones to less frequent data. Once each byte has its own code representation in the tree, the digits can be encoded by mapping to one of the nucleotides while ensuring that the previous encoded nucleotides are not used to encode the current digit. A representation is shown on the Figure 3.3 for the mapping of the digits regarding the last encoded nucleotide.

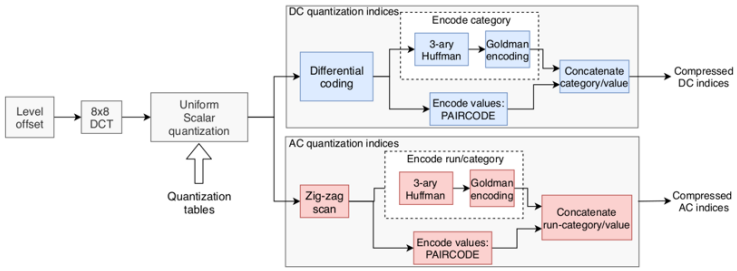


Figure 3.4: Schema of the JPEG DNA codec

3.2.2 PAIRCODE

PAIRCODE [15] is a technique described [15] in 2019 to encode any symbol (not only binary data) to fixed length codewords. The quaternary constrained codeword $\mathcal{C}*$ is constructed using mainly pair-elements from the following dictionary:

$$\mathcal{C}_1 = \{AT, AC, AG, TA, TC, TG, CA, CT, GA, GT\}$$

It is possible to build any codeword of even length by concatening words from these 10 elements. To build a codeword of odd length, it is possible to concatenate an even codeword with one of the words from the dictionary:

$$\mathcal{C}_2 = \{A, C, G, T\}$$

This construction guarantees that the final encoded DNA stream does not contain homopylmers and a GC-percentage higher than the AT-percentage.

3.2.3 JPEG DNA codec

The purpose of the JPEG DNA codec is to encode a raw image to DNA code using the JPEG codec and encoding the coefficients to actual DNA nucleotides thanks to the PAIRCODE and Goldman coding.

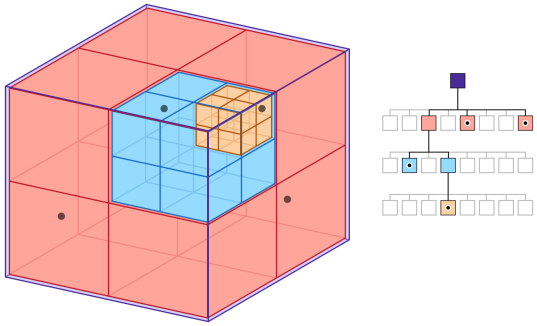
Before turning an image into a DNA stream, the first step is to extract the DCT coefficients to be encoded.

With a raw image as input of shape either H×W (gray image) or H×W×3 (RGB image), the first step is to shift the values of the image by -128 to center it around 0 since the values of the pixels are encoded on *uint8* values and are thus betweem 0 and 255. Then the discrete cosine transform is applied on these centered values to extract the (8×8) block DCT coefficients. Next, since the codec is dealing with visual information, a uniform scalar quantization is used to turn continuous values to discrete coefficients thanks to predefined quantization tables that give more importance to low frequency cosines (because the human eye is more sensitive to low frequency signals than high frequency ones). The quantization introduced a lot of zero for high frequencies on the lower right part of the block. The last step is then to flatten the block DCT quantized coefficients in a zigzag maneer starting from the upper left corner, this way, there will be a large part of zeros at the end of the 64-long flatten array.

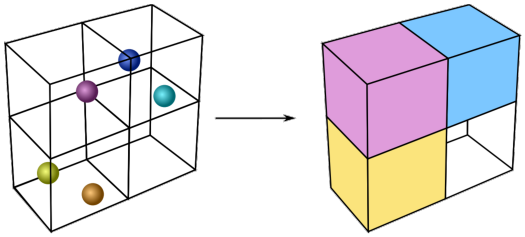
The flattened array is now composed of the DC value (constant intensity) as first element and then 63 AC values corresponding to each particular cosine frequency. As we will see, the DC and AC values are encoded into DNA in two different ways.

In the classical JPEG workflow, each category is mapped to a binary representation thanks to the Huffman tree that will encode the most frequent AC/DC categories to the shortest words. For the DNA version, they used a 3-ary Huffman tree to transform the category into trits and then encode these categories with the Goldman algorithm, the category are determined by the range in which the value falls. The AC and DC coefficients are then encoded using a codebook with fixed length l codewords, the codebooks have been generated using PAIRCODE.

For example, here is how the absolute AC values are categorized:



(a) Octree representation of a point cloud



(b) Voxelized representation of a point cloud

- $\{0\} \rightarrow 0$
- $[1, 5] \rightarrow 1$
- $[6, 17] \rightarrow 2$
- $[8, 82] \rightarrow 3$
- $[83, 375] \rightarrow 4$
- $[376, 1263] \rightarrow 5$
- $[1264, 5262] \rightarrow 6$
- $[5263, 17579] \rightarrow 7$

And for each category, we have a fixed number of codewords with fixed length in a specific codebook. Here is the example of the codebooks used in the JPEG DNA codec algorithm per category (as shown on Figure 3):

- $0 \rightarrow$ 10 words of length 2
- $1 \rightarrow$ 24 words of length 3
- $2 \rightarrow$ 130 words of length 4
- $3 \rightarrow$ 586 words of length 5
- $4 \rightarrow$ 1776 words of length 6
- $5 \rightarrow$ 7998 words of length 7
- $6 \rightarrow$ 24634 words of length 8
- $7 \rightarrow$ 110660 words of length 9

The only difference between the AC and the DC coding is that the DC coding is always encoded with the category and then the codeword in a particular codebook. Yet, due to the nature of the quantization and the zigzag, there is a lot of following zeros in the AC coefficients, so each time there is a 0 a counter is incremented at encoding time until it reaches 16, in which case, a particular codeword is appended to the stream and the counter is reinitialized to 0. If a non-zero value happens before the counter reaches 16, the number of zeros (between 1 and 15) is coupled with the category of the current AC coefficient, the counter is reinitialized to 0 and the values are encoded the same way as described for the DC values.

3.3 Point cloud compression

Numerous methods have been proposed to compress point clouds in the literature. They are usually based on different structures than a standard list of coordinates. Octrees, for example, have been widely used for this purpose [16]. The octree representation consists of dividing recursively the three dimensional space as nodes of a tree as shown on Figure 3.5a, where at each iteration the space is divided in eight subspaces with the same dimensions.

Compression algorithms using learning based auto-encoders architectures have also demonstrated good performance. While some take as input point coordinates [17], others take as input voxelized versions of the point clouds. Voxelized point clouds consist of an occupancy grid of regular spaced points, where several points are merged together in a single voxel as shown on Figure 3.5b. A voxel can be seen as a three dimensional pixel. These three dimensional grids can be then used as input for a *3D convolutional layer*. Unfortunately with this

structure, we loose all edge information.

The current state of the art for point cloud compression employed in this project is a model called "Latent Space Slicing For Enhanced Entropy Modeling in Learning-Based Point Cloud Geometry Compression" and has been developed by Nicolas Frank, Davi Lazzarotto and Touradj Ebrahimi at the MMSPG laboratory (EPFL).

3.3.1 Model architecture

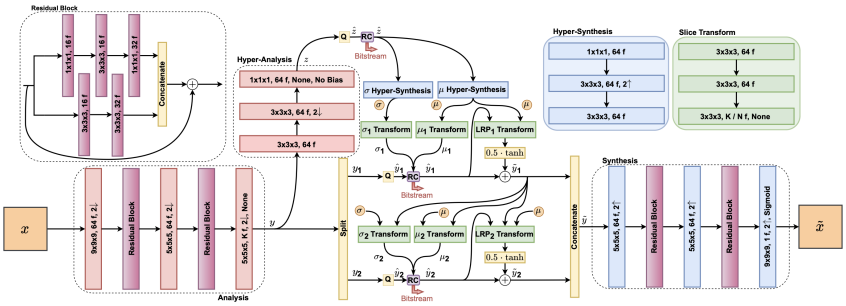


Figure 3.6: The "Latent Space Slicing For Enhanced Entropy Modeling in Learning-Based Point Cloud Geometry Compression" architecture with 2 slices.

This model is shown on Figure 3.6 and consists in a 3D auto-encoder architecture with latent entropy coding and is inspired by "Channel-wise auto-regressive entropy models for learned image compression" developed by D. Minnen and S. Singh [18]. The input of the model is an occupancy cubic grid with $k \times k \times k$ voxels, each of which is represented by 1 when occupied and 0 otherwise.

The first block (Analysis transform) of the model is composed of three *3D convolutional layers* and two *convolutional residual blocks* arranged staggered. This produces a latent representation y of shape $l \times l \times l \times d$ with d being the latent dimension. The nature of the convolutional layer allows inputs of different shapes.

This latent representation is then fed into an Hyper-Analysis transform block yielding z. This hyper-prior is passed to the bitstream as side information after quantization, and is used to model the entropy of the quantized latent features $\hat{y}$ after going through the hyper-synthesis.

While in other solutions for learning-based point cloud compression the hyperprior would be the only variable used to estimate the scale and mean of $\hat{y}$, here previously decoded channels for entropy modeling are employed.

The latent representation y is sliced along the channel dimension into N non-overlapping and equally sized tensors y_i with $i \in \{1, \ldots, N\}$.

Once the latent representation y is sliced, they compute the entropy parameters (μ_i, σ_i) for each slice y_i from the global entropy parameters (μ, σ) and the previously decoded latent representation slices $\tilde{y}_j \, \forall j \in \{0, \ldots, i-1\}$.

The final latent representation reconstruction $\tilde{y}_i$ is produced from $\hat{y}_i$ after going through a latent residual prediction (LRP) transformation that predicts the quantization error $y_i - \hat{y}_i$. This takes into account the error from the global entropy parameters (μ, σ), the current $\hat{y}_i$ and the previously decoded latent representation slices $\tilde{y}_j \, \forall j \in \{0, \ldots, i\}$. A *tanh* non-linearity scaled by a factor 0.5 is applied to the output of the transform to keep the output of the LRP within the range of quantization error. The predicted residuals are then added to $\hat{y}_i$, generating $\tilde{y}_i$. These slices are then concatenated along the channel dimension before going through the synthesis transform. This last learned block finally generates the output block $\tilde{x}$ containing a probability estimation for the occupancy of each voxel in the grid.

The model is trained from end-to-end between the input occupancy grid and output probability occupancy grid as a rate-distortion minimization problem represented by a loss function expressed as $\mathcal{L} = R + \lambda D$. The trade-off parameter λ is used to balance the importance of compression rate against reconstruction quality. The distortion is computed from a focal loss (FL) which can be expressed as:

$$\mathrm{FL}(x, \tilde{x}) = -x\alpha(1-\tilde{x})^{\gamma}\log(\tilde{x}) - (1-x)(1-\alpha)\tilde{x}^{\gamma}\log(1-\tilde{x})$$

with α and γ being configurable hyperparameters. This focal loss is a so-called binary

weighted focal crossentropy loss.

The last step to recover the actual binary occupancy grid for each voxel from the output probability occupancy grid $\tilde{x}$ is to apply a threshold t and round every value above t to 1 and 0 otherwise. While a naive approach would be to set this threshold t to 0.5, it was decided to find the best threshold $t \in [0, 1]$ at compression time such that it minimizes the point-to-point MSE [19] between the original and reconstructed rounded block.

This technique was introduced by [20] and has proven to enhance reconstruction quality without significantly impacting the bitrate as it is only required to add a threshold $t \in [0, 100]$ in the final bitstream.

Chapter 4

Implementation

In this section, we want to build a complete pipeline from a raw point cloud to a DNA strand that will be later synthesized in a medium and kept for a long period (up to a dozen or even hundreds of years). After this long period of time, we would like to recover our point cloud as faithful as possible to the original.

From the presented state-of-the-art, we thus have all the building blocks to construct this pipeline and be able to encode and decode point clouds. However, this pipeline will not be constructed to be the most efficient and optimized DNA code point cloud compression algorithm. Instead, it will be a baseline to compare against future models.

4.1 Point cloud latent representation

The first step is to turn the point cloud into a smaller latent representation. For this purpose, we can use the point cloud compression model previously described in section 3.3 that has already learned the principal features of point clouds.

The latent representation y contains a compressed representation with less information but still all the necessary information to reconstruct the original point cloud. From a block of shape $k \times k \times k$, once passed through the Analysis Transform, the latent representation y has a shape $\lceil \frac{k}{8} \rceil \times \lceil \frac{k}{8} \rceil \times \lceil \frac{k}{8} \rceil \times d$ with d being the arbitrary latent depth. Therefore if the latent depth d is well chosen, the latent representation contains all information to retrieve the original point cloud while being smaller.

In our particular implementation, we chose a latent depth of 160 which means that we indeed have a smaller number of elements since $8 \times 8 \times 8 \geq 160$.

We also have to make a choice on which model to use for the transfer learning of the weights since several models are available for different bitrate/distortion trade-offs. As we will see in the next sections, the nucleotide rate mostly depends on the input shape without impacting the distortion no matter which one is chosen. Thus, to get the best reconstruction possible, we will choose the highest λ value (1750) for the model. The λ value is weighting the focal loss in the global loss used to train the model as explained in Section 3.3.

Now that we have a latent representation, we have to go to the next step, which is the JPEG DNA codec described in section 3.2.3. This codec has some requirements that have to be met in order to achieve the best quality and nucleotide rate.

4.2 Latent representation with JPEG DNA codec

In this part, we will use the JPEG DNA codec but not the full one, we will drop all the part related to JPEG and thus encode directly the coefficients that are normally produced by the last DCT transform. So the input of the codec will be quantized discrete coefficients. To meet these requirements, we will have to change few things from the latent representation y.

4.2.1 Dimensionality

The output of the Analysis block is a $l \times l \times l \times d$ latent representation y but since the codec is built for image purpose, it only accepts an image as input which is either of shape $H \times W$ (gray image) or $H \times W \times 3$ (RGB image). Consequently, we have to tweak our latent

representation y so that it satisfies this requirement. Since we have several possibilities for that, I tried two different approaches:

- The first approach is to merge the two inner dimension together, in that case we have a $l \times l^2 \times d$ shape. We can then encode several $l \times l^2$ images alogn the latent features dimension. This approach allows to process each feature "independently";
- The second approach is more simple and consists of merging the three first dimensions together, hence we end up with a $l^3 \times d$ image. We can thus encode directly the full image with the codec.

Each approach has its own advantages and disadvantages.

The first one aims at treating each feature separately so that we can encode in regard of each feature's distribution and as such, hope for less quantized values and consequently a smaller number of oligos. Although, when encoding the final flat nucleotide stream, we have to add the length of each latent oligo length which is avoidable.

In the second alternative, it is easier to compute since it is a single gray image like, so we have in the end directly a flat nucleotide stream for a block.

Finally, as the first one does not offer better performance while enlarging the final stream length (and thus impact the final nucleotide rate), we will simply merge the first three dimensions (dimensions of the feature block) as proposed in the second approach to produce a 2D tensor compatible with the codec.

4.2.2 Quantization

The other important aspect of the codec is the value type. The Analysis block produces continuous values $y \in [-\inf, \inf]$ due to the nature of convolutional layer and the linear activation function of the last layer but the codec is built to encode discrete integer values as it is the case for the DCT coefficients. This means that we have to convert the latent representation y to an integer value type while ensuring the y range is mapped to a range $[0, span]$ as a bijective maneer. Here the $span$ value is a fixed value chosen at compression time as a quality parameter. It is used as a nucleotide rate/distortion trade-off. The greater the $span$ value, the better the quality and the greater the nucleotide rate. In contrast, the lower the $span$ value, the worse the quality and the lower the nucleotide rate.

To implement such a quantization transform, we can naively do a linear mapping between $[\min y, \max y]$ to $[0, span]$. With this transform, we create $span + 1$ ranges of equal length $\frac{\max y - \min y}{span}$ in which the y values will fall. It assumes that y is uniformly distributed because it gives the same range length in which to fall for each y value.

Finally, to get the quantized integer value, we can simply round the final mapped values and it will gives us at most $span + 1$ integer values to encode with the JPEG DNA codec.

The mapping can simply be described as:

$$z = \text{round}\left(span \frac{y - \min y}{\max y - \min y}\right)$$

.

4.2.3 DCT

In the original JPEG codec, each 8×8 block of the images are first centered around 0 (simply shift by -128) and then transformed into the 8×8 DCT values before being quantized with a default perceptual quantization table. For our purposes, we will use a custom way to do the DCT that is better adapted to our needs considering that we already have quantized coefficients.

Because they are built for perceptual fidelity, we will not be dividing by quantization tables and thus will not have the long tail of 0's after the zigzag transform that normally appears for high frequencies DCT coefficients. All coefficients will most likely not be 0, so we will have to encode them all.

All these transforms can be seen as a block represented on Figure 4.1 that first reshape the 4 dimensional tensor into a 2 dimensional one and then quantize it with the approach seen above.

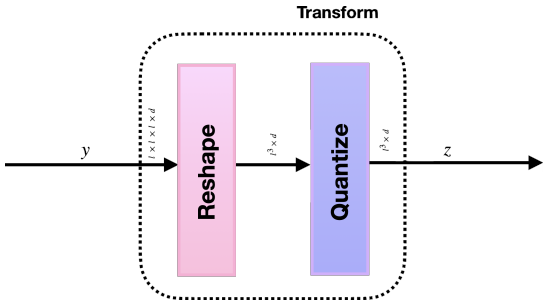


Figure 4.1: Reshape + quantization layers

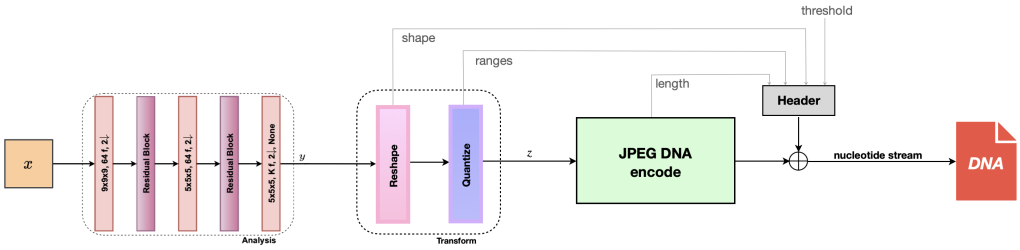


Figure 4.2: The full end-to-end mode that turns voxelized point clouds into a DNA stream

4.3 Nucleotide stream

We now have a full pipeline that can be used to encode and decode point clouds. The last step is to produce the actual nucleotide stream from the output of the codec and all intermediate information that are needed in order to fully decode the stream.

The intermediate information needed to produce the nucleotide stream is:

- The threshold used to turn the reconstructed block $\tilde{x}$ (the probability occupancy grid described in 3.3.1) as an *uint8* value;
- The oligo length used by the JPEG DNA codec (by default set to 200);
- The quantization range used for the latent representation y as described in 4.2.2;
- The shape of the latent representation y in order to reshape the decoded array of the codec since we merge all inner dimensions together to encode with the codec;

To turn a byte array into a nucleotide stream, we naively assume that each two consecutive bits can be considered as a nucleotide. We can use the mapping $00 \rightarrow$ A, $01 \rightarrow$ C, $10 \rightarrow$ G, $11 \rightarrow$ T. With this technique we can produce a 4 nucleotide stream for each byte of the byte array.

Here is how many nucleotides we can produce to represent our intermediate information:

- The threshold is a single byte since it is a *uint8* value $\in [0, 100]$ so we can produce 4 nucleotides;
- The oligo length is also encoded in a single byte, it thus produces 4 nucleotides as well;
- The quantization range are represented as two *float32* values, each value is encoded on 4 bytes so we can produce a total of 32 nucleotides for the quantization range;
- The shape of the latent representation has the form $l \times l \times l \times d$ and each dimension is encoded on a single byte so we can produce a total of 16 nucleotides for the shape;

The final stream is obtained by concatenating all the additional information and the nucleotide stream. In the end we have a total of $4 + 4 + 32 + 16 + n$ nucleotides where n is the number of nucleotides produced by the codec.

4.4 End-to-end model

With all theses changes, we now have a fully working end-to-end model capable of compressing a voxelized point cloud into a DNA stream respecting some constraints (homopolymers and kmer) thanks to PAIRCODE and the Goldman encoding. The model is shown on Figure 4.2.

4.5 Reconstruction

Starting from the nucleotide stream, we can fully reconstruct the point cloud by applying all steps in reverse order. First, we extract all the additional information and the codec stream from the nucleotide stream. Then, we can decode the codec stream to reconstruct the latent representation $\tilde{y}$ and reshape it to the shape that was encoded in the stream. Then, we can decode the latent representation $\tilde{y}$ using the dequantization transform with the quantization range that was also encoded in the stream in order to obtained the reconstructed block $\tilde{x}$ Finally, we can round the probability occupancy grid $\tilde{x}$ thanks to the threshold that was encoded in the stream and we can obtain the reconstructed binary occupancy grid.

Chapter 5

Performance

5.1 Visual performance

Now that we have a fully working model, it is time to evaluate its performance. On the Figure 5.1, we can see the impact of the span value on the distortion of the *mitch* point cloud with a voxelized depth of 8, which corrresponds to a voxelized point cloud of size $256 \times 256 \times 256$. The whole point cloud has been partitioned into $64 \times 64 \times 64$ smaller point clouds, which have been fed to the analysis transform.

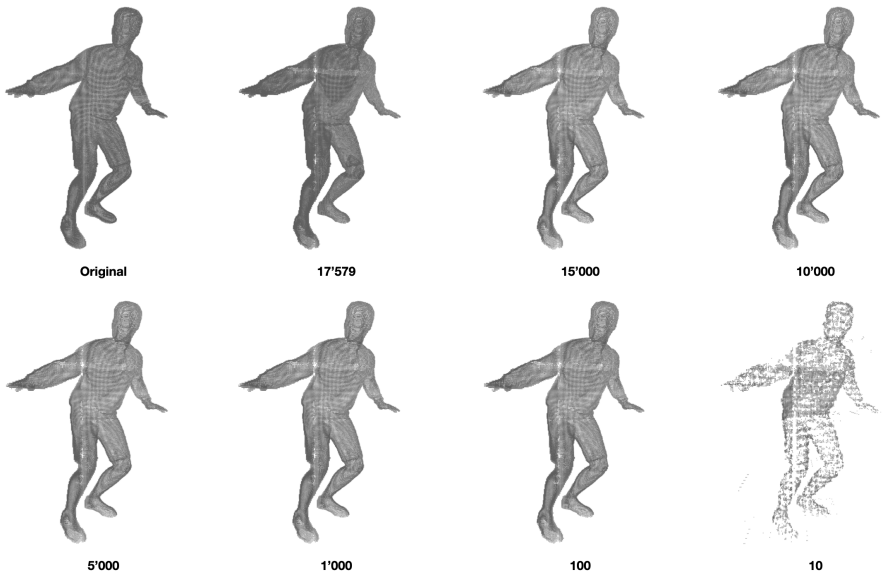


Figure 5.1: Mitch point cloud distortion regarding the span value with a voxelized depth of 8

We see some artifacts on the edges of the point cloud blocks, but the distortion is still low. These artifacts are probably due to the linear quantization since it is the only lossy part of the algorithm. The distribution of the latent representation plays a role in these artifacts but we will have a closer look with the rate-distortion analysis.

5.2 Rate-distortion analysis

To evaluate how well the model is performing, we can use a rate-distortion plot to compare how well the model can reconstruct the point cloud at a particular nucleotide rate. With this kind of plot, we can find a trade-off to maximize the reconstruction fidelity while minimizing the number of nucleotides required to encode it. In this case, the distortion is measured with a D1 PSNR metric which is the PSNR of the point to point MSE between the reconstructed point cloud and the original point cloud (regarding the closest neighbour to each point).

As we can see in the Figure 5.2, the rate-distortion can be easily controlled by the span value.

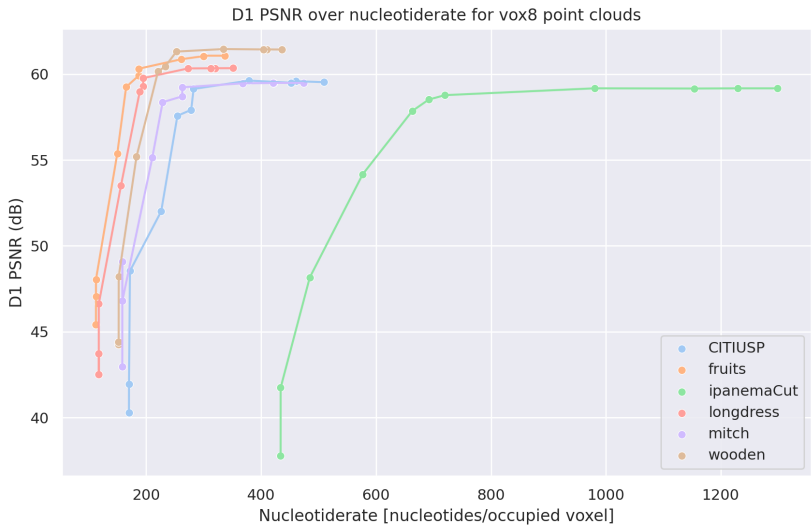


Figure 5.2: Rate-distortion analysis regarding the nucleotiderate (controlled by the span value)

These specific nucleotide rates have been computed for the following span values:

$$\text{span values} \in \{5, 7, 10, 20, 40, 50, 100, 1000, 5000, 10000, 17579\}$$

We see that the two first span values end up with the same nucleotide rate, while the distortion is much lower for the span value of 7. We also see that the PSNR is rising quite quickly for the first values without impacting the nucleotide rate much. Both can be explained by the codewords lengths that are small for the first codebooks (2 and 3 codewords lengths) and especially for the span value of 7, it has 2 coefficients (6 and 7) that are encoded on length 3 codewords while the rest is encoded on length 2 codewords (because falling in the first category codebook). So if the frequency of the 2 coefficients is not too high, it does not change the final stream length. As we see in the Figure 4b, the distribution of the latent representation is not concentrated close to the minimum or the maximum value so there will not be too many extreme values encoded in the final stream.

Weirdly, the ipanemaCut 5e shows a much higher nucleotide rate than all the others point clouds, it could be explained by the number of occupied voxels that are quite low compared to the overall number of voxels. But the CITIUSP 5d has a similar structure and has a better nucleotide rate over distortion ratio.

Finally, we can observe that we obtain a maximum PSNR around 60 quite quickly and that it is no longer increasing with a higher nucleotide rate. It can be explained by the linear quantization that is the only lossy part of the algorithm. Since the distribution of the latent representation is non uniform, as seen on Figure 4b, it can explain why the quantization is sub-optimal to turn continuous values into discrete ones.

If we compare with the rate-distortion analysis for the standard model, we can see that the PSNR is higher even with the smallest λ value. These bitrates have been computed for the following λ values:

$$\lambda \in \{40, 200, 400, 700, 1000, 1750\}$$

We can observe that the bitrate and the distortion depend more on the point cloud that is being encoded than on the DNA version. It is due to the fact that in this case, the latent representation y is entropy encoded per slice, using hyper-analysis entropy parameters and previous slices parameters, which helps a lot quantizing the actual learned distributions per slice. It is thus easily adapted to each point cloud which is not the case of the DNA model since only the whole quantized latent representation is encoded by the JPEG DNA codec. Furthermore, the LRP of the standard point cloud compressor helps predicting the error introduced by the quantization and this part is clearly not present in our DNA version, meaning that the quantization is always lost.

If we had wanted to do a naive mapping of the compressed bits to nucleotides, we would have had mapped a single nucleotide every 2 bits. So we would have ended up with a nucleotide rate of half the bitrate we have on this plot for the same distortion. Unfortunately, this naive mapping does not take into account the constraints of the GC-content and homopolymers

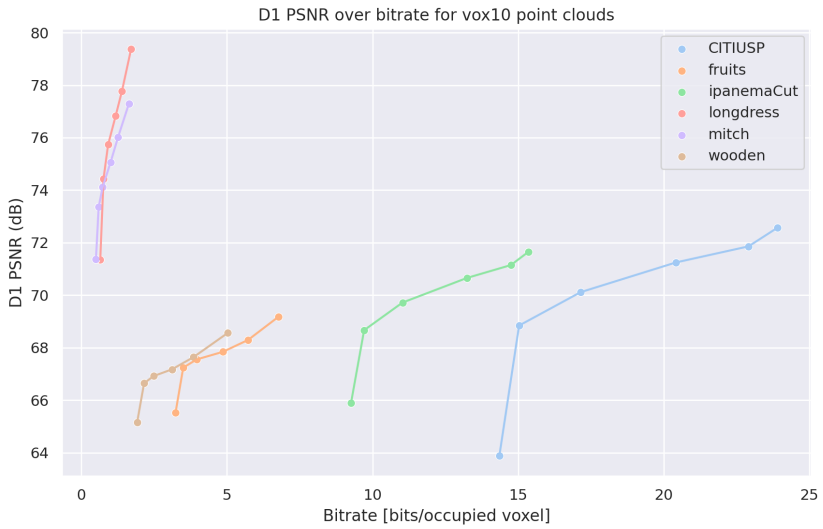


Figure 5.3: Rate-distortion analysis regarding the bitarte for the standard model described in section 3.3

that are taken into account in the PAIRCODE and Goldman algorithms used in the JPEG DNA codec.

5.3 DNA storage simulation

In order to see how robust the model is against DNA synthesis, storage and sequencing errors, we will simulate a compressed DNA stream with the MESA simulator. To do that we need to split our full stream into fixed length sequences, called *oligos*. In this simulation we will used the default parameters **??** with oligos of length 200, a storage period of 2 months, an ErrASE synthesis, 40 PCR cycles to amplify the oligos, a Taq polymerase and we will store the oligos in a E coli host. With these settings, we can now pass a full *fasta* file to the MESA server that will simulate the full process of synthesis, storage and sequencing and give us back the modified sequences. After merging back together all modified sequences into a single stream, we can pass stream to our decoder that will reconstruct the point cloud from it.

On the Figure 5.4, you can observe a toy example that has been simulated with the MESA simulator and the parameters specified above. This point cloud is a sphere of size $64 \times 64 \times 64$.

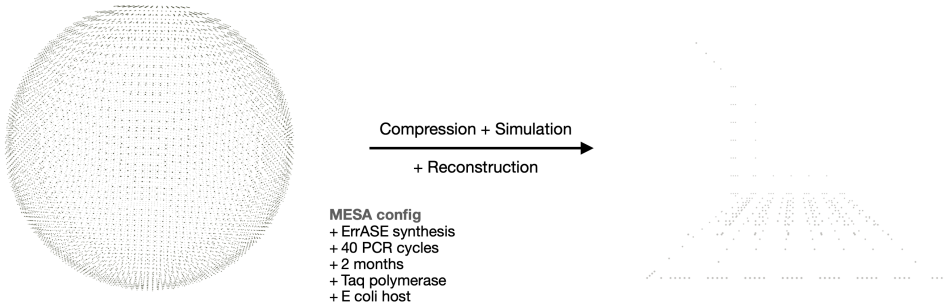


Figure 5.4: Simulated DNA stream for a 64 voxels width sphere

As you can observe, the reconstructed point cloud is not the same as the original one. This is due to the JPEG DNA codec behavior that cannot reconstruct a block if the codeword can not be found in the codebook belonging to the decoded category, in which case it outputs a 8×8 block of zeros.

If we observe on Figure 5.5 the latent representation distribution directly after the analysis block (not quantized) and the distribution of the reconstructed latent representation $\tilde{y}$ after simulating the DNA stream through the MESA simulator with the parameters mentionned above. We can clearly observe that the initial y is concentrated around 0 while the reconstructed $\tilde{y}$ is concentrated around -31 which corresponds to the minimum value of y. It

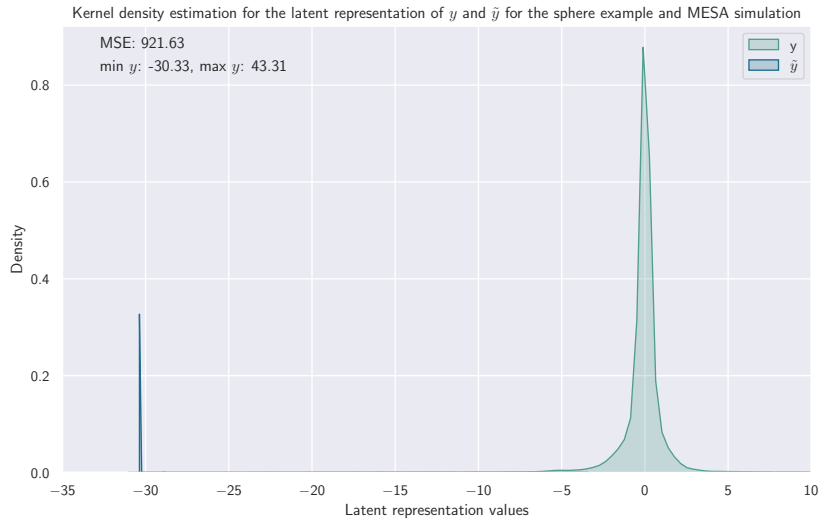


Figure 5.5: Latent representation y before simulation and reconstructed latent representation $\tilde{y}$ after simulation distributions

means that the JPEG DNA codec was not able to decode the stream and thus it produced a lot of 0 in the decoded array which is mapped to the minimum value of y once dequantized.

Chapter 6

Future directions

Some improvements can be made to obtain better results, performance and/or robustness.

6.1 Nucleotide rate and distortion

As seen in the performance section 5, the model suffers from limitations on the minimum distortion achievable. This is mostly due to the linear quantization which is the only lossy part of the algorithm, so several refinements are possible to get better results. The main problem is that the latent representation y is definitely non uniform, it is thus non optimal to divide the space in equally long gaps, instead we would have to divide the space in equally probable gaps. So in our case we would have to construct small gaps around 0 and bigger ones at the ends so that each one is equiprobable regarding the actual y distribution. For that purpose, it is possible to assume a normal distribution and instead of keeping track of the minimum and maximum value of y in order to quantize and dequantize, we would have to track the mean and the standard deviation of y so we can shift it exactly around 0 and control the spread of the distribution to control the nucleotide rate. One advantage is that we would have negative and positive values around 0, so we would give small codewords for the most frequent y value, even for the negative values. It would impact the final nucleotide rate in a good way.

The easier way of actually taking into account the distribution and quantize it accordingly is to quantize the latent representation with the entropy model used in the standard model that has learned all parameters from the distribution, it would give us better quantized symbols. But then, we would have to apply the algorithm slice per slice as it is the case with the classical model instead of directly encode the full latent representation. It would give us better rate over distortion results because it has been trained directly on that metric. And we would have the already trained models for different bitrates to control the rate-distortion ratio.

In this algorithm we used the JPEG DNA codec that is especially built to deal with visual information with all the JPEG parts (DCT, quantization tables, ...) in order to encode the DC and AC coefficients. Therefore it would be a great idea to build a general purpose coefficient coder instead of zigzaging the 8×8 blocks and encode the first value as DC coefficient and the 63 next as AC coefficients. In our case, it is unlikely to have a lot of following zeros, which is not a problem since the AC coder is traditionally encoding the value with its category and its corresponding codeword if no 0 are encountered. But the first DC value of each block are not necessarily close between adjacent blocks as it is the case for visual image, so instead of encoding the difference between adjacent blocks, we could encode the value like the 63 following AC coefficients. It would thus worth to encode the full flatten input array instead of diving it into 8×8 blocks.

6.2 Simulation robustness

As we saw in the simulation section 5.3, the model is not thoroughly robust against errors in the DNA stream, due to the JPEG DNA codec behavior. Without changing the whole structure, we can adapt the codec so that there is no categories anymore. This would allow us to reconstruct the final array even with errors introduced, it would directly affect the coefficient values but it would be almost always possible to decode the value. The problem

with this technique is that, to use the PAIRCODE algorithm, we would have to choose a codebook with constant length codewords to encode all the coefficients, which would directly affect the nucleotide rate. But it can be a tradeoff to choose at compression time, we encode all coefficients from the same codebook that will ensure reconstruction while increasing the nucleotide rate.

Another approach would be to train a learning based model with the same loss taking into account the distortion and the nucleotide rate but simulating the compressed DNA representation by simply mapping a nucleotide every 2 bits from the entropy model of the standard model. It would allow the model to learn how to compress the point clouds so that the compressed representation is as little error-prone as possible during the sequencing, storage and synthesis. To build such a model, we need a simulator that keeps track of the gradients but currently, the simulator is built on top of numpy which is not suitable for gradient learning models.

Chapter 7

Conclusion

After observing all the results, we can say that we have indeed built a working end-to-end point cloud compressor for DNA based storage capable of producing DNA stream sequences and recover the point clouds from them. We can also say that the end-to-end point cloud compressor is adaptive, the quality over rate can be tuned by the user as seen on the rate-distortion tradeoff curve.

The model is one of the first to be used in the field of point cloud compression for DNA storage and thus is a good starting point for future work but can not be considered as a complete and optimal solution since it uses several implementations that are not initially built for our purpose.

Improvements have to be done on the nucleotide rate and the achievable reconstruction distortion with the highest quality parameters. Improvements have also to be done on the robustness of the model and the ability to handle noise in the compressed domain. These improvements can be done either by entirely rebuilding the model architecture or by adapting the different model parts to our needs.

As previsouly said, this is one of the first model in the world that compress point clouds into DNA sequences. It therefore opens the door to this interesting and topical subject and can serve as a baseline to compare future models.

Bibliography

[1] A. Dellinger, "The environmental impact of data storage is more than you think — and it's only getting worse," jun 2019. https://www.mic.com/impact/the-environmental-impact-of-data-storage-is-more-than-you-think-its-only-getting-worse-18017662.

[2] J. Bohannon, "Dna: The ultimate hard drive," aug 2012. https://www.wired.com/2012/08/dna-data-storage.

[3] D. Panda, K. A. Molla, M. J. Baig, A. Swain, D. Behera, and M. Dash, "Dna as a digital information storage device: hope or hype?," *3 Biotech*, vol. 8, pp. 239–239, May 2018. 29744271[pmid].

[4] Y. Erlich and D. Zielinski, "DNA Fountain enables a robust and efficient storage architecture," Mar. 2017.

[5] K. Matange, J. M. Tuck, and A. J. Keung, "Dna stability: a central design consideration for dna data storage systems," *Nature communications*, vol. 12, p. 1358, March 2021.

[6] K. B. Mullis, "The unusual origin of the polymerase chain reaction," *Scientific American*, vol. 262, p. 56–65, Apr 1990.

[7] M. Schwarz, M. Welzel, T. Kabdullayeva, A. Becker, B. Freisleben, and D. Heider, "MESA: automated assessment of synthetic DNA fragments and simulation of DNA synthesis, storage, sequencing and PCR errors," *Bioinformatics*, vol. 36, pp. 3322–3326, 03 2020.

[8] S. Kosuri and G. Church, "Large-scale de novo dna synthesis: Technologies and applications," *Nature methods*, vol. 11, pp. 499–507, 04 2014.

[9] M. Jensen, M. Fukushima, and R. Davis, "Dmso and betaine greatly improve amplification of gc-rich constructs in de novo synthesis," *PloS one*, vol. 5, p. e11024, 06 2010.

[10] A. Fazekas, R. Steeves, and S. Newmaster, "Improving sequencing quality from pcr products containing long mononucleotide repeats," *BioTechniques*, vol. 48, pp. 277–85, 04 2010.

[11] D. Laehnemann, A. Borkhardt, and A. McHardy, "Denoising dna deep sequencing data–high-throughput sequencing errors and their correction," *Briefings in bioinformatics*, vol. 17, 05 2015.

[12] J. Weirather, M. d. Cesare, Y. Wang, P. Piazza, V. Sebastiano, X.-J. Wang, D. Buck, and K. Au, "Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis," *F1000Research*, vol. 6, p. 100, 06 2017.

[13] M. Schirmer, R. D'Amore, U. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: Resolving fine-scale variation in metagenomic sequencing data," *BMC Bioinformatics*, vol. 17, 03 2016.

[14] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, pp. 77–80, Feb 2013. 23354052[pmid].

[15] M. Dimopoulou, M. Antonini, P. Barbry, and R. Appuswamy, "A biologically constrained encoding solution for long-term storage of images onto synthetic dna," in *2019 27th European Signal Processing Conference (EUSIPCO)*, pp. 1–5, Sep. 2019.

[16] J. Kammerl, N. Blodow, R. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proceedings - IEEE International Conference on Robotics and Automation*, 05 2012.

[17] X. Wen, X. Wang, J. Hou, L. Ma, Y. Zhou, and J. Jiang, "Lossy geometry compression of 3d point cloud data via an adaptive octree-guided network," in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2020.

[18] D. Minnen and S. Singh, "Channel-wise autoregressive entropy models for learned image compression," 2020.

[19] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3460–3464, 2017.

[20] M. Quach, G. Valenzise, and F. Dufaux, "Improved deep point cloud geometry compression," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, 2020.

Appendix

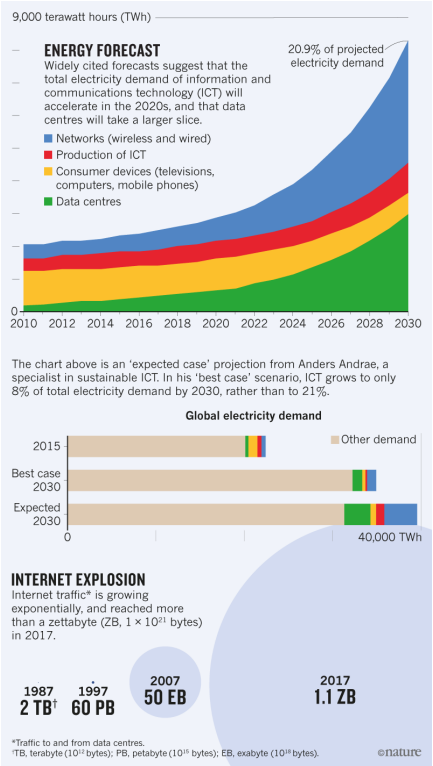


Figure 1: Energy consumption forecast

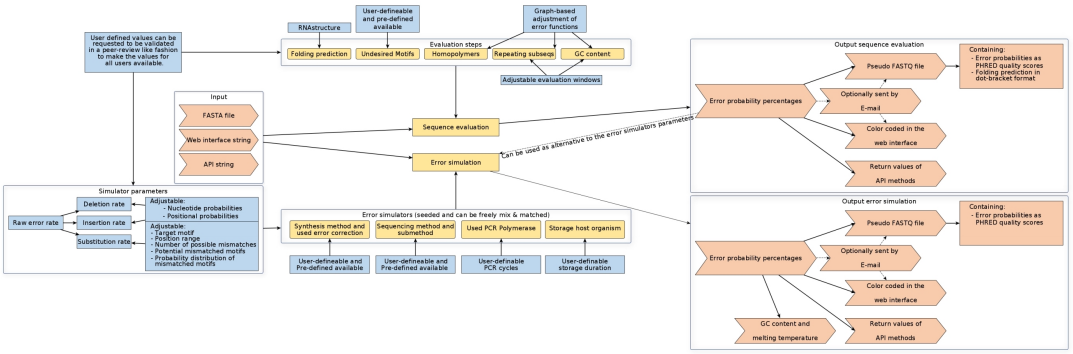


Figure 2: Mesa workflow

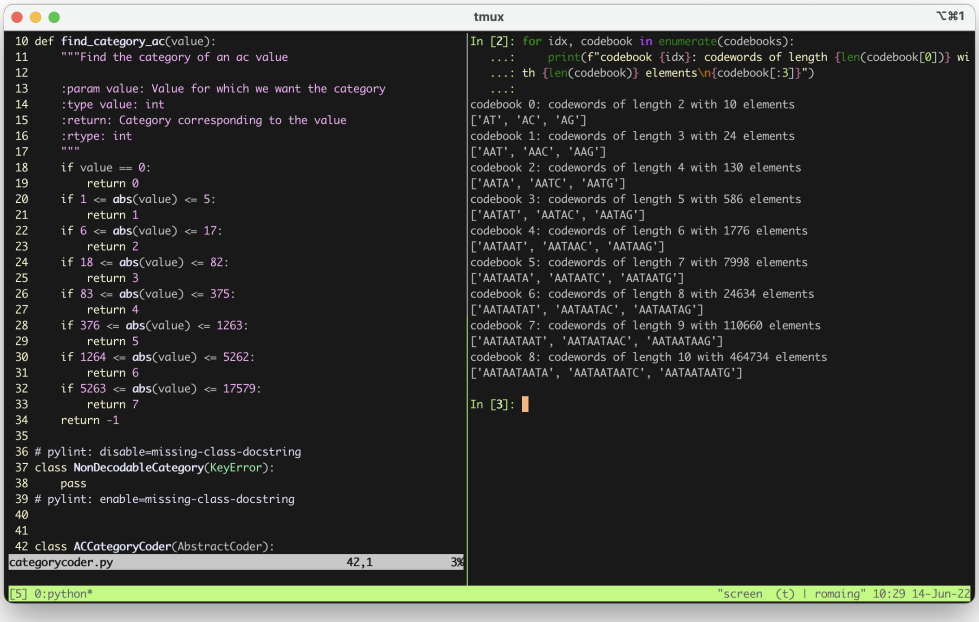
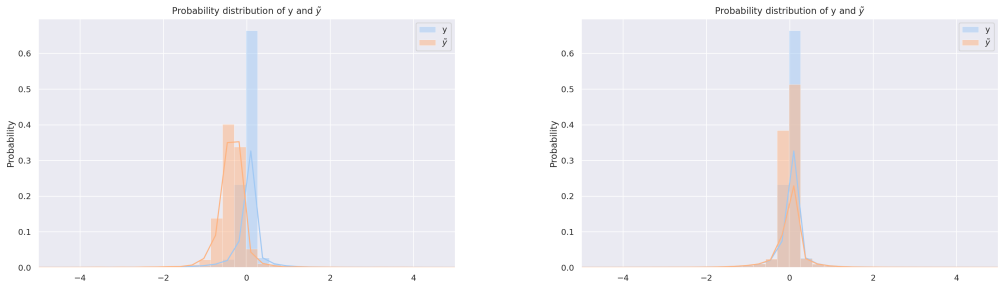


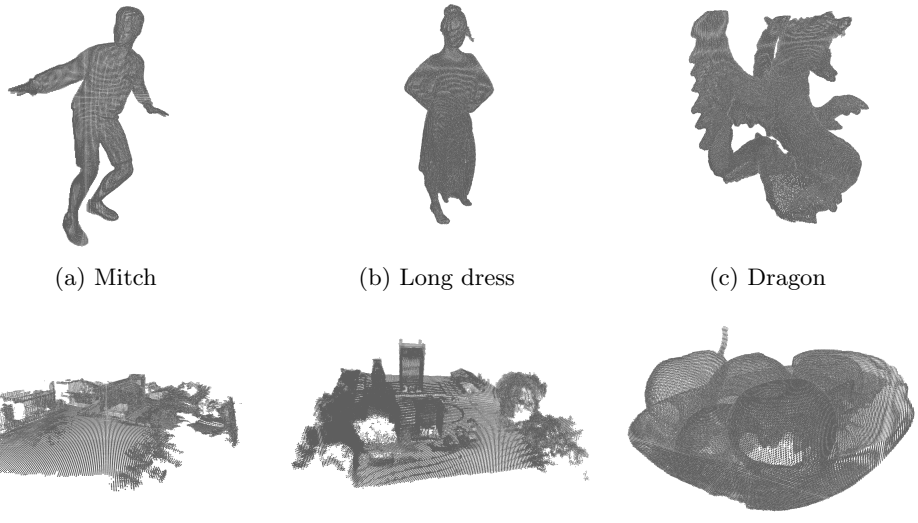
Figure 3: The ranges for the AC coefficients categories and the codebooks associated used by the JPEG DNA codec



(a) Span value of 50

(b) Span value of 1000

Figure 4: Latent representation y and reconstructed representation $\tilde{y}$ distribution for two different span values



(a) Mitch

(b) Long dress

(c) Dragon

(d) CITIUSP

(e) Ipanema Cut

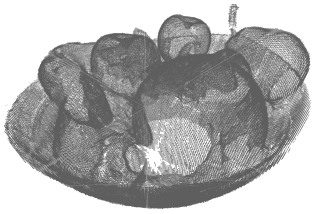
(f) Fruits

Figure 5: All original vox8 point clouds

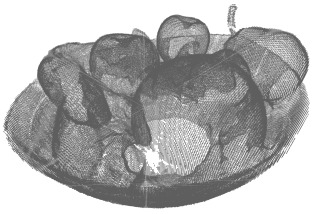
(a) Longdress 50

(b) Longdress 1000

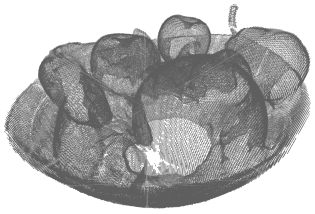
(c) Longdress 17579



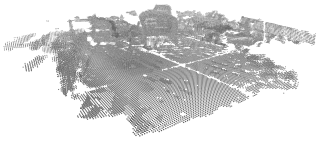
(d) Fruits 50



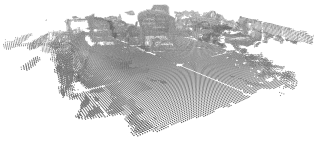
(e) Fruits 1000



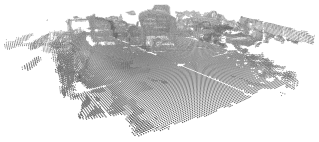
(f) Fruits 17579



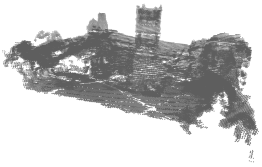
(g) CITIUSP 50



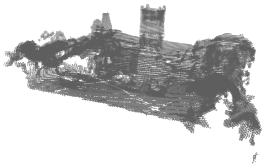
(h) CITIUSP 1000



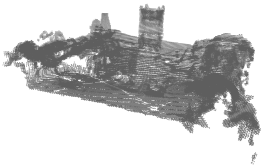
(i) CITIUSP 17579



(j) Ipanema Cut 50



(k) Ipanema Cut 1000



(l) Ipanema Cut 17579

(m) Dragon 50

(n) Dragon 1000

(o) Dragon 17579

Figure 6: All reconstructed vox8 point clouds with different span values