Università degli Studi di Padova

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Deep learning approaches for the identification of erroneous regions in metagenome-assembled genomes

Supervisor
Prof. Marco Baiesi
Università degli Studi di Padova

Co-supervisor
Dr. Paolo Manghi
Edmund Mach Foundation

Master Candidate
Marco Chiloiro

Student ID
2087630

Academic Year
2024-2025

"Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less."
— Marie Curie

Abstract

Metagenomics is the study of the genetic material of microorganisms sampled directly from the environment. This approach expands the field of study of microorganisms, adding results to those obtained with traditional microbiological techniques. One of the main tasks of metagenomics is the reconstruction of genomes contained in environmental samples, which often contain unknown species. To do this, the genetic content of an environmental sample is first sequenced, resulting in short DNA sequences called “reads”. To assemble reads into longer DNA sequences, called “contigs”, in order to group them into bins that will form the reconstructed genomes, so-called reference-free assembly tools are needed, which assemble the reads together without prior knowledge. The goal of this thesis is to develop an assembly quality assessment tool using deep learning to improve the quality of reconstructed genomes, also called metagenomic assembled genomes (MAGs).

Particular attention is paid to the identification of assembly errors due to repetitions within the genomes of different microorganisms. The starting dataset is the collection of over 1.5 million microbial genomes hosted at the University of Trento. A synthetic dataset of reads is built by simulating the Illumina sequencing process on several genomes almost perfectly complete and free from contamination. The assembly procedure is then applied to simulated reads from previously selected high-quality genome pairs having highly similar genomes, to increase the possibility of assembly errors due to DNA repeats in the obtained contigs. The simulated reads are then re-mapped onto the assembly contigs, in order to assign a degree of error per position as a function of the depth of coverage, i.e. how many reads align at a given position along a given contig. Briefly, if a region is not completely mapped by either genome, it is considered incorrectly assembled. Two different deep learning models, together with a benchmark model and two ensemble models, are trained on these data to predict the presence of misassembled regions in contigs, with the aim of developing a post-assembly quality control tool.

The models generalize quite well, particularly when combined into an ensemble, achieving AUC scores of ~ 0.8 on training-related genomes and ~ 0.7 on external genomes. While to be optimized, the presented models are capable of learning DNA-related properties of the genomic sequences in order to distinguish between correctly and erroneously assembled genomic regions in bacteria.

Contents

List of figures

List of tables

List of acronyms

DBG	De Bruijn Graph
MAG	Metagenome-Assembled Genomes
HTS	High-Throughput Sequencing
SGB	Species-level Genome Bin
NN	Neural Network
CNN	Convolutional Neural Network
NLP	Natural Language Processing
PDF	Probability Density Function
MLP	Multi-Layer Perceptron
AUC	Area Under the Curve
TPR	True Positive Rate
TNR	True Negative Rate
PPV	Positive Predicted Value
NPV	Negative Predicted Value

1
Introduction

1.1 DNA and the genetic code

Deoxyribonucleic acid (DNA) is a chemical compound that carries genetic instructions for the functioning and reproduction of all known organisms. The molecules composing DNA are made up of two paired twisted strands, known as "double helix". Each DNA strand is composed of four units called nucleotide bases, i.e. adenine (A), thymine (T), guanine (G) and cytosine (C). These four bases constitute the genetic alphabet that encodes information within DNA. In fact, the order of these bases determines the content of information in the DNA molecule, and this is why we often speak of DNA "sequences". Within the double helix, bases on opposite strands always have a specific pairing pattern: A pairs with T and C pairs with G. Therefore, knowing the identity of one of the bases in the pair automatically determines the other base in the pair, and then to report a DNA sequence is sufficient to report the sequence of bases in only one of the two strands. Two other important concepts to define are *genome* and *gene*. A genome is defined as the entire genetic information of an organism, while a gene is a section of DNA that contains instructions for making a functional product, such as a protein.

The building blocks of an organism are proteins that are formed by chains of amino acids. The sequence of amino acids in a protein determines its structure and functions. DNA contains the instructions to synthesize amino acids, and thus proteins. These instructions are encoded by *codons* and the set of rules that translates the DNA code, composed by 4 letters, to the codons code is named *genetic code*. Codons are DNA sequences made up of three nucleotide bases, and each codon encodes a signal to start or stop protein synthesis or for a specific amino acid. While there are 64 possible codons, there are 20 amino acids used to build proteins. This fact implies the so-called redundancy of the genetic code, that is, that some amino acids are encoded by more than one codon. A crucial aspect of genetic code is that the same codons code for the same amino acids in nearly all species, meaning that genetic code is universal across almost all living organisms. We can say then that DNA is the universal language of life, which makes its understanding and study a crucial aspect of biology.

1.2 Illumina sequencing

DNA *sequencing* is the process of determining the exact order of nucleotide bases in DNA. One of the most used technologys in the field of DNA sequencing is Illumina sequencing, which is based on the *sequencing-by-synthesis* method. In the following, we give a brief overview of the main steps in Illumina sequencing.

The first step in Illumina sequencing is the fragmentation of the DNA contained in the sample on which we want to perform sequencing. This step can be done using techniques such as those based on the use of ultrasonic sound waves, which break the DNA strands into smaller pieces, usually about 50-500 base pairs (bp) long. Once fragmented, *adapters*, that are short synthetic DNA sequences, are added to the edges of each DNA fragment. Adapters are made by three main parts: a sequence that is complementary to the small DNA sequences anchored to the flow cell, the barcode sequence, that serves to identify samples since Illumina sequencing can handle multiple samples in parallel, and by a binding site for the sequence primer. A primer is a short sequence of DNA which acts as a trigger for the synthesis of a new strand of DNA. The result of this process is called DNA library, that contains millions of different fragments. Then, the prepared DNA library is loaded onto a flow cell, which is a small glass surface with thousands of short nucleotide sequences anchored that can bind to the first part of the adapters and act as a support to hold the DNA strands during the process. Through bridge amplification, a technique used to duplicate DNA strands, many identical copies of each fragment are generated. These amplified fragments form dense clusters of identical DNA sequences. This is to ensure that the sequencing process produces reliable results. After amplification, the true sequencing-by-synthesys process begins. A primer attaches the strands adapter primer binding site and a polymerase starts adding fluorescently tagged nucleotides to the strands. A polymerase is the enzyme responsible for the synthesis of the DNA strand complementary to a given strand. Each of the four fluorescently tagged bases can be excited in order to emit a unique wavelength, so by recording all the emissions the machine records which bases are added during the synthesis.

A key concept in DNA sequencing is *depth of coverage*. Depth of coverage, sequencing coverage, or simply coverage, is defined as the average number of times each nucleotide base in a genome is sequenced. It indicates how well a sample is covered by sequencing reads: the higher the value, the better the analysis and the subsequent sequencing steps.

The main advantage of Illumina sequencing is the ability to sequence millions of DNA fragments in parallel in a short amount of time and at a reduced cost. This technology is also called high-throughput sequencing (HTS) and generates sequencing data known as *reads*, which are essentially transcripts of short DNA sequences of length 50-300 bp.

1.3 Metagenomics

The study of microorganisms, particularly bacteria, traditionally involves bacterial cultivation. This technique involves isolating and growing microorganisms in the laboratory under controlled conditions. However, this technique has an important limitation: a significant portion of existing microorganisms cannot be easily grown in the laboratory, due to the difficulty in recreating the environmental conditions in which these organisms live.

With the development of high-throughput sequencing technologies such as Illumina sequencing, it becomes possible to study the genetic material from entire ecosystems. Specifically, the study of the genetic material of microorganisms sampled directly from the environment is called *metagenomics*. This approach avoids the need to culture individual organisms prior to DNA sequencing, thus enabling new studies and research in the field of microbial studies. Metagenomics has allowed the study of microbial communities in their natural environments, such as soil [1][2], oceans [3], human gut [4] and food [5]. In particular, it has led to the identification of new species, the determination of the functional roles of different organisms in different environments, and the interactions that occur between different species.

1.4 Reference-free assembly: reconstructing genomes

One of the critical challenges in reconstructing a genome from raw DNA sequencing data is assembling the sequencing reads in order to form longer DNA sequences that actually belong to the genome one wants to reconstruct. Assembly is performed by the usage of informatic tools, that are traditionally reference-based. Reference-based assembly tools rely on a reference genome (an already reconstructed and characterized genome) that serves as a guide to assemble sequencing reads into continuous DNA sequences. In the context of metagenomics, where the species in a sample are typically unknown, this kind of approach is not useful. Therefore, *reference-free* assembly, or de novo assembly, becomes essential. These kind of tools do not rely on prior knowledge, and they perform assembly solely from raw sequencing data.

One of the most used approaches for this type of assembly tools is based on the De Bruijn Graph (DBG) approach. A popular tool based on this method is MEGAHIT [6]. In this type of methods, sequencing reads are split into smaller overlapping sequences of length *k*, called *k-mers*. For example, an "ATGC" read can be split into 3-mers: "ATG", "TGC". Genomes are represented as directed graphs: each node in the graph is represented by a k-mer, and edges in the graph connect nodes that share overlapping sequences. Assembly algorithms based on the DBG approach aim to find multiple paths through the graph that collectively "explain" all the edges. In the DBG there are often multiple valid traversals, and identifying the correct one is the main source of computational complexity in the assembly. When ambiguities cannot be resolved, the assembly breaks the traversal and returns fragmented reconstructions of the original genome, i.e. DNA sequences called *contigs*. Contigs are usually returned in FASTA format, a format used in bioinformatics that is a textual representation of nucleotide sequences, where each sequence is preceded by the sequence name and comments.

In the field of metagenomics, reference-free assembly is commonly used to reconstruct the genomes of individual organisms present in an environmental sample. The main steps to perform this operation are as follows:

- **Samples collection and DNA extraction**: collection of environmental samples, followed by DNA extraction in order to obtain the genetic material from all microorganisms present in the sample.
- **Sequencing data acquisition**: the DNA extracted is then subjected to HTS technologies, such as Illumina sequencing, generating numerous short reads that represent fragments of genomes from the entire microbial community present in the collected sample.
- **Reads preprocessing**: the raw sequencing data are then processed to remove low-quality reads, adapters and other artifacts.
- **Reference-free assembly**: the cleaned reads are assembled into contigs by using reference-free assembly algorithms.
- **Binning**: assembled contigs are grouped into bins, which represent the overall structure of the genomes contained in the collected sample. Binning tools, such as MetaBAT [7], ensures that the contigs are ordered correctly within bins. The output reconstructed genomes are called metagenome-assembled genomes (MAGs).
- **Refinement**: the resulting MAGs are then refined in order to correct any remaining errors. Tools such as CheckM [8] are used to assess the quality of MAGs.
- **Annotation and analysis**: high-quality MAGs are then annotated. Annotation involves identifying genes and other functional regions in the genome. This can be done through computational methods and by comparing the assembled genome to known databases of gene sequences. The final assembled and annotated genome is then ready for further analysis.

In Figure 1.1 the critical steps for the reconstruction of the genomes present in an environmental sample are schematized.

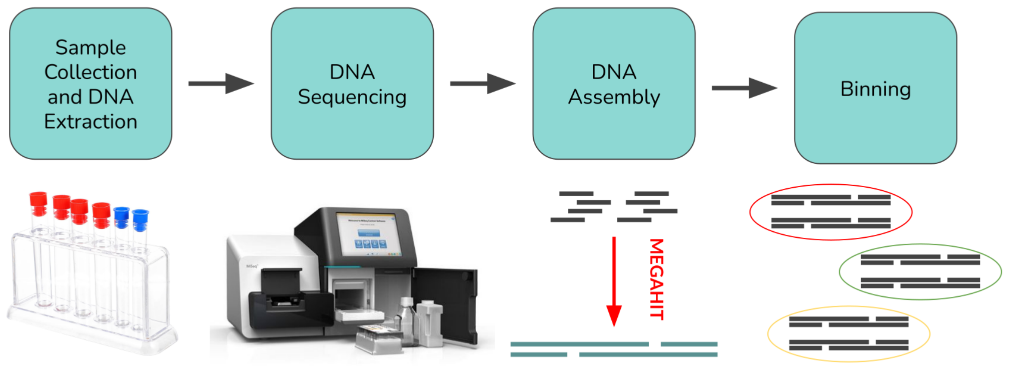


Figure 1.1: Main steps for genomes reconstruction for a metagenomic sample. The first step is to recover the sample from the environment, such as living organisms (human gut), soil, or water. After the DNA has been fragmented into small pieces of controlled length, DNA sequencing is performed using a specialized machine. The ouput of DNA sequencing are text-based files (FASTQ) containing all the information of sequencing, in particular the nucleotide base sequences of short DNA fragments, i.e. sequencing reads. After preprocessing this data, a reference-free assembly tool is applied to create longer DNA sequences, called contigs, belonging to the different genomes contained in the original sample. One of the most commonly used tools is MEGAHIT. These contigs are then binned, or scaffolded, to be sorted and clustered into reconstructed genomes, i.e. a MAG, through a specific bioinformatics tool.

1.5 Deep learning in metagenomics

The development of backpropagation and the increase in computational power in recent years have made deep learning, a branch of machine learning, highly popular due to its ability to identify complex patterns in large and intricate datasets. Deep learning focuses on algorithms based on multi-layered neural networks (NNs), which is where the term "deep" comes from. In these networks, each layer receives and processes information from the previous layers [9].

Metagenomic datasets are typically very large, complex, and often unlabeled or unannotated, making traditional methods often insufficient. Deep learning offers valid alternative methods to overcome these limitations. Various deep learning-based tools have been developed in the field of metagenomics, from protein structure prediction to taxonomic classification, i.e. the way of organizing living organisms into groups based on their similarities and differences, of assembled contigs [10].

A deep learning application we are interested in is the classification of raw DNA sequences. This task falls under *supervised* deep learning, where labels, i.e. the information we want the model to learn to predict, are provided to the model beforehand. In this context, the main goal of the deep learning model training is to minimize a loss function, which measures how well the model performs in making accurate predictions. This is achieved by adjusting the weights of the neurons through backpropagation at each epoch. By providing the model with appropriately encoded DNA sequences, most of the feature extraction is managed directly by the NN, removing the need for prior knowledge or manual feature engineering. For this type of sequence classification task, CNN architectures have shown the best performance [11].

Another interesting type of NN architectures is based on natural language processing (NLP). These models use techniques like attention mechanisms, word embeddings, and transformers [12] to understand and model the meaning of text. An interesting analogy is between text and DNA sequences: DNA can be seen as a kind of text, with its own alphabet (nucleotide bases) and words (k-mers). Because of this similarity, attention-based tools, particularly transformers, seem well-suited for raw DNA sequence classification. For example, tools like VirNet [13] and DLMeta [14] use such models for classification tasks. VirNet employs a deep attention model to identify viruses, while DLMeta combines CNNs with transformers to perform various metagenomic identification tasks, such as viral identification.

1.6 Assessment of assembly quality

A key aspect of metagenomics is the construction of reliable MAGs, which are derived by binning assembled contigs into distinct genomic bins. However, metagenomic assembly is computationally challenging due to the complexity of managing DNA sequences that involve both intra-genomic repeats (repeated DNA sequences within the same organism) and inter-genomic repeats (DNA sequences shared between different organisms). Although inter-genomic repeats in bacteria are typically small (usually under 10^4 bp), intra-genomic repeats can be much larger, sometimes almost as long as the genome itself. This is due to the fact that certain genes can differ between closely related bacterial genomes within a microbial community, leading to almost the entire genome potentially being regarded as an inter-genomic repeat. As a consequence, metagenome assemblies are often incomplete and likely contain errors.

The main factors that can influence the performance of DBG assemblers are sequencing errors, sequencing coverage depth, and the presence of repeats. Especially due to this last source of assembly error, it can happen that so-called *chimeric assemblies* are formed, in which sequences from different genomes are mistakenly joined.

Two main approaches are commonly used to assess assembly quality: *reference-based* and *reference-free* methods. In reference-based error detection, assembly errors are found by comparing the assembled data with reference genomes that have already been sequenced. On the other hand, reference-free methods look at the features within the assembled data itself to find inconsistencies that might indicate errors.

Reference-based methods work well when testing assemblies of communities with known compositions. However, these methods are less effective when applied to metagenomics data. For example, if a metagenomic sequence comes from a genome that does not have available reference, the reference-based approach cannot be used to check for errors. Furthermore, it is difficult to state if differences between an assembled sequence and the reference are actual errors or if they are real variations between the genomes in the metagenomic sample. An example of a reference-based approach is provided by

MetaQUAST[15].

Reference-free methods, on the other hand, offer a more flexible and widely applicable solution for assessing the quality of metagenomic assemblies.

1.7 Assembly reference-free assessment tools: related works

Existing reference-free strategies are consistency-based. These methods identify errors by aligning sequencing reads with assembled sequences. In this context, the alignment procedure consists in remapping the sequencing reads onto the contigs assembled from them, in order to find the exact position of each read along the assembly contigs. From alignment information, various features are extracted, such as sequencing coverage, mapping quality, alignment length and mismatch rates. These features are subsequently utilized in statistical or machine learning models to detect misassemblies. In the following, we report the most popular reference-free methods. ALE [16] assesses the quality of assemblies as the likelihood that the observed sequencing reads could be produced from a specific assembly, based on a model of the sequencing process. VALET [17] identifies misassemblies by analyzing a combination of different metrics derived from the alignments. metaMIC [18] is a machine learning-based tool that extracts features from read alignments and uses them to train its model for identifying misassemblies. SuRankCo [19] applies machine learning to rank contigs using features such as their lengths and coverage. Methods that use deep learning are DeepMAsED [20] and ResMiCo [21], which harness features analogous to those used by metaMIC. A novel reference-free method is DeepMM [22], which transforms alignments into images for feature learning and applies contrastive learning to better detect misassemblies from different perspectives.

1.8 Goal

As previously mentioned, all existing reference-free assembly quality assessment methods rely on features derived from aligning reads against assemblies. Our goal is to develop a deep learning algorithm capable of distinguishing well-assembled sequences from misassembled ones directly from raw DNA sequences, without the need for reads alignment, thereby avoiding a computationally expensive step. Furthermore, in our work, we focus particularly on identifying assembly errors caused by repeats, such as chimeric assemblies, which are assembly contigs obtained by mistakenly joining regions of DNA from genomes belonging to different species.

To achieve this, we create a synthetic dataset that highlights errors due to repeats while controlling ones due to coverage depth. The main idea is that by assembling reads from very similar genomes, chimeric assemblies are highly likely to occur. We set a high coverage value to simulate reads from high-quality MAGs coming from the same *species-level genome bins* (SGBs), and then we assemble these reads together. Species-level genome bins [23] are clusters of genome bins that span 5% genetic diversity. Essentially, genomes belonging to the same SGB share at least 95% of their genomes.

The high coverage value, set equally for all read simulations, serves two purposes: it limits assembly errors due to depth of coverage and establishes the expected coverage value for well-assembled sequences. Taking this last consideration into account, we develop two methods to define binary labels based entirely on coverage data. Once the labels are assigned, model training is performed only on raw assembly sequences, using an appropriate encoding. We demonstrate that the model can learn effectively from raw data, and the results obtained are very conservative, i.e., the model recognizes almost all well-assembled sequences. However, the algorithm is not yet optimal, as only a portion of misassembled sequences is identified. Nevertheless, the results are promising, as the algorithm can effectively recognize a part of misassembled sequences without discarding well-assembled ones, potentially improving the quality of the binning step.

2
Theoretical Background

In this chapter, we present a brief overview of the primary neural network architectures employed in this work.

2.1 Multilayer perceptron

A *multilayer perceptron* (MLP) is a feedforward neural network consisting of an input layer, an output layer, and at least one hidden layer. At each layer, except for the input layer, there is an associated weight matrix and a bias vector. The output of each non-input layer is computed as a vector resulting from a vector-matrix multiplication between the output vector of the previous layer and the weight matrix associated with the current layer, summed with a bias vector, and then passed through an element-wise nonlinear activation function.

Consider an MLP with $N+1$ layers, where D_d denotes the dimensionality of the output of the d-th layer for $d \in \{0, \dots, N\}$. Here, $d = 0$ corresponds to the input layer, and $d = N$ corresponds to the output layer. Given the output vector of the d-th layer, $\vec{x}_d \in \mathbb{R}^{D_d}$, the output of the $(d+1)$-th layer is computed as

$$\vec{x}_{d+1} = \sigma_{d+1}(\mathbf{W}_{d+1} \cdot \vec{x}_d + \vec{b}_{d+1}), \tag{2.1}$$

where $\vec{b}_{d+1} \in \mathbb{R}^{D_{d+1}}$ is the bias vector, $\sigma_{d+1} : \mathbb{R}^{D_{d+1}} \to \mathbb{R}^{D_{d+1}}$ is the element-wise nonlinear activation function and $\mathbf{W}_{d+1} \in \mathbb{R}^{D_{d+1} \times D_d}$ is the weight matrix associated with the $(d+1)$-th layer.

The total number of trainable parameters M_{MLP} of a MLP is given by the following equation:

$$M_{MLP} = \sum_{d=0}^{N-1} (D_d D_{d+1} + D_{d+1}). \tag{2.2}$$

The time complexity of a forward pass of such architecture is equal to:

$$O\left(\sum_{d=0}^{N-1} D_d D_{d+1}\right). \tag{2.3}$$

2.2 Convolutional Neural Network

A CNN is a type of feedforward neural network that is typically used in deep learning for tasks such as image processing and computer vision. Although CNNs are typically applied to images, they can also be adapted to one-dimensional sequences, which is the case we consider in this section.

The building block of a CNN is the *convolutional layer*. The key idea is to convolve a kernel, or filter, with the input sequence. Note that sequences are often encoded and therefore represented as matrices with a length, which corresponds to the sequence length, and a depth, which is the dimensionality of the encoding for each element in the sequence. Kernels are matrices with the same depth as the input but with a shorter length. The trainable parameters of the layer are represented by the kernels, together with the bias term added after the convolution operation. Two other hyperparameters control the convolution operation: stride S, which defines how the kernel moves along the sequence, and padding P, which determines how the input sequence is augmented at its boundaries, e.g., for zero padding with zeros.

Given an input $\mathbf{x} \in \mathbb{R}^{L_{\text{in}} \times C_{\text{in}}}$ of length L_{in} and depth C_{in}, the convolution operation with a kernel $\mathbf{w} \in \mathbb{R}^{K \times C_{\text{in}}}$ of length K at position i along the padded sequence, also considering the bias term b, produces the output:

$$y_j = \sum_{m=0}^{K} \sum_{n=0}^{C_{\text{in}}} x^P_{i+m,n} w_{m,n} + b, \tag{2.4}$$

where $\mathbf{y} \in \mathbb{R}^{L_{\text{out}}}$ is the output sequence of length L_{out} and $\mathbf{x}^P \in \mathbb{R}^{(L_{\text{in}}+2P) \times C_{\text{in}}}$ is the padded input.

The final output is the concatenation of all output vectors for each applied kernel. Its length L_{out} is determined by

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2P - K}{S} \right\rfloor + 1, \tag{2.5}$$

and its depth C_{out} corresponds to the number of filters applied. After the convolution, an element-wise nonlinear activation function is applied to the output.

The total number of trainable parameters M_{CL} in a convolutional layer is given by:

$$M_{\text{CL}} = (KC_{\text{in}} + 1)C_{\text{out}}. \tag{2.6}$$

It is important to note that the number of parameters does not depend on the input length L_{in}. This is because the parameters are shared across the entire input sequence through the convolution operation. On the other hand, time complexity of a forward pass depends on it, and it is

$$O(L_{\text{in}} K C_{\text{in}} C_{\text{out}}), \tag{2.7}$$

since from Equation 2.5 we see that L_{out} strictly depends on L_{in}.

CNNs usually implement convolutional layers to extract features, pooling layers to reduce data size and improve efficiency, and fully connected layers, or MLPs, to combine features for final predictions in supervised learning tasks.

2.3 Self-attention mechanism

The *self-attention* mechanism enables a deep learning model to quantify the importance of each element in a sequence in relation to all other elements. Unlike recurrent neural networks (RNNs), which process input sequences sequentially, self-attention processes the entire input at once. This fact enables the model to capture long-range dependencies within the sequence more effectively, addressing a key limitation of RNNs where information tends to degrade over long sequences.

Given an encoded input sequence $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$ of length L and encoding depth d_{model}, the self-attention mechanism begins by applying three linear projections of $\mathbf{X}$, obtaining three distinct representations: the *value* $\mathbf{V} \in \mathbb{R}^{L \times d_V}$, *query* $\mathbf{Q} \in \mathbb{R}^{L \times d_K}$, and *key* $\mathbf{K} \in \mathbb{R}^{L \times d_K}$ representations. Specifically, the three projections are performed as follows:

$$\mathbf{V} = \mathbf{X}\mathbf{W}^V; \quad \mathbf{Q} = \mathbf{X}\mathbf{W}^Q; \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \tag{2.8}$$

where $\mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_V}$, $\mathbf{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_K}$, and $\mathbf{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_K}$ are weight matrices containing learnable parameters. Query and key representations are used to compute attention weights, which are then applied in a weighted average of the values for each position within the sequence. This operation is carried out by the attention function, defined as follows:

$$\text{Attention}(\mathbf{V}, \mathbf{Q}, \mathbf{K}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_K}}\right)\mathbf{V}. \tag{2.9}$$

In the equation above, the softmax function is applied row-wise in order to normalize the weights. This ensures that the weighted average is simply a weighted sum. The factor $(d_K)^{-1/2}$ has the function of counteracting the growth of the dot product as d_k increases. Without this scaling, the dot product could become excessively large, pushing the softmax into regions where the gradient becomes very small, making learning difficult.

In summary, the attention function in Equation 2.9 computes a sequence representation where each element, i.e. each row of the output matrix, is a weighted sum of all other elements in the sequence, where the weights, which are calculated using the query and key representations, capture contextual relationships.

2.4 Multi-head self-attention

Multi-head self-attention involves applying the self-attention mechanism h times in parallel to the input $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$. The resulting matrices for each head are then concatenated, and a final linear projection is applied to this combined representation. This approach allows the model to learn information from multiple representation subspaces. Specifically, we have that:

$$\begin{aligned} \text{MultiHead}(\mathbf{X}) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^0 \\ \text{with head}_i &= \text{Attention}(\mathbf{X}\mathbf{W}_i^V, \mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K), \end{aligned} \tag{2.10}$$

where $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_V}$, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_K}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_K}$ are the weight matrices of the i-th self-attention head, with $i \in \{1, \dots, h\}$, and $\mathbf{W}^0 \in \mathbb{R}^{hd_V \times d_{\text{model}}}$ is the weight matrix associated with the last linear projection.

Since $d_{\text{model}} \simeq d_V \simeq d_K$ is the most common case, we consider $d_{\text{model}} = d_V = d_K = d$ for simplicity. Therefore, the total number of learnable parameters M_{MHA} of a multi-head attention layer is:

$$M_{\text{MHA}} = d^2(3 + h). \tag{2.11}$$

The initial linear projections are performed in parallel as matrix-matrix multiplications, resulting in a time complexity of $O(Ld^2)$. The final projection has the same time complexity. The attention computation requires $O(L^2d)$, and this operation is also performed in parallel across all attention heads. Therefore, the overall time complexity of a forward pass is:

$$O(Ld^2 + L^2d). \tag{2.12}$$

2.5 Transformer Encoder

A transformer encoder layer is composed of two sub-layers: a multi-head self-attention mechanism layer followed by an MLP. Each of these sub-layers is accompanied by a residual connection, and after the addition of the residuals, layer normalization [24] is applied. The output of each sub-layer is given by $\text{LayerNorm}(\mathbf{X} + \text{Sublayer}(\mathbf{X}))$, where Sublayer($\mathbf{X}$) represents the operation performed by the sub-layer. The MLP sub-layer is applied to each position separately and identically. It is made up of one hidden layer, after which the ReLU activation function is applied. The dimensionality of input and output layers is d_{model} while the hidden layer has dimensionality d_{ff}. A diagram of this architecture is shown in Figure 2.1.

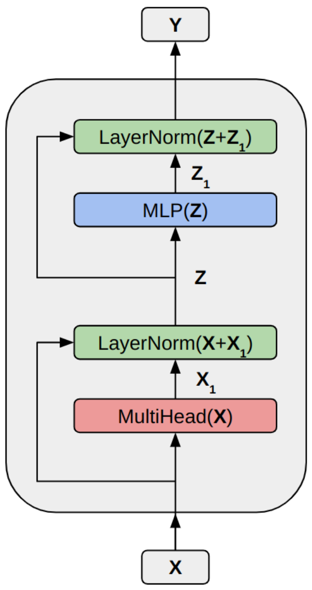


Figure 2.1: Diagram of a transformer encoder layer. Given an input sequence $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$, the output **Y** has the same dimensionality.

3 Methods

3.1 Chimeric assembly contigs dataset

Our goal is to train a deep learning model capable of distinguishing between well-assembled and misassembled DNA sequences. To achieve this, we build a dataset that includes examples of both scenarios. The main idea is that by assembling reads coming from organisms that have very similar genomes, it is very likely that the resulting assembly may contain errors caused by repeats and chimeric assembly contigs.

The starting dataset consists of more than 10^6 genomes originated from various samples. While this dataset is not yet publicly available, further details can be found in the referenced paper [25]. Each genome within this dataset is stored as a FASTA file. The dataset is organized into Species-level Genome Bins (SGBs). We select from this dataset only bacterial genomes with *completeness* greater than or equal to 98% and a *contamination* less than or equal to 2%. Completeness and contamination are commonly used metrics to evaluate the quality of assembled genomes, particularly in the case of MAGs. Completeness measures whether a genome is complete, that is, whether it includes all expected genes. Contamination measures the presence of external genetic material within a genome. Both metrics were estimated using the tool CheckM. From these selected genomes, only SGBs that contain at least 50 genomes are considered. Then, for each SGB, only the best 50 genomes are kept. The metric used for the last step is completeness -3contamination, a commonly used metric to evaluate the quality of a genome. The resulting dataset is made up of 3800 high-quality genomes: 50 genomes per 76 SGBs.

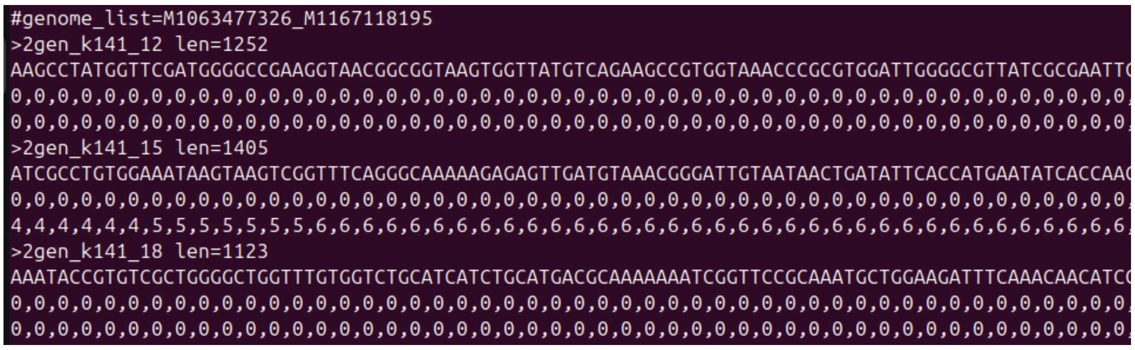
Since we do not have the original reads from which the genomes were assembled, we need to simulate them. The simulation is performed using the tool CAMISIM [26], which returns reads in FATSQ format from a given genome in FASTA format. The following parameters are used for the simulation.

- The Illumina HiSeq 2500 system is selected, a widely used high-throughput sequencing platform.
- Paired-end simulation is performed, meaning that each DNA fragment is sequenced from both ends, generating two reads per fragment.
- The length of the reads is set to 150 bases, a typical choice for Illumina sequencing.
- The coverage is set to 100, which means that each base in the genome is sequenced 100 times on average.
- The mean size of the DNA fragments that are being sequenced is set to 400 base pairs with a standard deviation of 10 base pairs.

The next step is to assemble reads simulated from different genomes belonging to the same SGB. For each SGB, 10 genome pairs are randomly sampled from the 50 genomes, and the corresponding simulated reads are assembled for each of these pairs: the read files are concatenated and then assembled using the tool MEGAHIT. The results of this procedure are 760 FASTA files. Contigs with less than 1000 bases are removed from these files, as short sequences are of little significance in the process of reconstructing a genome.

Once we have obtained the assembly contigs, we need to compute alignment coverages in order to calculate a label per position, as we will explain in detail later. Coverage is defined as the number of reads aligned at a given position along the DNA sequence. To obtain these coverages we need to align the simulated reads on the corresponding assemblies. Since each assembly comes from a pair of genomes, two alignment operations per file are needed. The alignments are performed using the tool Bowtie2 [27] by setting the "very-sensitive-local" mode. In order to consider only high-quality alignments, a filter is applied on the obtained BAM files: only alignments with mapping quality (MAPQ) greater than 20 and which have at least 120 alignment matches are considered.

Once the assembly and alignment files are obtained, all necessary information per assembly can be stored in a single file, hereafter referred with the *seqcov* (sequence-coverage) extension. An example of this file is shown in Figure 3.1.
It is important to say that all the software tools mentioned and used (CheckM, CAMISIM, Megahit, Bowtie2) are well known and considered reliable by the scientific community dealing with metagenomics.



```
#genome_list=M1063477326_M1167118195
>2gen_k141_12 len=1252
AAGCCTATGGTTCGATGGGGCCGAAGGTAACGGCGGTAAGTGGTTATGTCAGAAGCCGTGGTAAACCCGCGTGGATTGGGGCGTTATCGCGAATT
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
>2gen_k141_15 len=1405
ATCGCCTGTGGAAATAAGTAAGTCGGTTTCAGGGCAAAAAGAGAGTTGATGTAAACGGGATTGTAATAACTGATATTCACCATGAATATCACCAA
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4,4,4,4,4,4,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
>2gen_k141_18 len=1123
AAATACCGTGTCGCTGGGGCTGGTTTGTGGTCTGCATCATCTGCATGACGCAAAAAAATCGGTTCCGCAAATGCTGGAAGATTTCAAACAACATC
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 3.1: Example of a seqcov file. The main header lists the genomes used to create the chimeric assembly. Following this, there is a header for each contig (indicated by a ">" at the beginning of the line) that specifies the name of the contig and its length. Below the header, the nucleotide sequence is provided, followed by the coverage for each genome, with coverage values listed per position and separated by commas.

3.2 Labels

Given an assembly contig, the main challenge is to define a binary label that can indicate whether a position within the sequence belongs to a well-assembled sequence, therefore belonging to a true own genomic sequence, or if it belongs to an artifact sequence. As anticipated, alignment coverages per genome are used to compute these labels. The idea is that since the reads from which the assemblies are derived are simulated with a coverage of 100, well-assembled regions are expected to show a similar coverage value, while misassembled regions do not. Therefore, by somehow setting a threshold on the coverage values, it is in principle possible to determine such a label. The reason for choosing high coverage in read simulations is that this leads to less ambiguity in defining labels, since statistical fluctuations in coverage are less significant.

To understand the assembly coverage statistics, the global histogram that collects all coverage values in all assembly files is shown in Figure 3.2. For the following observations and for simplicity of calculation, from now on we consider the coverage values as i.i.d. continuous random variables. From the histogram, we note that the distribution of coverage values appears to be separated into three parts: for large coverage values, the frequencies appear to follow a normal distribution centered in 100, for lower coverage values seem to follow a uniform distribution and finally there is a frequency peak corresponding to the bin $[0, 5)$. The first part is not surprising. Since the reads were simulated with average coverage equal to 100, we expect that along the well-assembled regions the alignment coverage values follow a normal distribution centered in 100. The second part is due to the "tails" of well-assembled sequences. In a well-assembled sequence, on average, the coverage is lower at the beginning and end because fewer reads can align in these regions. As you move toward the central zone, coverage increases until it stabilizes, here following a normal distribution. The same pattern shows in reverse toward the end, where coverage gradually decreases. This happens because there are fewer alignment possibilities at the edges of the sequence. In summary, for positions within the tails of well-assembled regions, we assume that they follow normal distributions with variable means depending on the position in the tail: for positions in the tails closer to the central zone of the corresponding well-assembled sequence, these distributions have larger means and vice versa. As a result, the overall distribution of these positions appears to follow a uniform-like distribution, as seen in the histogram for coverage interval $[5, 70]$. It is also important to note that there are far fewer positions within the tails than those within well-assembled sequences or within the frequency peak for 0 coverage value. For sequence positions within misassembled regions, we expect coverage values equal to 0, apart from small fluctuations. The frequency peak corresponding to the first bin of the histogram is due to this last fact and the fact that we are considering the coverage data for the alignments of both genomes used for the assembly. Therefore, in addition to the misassembled regions in which for a given assembly neither of the two genomes has coverage values other than 0, there are also regions in which for a genome the sequence is well-assembled, therefore with coverage values that follow the normal distribution centered in 100, while for the other genome the coverage value is equal to 0. This happens because the initial genomes are different, so there are regions that are not in common.

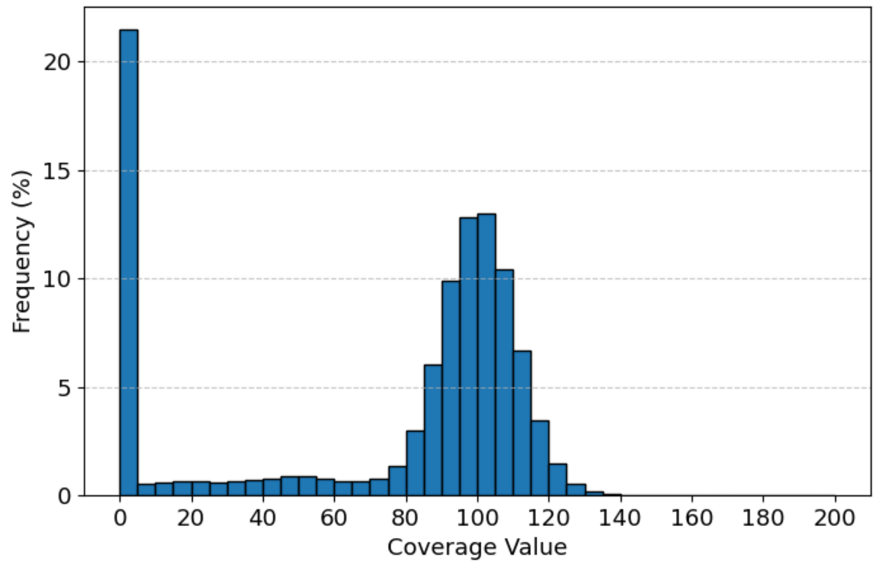


Figure 3.2: Histogram that collects all coverage values in all assembly files. The values reported on the y-axis represent the percentage frequency, while the x-axis shows the coverage values separated into bins of size 5.

In light of these considerations, we want to define a way to state that a certain position in the sequence, given the corresponding coverage values, does not belong to a well-assembled sequence. To simplify the calculation, we make the following assumptions: coverage values for positions within well-assembled sequences are continuous random variables i.i.d. according to the normal distribution $\mathcal{N}(\mu = 100, \sigma^2)$, and coverage values greater than 70 coming from well-assembled sequence tails do not significantly influence this distribution. The first hypothesis, as already mentioned, derives from the fact that the reads are simulated with coverage 100. The second one comes from the fact that, as previously observed, the positions belonging to the tails are few compared to those belonging to the central regions of the well-assembled sequences. Since the overall distribution of coverage values is a mixed distribution, it is difficult to separate these two parts and, to simplify the calculation, we make this assumption. Furthermore, the positions belonging to the tails also come from well-assembled regions, so mistakenly considering the coverage values of these positions as belonging to the normal distribution would not be a problem for our purposes. We perform a Bayesian inference to estimate the variance σ^2 of the normal distribution $\mathcal{N}(\mu = 100, \sigma^2)$. This involves determining the conditional *probability density function* (PDF) of the variance given the observed data D, in other words the posterior distribution $p(\sigma^2|D)$, given the likelihood $p(D|\sigma^2)$ and the prior $p(\sigma^2)$ PDFs. The relationships between these PDFs comes from the Bayes Theorem, as reported in the following proportion:

$$p(\sigma^2|D) \propto p(D|\sigma^2)p(\sigma^2). \tag{3.1}$$

The likelihood PDF is defined as:

$$p(D|\sigma^2) = \prod_{i=1}^{n} \mathcal{N}(c_i; \mu = 100, \sigma^2) = (2\pi)^{-n/2}(\sigma^2)^{-n/2} \exp\left\{-\frac{\sum_{i=1}^{n}(c_i - \mu)^2}{2\sigma^2}\right\}, \tag{3.2}$$

where $c_i \in D$, with $i = 1, ..., n$, are the coverage values and $n = |D|$ the number of samples. As a prior distribution we choose a scaled inverse chi-squared distribution, defined as follows

$$f(x; \nu, \tau^2) = \frac{(\tau^2\nu/2)^{\nu/2}}{\Gamma(\nu/2)} x^{-(1+\nu/2)} \exp\left\{-\frac{\nu\tau^2}{2x}\right\}, \tag{3.3}$$

which is the conjugate prior of a normal distribution with known mean and unknown variance. The chosen prior is defined as $p(\sigma^2) = f(\sigma^2; \nu = 1, \tau^2 = 10)$. With these choices, the posterior is a scaled inverse chi-squared distribution with parameters $\left(\nu + n; \frac{\nu\tau^2 + \sum_{i=1}^{n}(c_i - \mu)^2}{\nu + n}\right)$. The estimated value of the variance $\tilde{\sigma}^2$ is the one that maximizes the posterior, in other word the mode of the posterior, which is equal to:

$$\tilde{\sigma}^2 = \frac{\nu\tau^2 + \sum_{i=1}^{n}(c_i - \mu)^2}{n + \nu + 2}. \tag{3.4}$$

Only coverage values within the range $[70, 130]$ are considered for inference, range where it is assumed that data not coming from well-assembled sequences does not affect significantly the normal distribution. Finally, substituting the chosen parameters within equation (3.4), we obtain an estimated standard deviation of $\tilde{\sigma} = 10.3$. The estimated likelihood and the histogram for data within the range considered for inference are shown in Figure 3.3.

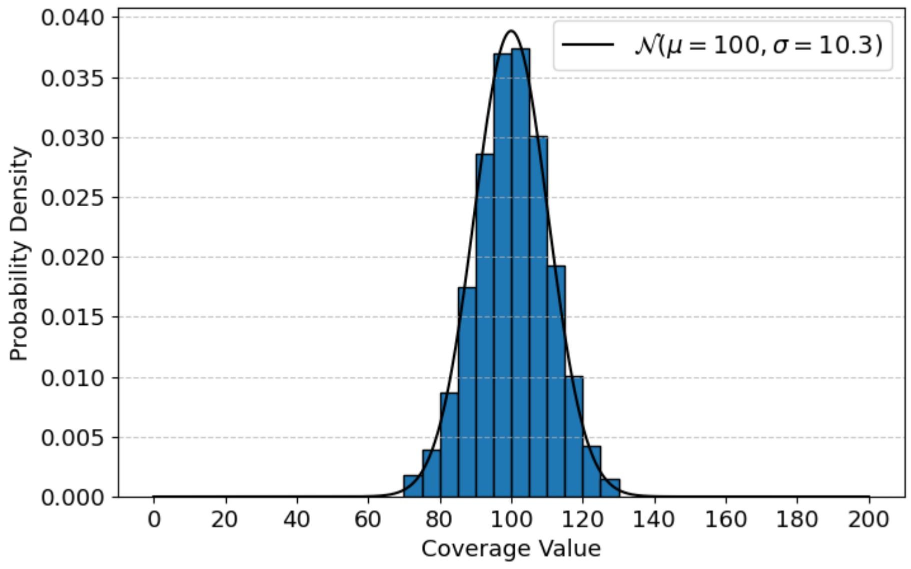


Figure 3.3: Histogram for coverage values in the interval $[70, 130]$ with bin size 5 and the corresponding inferred normal distribution $\mathcal{N}(\mu = 100, \sigma = 10.3)$. The histogram is normalized such that its total area is equal to 1, so that it can be compared to a PDF.

From the estimated PDF, we can state that a position belonging to a well-assembled sequence has a coverage value greater than $\mu - 2\tilde{\sigma} \simeq 80$ with probability of 97.4%. On the other hand, as we said before, it is very unlikely that a position belonging to an artifact sequence has a coverage value significantly greater than 0. Besides this, positions belonging to tails are assumed to be few with respect to the ones belonging to well-assembled sequences, and they have typically lower coverage values. Then it is reasonable, from now on, to set the coverage value threshold at 80 to define whether a position belongs to a well-assembled sequence or not.

For each position within the assembly contigs, there are two coverage values, one for each of the genomes used for the reads simulation. Therefore, for each position, there are three possible cases: both coverage values are greater than the threshold, one coverage value is greater and the other is less than the threshold, and both coverage values are less than the threshold. In the first case, the position belongs to a well-assembled region with respect to both the genomes, so that position actually exist within both of them. In the second case, the position only exists in one of the two genomes. Therefore, in the first two cases, we can say that the position comes from a well-assembled sequence. On the other hand, in the latter case we have a position that does not belong to either of the two genomes, so the position does not come from a well-assembled sequence. Having made these observations, we illustrate the process of assigning labels to each position of the assembly. Consider the two coverage sequences separately. If a coverage value is above the defined threshold of 80, it is very likely that the corresponding position belongs to a well-assembled DNA sequence. Then, for each coverage sequence, the labels are assigned position by position: label 1 if coverage is above the threshold, label 0 otherwise. Occasionally fluctuations in labels occur along the well-assembled sequence, that is, some positions have the label 0. The ideal case would be that well-assembled sequences were labeled continuously, without fluctuations, since the ultimate goal is to return clean and reasonably long sequences to the binner. Then, we first "smooth" the sequence of labels, i.e. if within a sequence of labels 1 there are short fluctuations of labels 0, we change them to 1. This is done using a sliding window of length 21. The most frequent class within the window is assigned to the considered position, which is the central one in the sliding window. After this operation, only sequences consecutively labeled as 1 longer than 300 are retained; otherwise, the labels are set to 0. Once obtained the sequence labels considering the two genomes, we need to compute an unique set of labels. There are two possible ways to label a position, one for genome. These labels consist of sequences of 1, longer than 300, and sequences of 0. If there is no overlap between sequences labeled as 1, then the final labels are the sequences themselves. If instead there is overlap, only the longest sequence of labels is retained. Recall that the goal is to improve the performance of the binner, so the longer the clean sequences are, the better is. This last step is represented in Figure 3.4.

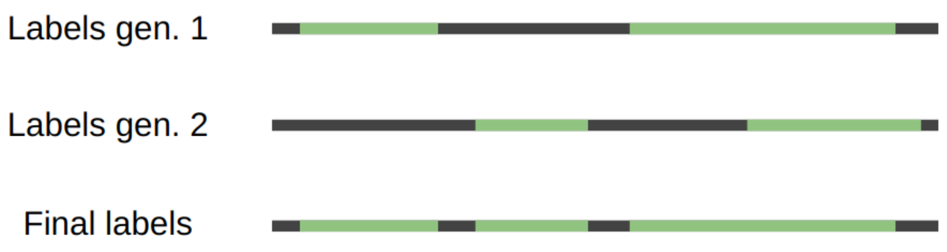


Figure 3.4: Visual representation of the final step to compute labels. The first two lines represent the labels calculated separately from the coverages of the two genomes. The last row represents the final labels. 1 labels are represented in green; 0 labels in black. In the last row, representing the final labels, we can see that the first sequence of labels 1 comes from the first genome while the second sequence comes from the second genome. For these two cases, there is no overlap between sequences labeled as 1 of the two genomes. On the other hand, the third sequence of labels 1 derives from an overlap. Since the one belonging to the first genome is the longest of the two overlapping sequences, this will be the final labeling.

To include the information that a position belongs to a well-assembled sequence tail, the label sequences are smoothed by applying a moving average with a sliding window of size 301. The average coverage value in the sliding window is assigned to the center position. With this definition of label, labels no longer represent binary classes but probabilities, in particular, the probability that a given position belongs to a well-assembled sequence. This last operation is performed to make the model better understand, through the loss computation, when a position belongs to a tail of a well-assembled sequence, distinguishing this situation from the one in which the position is instead found in the center of the sequence.

So far, labels are defined by combining labels assigned based on coverage relative to individual genome alignments. Another possible way to assign labels is to consider, for each position, the coverage values for each of the two alignments simultaneously. To do this, for each position, we calculate the average of the strictly positive coverages $\mathrm{PA}_i(c_{i,1}, c_{i,2})$, as defined in Equation 3.5, where $c_{i,1}$ and $c_{i,2}$ are the coverage values with respect to the two alignments at the position i.

$$\mathrm{PA}_i(c_{i,1}, c_{i,2}) = \begin{cases} \dfrac{c_{i,1} + c_{i,2}}{2} & \text{if } c_{i,1} > 0 \text{ and } c_{i,2} > 0 \\ c_{i,1} & \text{if } c_{i,1} > 0 \text{ and } c_{i,2} = 0 \\ c_{i,2} & \text{if } c_{i,2} > 0 \text{ and } c_{i,1} = 0 \\ 0 & \text{if } c_{i,2} = 0 \text{ and } c_{i,1} = 0 \end{cases} \tag{3.5}$$

After that, a threshold of 80 is applied to obtain the label for all positions. The same operations as before are then applied: fluctuations and short sequences are removed and the tails of the sequences labeled as 1 are smoothed with a moving average. This labeling method is computationally preferable to the first. The first method assigns labels based on the coverage of all genomes used to create an assembly that, in principle, can involve more than two genomes. The second method, instead, directly computes the average coverages, which is more efficient, especially if the work is extended to include a larger number of genomes for each assembly.

As an example, in Figure 3.5, the coverage values are shown relative to the position within an assembly contig for both alignments. In Figure 3.6, the corresponding labels for both proposed methods are displayed. From now on, we will refer to the two labeling methods as "heuristic" labels and "positive average" labels.

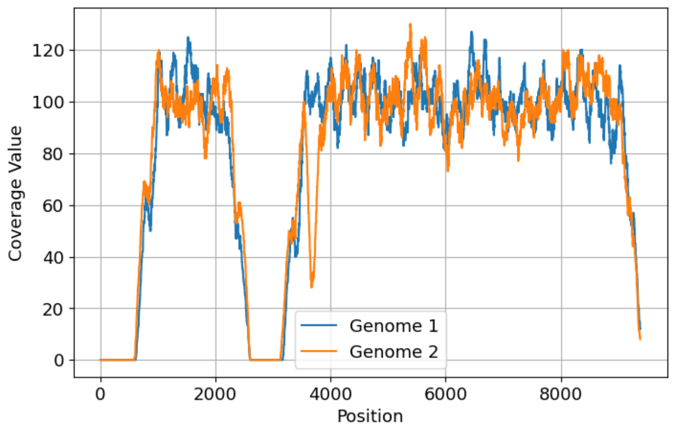


Figure 3.5: Example of coverage values for an assembly contig. The plot shows the coverage values at each position relative to both of the genomes used to simulate reads for assembly.

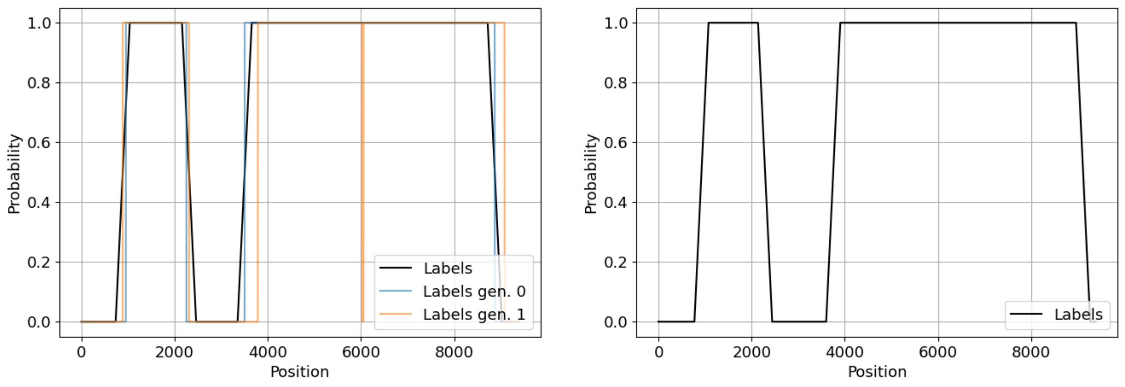


Figure 3.6: Example of an assembly contig labeling, whose coverage values are shown in Figure 3.5. On the left, the heuristic labeling is shown, where the two alignments are considered separately. The labels corresponding to the individual genomes are illustrated in blue and orange, while the final labels are shown in black. On the right is shown the positive average labeling, which is based on the average of the strictly positive coverage values.

3.3 Models

The models considered are *seq2pos* models, which take as input an encoded DNA sequence $\vec{x}_i$, where i represents the position within a given assembly contig at the center of the input sequence. The output of the model is a scalar value $h(\vec{x}_i)$ within the interval $[0, 1]$, representing the probability that the position i is part of a well-assembled sequence. The predicted label $C(\vec{x}_i)$ is then determined by applying a probability threshold of 0.5 to the model output, such that:

$$C(\vec{x}_i) = \begin{cases} 0 & \text{if } h(\vec{x}_i) < 0.5 \\ 1 & \text{if } h(\vec{x}_i) \geq 0.5. \end{cases} \tag{3.6}$$

The choice of this kind of models allows flexibility in the length of an assembly contig given as input, as classification is performed for each position. This fact is particularly useful since contig lengths are highly variable: they can range from 10^3 to 10^5 positions.

In order to choose the fixed length of the input sequence, we have to make some considerations. Including more positions in the input window gives the model more information. Therefore, the longer the sequence length, the greater the predictive ability of the model. On the other hand, in general, considering longer sequences requires more computational resources for both training and final predictions. Another consideration is about padding. Since assembly contigs are assumed to be part of longer sequences, i.e. genomes, complete ignorance of the bases before and after a contig is assumed. This is equivalent to assigning to these positions the same probability of having any base. Having longer windows corresponds to having a greater number of padded positions for windows close to contigs edges, thus having windows with greater uncertainty about the sequences they contain. Trying different settings, we choose a fixed input length of 501, which is a compromise between the previous considerations. The length is chosen odd, since we want the same amount of positions before and after the central one.

4

Training and Results

4.1 DNA Sequence Encoding

The first step in the processing of DNA sequences is selecting an encoding method. To do this, we implement *one-hot encoding*. In one-hot encoding, each nucleotide in the DNA sequence is represented by a binary vector, where each position in the vector corresponds to a specific nucleotide, and the vector consists of a single 1 in the position corresponding to the nucleotide and 0s elsewhere. Specifically, the nucleotides are encoded as follows:

$$\begin{aligned} \texttt{A} &\rightarrow [1, 0, 0, 0] \\ \texttt{C} &\rightarrow [0, 1, 0, 0] \\ \texttt{G} &\rightarrow [0, 0, 1, 0] \\ \texttt{T} &\rightarrow [0, 0, 0, 1]. \end{aligned}$$

However, DNA sequences often contain positions with uncertain nucleotides. These degenerate positions are represented using the IUPAC code [28]. Each of these symbols corresponds to uncertainty in the assignment of the nucleotide, so for each symbol, we assign an identical probability in the corresponding component of the one-hot encoding vector. The complete encoding is illustrated in Table 4.1.

Symbol	Base	Encoding
A	A	$[1, 0, 0, 0]$
C	C	$[0, 1, 0, 0]$
G	G	$[0, 0, 1, 0]$
T	T	$[0, 0, 0, 1]$
R	A or G	$[1/2, 0, 1/2, 0]$
Y	C or T	$[0, 1/2, 0, 1/2]$
S	G or C	$[0, 1/2, 1/2, 0]$
W	A or T	$[1/2, 0, 0, 1/2]$
K	G or T	$[0, 0, 1/2, 1/2]$
M	A or C	$[1/2, 1/2, 0, 0]$
B	Not A	$[0, 1/3, 1/3, 1/3]$
D	Not C	$[1/3, 0, 1/3, 1/3]$
H	Not G	$[1/3, 1/3, 0, 1/3]$
V	Not T	$[1/3, 1/3, 1/3, 0]$
N	Any nucleotides	$[1/4, 1/4, 1/4, 1/4]$

Table 4.1: IUPAC symbols, correspondig bases and encoding.

Padding values are set to represent complete uncertainty about bases and are then denoted by the symbol "N".

4.2 Model architectures

4.2.1 CNN model

The main advantages of using a CNN are its efficiency from a computational point of view and its ability to capture local patterns along a sequence. By concatenating several alternating CNN layers with maximum pooling layers, the model can capture increasingly global patterns, condensing sequence information into a shorter but deeper sequence. This condensed sequence is finally flattened and provided as input to a multilayer perceptron (MLP) for classification. The model considered is made up of 4 convolutional layers alternating with 3 max pooling layers, followed by an MLP with one hidden layer. The structure of the model used is described in detail in Table 4.2.

4.2.2 CNN-Transformer Encoder Model

The second model considered is a combination of a CNN and a transformer encoder, followed by the usual MLP for classification. The initial CNN aims to learn a representation of the sequence given as

Layer	Type	Input Size	Output Size	Kernel Size
1	1D CNN + MaxPool	501×4	247×16	8
2	1D CNN + MaxPool	247×16	120×32	8
3	1D CNN + MaxPool	120×32	57×64	7
4	1D CNN	57×64	51×128	7
5	Dense	6528	32	/
6	Dense	32	1	/

Table 4.2: Architecture of the Convolutional Neural Network (CNN) used for classification. The CNN layers considered have stride 1. The first five layers are followed by the ReLU activation function. The first three layers, immediately after the activation function, are followed by a max pooling layer (kernel size 2, stride 2). After the last layer, the sigmoid activation function is applied to produce an output value between 0 and 1. The total number of trainable parameters used in this architecture is 285489.

input, capturing local correlations. This representation of the sequence is then given as input to a transformer encoder, which using the self-attention mechanism takes into account long-range relationships within the sequence. The main advantage of the transformer compared to RNN, such as LSTM and GRU, is due to its well-known ability to encode long-range relationships. Therefore, after these two blocks, the input sequence is encoded in a representation that takes into account both local and global correlations. Finally, by flattening this representation vector, the MLP is used for classification. Similar work was done in [29]. In Table 4.3 is shown in detail the architecture used.

Layer	Type	Input Size	Output Size	Kernel Size
1	1D CNN + MaxPool	501×4	247×32	8
2	1D CNN + MaxPool	247×32	120×64	8
3	1D CNN + MaxPool	120×64	57×64	7
4	TransformerEncoder ($\times 4$)	57×64	57×64	/
5	Dense	3648	32	/
6	Dense	32	1	/

Table 4.3: Architecture of the CNN-Transformer Encoder used for classification. The convolutional layers considered have stride 1. The first 3 layers, immediately after the ReLU activation function, are followed by a max pooling layer (kernel size 2, stride 2). The transformer encoder layer consists of four identical stacked layers, each with the following specifications: an embedding dimension of 64, 4 attention heads, a feedforward network dimension of 1024, ReLU activation applied after the intermediate layer, and a dropout rate of 0.3. After the last layer, the sigmoid activation function is applied to produce an output value between 0 and 1. The total number of trainable parameters used in this architecture is 759265.

4.2.3 Benchmark model

In order to verify that our objective is both achievable and reasonable, and to evaluate whether the model can effectively learn from the data, it is important to establish a benchmark model. This benchmark is used to evaluate the performance of more sophisticated models and to confirm that the complexity we aim to incorporate is justified. The benchmark model we chose is simply the MLP used for classification applied directly to the flattened input sequence, as shown in Table 4.4.

Layer	Type	Input Size	Output Size
1	Dense	2004	32
2	Dense	32	1

Table 4.4: Simple MLP used as benchmark model. Biases are considered, ReLU activation function si applied after the first layer, sigmoid activation function is applied to produce an output value between 0 and 1 after the last layer. The total number of trainable parameters used in this architecture is 64161.

4.2.4 Models combination

In addition to the two models just described, we propose two possible ways of combining them, with the aim of obtaining a single model capable of exploiting the strengths of both and thus obtaining better performances.

The first combining method consists of simply averaging the two output probabilities. Let us define the input sequence corresponding to the position i of a given assembly contig as $\vec{x}_i$ and the output probabilities of the two models as $f(\vec{x}_i)$ and $g(\vec{x}_i)$ respectively. The final output of such models combination $h(\vec{x}_i)$ is

$$h(\vec{x}_i) = \frac{f(\vec{x}_i) + g(\vec{x}_i)}{2}. \tag{4.1}$$

The second combining method consists in building a simple meta-model that combines the two model outputs. This meta-model, given the input sequence $\vec{x}_i$, computes the output probabilities for both models and combines these two predictions by applying a dense layer with bias followed by a sigmoid activation function. This model has 3 trainable parameters, named w_1, w_2 and c. Therefore, the final output probability $h(\vec{x}_i)$ of such a model is

$$h(\vec{x}_i) = \sigma(w_1 f(\vec{x}_i) + w_2 g(\vec{x}_i) + c). \tag{4.2}$$

4.3 Training, Validation and Test set definition

To compare the models considered, it is necessary to define fixed datasets: one used for training, one for validation, and finally one to test the model ability to generalize outside the training set. Actually, due to how the dataset is structured, we can think of three levels of generalization capacity of a model: the ability to generalize on samples belonging to the same assembly files used for training, on different files but belonging to the same SGBs used for training, and different files belonging to different SGBs. For this reason, we divide the test set into three parts, respecting the hierarchy just described. Intuitively, we expect the model to generalize worse following the order of describing these 3 test datasets.

The procedure to define such datasets is the following: 25 SGBs are selected, from which, for each of them, 6 assembled files are selected. For each of these, 7680 windows are taken for training. From each of the same files, 3840 windows are taken for validation and the same number for testing. In total, we therefore used 1152000 windows for training and 288000 for both validation and testing. The windows used for validation and testing are chosen to have no overlap with the training windows. The test set just described corresponds to the first level of testing for the generalization ability of the model, i.e. whether it is able to generalize on samples belonging to the same assemblies used for training. The second test set, following the hierarchical order of generalization, is made up of assembly files not used for training but belonging to the same SGBs. Furthermore, the files were chosen in such a way that they were not assembled from the genomes used to assemble the files considered for training. The files that respect this last condition are 41. From each of these files, 7040 windows are selected, so the second test set is made up of 288640 windows. The third test set is constructed by selecting 26 SGBs not used for training and from each of these choosing 5 assembly files. For each of these files, 3840 windows are drawn, so the third test set is made up of a total of 499200 windows.

4.4 Model training and evaluation

The models are trained on the dataset defined earlier, utilizing binary cross-entropy as the loss function. Adam optimizer is selected with PyTorch default values. Batch gradient descent is used as the weight update mechanism, with data processed in batches of 64 samples.

To mitigate overfitting, several strategies are implemented: early stopping, a learning rate scheduler, and L2 regularization. As regards early stopping, the loss on the validation set calculated at the end of each epoch is used as a criterion. The criterion for stopping training is as follows: if the validation loss does not decrease within 30 epochs, the training is stopped. The difference between the best loss and the current one is used as metric: if this difference is greater than 10^{-4}, then the best loss is updated with the current one and the count of epochs without improvement is reset to zero. The learning rate scheduler consists of two phases. In the first phase, which is the warm-up phase, the learning rate increases linearly from 0 to 10^{-4} at each training step during the first 10 epochs. In the second phase, the learning rate decreases linearly over the next 90 epochs, reaching 0 by the end of the 100-th epoch. L2 regularization is applied with a weight decay set to 10^{-4}. After each epoch, the training dataset is shuffled to ensure that the model does not learn any unintended patterns from the order of the data, promoting better generalization and preventing overfitting.

The meta-model is trained on the training set after the training of the CNN and the CNN-Transformer Encoder have been completed. The training is performed without a learning rate scheduler, early stopping, and L2 regularization. The optimizer and loss function are the same as those used during the previous training. The fixed learning rate is set to 10^{-4}, and the training lasts for 5 epochs.

To evaluate the performance of different models, we calculate the area under the curve (AUC) and construct confusion matrices for each of the three test sets. AUC provides an overall measure of model performance and is preferred over accuracy because it is not affected by class imbalance. From the confusion matrices, we also compute four additional metrics: true positive rate (TPR), true negative rate (TNR), positive predicted value (PPV) and negative predicted value (NPV). TPR measures the ability of the models to identify well-assembled sequences, ensuring that most of them are preserved, which is a key factor in our goal. TNR evaluates the capacity of the models to detect misassembled sequences. PPV and NPV assess the accuracy of the models when predicting positive and negative cases, respectively.

The system used for training has 4 Intel Xeon Gold 6140 CPUs (144 threads total), large caches (L1: 2.3 MiB, L2: 72 MiB, L3: 99 MiB), and a 4-node NUMA configuration, designed for high-performance tasks. All models are defined using the PyTorch API, along with a data generator for efficient data feeding during training. Furthermore, the training process is designed to run also on a GPU with CUDA support for enhanced performance. The developed code for training and testing is available on GitHub[1].

4.5 Results and discussions

4.5.1 Heuristic labels

From Figure 4.1, which shows the training histories of the trained models, it is possible to note that the benchmark model triggers early stopping after a few epochs. This behavior suggests that the model struggles to generalize effectively in the validation set. In contrast, the other two models trigger early stopping after significantly more epochs and for a lower value of loss, indicating a stronger ability to generalize on the validation set compared to the benchmark model. This shows that the CNN and CNN-Transformer Encoder models can capture patterns in the data more effectively than the benchmark model. Furthermore, the CNN-Transformer Encoder model overfits quicker than the CNN model, i.e. during training early stopping occurs after fewer epochs.

[1]https://github.com/marco-chiloiro/Deep-learning-methodologies-to-approach-the-problem-of-misassembled-microbial-genomes

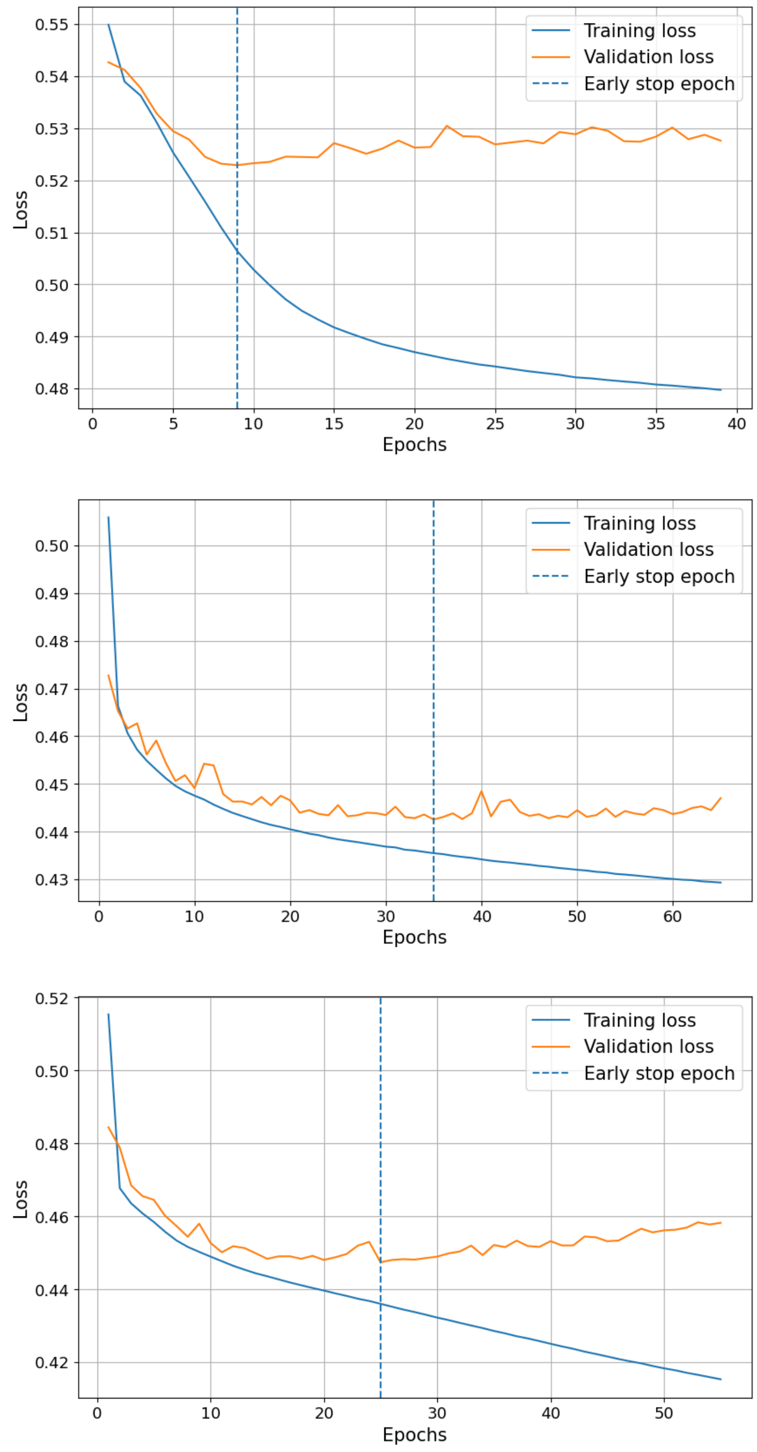


Figure 4.1: Training history plots considering the heuristic labeling. The top plot represents the benchmark model, the middle plot corresponds to the CNN model, and the bottom plot depicts the CNN-Transformer Encoder model.

In Table 4.5 are reported the results for all the 5 models considered on the 3 test set chosen. Finally, all the 15 confusion matrices are shown in Figure 4.2. From these results, it is possible to make several considerations. Firstly, it is noteworthy that even the benchmark model, despite its simplicity, is capable of learning from raw DNA sequences. This is evidenced by the fact that the AUC is significantly larger than 0.500. Furthermore, it is able to recognize most of the well-assembled sequences, since it obtains a TPR higher than 0.950 in all the test datasets. However, as indicated by the low TNR and NPV values, the benchmark model has a limited ability to recognize misassembled sequences and a low accuracy in predicting them. On the other hand, the two considered models, i.e. the CNN and CNN-Transformer Encoder models, demonstrate significantly improved classification performance, as all the considered evaluation metrics show higher values. In particular, these models exhibit a TNR value that is almost double that of the benchmark model, demonstrating a better ability to recognize misassembled sequences. For these models, the NPV values are comparable to the PPV values, both around 0.800, indicating a good accuracy in classifying both positive and negative classes. Furthermore, these models are more conservative than the benchmark model, as evidenced by their higher TPR values. Overall, the CNN model performs slightly better than the CNN-Transformer Encoder model, while maintaining comparable performance.

Model	**AUC**	**TPR**	**TNR**	**PPV**	**NPV**
Test set 1					
Benchmark	0.673	0.965	0.221	0.805	0.654
CNN	0.775	0.975	0.409	0.845	0.833
CNN-TransformerEncoder	0.764	0.971	0.405	0.844	0.810
Average model	0.780	0.980	0.397	0.843	0.860
Meta-model	0.777	0.977	0.403	0.845	0.838
Test set 2					
Benchmark	0.680	0.962	0.234	0.833	0.610
CNN	0.763	0.971	0.428	0.871	0.789
CNN-TransformerEncoder	0.758	0.965	0.428	0.870	0.754
Average model	0.774	0.977	0.418	0.870	0.824
Meta-model	0.767	0.971	0.428	0.871	0.789
Test set 3					
Benchmark	0.657	0.965	0.215	0.806	0.645
CNN	0.714	0.988	0.368	0.841	0.903
CNN-TransformerEncoder	0.713	0.981	0.373	0.841	0.850
Average model	0.720	0.990	0.364	0.840	0.912
Meta-model	0.719	0.986	0.368	0.841	0.884

Table 4.5: Area Under the Curve (AUC), True Positive Rate (TPR), True Negative Rate (TNR), Positive Predicted Value (PPV) and Negative Predicted Value (NPV) for all models evaluated on the 3 test set, considering heuristic labeling.

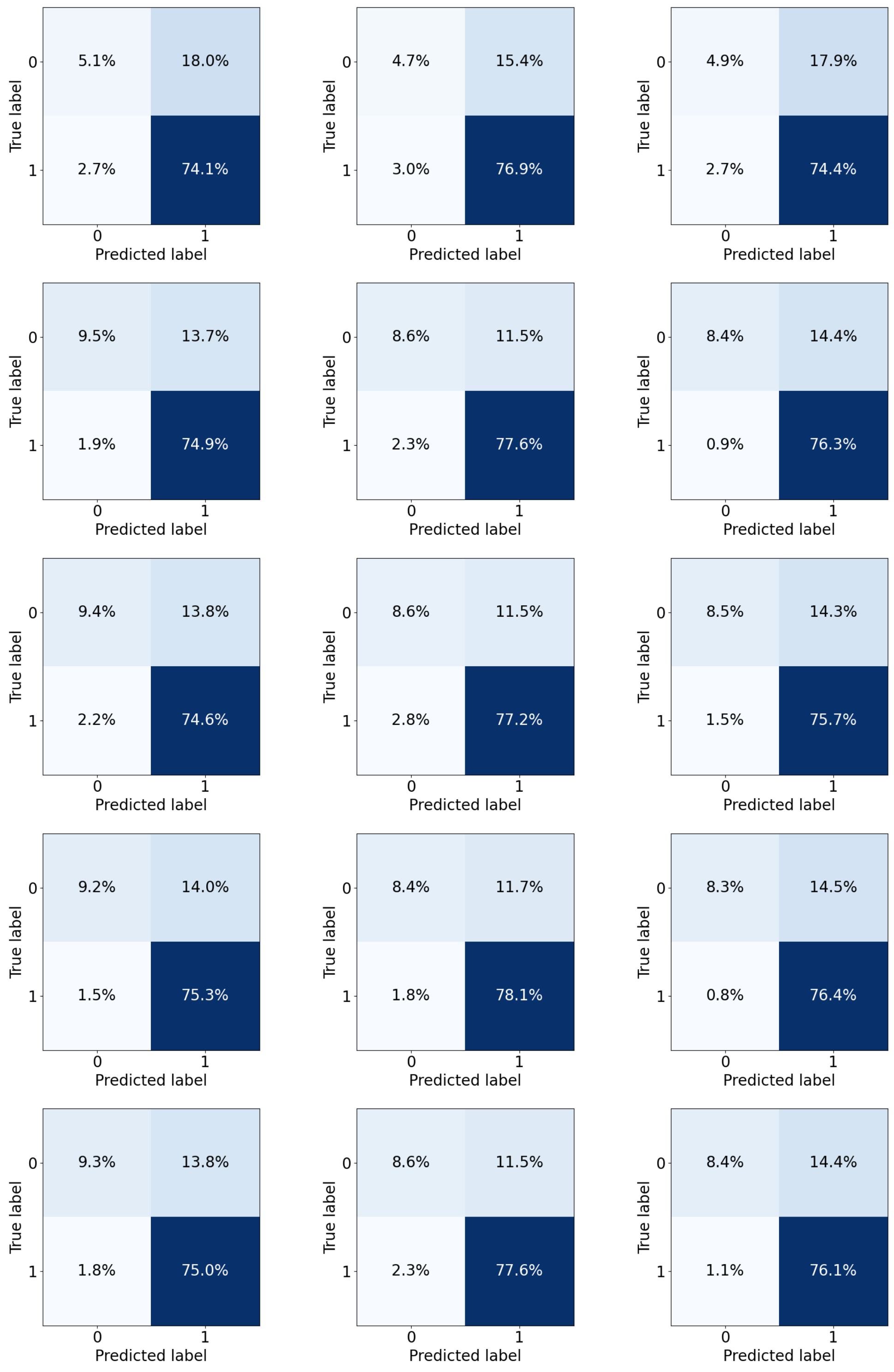


Figure 4.2: Confusion matrices for all models considering the heuristic labeling. Rows correspond to the models in the following order: Benchmark, CNN, CNN-TransformerEncoder, average, and meta- model. Columns represent the test sets in the order: test set 1, 2, and 3.

As can be seen from the results, both models combinations show better performance than the models considered individually, highlighting slightly higher AUC values in all test datasets. Regarding the meta-model training, the three learned parameters are $w_1 = 1.99$, $w_2 = 3.60$ and $c = -2.89$, indicating that the meta-model gives more weight to the CNN-Transformer Encoder model. Nevertheless, the model combination based on the average of the predictions, as described in Equation 4.1, performs slightly better. Furthermore, this latter model has a lower TNR value and a higher NPV value than those of the single models, demonstrating its ability to predict a slightly smaller fraction of misassembled sequences but with greater accuracy.

As for performance in the three different test sets, as one might expect, all models perform best in the first test set, with progressively worse performance on the second and third test sets. This trend is consistent with the increasing generality and externality of the data in these test sets compared to the training dataset.

4.5.2 Positive average labels

The training histories for all the models considered are shown in Figure 4.3. In Table 4.6 are reported the results for all the 5 models considered on the 3 test set chosen. All the 15 confusion matrices are shown in Figure 4.4. Regarding the training of the meta-model, the three parameters that the model learned are $w_1 = 1.79$, $w_2 = 4.02$ and $c = -3.11$. The considerations on the results obtained with this labeling method remain the same as those discussed in the previous subsection, indicating that the two labeling methods produce similar results in terms of label assignment.

Model	**AUC**	**TPR**	**TNR**	**PPV**	**NPV**
Test set 1					
Benchmark	0.692	0.956	0.254	0.813	0.630
CNN	0.805	0.974	0.437	0.853	0.833
CNN-TransformerEncoder	0.797	0.966	0.441	0.853	0.795
Average model	0.812	0.978	0.430	0.853	0.852
Meta-model	0.810	0.968	0.450	0.856	0.805
Test set 2					
Benchmark	0.670	0.954	0.274	0.843	0.593
CNN	0.802	0.968	0.469	0.882	0.780
CNN-TransformerEncoder	0.794	0.960	0.469	0.881	0.742
Average model	0.808	0.973	0.459	0.880	0.804
Meta-model	0.805	0.960	0.477	0.882	0.746
Test set 3					
Benchmark	0.671	0.956	0.247	0.816	0.618
CNN	0.737	0.988	0.388	0.848	0.906
CNN-TransformerEncoder	0.729	0.977	0.397	0.849	0.832
Average model	0.740	0.988	0.386	0.848	0.905
Meta-model	0.738	0.979	0.393	0.848	0.846

Table 4.6: Area Under the Curve (AUC), True Positive Rate (TPR), True Negative Rate (TNR), Positive Predicted Value (PPV) and Negative Predicted Value (NPV) for all models evaluated on the 3 test sets, considering positive average labeling.

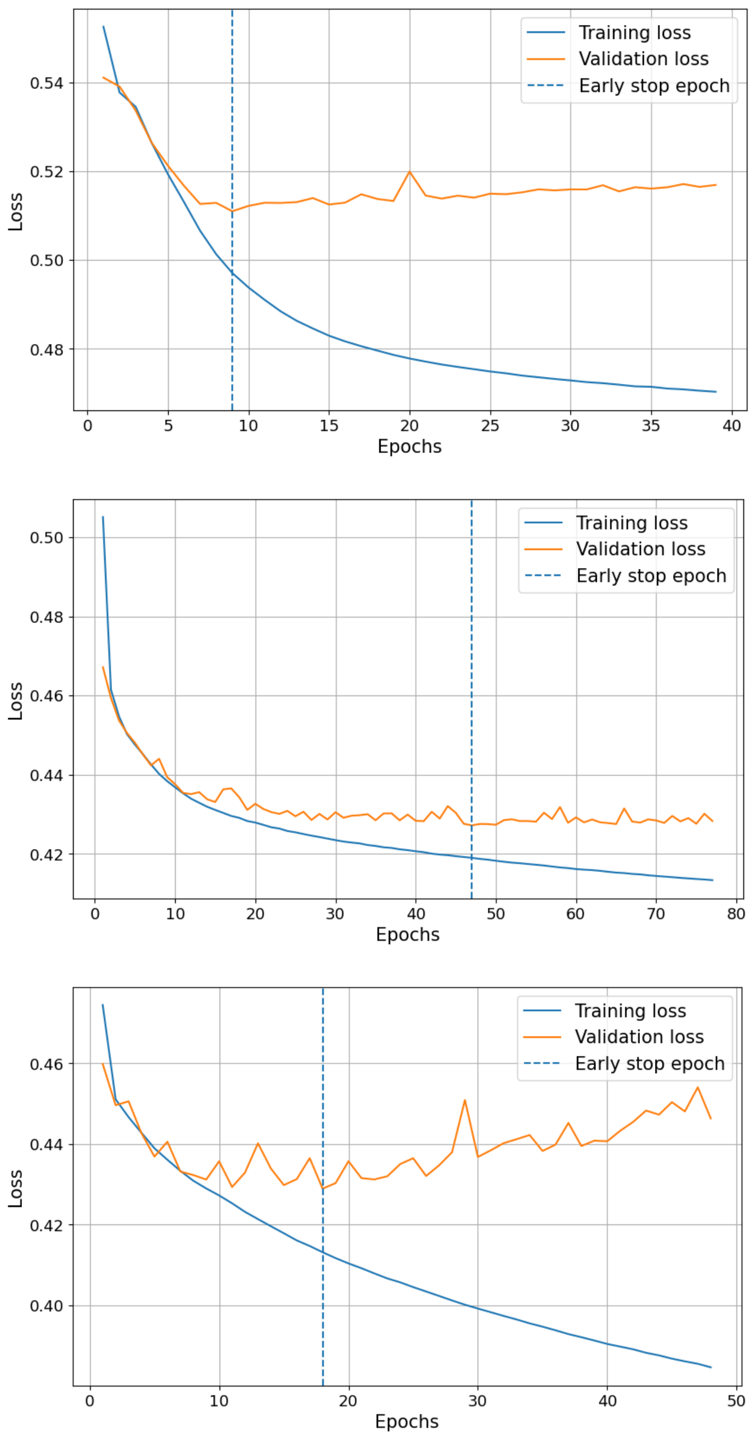


Figure 4.3: Training history plots considering the positive average labeling. The top plot represents the benchmark model, the middle plot corresponds to the CNN model, and the bottom plot depicts the CNN-Transformer Encoder model.

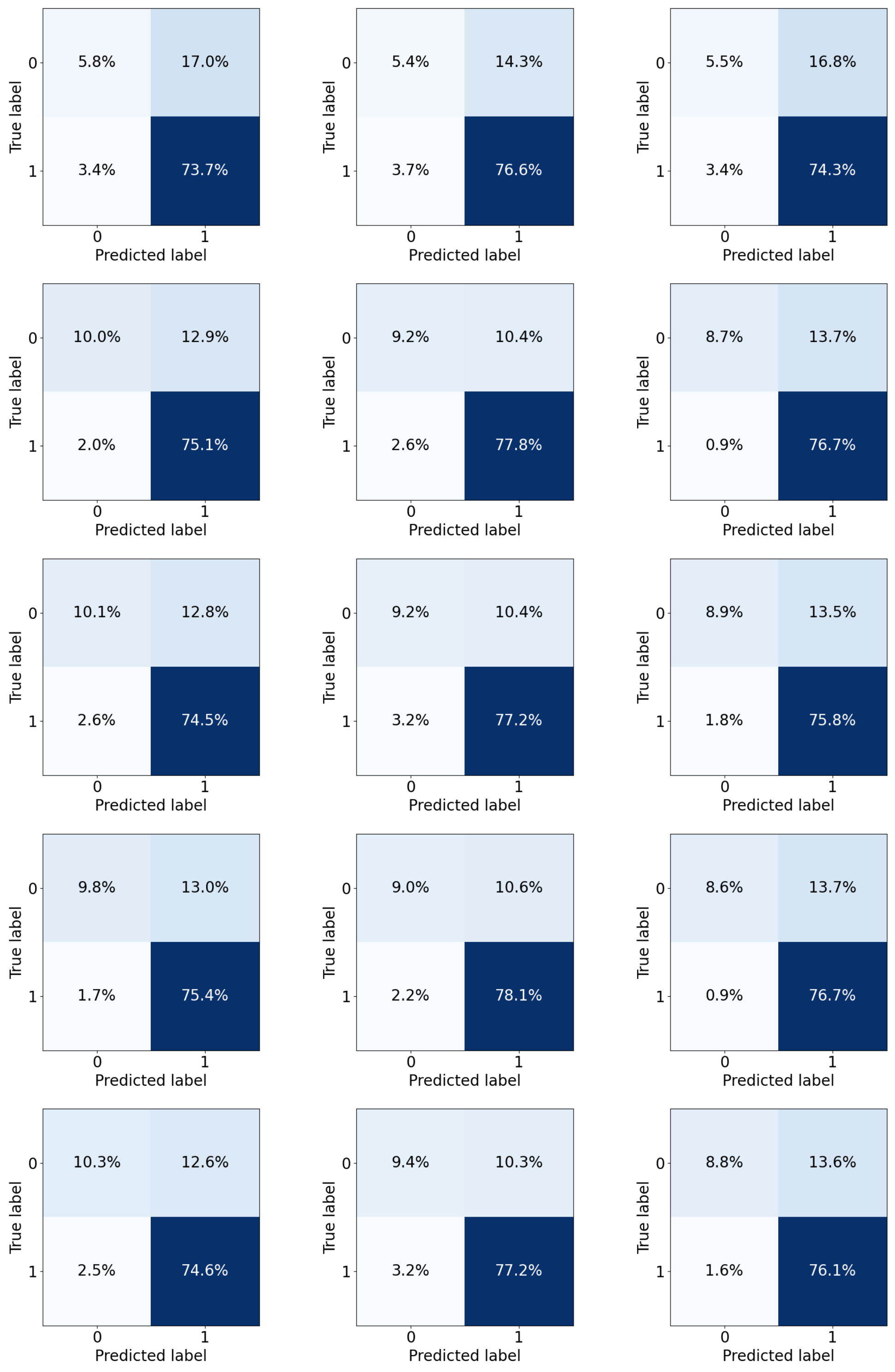


Figure 4.4: Confusion matrices for all models considering the positive average labeling. Rows correspond to the models in the following order: Benchmark, CNN, CNN-TransformerEncoder, average, and meta- model. Columns represent the test sets in the order: test set 1, 2, and 3.

5 Conclusion

This thesis aims to develop a reference-free assembly assessment tool designed to identify misassembled sequences directly from raw DNA sequences. Special attention is paid to assembly errors caused by repeats, such as in the case of chimeric assemblies, which are simulated by assembling synthetic reads derived from similar genomes.

The results demonstrate that deep learning models can effectively learn from raw DNA sequences for this specific task. The benchmark model achieves an AUC score greater than 0.650 on all test datasets, demonstrating its learning ability. The two models, based on CNN and transformer encoder architectures, show improved performance, particularly in their ability to identify misassembled sequences with greater accuracy. The model combination approach which averages the predicted probabilities of individual models is more effective than single models. This ensemble method not only increases the overall AUC score but also enhances the accuracy of identifying misassembled sequences. As expected, the performance of all models decreases when tested on external and more generalized datasets than the training set. Finally, the two labeling methods considered result almost equivalent from classification performance point of view, making the positive average labeling preferable over the heuristic labeling given its computationally efficiency.

This work demonstrates that, using appropriate deep learning models, it is possible to identify a portion of the misassembled regions within contigs with reasonable accuracy, while preserving most of the well-assembled ones. In fact, the models considered exhibit conservative behavior, identifying most of the well-assembled sequences. However, they struggle in detecting all misassembled sequences, although they still achieve a reasonable level of accuracy in doing so. This balance is consistent with our goals, prioritizing reliable, even if not exhaustive, identification of misassembled sequences, while adopting a conservative approach to well-assembled sequences. This approach, with further refinement, has significant potential in the field of metagenomics. It could improve the binning quality by reliably excluding some assembly errors.

While this work represents a promising initial step, it is not intended as a definitive tool. The training was performed on a dataset limited in scope, focusing on a specific type of assembly error, and the synthetic nature of the training data may not fully capture the complexity of real-world samples. Additionally, the nature of metagenomic data makes training deep learning models computationally intensive, both in terms of time and resources. As a result, both the complexity of the models and the volume of training data had to be reduced to achieve meaningful results for the purposes of this work, ultimately limiting their generalization capabilities and overall performance.

Despite these limitations, the results are encouraging and there are many possibilities for further development and improvement. More complex datasets, such as contigs involving the assembly of more than two genomes from the same SGB, could be synthesized to better reflect real-world metagenomic challenges. Furthermore, by increasing the volume of training data and using more powerful hardware and optimized code, it would be possible to improve the complexity of the models. This would also allow for the exploration of alternative architectures and model ensembles, which have been shown to have the potential for better generalization and classification performance. With access to additional computing resources, a more robust hyperparameter optimization process, such as Bayesian optimization, could be implemented to fine-tune the model performance. Further testing could also be conducted using increasingly realistic datasets. For example, metagenomic samples could be simulated by combining multiple genomes in realistic proportions and assembling them together, resulting in a more accurate representation of real-world scenarios. Finally, the performance of the tool could be compared with existing methods to assess its effectiveness and potential advantages.

This study highlights the potential of deep learning to address critical challenges in metagenomic assembly. With further refinement and exploration of the proposed directions, this approach could evolve into a powerful tool, providing a robust framework to reliably identify misassembled sequences while preserving well-assembled ones. This tool has the potential to significantly improve the quality of MAGs and, consequently, the correctness of downstream processes such as binning and taxonomic classification.

References

[1] B. Ma, C. Lu, Y. Wang, J. Yu, K. Zhao, X. Ran, H. Ren, X. Lv, R. Pan, J. Zhang, Y. Zhu, and J. xu, "A genomic catalogue of soil microbiomes boosts mining of biodiversity and genetic resources," *Nature Communications*, vol. 14, 11 2023.

[2] R. Daniel, "The metagenomics of soil," *Nature reviews microbiology*, vol. 3, no. 6, pp. 470–478, 2005.

[3] S. Sunagawa, S. Acinas, P. Bork, C. Bowler, D. Eveillard, G. Gorsky, L. Guidi, D. Iudicone, E. Karsenti, L. Fabien, H. Ogata, S. Pesant, M. Sullivan, P. Wincker, and C. de Vargas, "Tara oceans: towards global ocean ecosystems biology," *Nature Reviews Microbiology*, vol. 18, 05 2020.

[4] E. Pasolli, F. Asnicar, S. Manara, M. Zolfo, N. Karcher, F. Armanini, F. Beghini, P. Manghi, A. Tett, P. Ghensi, M. C. Collado, B. L. Rice, C. DuLong, X. C. Morgan, C. D. Golden, C. Quince, C. Huttenhower, and N. Segata, "Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle," *Cell*, vol. 176, no. 3, pp. 649–662.e20, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867419300017

[5] N. Carlino, A. Blanco-Míguez, M. Punčochář, C. Mengoni, F. Pinto, A. Tatti, P. Manghi, F. Armanini, M. Avagliano, C. Barcenilla, S. Breselge, R. Cabrera-Rubio, I. Calvete-Torre, M. Coakley, J. Cobo Díaz, F. De Filippis, H. Dey, J. Leech, E. Klaassens, and G. Hermes, "Unexplored microbial diversity from 2,500 food metagenomes and links with the human microbiome," *Cell*, vol. 187, pp. 1–21, 08 2024.

[6] D. Li, C.-M. Liu, R. Luo, K. Sadakane, and T.-W. Lam, "Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph," *Bioinformatics*, vol. 31, no. 10, pp. 1674–1676, 01 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv033

[7] D. D. Kang, J. Froula, R. Egan, and Z. Wang, "Metabat: an efficient tool for accurately reconstructing single genomes from complex microbial communities," *PeerJ*, vol. 3, p. e1165, 2015.

[8] D. Parks, "Checkm: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes," *PeerJ PrePrints*, vol. 2, 10 2014.

[9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, 05 2015.

[10] E. L. Karin and M. Steinegger, “Cutting edge deep-learning based tools for metagenomic research,” *National Science Review*, p. nwaf056, 02 2025. [Online]. Available: https://doi.org/10.1093/nsr/nwaf056

[11] G. Roy, E. Prifti, E. Belda, and J.-D. Zucker, “Deep learning methods in metagenomics: a review,” *bioRxiv*, 2023.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[13] A. O. Abdelkareem, M. I. Khalil, M. Elaraby, H. Abbas, and A. H. A. Elbehery, “Virnet: Deep attention model for viral reads identification,” in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, 2018, pp. 623–626.

[14] Y. Zhang, C. Li, H. Feng, and D. Zhu, “Dlmeta: a deep learning method for metagenomic identification,” in *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2022, pp. 303–308.

[15] A. Mikheenko, V. Saveliev, and A. Gurevich, “Metaquast: evaluation of metagenome assemblies,” *Bioinformatics*, vol. 32, no. 7, pp. 1088–1090, 11 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv697

[16] S. C. Clark, R. Egan, P. I. Frazier, and Z. Wang, “Ale: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies,” *Bioinformatics*, vol. 29, no. 4, pp. 435–443, 01 2013. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts723

[17] N. D. Olson, T. J. Treangen, C. M. Hill, V. Cepeda-Espinoza, J. Ghurye, S. Koren, and M. Pop, “Metagenomic assembly through the lens of validation: recent advances in assessing and improving the quality of genomes assembled from metagenomes,” *Briefings in Bioinformatics*, vol. 20, no. 4, pp. 1140–1150, 08 2017. [Online]. Available: https://doi.org/10.1093/bib/bbx098

[18] S. Lai, S. Pan, L. P. Coelho, W.-H. Chen, and X.-M. Zhao, “metamic: reference-free misassembly identification and correction of de novo metagenomic assemblies,” *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/06/23/2021.06.22.449514

[19] M. Kuhring, P. Dabrowski, V. Piro, A. Nitsche, and B. Renard, "Surankco: Supervised ranking of contigs in de novo assemblies," *BMC bioinformatics*, vol. 16, p. 240, 07 2015.

[20] O. Mineeva, M. Rojas-Carulla, R. E. Ley, B. Schölkopf, and N. D. Youngblut, "Deepmased: evaluating the quality of metagenomic assemblies," *Bioinformatics*, vol. 36, no. 10, pp. 3011–3017, 02 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa124

[21] O. Mineeva, D. Danciu, B. Schölkopf, R. E. Ley, G. Rätsch, and N. D. Youngblut, "Resmico: Increasing the quality of metagenome-assembled genomes with deep learning," *PLOS Computational Biology*, vol. 19, no. 5, pp. 1–20, 05 2023. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1011001

[22] Y. Ding, J. Xiao, B. Zou, C. Yang, and L. Zhang, "Deepmm: Identify and correct metagenome misassemblies with deep learning," *bioRxiv*, 2025. [Online]. Available: https://www.biorxiv.org/content/early/2025/02/13/2025.02.07.637187

[23] E. Pasolli, F. Asnicar, S. Manara, M. Zolfo, N. Karcher, F. Armanini, F. Beghini, P. Manghi, A. Tett, P. Ghensi, M. C. Collado, B. L. Rice, C. DuLong, X. C. Morgan, C. D. Golden, C. Quince, C. Huttenhower, and N. Segata, "Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle," *Cell*, vol. 176, no. 3, pp. 649–662.e20, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867419300017

[24] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016. [Online]. Available: https://arxiv.org/abs/1607.06450

[25] E. Pasolli, F. Asnicar, S. Manara, M. Zolfo, N. Karcher, F. Armanini, F. Beghini, P. Manghi, A. Tett, P. Ghensi, M. C. Collado, B. L. Rice, C. DuLong, X. C. Morgan, C. D. Golden, C. Quince, C. Huttenhower, and N. Segata, "Extensive unexplored human microbiome diversity revealed by over 150,000 genomes from metagenomes spanning age, geography, and lifestyle," *Cell*, vol. 176, no. 3, pp. 649–662.e20, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867419300017

[26] B. A. Fritz, J. Hofmann, and et al., "Camisim: Simulating metagenomes and microbial communities," *Microbiome*, vol. 7, p. 17, 2019.

[27] B. Langmead and S. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, pp. 357–359, 2012.

[28] A. Cornish-Bowden, "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984," *Nucleic Acids Research*, vol. 13, no. 9, pp. 3021–3030, May 1985.

[29] S. He, B. Gao, R. Sabnis, and Q. Sun, “Nucleic transformer: Deep learning on nucleic acids with self-attention and convolutions,” *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/07/26/2021.01.28.428629

Acknowledgments

I thank my family and friends for their constant support. A special thanks to Paolo Manghi for his guidance and the opportunity to work on this project. I am grateful to the University of Trento for the resources provided and to the University of Padua and the Physics of Data professors for an incredible academic journey.