Master Project

# Point cloud compression for DNA based storage

*Author:*
Romain Graux

*Supervisors:*
Prof. Dr. Touradj Ebrahimi
Davi Lazzarotto

# Chapter 1

# Introduction

Each year, we produce more data than the previous one. All those datas are stored in multiple datacenters spread all over the world. Those datacenters require a lot of energy to maintain bits of information in spinning disks, tapes, capacities, transistors, ...

The Independent reported in 2016 that data centers will consume three times as much energy as they are currently using over the course of the next decade. [1]

It becomes naturally important to find more eco-friendly ways to store data.

# Chapter 2

# Problem defintion

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

We will first discover the state of the art of compression models for point cloud that leads to the best bitrates for various point clouds and then adapt it to output **A**, **C**, **G** and **T** symbols instead of the classical 0 and 1 binary base.

Then we will have to study the state of the art of DNA based storage and how we can store our long sequence of **A**, **C**, **G** and **T** in the most efficient way to minimise the cost of storing those datas while trying to recover the compressed point cloud in the most XXXX way. For this last part, we will have to study the influence of each parameters to store the DNA strand as small chunks in a solution. At the end, we will have to propose the best parameters to store a DNA strand for a certain period of time.

And finally, we will have to provide the reconstruct model to go from the compressed version to the point cloud as closed as possible from the original one.

## To continue

# Chapter 3

# State of the art

## 3.1 DNA based storage

As we produce more data every year, we need to find a way to store it efficiently. Currently, we store our data in big data center consuming a lot of energy to keep these informations in electronic devices, **Find the consumption of data centers**

It would be a good idea to find a way to store our data in a more efficient and ecological way. For storing information, hard drives don't hold a candle to 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]

But it can not be applied to all data types, for example, it is not possible yet to replace an USB stick by a DNA based USB stick and expecting the same experience. The information retrieval latency and high cost of the DNA sequencer and other instruments "currently makes this impractical for general use," says Daniel Gibson, a synthetic biologist at the J. Craig Venter Institute in Rockville, Maryland, "but the field is moving fast and the technology will soon be cheaper, faster, and smaller." Gibson led the team that created the first completely synthetic genome, which included a "watermark" of extra data encoded into the DNA. [2]

This does not mean that there are no applications for DNA based storage. DNA based storage can be used for long term media preservation archives (so called cold media storage) which are infrequently accessed and thus do not need low information retrieval latency.

### 3.1.1 Constraints

Unfortunately, nucleic acids have biological constraints and can not 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 when sequencing it. Unfortunately, the list of constraints is not exhaustive and in the real life, each arangement of nucleic acids has an impact on the strength of a strand, therefore we can only simulate the longevity of a strand thanks to the actual discoveries but not strictly respect 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 process, the explanation for each constraint comes directly from [3].

#### Synthesis

For example, to synthesis synthetic DNA, *in silico* designed constructs have to be split into smaller fragments [usually 200–3000 base pairs (bp)] [4]. The fragments are then splitted into several oligonucleotides (so-called oligos) [usually 40-100 bp] that are individually synthetized. Once synthetized, 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 synthetis 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 [5]. 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 [6].

#### 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

#### 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 [7]. Illumina and Nanopore sequencers also show an increased error rate in the presence of homopolymers [7]. 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 → CA and CG → TG, Nanopore data contain an increased amount of TAG → TGG and TAC → TGC substitutions [8] and a common substitution pattern in Illumina data is GGG → GGT [9].

### 3.1.2 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 [3] 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, without requiring programming experience. All functionality 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 step; With this simulator it is now possible to have a feedback of the strength of a particular DNA strand and thus help us to move towards the best DNA coding to ensure good information retrieval in the end.

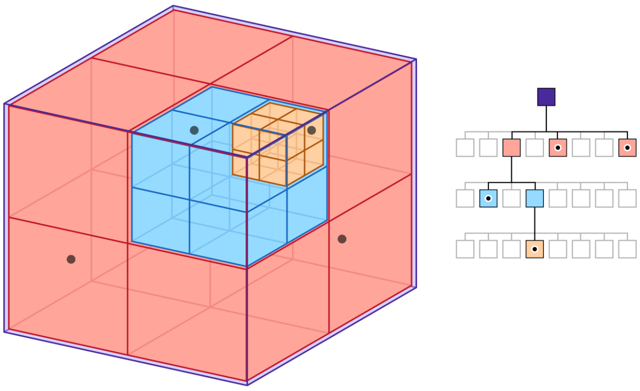

Figure 3.1: Octree representation of a point cloud

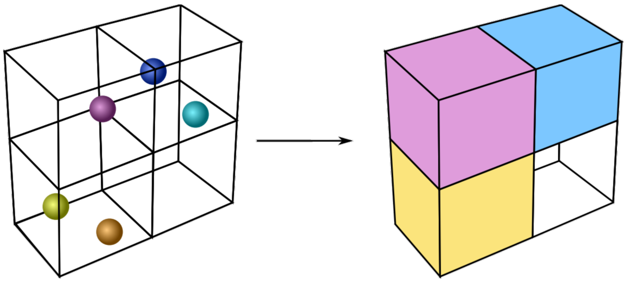

Figure 3.2: Voxelized representation of a point cloud

## 3.2 JPEG DNA codec

## 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 classical list of coordinates. Octrees, for example, have been widely used for this purpose [10]. The octree representation consists of diving recursively the three dimensional space as nodes of a tree as shown on Figure 3.1.

Compression algorithms using learning based autoencoders architectures have also demonstrated good performance. While some take as input point coordinates [11], others take as input voxelized versions of the point clouds. Voxelized point clouds consist of occupancy grid of regular spaced points so that several points are merged together in a single voxel as shown on Figure 3.2. A voxel is similar to a three dimensional pixel. These three dimensional grids can be then used as input for a *3D convolutional layer*.

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

### 3.3.1 Model architecture

This model is shown on Figure 3.3 and consists in a 3D autoencoder architecture with latent entropy coding. The input of the model is an occupancy cubic grid with $k \times k \times k$ voxels represented by 1 when occupied and 0 otherwise.

The first block (Analysis transform) of the model is composed of 3 *3D convolutional layers* and 2 *convolutional residual blocks* arranged staggered which produces a latent representation $y$ of shape $l \times l \times l \times d$ with $d$ being the latent dimension.

This latent representation is then fed into an Hyper-Analysis transform block yielding $z$. This hyperprior 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 $\hat{y}$, they use previously decoded

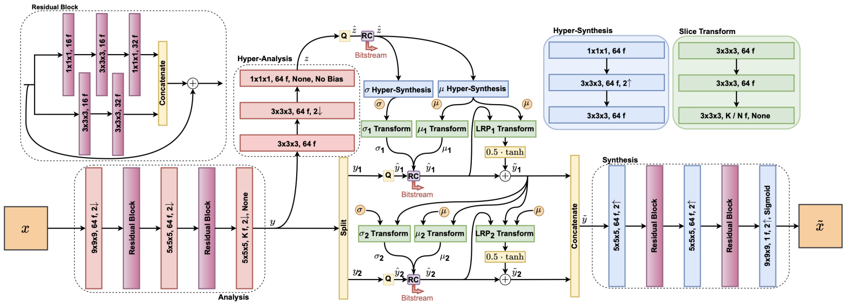

Figure 3.3: The "Latent Space Slicing For Enhanced Entropy Modeling in Learning-Based Point Cloud Geometry Compression" architecture with 2 slices.

channels for entropy modeling.

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 ($\mu_i$, $\sigma_i$) for each slice $y_i$ from the global entropy parameters ($\mu$, $\sigma$) 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) transform that predicts the quantization error $y_i - \hat{y}_i$ in order to take into account this error from the global entropy parameters ($\mu$, $\sigma$), the current $\hat{y}_i$ and 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.

They trained the model from end-to-end between the input occupancy grid and output probability occupancy grid as a rate-distortion minimization problem represented by the loss function expressed as $\mathcal{L} = R + \lambda D$. The trade-off parameter $\lambda$ 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 $\alpha$ and $\gamma$ 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, they decided to find the best threshold $t \in [0, 1]$ such that it minimizes the point-to-point MSE [12] between the original and reconstructed rounded block.

This technique was introduced by [13] 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 synthestized in a medium and kept for a long period (Could be dozen or even hundred of years). After this long period of time, we would like to recover our point cloud as faithful as possible to the original.

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 but it will be a baseline on which we can base ourselves to compare 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 reprsentation $y$ 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 shape since $8 \times 8 \times 8 \geq 160$.

Now that we indeed have a latent representation, we have to go to the next step, which is the JPeg DNA codec described in section 3.2, but this codec has some requierements that have to be met in order to achieve the best quality, nucleotide rate, all these details will be discuss in the next section.

## 4.2 Latent representation with Jpeg DNA codec

### 4.2.1 Dimensionality

We end up with 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, 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 pros and cons.

The first one aims at treating each feature separately so that we can encode regarding each feature distribution and hope for less quantized values and in the end, 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.

The second as for 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.

We will evaluate their performance in the section 5 to see which one is preferable over the other.

### 4.2.2 Quantization

The other important aspect of the codec is the value type. The latent representation $y$ has a *float* value type but the codec is built for *uint8* value type. This means that we have to convert the latent representation $y$ to an *uint8* value type while ensuring the $y$ range is mapped to the full *uint8* range $[0, 255]$.

To do so, we can use a *quantization* transform that will map the latent representation $y$ to the *uint8* range. This quantization transform is a *linear* transform that maps the latent representation $y$ to the *uint8* range and finally rounded. The mapping can simply be described as: $y_q = \text{round}\left(255\frac{y-\min(y)}{\max(y)-\min(y)}\right)$.

The problem with this naive quantizer is that it assumes that $y$ has an uniform distribution and thus divides the range $[\min(y), \max(y)]$ into 255 equally long gaps.

### 4.2.3 DCT

In the original JPEG codec, each $8 \times 8$ block of the images are first centered around 0 (simply shift by $-128$) and then trasnformed into the $8 \times 8$ 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 more adapted to our needs considering that our input is not a regular image. Since we already have quantized *uint8* values that are not perceptual values , we can directly used them as DCT coefficients. But then, we will not be able to divide by a quantization table because it would directly impact our coefficients.

However, in our case, not dividing by a quantization table is a plus since they are built in order to keep a perceptual fidelity which means that all the high frequency DC are usually dropped and it would affect badly our recovered coefficients and in the end the recovered point clouds because the model is more sensitive to high frequency details than our human eyes.

In the end, we have a codec that can encode our coefficients in a lossless manner. Unfortunately, one downside of this non DCT codec is that all non-zero coefficients are always encoded and thus never dropped, so we can not control the nucleotide stream length with this method. We will always have the best quality.

## 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 intermiediate information that are needed in order to fully decode the stream.

The intermediate informations needed to produce the nucleotide stream are:

- 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 → A, 01 → C, 10 → G, 11 → 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 intermerdiate informations:

- 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 informations 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 Reconstruction

Starting from the nucleotide stream, we can fully reconstruct the point doing by doing all steps in reverse order. First, we extract all the additional informations 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 reconstrcuted block $\tilde{x}$ Finally, we can round the probabibility 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

# Chapter 6

# Improvements

# Chapter 7

# Conclusion

# 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] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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 A

# Appendix

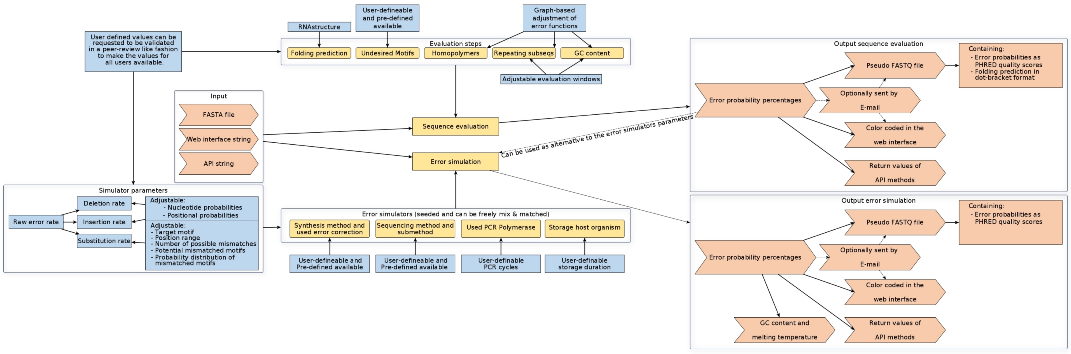
User-defineable and pre-defined available
RNAstructure
Graph-based adjustment of error functions
User defined values can be requested to be validated in a peer-review like fashion to make the values for all users available.
Evaluation steps
Folding prediction
Undesired Motifs
Homopolymers
Repeating subseqs
GC content
Adjustable evaluation windows
Input
FASTA file
Web interface string
API string
Sequence evaluation
Error simulation
Can be used as alternative to the error simulators parameters
Simulator parameters
Raw error rate
Deletion rate
Insertion rate
Substitution rate
Adjustable:
- Nucleotide probabilities
- Positional probabilities
Adjustable:
- Target motif
- Position range
- Number of possible mismatches
- Potential mismatched motifs
- Probability distribution of mismatched motifs
Error simulators (seeded and can be freely mix & matched)
Synthesis method and used error correction
Sequencing method and submethod
Used PCR Polymerase
Storage host organism
User-defineable and Pre-defined available
User-defineable and Pre-defined available
User-definable PCR cycles
User-definable storage duration
Output sequence evaluation
Error probability percentages
Pseudo FASTQ file
Containing:
- Error probabilities as PHRED quality scores
- Folding prediction in dot-bracket format
Optionally sent by E-mail
Color coded in the web interface
Return values of API methods
Output error simulation
Error probability percentages
Pseudo FASTQ file
Containing:
- Error probabilities as PHRED quality scores
Optionally sent by E-mail
Color coded in the web interface
GC content and melting temperature
Return values of API methods

Figure A.1: Mesa workflow