UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

MASTER THESIS No. 2472

# DETECTION OF MODIFIED NUCLEOTIDES USING NANOPORE SEQUENCING AND DEEP LEARNING METHODS

Sanja Deur

Zagreb, June 2021

UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

MASTER THESIS No. 2472

# DETECTION OF MODIFIED NUCLEOTIDES USING NANOPORE SEQUENCING AND DEEP LEARNING METHODS

Sanja Deur

Zagreb, June 2021

**UNIVERSITY OF ZAGREB**
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

Zagreb, 12 March 2021

# MASTER THESIS ASSIGNMENT No. 2472

| | |
|---|---|
| Student: | **Sanja Deur (0036498773)** |
| Study: | Computing |
| Profile: | Computer Science |
| Mentor: | prof. Mile Šikić |

Title: **Detection of Modified Nucleotides Using Nanopore Sequencing and Deep Learning Methods**

Description:

Nanopore sequencing is one of the state-of-the-art sequencing technologies. Passage of DNA through a pore changes its ionic current. Due to the pores' size, there are usually five nucleotides (5-mer) in the pore, influencing the measured signal. Each 5-mer produces different signals, and this information is used for basecalling (converting the raw signal to a sequence of nucleotides). The signal is approximately rectangular. The goal of the thesis is the development of a method for the detection of modifications of canonical nucleotides (A, C, T, G) such as 5mC methylations using DNA nanopore sequencing and deep learning methods. For evaluation of the results, use publicly available datasets. The solution should be implemented in Python with the PyTorch or similar computational library. The source code should be documented using comments and should follow the Google Python Style Guide when possible. The complete application should be hosted on GitHub under an OSI approved license.

Submission date: 28 June 2021

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 12. ožujka 2021.

# DIPLOMSKI ZADATAK br. 2472

| | |
|---|---|
| Pristupnica: | **Sanja Deur (0036498773)** |
| Studij: | Računarstvo |
| Profil: | Računarska znanost |
| Mentor: | prof. dr. sc. Mile Šikić |

Zadatak: **Određivanje modificiranih nukleotida koristeći sekvenciranje nanoporama i duboko učenje**

Opis zadatka:

Sekvenciranje nanoporama je jedna od vodećih tehnologija sekvenciranja danas. Prolaskom DNA kroz poru mijenja se ionska struja. Uslijed veličine pore, obično se u pori nalazi 5 nukleotida (5-torka) koji utječu na mjereni signal. Svaka 5-torka uzrokuje različit signal i ta informacija se koristi za pretvaranje sirovog signala u slijed nukleotida. Oblik signala je približno pravokutan. Cilj rada je razvoj nove metode za detekciju modifikacija kanonskih nukleotida (A, C, T, G) kao što je 5mC metilacija korištenjem DNA sekvenciranja nanoporama i metoda dubokog učenja. Za evaluaciju koristiti javno dostupne skupove podataka. Rješenje je potrebno implementirati u programskom jeziku Python koristeći PyTorch ili sličnu biblioteku za matrični izračun. Izvorni kod je potrebno dokumentirati koristeći komentare i razvijati prema Google Python Style Guide kada je to moguće. Cijeli programski proizvod potrebno je postaviti na GitHub pod jednom od OSI odobrenih licenci.

Rok za predaju rada: 28. lipnja 2021.

*Foremost, I am profoundly grateful to my mentor, Professor Mile Šikić, for his guidance, encouragement, and sharing his expertise with me over the past few years. Further on, I would like to thank my supervisor, Dominik Stanojević, for providing useful advice and thoughtful comments regarding this thesis. Finally, I am thankful to my family and friends, especially my parents, for their unconditional love and support throughout this entire process.*

# CONTENTS

# 1. Introduction

In recent years, scientists are increasingly aware of the importance of the the field of epigenetics, with more and more substantial advances being made. Epigenetics, translated from Greek as "over the genome", studies heritable changes caused by the activation and deactivation of genes without altering the underlying DNA sequence (Elnitski, 2021). One's epigenetics change with age, both as part of normal development and in response to one's behaviors (e.g. dietary options) and environment (e.g. exposure to pollutants) (CDC, 2020).

DNA methylation, specifically 5-Methylcytosine (5mC) which is the most abundant and biologically relevant type, is a normally occurring epigenetic modification (KGaA, 2021). However, both hypermethylation and hypomethylation of 5mC at CpG dinucleotides have been associated with various illnesses and health conditions, such as tumors of all types, which was first confirmed to occur in human cancer in 1983 (Weinhold, 2006). Therefore, it is of great importance to conduct further research about DNA modifications in order to improve understanding of cellular functions and to devise appropriate therapeutic tools.

Lately, various tools for the detection of CpG methylation, dependent on the Oxford Nanopore Technologies (ONT) sequencing, have been developed. Nanopore sequencing is a state-of-the-art sequencing technology which detects different electrical current signals for different canonical nucleotides (A, C, T, G), whilst a DNA or RNA strand passes through a nanopore. Then, basecalling, the process of translating this detected signal into a DNA sequence, is performed (Wick et al., 2019). In addition, Nanopore sequencing can be utilised for DNA modification detection, since signal shapes of modified nucleobases differ from the unmodified ones (Yuen et al., 2020).

The aim of this thesis is the development of a deep learning method for DNA base modification detection. Rockfish, one of the state-of-the-art methods based on a transformer architecture, is the backbone of the method developed in this work. Tombo's re-squiggle algorithm is a bottleneck in the Rockfish pipeline. Therefore, the main idea is to replace Tombo with another faster tool whose task is to remap signal points

at indels, a type of genetic variation in which a specific nucleotide sequence is present (insertion) or absent (deletion), hence correcting basecalling errors. The tool has been named Remapper, and it consists of two different approaches to resolving indels, later on compared with original Rockfish implementation, as well as to each other.

The contents of this thesis are organised in the following manner. Chapter 2 gives further details on biological concepts relevant to this work, as well as an overview of prior work done in the field of DNA modifications, concentrating on the aforementioned Rockfish implementation. Furthermore, Chapter 3 includes a brief description of utilised datasets and accompanying data analysis. In Chapter 4 the most important methods used in the implementation are thoroughly described and illustrated. Chapter 5 outlines the implementation details, external dependencies, overall code structure, and training procedure of the deep model. Chapter 6 presents comparison of results obtained by this implementation versus the original Rockfish implementation, offers a short discussion, and thoughts on possible future improvements. Finally, Chapter 7 states the conclusion of this thesis.

# 2. Background

This chapter contains biological background required for better understanding of the topic of this thesis. Next, an overview of the most significant efforts in this field is provided. At the end, goal of this thesis is formulated, alongside the differences in approach from prior related work.

## 2.1. Biological background

Epigenetics, the study of heritable phenotypic changes which do not involve alterations of the DNA sequence, plays a key role in gene activity and expression (Dupont et al., 2009). Gene expression indicates when and how often proteins are produced from the instructions within the genes. While genetic changes can alter which protein is created, epigenetic changes decide whether the gene is expressed ("turned on") or silenced ("turned off"). Types of epigenetic changes, that can interact with each other, comprise DNA methylation, histone modification, and non-coding RNA-associated silencing (CDC, 2020). DNA methylation, arguably the best known epigenetic process, is going to be the topic of interest in the proceeding text.

DNA methylations, including 5-methylcytosine (5mC), 5-hydroxymethylcytosine (5hmC), and N6-methyldeoxyadenosine (6mA), are introduced into a DNA molecule by adding methyl or hydroxymethyl groups to nucleotides. 5mC methylation, one of the most widespread and biologically relevant genomic modifications, is introduced by biochemical addition of a methyl group ($–CH_3$) at the fifth position of the pyrimidine ring of cytosines, as shown in Figure 2.1 (Liu et al., 2019a).

5mC is enriched at CpG sites, regions in which a cytosine nucleotide (C) is linked to a guanine nucleotide (G) by a phosphodiester bond (p). In mammals, the majority of CpG cytosines are methylated. Nonetheless, CpG islands, promoter regions of DNA which have higher concentrations of CpG sites, are free of methylation in normal cells. Conversely, in cancer cells these CpG islands are overly methylated, thus silencing the tumor-suppressor genes that should normally be expressed. This epigenetic abnor-

mality happens early in the development of a tumor, therefore epigenetics can help to detect cancers early on (Simmons, 2008).

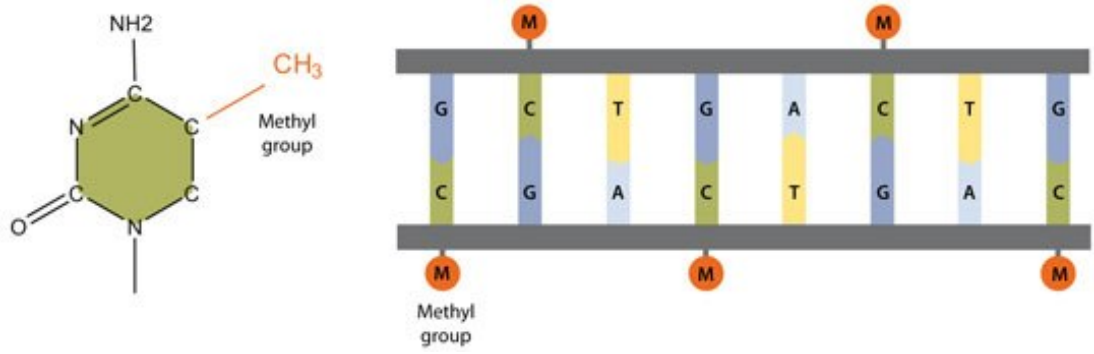


**Figure 2.1:** DNA methylation (KGaA, 2021)

Both hypermethylation and hypomethylation of 5mC at CpG dinucleotides have been shown to be associated with diseases. Except for tumors of almost all types, cognitive dysfunction, and respiratory, cardiovascular, reproductive, autoimmune, and neurobehavioral illnesses are linked with epigenetic mechanisms (Weinhold, 2006). Epigenetics change as one ages, both as part of normal development and aging, as well as in response to one's behaviours (e.g. lifestyle and dietary exposures) and environment (e.g. different pollutants) (CDC, 2020).

Whole-genome bisulfite sequencing (BS-seq), the traditional method of 5mC detection, converts cytosine to uracil, whereas 5mC is not influenced, therefore modified and unmodified cytosine can be differentiated. Disadvantages of BS-seq include inability to evaluate repetitive genomic regions by short-read sequencing (Liu et al., 2019a), DNA degradation, and sensitivity to the reaction conditions (Yuen et al., 2020).

Oxford Nanopore Technologies (ONT) long-read sequencing is one of the state-of-the-art sequencing technologies, which resolves obstacles present in previously mentioned bisulfite sequencing. As depicted in Figure 2.2 a DNA or RNA strand passes through a membrane via a nanopore and changes its ionic current. Detected electrical current signal, called the "squiggle", is the raw data obtained by an ONT sequencer. Then, the basecalling of raw data takes place, converting the raw Nanopore signals into the corresponding nucleotide sequence (Wick et al., 2019).

Moreover, it has been found that Nanopore sequencing can be used to detect DNA methylation (Ni et al., 2019), since the base modifications can be detected by their unique signal shapes, which differ from the equivalent unmodified base (Yuen et al., 2020).

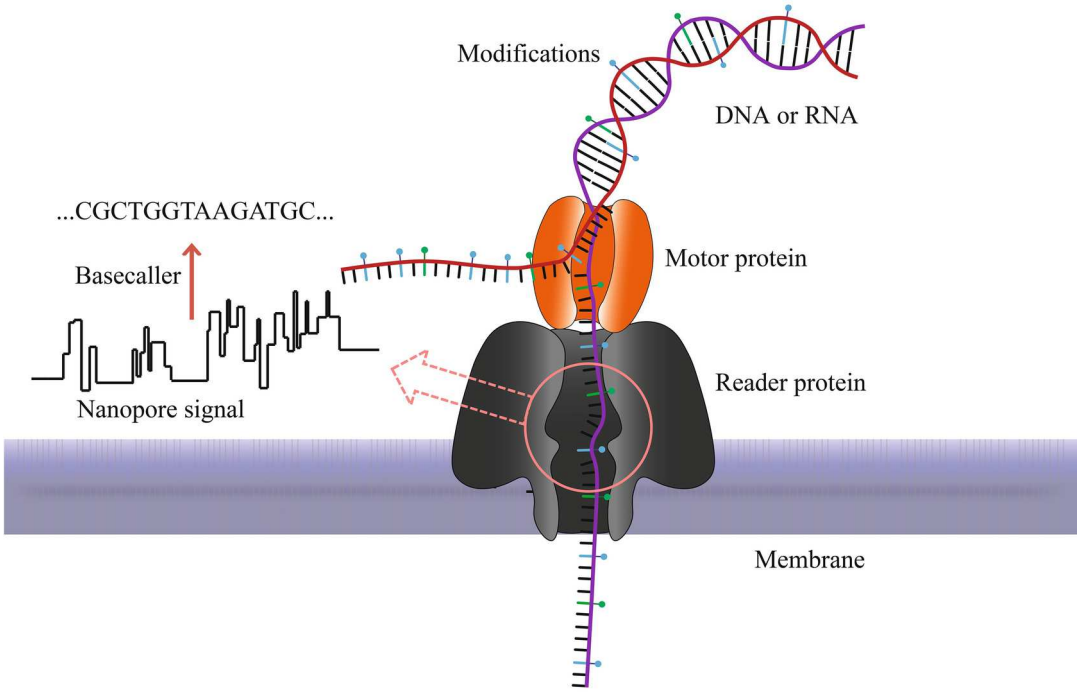


**Figure 2.2:** Oxford Nanopore sequencing (He et al., 2021)

The electrical resistance of a pore is determined by the number of nucleotides in the pore which is approximately five nucleotides per pore (5-mer), resulting in a large number of possible states: $4^5 = 1024$ for the basic four nucleotides, and $5^5 = 3125$ when 5mC modification is present. In addition, taking into account that signals come from single molecules, hence producing noisy and stochastic data, basecalling is considered a challenging task (Wick et al., 2019). All of the modern basecallers use neural networks, such as the one that is going to be used in this thesis and described in the upcoming chapters, called Guppy.

## 2.2. Related work

In recent years, multiple tools have been developed in order to predict the existence of methylation at CpG sites from Nanopore signals. They can be divided into three categories: statistical tools, tools based on the hidden Markov model, and deep learning tools.

NanoRaw (Stoiber et al., 2016), NanoMod (Liu et al., 2019b), and Tombo[1] belong to the group of statistical tools. Testing-based tools can detect any chemical modification without the need for prior training data. NanoRaw and NanoMod require two

[1] `https://nanoporetech.github.io/tombo/tutorials.html`

groups of reads, one from a sample with modifications and the other from a matched unmodified control sample, alike Tombo that needs only modified samples for modification detection. On the one hand, in NanoRaw's implementation Mann–Whitney U-test combined with Fisher's method is used. On the other hand, NanoMod replaces Mann–Whitney U-test with Kolmogorov–Smirnov test, and Fisher's method with Stouffer's method, thus improving performance. Lastly, the basis of the Tombo framework is its re-squiggle algorithm which defines a new assignment from squiggle (raw Nanopore signal) to reference sequence based on an expected current level model, since basecalling may contain some errors compared to a reference sequence. Afterwards, modified base detection takes place, using different statistical tests.

SignalAlign (Rand et al., 2017) and Nanopolish[2] use hidden Markov model (HMM) for modification prediction. SignalAlign is a generative model which is consisted of a variable-order hidden Markov model (HMM) combined with a hierarchical Dirichlet process (HDP) used to learn ionic current distributions, referred to as an HMM-HDP model. Nanopolish uses HMM to compare likelihoods of both modified and unmodified k-mers, nucleotide strings of length k, which contain at least one CpG site. If there is more than one CpG present in a k-mer, only a k-mer level prediction is done.

Some of the most popular deep learning approaches are the following: Guppy[3], DeepMod (Liu et al., 2019a), DeepSignal (Ni et al., 2019), DeepSignal2[4], Megalodon[5], and, lastly, Rockfish[6] whose implementation is given in its own section (Section 2.3) because of its importance for this thesis. Guppy is used for basecalling the raw signals in this thesis, and described in more details in Subsection 5.1.1. DeepMod is a pure LSTM model, whereas DeepSignal combines LSTM and CNN architecture. DeepSignal2 is a much smaller deep learning model in size, and it achieves slightly better performance in 5mCpG detection of human data, than the original DeepSignal. Last but not least, Megalodon performs basecalling exactly as in Guppy, performs reference anchoring using Mappy which is also used in this thesis, and explained in Subsection 5.1.2. However, Megalodon resolves indels (insertions and deletions) simply by assigning signal points to the previous nucleobase, alike the methods developed in this work (consult Section 4.1 and 4.2).

Currently, DeepSignal2 and Megalodon are state-of-the-art modification detection tools, whilst Rockfish is still in the development, but already shows promising results.

[2] https://nanopolish.readthedocs.io/

[3] https://community.nanoporetech.com/protocols/Guppy-protocol/

[4] https://github.com/PengNi/deepsignal2

[5] https://github.com/nanoporetech/megalodon

[6] https://github.com/lbcb-sci/Rockfish

## 2.3. Rockfish

Rockfish[7] is a deep learning method for detecting DNA base modifications from Nanopore signal, developed by my supervisor Dominik Stanojević. The method can be used for several different tasks, such as read-level and genomic-level 5mC modification detection, cross-dataset generalization, and bisulfite sequencing. Rockfish has been tested on sequenced Escherichia coli Repli-G/M.SssI data and NA12878 human data, described in more details in chapters 3.1 and 3.2, respectively. The testing shows that Rockfish achieves state-of-the-art results, or at least comparable performance, as the methods described in Section 2.2.

The Rockfish code consists of the following three parts:

1. extract features
2. train
3. inference

Figure 2.3 illustrates Rockfish pipeline which is composed of four parts. First, Nanopore reads are basecalled using Guppy basecaller, in order to infer nucleobase sequence from the raw signal.

Next, Tombo's re-squiggle algorithm is used to map basecalled reads to the given reference using Minimap2 (Li, 2018), and to map signal points to the reference, thus correcting possible basecalling errors.

Third step is feature extraction using event table which is the output of re-squiggle algorithm. For every read, CpG motifs must be found, whilst taking care of alignment strand (forward and reverse), thus obtaining CpG regions of 17 nucleobases, because the window parameter is set to 8 by default, as shown in Figure 2.4. It is also possible to change the motif, which is "CG" by default, and central position index, which defaults to zero, thus respresenting the nucleobase "C". For example, motif "AATG", index 2, and window 7 might be provided, meaning that the nucleobase "T" is in the center, and length of the region equals 15. Nucleobases in the said regions get 20 signal points sampled from the corresponding event. If an event is longer than 20 points, some of the signal points are removed, and if it is shorter than 20 points, some of the points are repeated. The resulting signal vectors have exactly 340 elements, and they are stored in a binary file, together with event lengths, sequences of 17 nucleobases, and labels. Labels are provided for synthetic datasets, but for native ones need to be determined from

[7] `https://github.com/lbcb-sci/Rockfish`

bedMethyl file, containing per-site coverage and methylation percentage, as depicted in Figure 2.3.

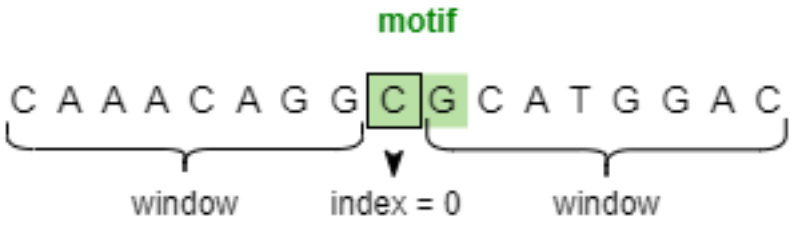


**Figure 2.4:** An example of CpG region, window = 8

Finally, the model is trained on the aforementioned binary file, in addition to encoded nucleobase vector, which is attained by assigning different integers to different nucleobases in the relevant region. The length of these vector should be 340 such as the signal vector, which is achieved by repeating each label 20 times, hence mapping 20 signal points to every nucleobase. The Rockfish model consists of encoder network and transformer module, and it outputs modification probability for the given CpG region. The encoder network is used to build latent representations of the input region, and to increase latent dimension for every timestamp. The encoder has three convolutional blocks, comprised of one-dimensional convolutional layer, GELU activation function (Hendrycks i Gimpel, 2016) and instance normalization (Ulyanov et al., 2016). Further on, data is processed using transformer module, i.e transformer encoder and decoder. The transformer encoder is equivalent to the encoder defined in (Vaswani et al., 2017). Furthermore, the transformer decoder consists of global average pooling operation which reduces sequence dimension and a linear layer which outputs logit value, i.e. the wanted modification probability.

Inference takes trained model checkpoint file and re-segmented fast5 files as input, and outputs the modification probability for relevant CpG regions, alongside with a few other important information, such as contig, read name, alignment strand, and position of the central nucleobase (C) in the reference. The last output value is the predicted modification label, which equals 1 if the logit value is greater than 0, meaning that the modification occurred, and zero otherwise, which means there is no modification.

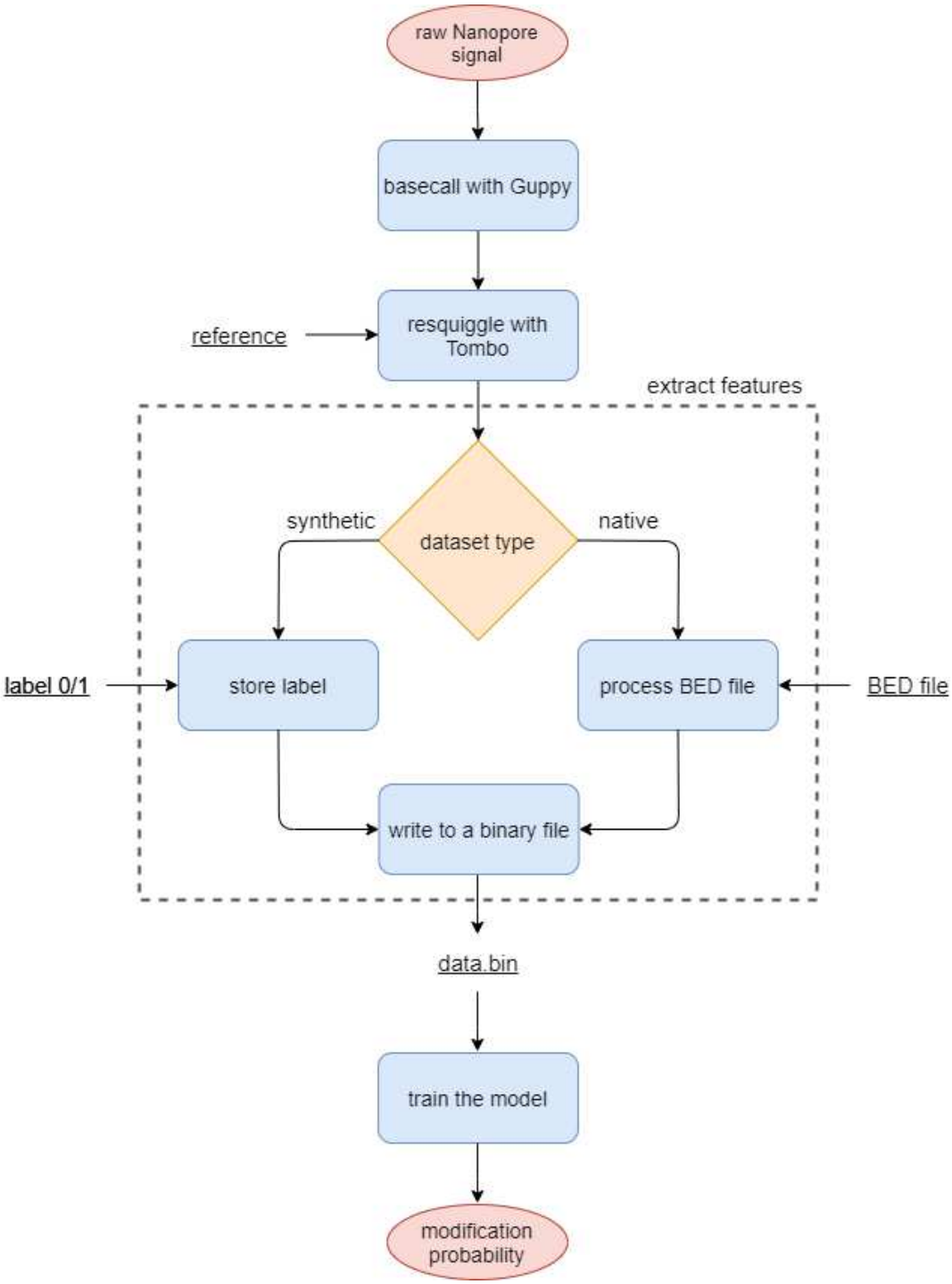


**Figure 2.3:** Rockfish pipeline

# 3. Dataset

This chapter gives a brief description of datasets used in this thesis, Escherichia coli and Homo sapiens data. Moreover, data analysis, crucial for the subsequent chapters, is provided. The detailed data analysis is crucial, since it helps whilst making decisions on what algorithms to implement, what parameters to use, and what thresholds to put.

## 3.1. Escherichia coli data

Escherichia coli strain K12 MG1655 has been kindly gifted to us by Dr. Swaine Chen's laboratory in Genome Institute of Singapore, A*STAR, Singapore. The modifications on the genomic DNA obtained from the grown E. coli were eliminated using REPLI-g Mini Kit. Afterwards, the resulting whole genome amplified sample was treated with M.SssI methyltransferase. The obtained synthetic E. coli data is primarily used for the data analysis, since it is known which reads are modified. For that purpose, 1,000 modified and 1,000 unmodified reads are examined.

The reference genome has been downloaded from NCBI (National Center for Biotechnology Information) GenBank under accession number NC_000913.3. There is a total of 346,793 CpG sites in the reference genome.

## 3.2. Homo sapiens data

NA12878 Homo sapiens native dataset (Jain et al., 2018) has been obtained from European Nucleotide Archive under acession number PRJEB23027. Human genome assembly GRCh38 (Schneider et al., 2017) was used as the input reference for data extraction. Bisulfite sequencing used as a ground truth for NA12878 data was acquired from ENCODE Project (Consortium et al., 2012) under accession number ENCFF835NTC.

The described human dataset consists of 406,821 reads in total, mapped to the chromosome 21 and 22. The data is partitioned into training, validation, and test set by 80%, 10%, and 10%, respectively, as can be seen in the Table 3.1.

**Table 3.1:** Distribution of Homo sapiens data

| Chromosome | Training | Validation | Test |
|---|---|---|---|
| chr21 | 176,040 | 22,005 | 22,006 |
| chr22 | 149,416 | 18,677 | 18,677 |
| **Total** | 325,456 | 40,682 | 18,677 |

For training the model only the high-confidence CpG positions, i.e. the positions which have at least 10 mapped reads, are included. This definition is in the accordance with DeepSignal (Ni et al., 2019). Furthermore, only the positions with unambiguous methylation are considered, meaning that the position is labeled as unmethylated if the methylation frequency is 0%, whereas it is labeled as methylated if the frequency is 100%. The final outcome is 5,084,927 unmethylated and 6,211,372 methylated high-confidence CpG positions.

## 3.3. Data analysis

The mentioned synthetic Escherichia coli dataset was thoroughly examined and analysed, taking into consideration if the reads are modified or unmodified. The native NA12878 human dataset is very large, and is not clearly separated according read modification, thus it is not analysed in this section.

A large number of different analyses has been made, however, only the most important ones are presented in this thesis. In Subsection signal lengths are plotted and commented. Subsequently, in Subsection alignment between reference and queries, i.e. basecalled reads, is explored and the most important findings written down. At last, start index of raw signals is analysed, compared against the end of the signal, and plotted in Subsection .

The data analysis is important because it points us in the right direction regarding the selection of methods to implement, what exactly to keep in mind while coding them, on which values to set the parameters, etc.

### 3.3.1. Signal length

This subsection briefly gives raw signal lengths of 1,000 modified and 1,000 unmodified E.coli reads, presented in Figure 3.1. It can be observed that a lot of unmodified

reads have shorter signal lengths, and then also a few of them have extremely long signal lengths, around 500,000 to 600,000 signal points. On the contrary, modified reads' signal lengths are distributed more evenly, mostly below 150,000, and with a maximum value below 400,000.

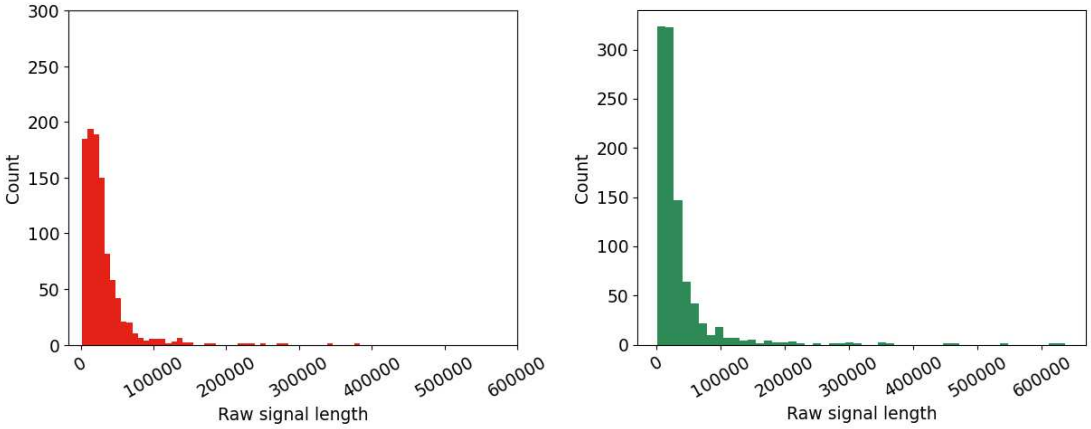


**Figure 3.1:** Distribution of raw signal lengths for modified vs. unmodified reads

### 3.3.2. Alignment

Alignment is obtained by mapping basecalled reads to the reference, and it is consisted of the four following CIGAR operations: match, mismatch, deletion, and insertion. The values in Table 3.2 are calculated as the amount of certain operation in alignment divided by the length of alignment.

First two rows show the distribution of operations on all bases, from which it can be concluded that alignments are pretty accurate, considering they have more than 90% of matches. It can also be seen that modified reads have less matches, and more mismatches, deletions, and insertions. In conclusion, modified reads are harder to map correctly, because, as their name states, they contain modifications.

Last two rows show the comparison of operations at CpG positions, i.e. if cytosine is matched, mismatched, deleted, or inserted. We can conclude that modified reads once again have less matches, more mismatches and deletions, but, surprisingly, less insertions. Therefore, inserted cytosine is not going to be considered as a modification at CpG context. Reference anchoring is going to be implemented, and only the CpG positions on the reference are going to be observed, as it is considered to be the ground truth.

**Table 3.2:** Average amount of CIGAR operations across the alignments [%]

| Position | Modification | Match | Mismatch | Deletion | Insertion |
|---|---|---|---|---|---|
| Any | mod | 91.177 | 3.217 | 3.295 | 2.311 |
| | nomod | 93.867 | 2.260 | 2.403 | 1.470 |
| CpG | mod | 92.310 | 4.841 | 2.338 | 0.510 |
| | nomod | 96.007 | 2.601 | 0.857 | 0.535 |

After aligning basecalled reads to the reference using Mappy, we have noticed that some reads do not yield any alignment. The further investigation was conducted, and the findings were that 97.794% of reads with no alignments have mapping quality of zero. In general, reads with lower mapping quality have either no alignment, or short and quite incorrect one.

Based on the distribution of mapping qualities, shown in Figure 3.2, it is decided to put the mapping quality threshold to 10, meaning that all alignments that have the mapping quality below said threshold are discarded. It can be observed that Mappy has done a pretty good job, because a vast majority of alignments have the maximum mapping quality of 60. Lastly, the average mapping quality is, as can be expected, higher for unmodified reads, because they have less mistakes as has been previously shown in Table 3.2.

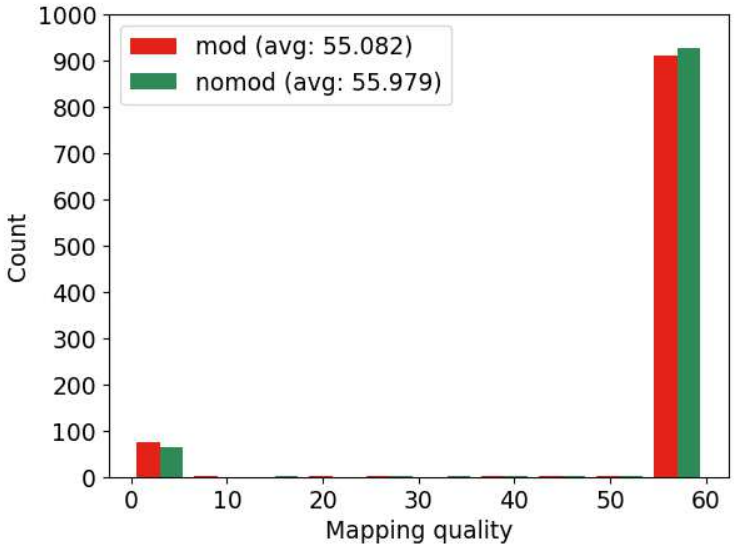


**Figure 3.2:** Mapping quality of alignments for modified vs. unmodified reads

### 3.3.3. Start raw index

Start index of raw signals is the index at which the signal actually starts, that can be noticed by a sudden peak in the signal amplitude. The assumption is that the first N signal points before the mentioned peak have a small amplitude, and thus a small standard deviation.

First, distribution of start raw indices may be seen in Figure 3.3 and it can be seen that those indices are usually lower for unmodified reads. The reasoning behind that might lie in the fact that unmodified reads are easier to basecall, therefore they have smaller starting area with low amplitude, i.e. the real signal values start before than in modified reads.

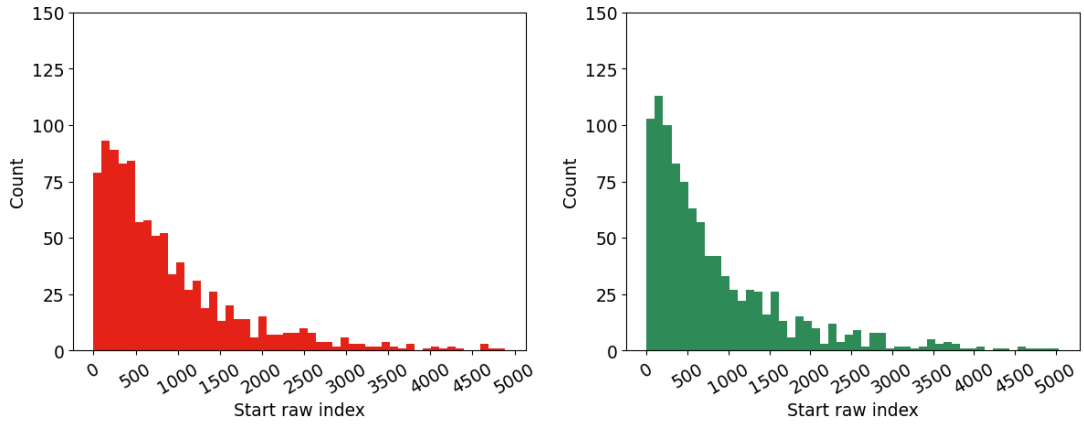


**Figure 3.3:** Distribution of start index of raw signals for modified vs. unmodified reads

In order to further explain the idea around the start raw index, the first and the last 1,000 signal points are drawn for one representative modified read, and one unmodified, as shown in Figure 3.4 and 3.5. It can be noticed that signal remains still until the sudden start raw index peak, and then continues to deviate around the center value. Furthermore, it can be concluded that there exists no such thing as an end raw index, since the signal has larger amplitude until the very end of the read.

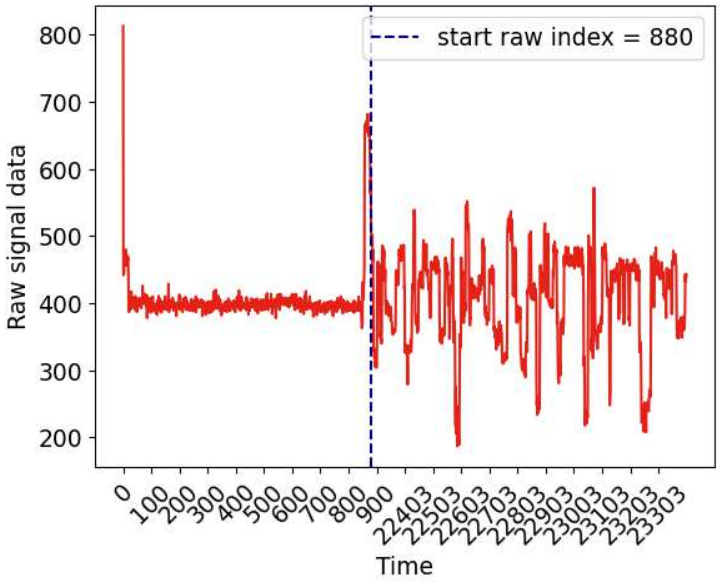


**Figure 3.4:** The first and last 1,000 signal points for a representative modified read

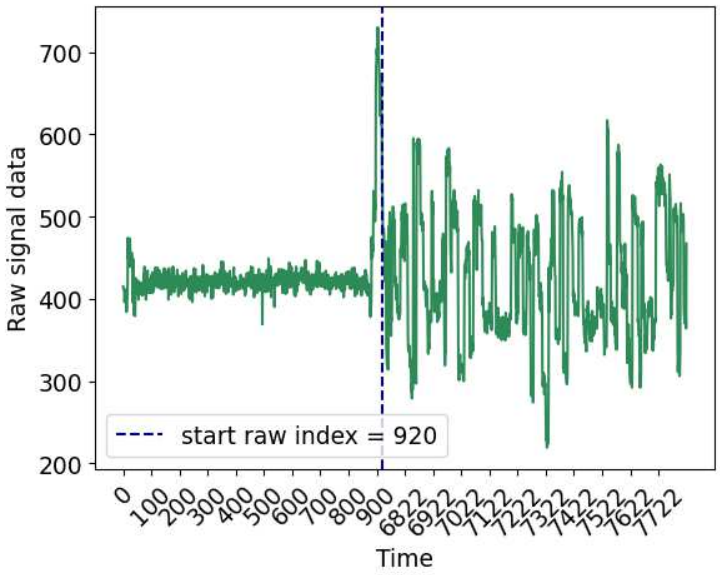


**Figure 3.5:** The first and last 1,000 signal points for a representative unmodified read

We have also decided to look at the standard deviation of signal points before jumping to any conclusions. As can be concluded from Figure 3.5 and 3.6, standard deviation is close to zero until the abrupt appearance of start raw index for both modified and unmodified representative read. Further on, the standard deviation diverges until the very end of the signal, which confirms that there does not exist an end raw index.

Based on the findings in this subsection, it is decided to trim the signal, so it begins from the start raw index until the end of the signal, thus obtaining only the relevant

signal points.

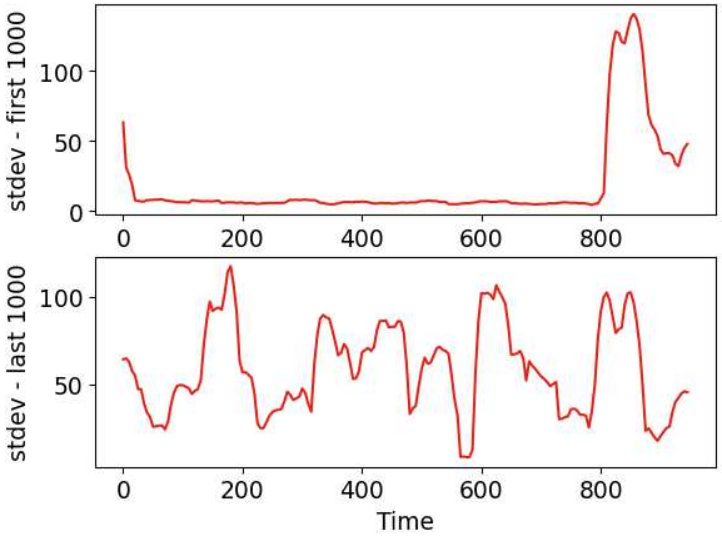


**Figure 3.6:** Standard deviation of the first and last 1,000 signal points for a modified read

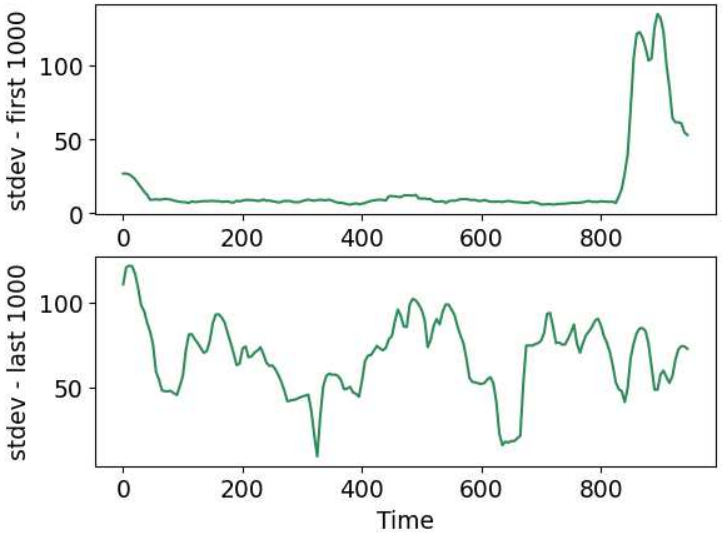


**Figure 3.7:** Standard deviation of the first and last 1,000 signal points for an unmodified read

# 4. Methods

This chapter explains underlying concepts and algorithms necessary for the final Remapper implementation described in Chapter 5. Remapper is a tool developed for the sake of replacing Tombo framework used in the original Rockfish code described in 2.3, and hopefully to lower the overall execution time. After aligning basecalled reads to the reference using Mappy, signal points at indels should be remapped, hence the name Remapper. First, insertions are resolved as depicted in Section 4.1 using the Half-half method (see Subsection 4.1.1). Next, as explained in Section 4.2, the deletions are dealt with in one of the two possible ways, Concatenate and divide method (see Subsection 4.1.1) or Longer neighbour method (see Subsection 4.2.2). At last, Section 4.3 describes Binary writer, used for storing the extracted features into binary file, later on used for training.

## 4.1. Resolving insertions

Insertions occur when a basecalled read contains a nucleobase, or several consecutive nucleobases, which are not present in the reference at the same position in the alignment. The reference is considered to be the ground truth, and insertions to mainly be mistakes made during the basecalling process. Therefore, it is necessary to remove insertions and remap their signal points to the neighbouring bases. In order to do so, the Half-half method, described in the succeeding subsection, has been developed. As shown in Algorithm 1 the method takes alignment obtained using Mappy - "al", and signal points intervals for the observed read - "raw", as inputs. For more details on how the inputs are obtained consult second and third step of Remapper method in Section 5.2. The outputs are signal points intervals mapped to the reference, and intervals of indices at which deletions have taken place, that facilitate future handling of deletions.

**Algorithm 1** Resolve insertions

```
function RESOLVE_INSERTIONS(al, raw)
    cigar ← al.cigar if al.strand == 1 else reversed(al.cigar)
    r_pos, q_pos ← 0, al.q_st
    r_len ← al.r_en − al.r_st
    intervals ← [None] * r_len
    insertion ← False
    deletion_idx ← []

    for length, operation in cigar do
        if operation in {0, 7, 8} then
            if insertion then
                intervals[r_pos] ← (center, raw[q_pos].end)
                insertion, length ← False, length − 1
                r_pos, q_pos ← r_pos + 1, q_pos + 1
            for i = 0 to length do
                intervals[r_pos + i] ← raw[q_pos + i]
            r_pos, q_pos ← r_pos + length, q_pos + length

        else if operation == 1 then
            ins_interval ← (raw[q_pos].start, raw[q_pos] + length].start)
            center ← int(np.mean(ins_interval))
            intervals[r_pos − 1] ← (intervals[r_pos − 1].start, center)
            insertion ← True
            q_pos ← q_pos + length

        else if operation in {2, 3} then
            deletion_idx.append((r_pos, r_pos + length))
            if insertion then
                intervals[r_pos] ← (center, raw[q_pos].start)
                insertion, length ← False, length − 1
                r_pos ← r_pos + 1
            for i = 0 to length do
                intervals[r_pos + i] ← (raw[q_pos].start, raw[q_pos].start)
            r_pos ← r_pos + length
    return intervals, deletion_idx
```

### 4.1.1. Half-half method

The thinking process behind the occurrence of inserted bases is that they should not be present, and that they contain signal points which in reality belong to their neighbouring bases. For that reason, the Half-half method deals with insertions applying a simple heuristic of assigning half of their signal points to the left neighbour, and half of their points to the right neighbour, as shown in Figure 4.1. If there is an odd number of signal points, then the right neighbour gets one point more. For example, a total of 5 signal points would be divided into 2 and 3 points, and assigned to the left and right neighbour, respectively.

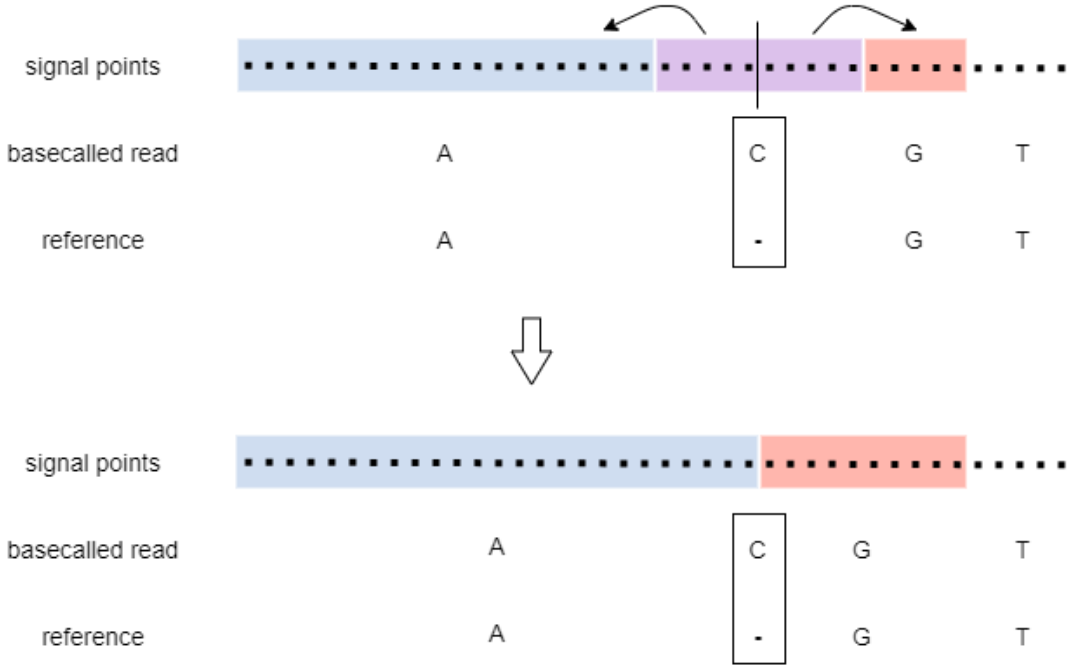


**Figure 4.1:** Half-half method for resolving insertions

Algorithm 1 demonstrates pseudocode for resolving insertions using the Half-half method. Input arguments are alignment obtained by aligning basecalled read to the reference using Mappy, and raw signal points that are written as intervals, from which the exact signal point values are easily attainable. Start of mentioned intervals is inclusive, whilst the end is exclusive.

Firstly, CIGAR operations and corresponding lengths are extracted from alignment, depending on the alignment strand. A few other variables are initialized, amongst which the starting reference and query positions that are going to be crucial in the continuation.

Subsequently, we iterate through CIGAR, and check if an operation is match (or

mismatch), insertion, or deletion, according to SAM format specification[1]. If the operation is insertion (consult line 17 in Algorithm 1), insertion interval is found, taking into account number of consecutive insertions, then the center index of mentioned interval is remembered. The end index of left neighbour is set to the center index, insertion flag is marked as True, and query position is updated depending on number of insertions. Only the query position is updated because insertions are the type of operation which consume query, i.e. they appear on the query, and are missing on the reference.

Next, if the operation is match or mismatch (see line 9 in Algorithm 1), and the insertion flag is set, i.e. insertion occurred in the last iteration, then the start index of the current base is set to the remembered center index. Then, the insertion flag is set to False, length of match operation is decreased by one, because the current base is "resolved", and positions on both query and reference are incremented, which concludes adjustment of the right neighbour. Then, remaining matches are resolved by simply mapping the signal points intervals from query to reference. Matches and mismatches consume both query and reference, thus both of the positions must be increased by the length of the matches.

Moreover, handling of deletions can be seen at line 23 in Algorithm 1, which is done in a similar fashion as the previously described matches, with the main difference lying in remembering deletion intervals essential for the following portion of the Remapper implementation. Again, if the insertion happened in the last iteration, then half of the insertion's signal points are given to the deletion, i.e. the right neighbour. The variables are adjusted similarly as for the matches, with the difference that only the position on the reference is incremented, because deletions consume reference. Afterwards, or immediately if there were no previous insertions detected, the signal intervals are assigned to deletions, but in a way that they get zero signal points, e.g. interval (350, 350). If they have not gotten any signal points from possible insertions, deletions enter the next phase with having assigned intervals, but containing zero points. Lastly, the reference position is increased by the number of deletions.

Finally, signal intervals assigned to the reference with resolved insertions and deletions written as empty intervals, together with a list of indices where the deletions occurred, are returned as the output.

[1] `https://samtools.github.io/hts-specs/SAMv1.pdf`

## 4.2. Resolving deletions

Deletions appear when reference contains certain nucleobases which are "deleted", i.e. not present, in the basecalled read at the same positions in the alignment. This event is considered to be the result of mistakes whilst basacalling the reads, because bases, which should be a part of the sequence, are wrongly omitted and "contain zero signal points". Those deleted bases should have certain amount of signal points, mistakenly attributed to their neighbours. For this purpose, two methods for resolving deletions are implemented and compared. One the one hand, there is Concatenate and divide method, described in Section 4.2.1, which attempts to remap signal points from neighbours to deletions uniformly. One the other hand, Longer neighbour method is created and explained in Section 4.2.2, giving deletions signal points from the neighbour that holds more of them. The Algorithm 2 presents the resolve deletions algorithm, which takes outputs from the resolve insertions algorithm, and the type of deletion method as the input. At the end, the algorithm outputs the resulting signal points intervals remapped at indels and mapped to the reference.

**Algorithm 2** Resolve deletions

```
function RESOLVE_DELETIONS(intervals, deletion_idx, del_method)
    for del_st, del_en in deletion_idx do
        left ← intervals[del_st − 1]
        right ← intervals[del_en]

        if del_method ==′ concatenate_and_divide′ then
            sig_st, sig_en ← left.start, right.end

        else if del_method ==′ longer_neighbour′ then
            left_len ← left.end − left.start
            right_len ← right.end − right.start
            del_len ← del_en − del_st
            if left_len > right_len and left_len > del_len then
                sig_st, sig_en ← left.start, left.end
            else if right_len > left_len and right_len > del_len then
                sig_st, sig_en ← right.start, right.end
            else
                sig_st, sig_en ← left.start, right.end

        points ← np.array_split(range(sig_st, sig_en), del_en − del_st + 2)

        if len(points[−1]) == 0 then
            while len(points[−1]) == 0 do
                points.pop(−1)
            interval ← points.pop(−1)
            intervals[del_en] ← (interval[0], interval[−1] + 1)

        for i = del_st − 1 to del_en + 1 do
            if len(points) == 0 then
                if i < del_en then
                    intervals[i] ← intervals[del_en].start, intervals[del_en].start)
                continue
            interval ← points.pop(0)
            intervals[i] = (interval[0], interval[−1] + 1)
    return intervals
```

### 4.2.1. Concatenate and divide method

Deletions are bases which exist on the reference, but are deleted on the basecalled read, as shown in Figure 4.2, where "C" in the CpG context is modified, or more precisely deleted, on the read. One of the possible heuristics to deal with that issue is Concatenate and divide method, whose goal is to divide signal points between deletions and their first neighbours uniformly. The signal points from left and right neighbour are concatenated and divided into equal parts. If there exists a remainder whilst dividing the points, quite the opposite from the aforementioned insertion resolving process, the bases to the left get more points. For example, in Figure 4.2, 25 points shall be divided to three bases, which is achieved in the following way: left neighbour gets 9 points, deleted base gets 8 points, and, finally, 8 points are assigned to the right neighbour.

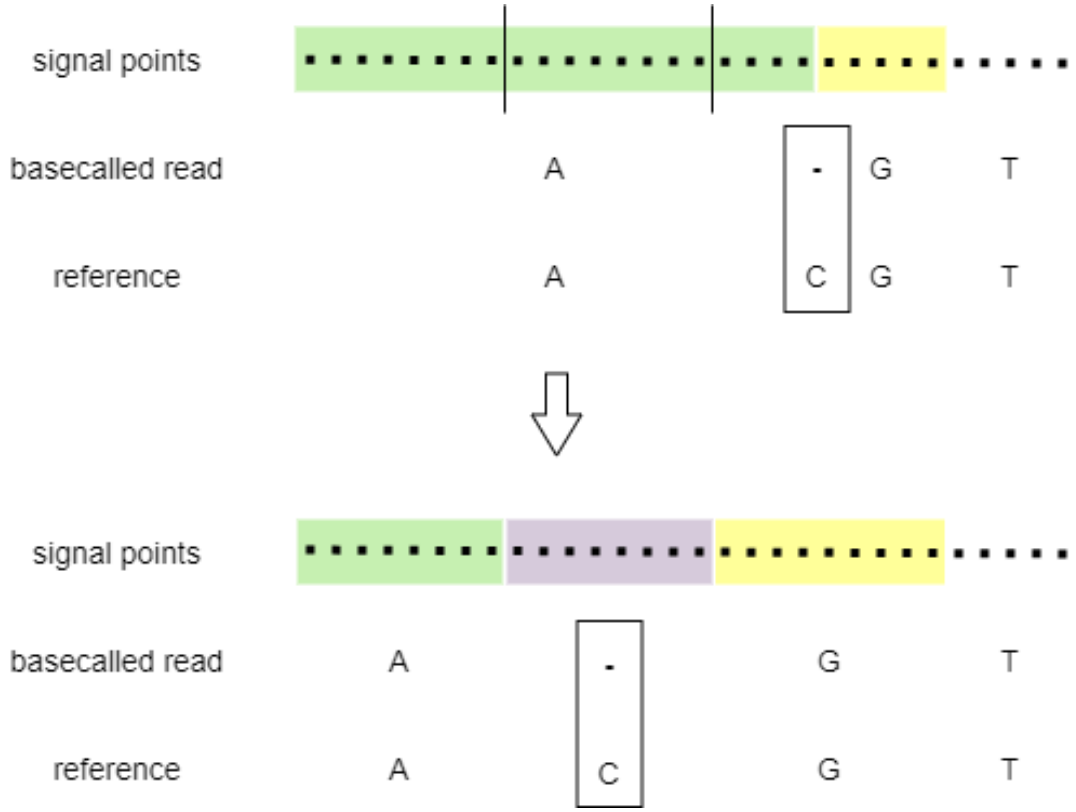


**Figure 4.2:** Concatenate and divide method for resolving deletions

The pseudocode for the Concatenate and divide method is given in Algorithm 2. The inputs are signal intervals mapped to the reference already modified by the resolve insertions method, intervals at which deletions occurred, and wanted deletion method. The resolve deletions method begins with a for loop going through deletion intervals,

and finding an interval for the left and right neighbour.

Next, signal start is defined as the start position of left neighbour, and the signal end as the end position of right neighbour, thus concatenating necessary signal points. At line 17 in Algorithm 2 points are divided uniformly between neighbours and deletions, depending on the number of deletions.

The case where there is not enough signal points to divide amongst bases is covered from line 18 to 22. This part of code makes sure that the right neighbour keeps at least one signal point. It can be observed that the left neighbour always has at least one point, at least one point is explicitly given to the right one, and deletions might have zero or more points. For example, if left neighbour has one point, the right one two points, and there are two deletions, then the points are given as follows: one to the left neighbour, one to the first deletion, zero to the second deletion, and, lastly, one to the right neighbour.

Finally, the program loops through the relevant positions on the reference and maps points to them, more precisely it adjusts the signal interval start and end. If there are zero signal points, then an empty interval is made, e.g. (350, 350). After the adjustment of all deletion intervals, new signal point intervals is returned as the output.

### 4.2.2. Longer neighbour method

Concatenate and divide method might seem a bit unfair, in sense that it divides signal points in a uniform way, disregarding the cases in which one neighbour has a lot more points than the other. For instance, if there is one deletion, and one neighbour has 5 points, whilst the other has 85 points, is it fair to divide points as 30-30-30, taking away a vast number of points from the latter neighbour? We have decided to analyse the direct neighbours on the E. coli dataset (see Subsection 3.1), and based on the results think about an implementation of an alternative method.

The first assumption is that the bases around deletions have more signal points than those around matches. The reasoning behind that lies in the previously explored thought that the basecaller has made a mistake in not detecting the deleted base, and that it assigned its signal points to the one of the neighbours. Therefore, the deletion should have probably been in the place where there are more signal points than the average. The Figure 4.3 confirms the assumption that there are more signal points in average around deletions than matches. It can be observed that matches have larger values for 5 signal points, for 10 signal points values are similar, and for larger values, deletions start to take over. Lastly, it can also be seen in the right figure that unmodified

reads have larger values for smaller number of signal points, and modified ones take over after 15 points. This can be interpreted in a way that modified reads have more deletions at CpG positions.

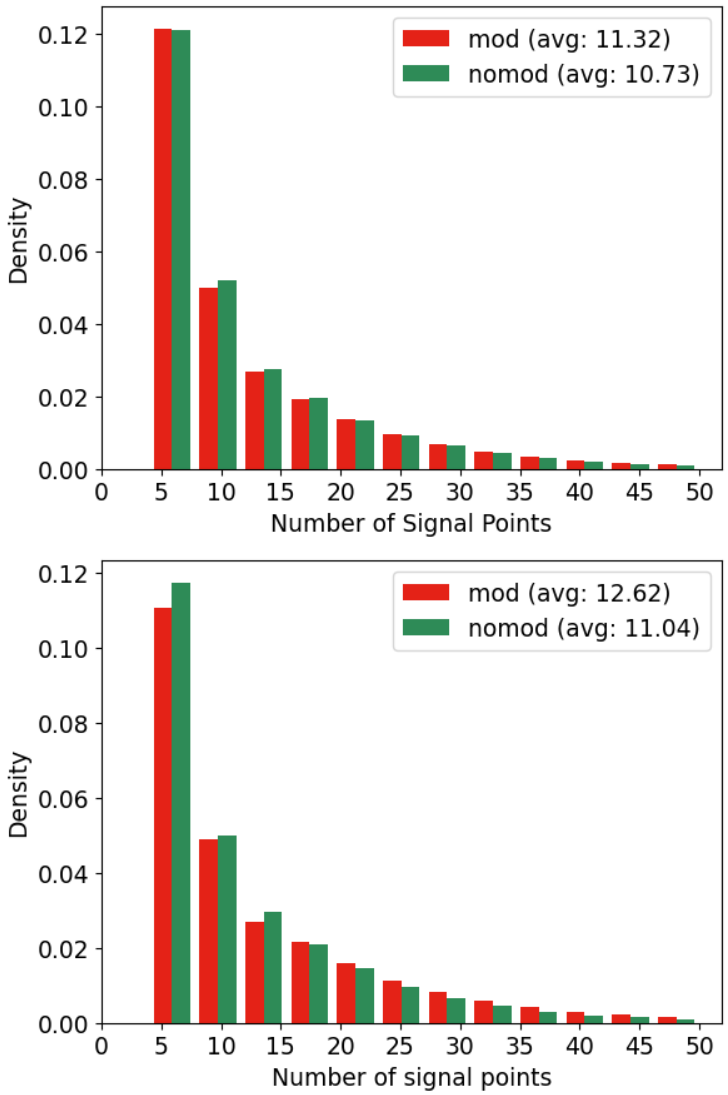


**Figure 4.3:** Distribution of signal points around matches vs. deletions

Now that we have established that there are indeed more points around deletions than usual, we can proceed to formulate the second assumption. It looks at the signal ratio, i.e. the neighbour with less signal points divided by the neighbour with more signal points, which is a number between zero and 1. So, the second assumption, which is the foundation of the Longer neighbour method, claims that the signal points from the supposed deleted base have been assigned to the neighbour who now has more points. We wonder how often the case that one neighbour has more points than the other happens. Also, we are interested in signal ratios, i.e. how many more signal points does

the larger of two neighbours have. Ratios below 0.5 prove a large difference between neighbours, for instance, ratio for opening example of 5 and 85 points is 0.0625. On the contrary, ratios above 0.5 show a smaller difference between neighbours, whilst the ratio equal to 1 means that the two neighbours have the same amount of signal points.

Figure 4.4 shows distribution of signal points ratios for E. coli reads, and interestingly strongly confirms the aforementioned statement that modified reads have more deletions than unmodified. Moreover, there are indeed a vast number of deletions with ratios below 0.5 which arises the need for the alternative method which will take this discovered fact into consideration. There is also quite a lot of cases with ratio equal to 1, which can be covered with the previously implemented Concatenate and divide method. Finally, the interesting fact is that one or two consecutive deletions prevail, confirming that the basecaller works well, i.e. that it has not basecalled a lot of consecutive deletions.

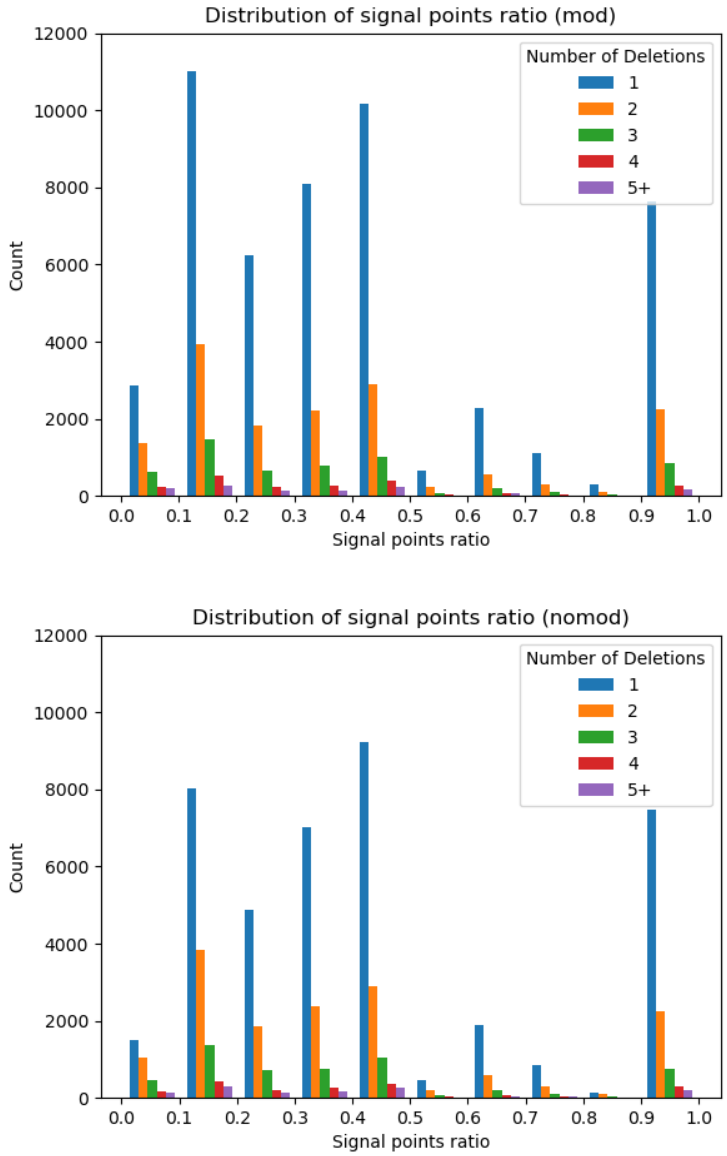


**Figure 4.4:** Distribution of signal points ratio for modified vs. unmodified reads

Longer neighbour method first compares the number of signal points of left and right neighbour and decides which one is longer. Then, the signal points of the longer neighbour are divided between the neighbour and the deletions in a uniform way, same as in the Concatenate and divide method. For further explanation consult Figure 4.5 where the same starting example from Figure 4.2 is now resolved using the alternative method.

There exist two edge cases, left and right neighbour having the same amount of signal points, and not having enough signal points to divide between all the deletions. The latter case is quite rare because there are mostly one or two consecutive deletions

as proved above, and there are usually enough signal points to assign to them. Nevertheless, both cases are solved with Concatenate and divide approach.

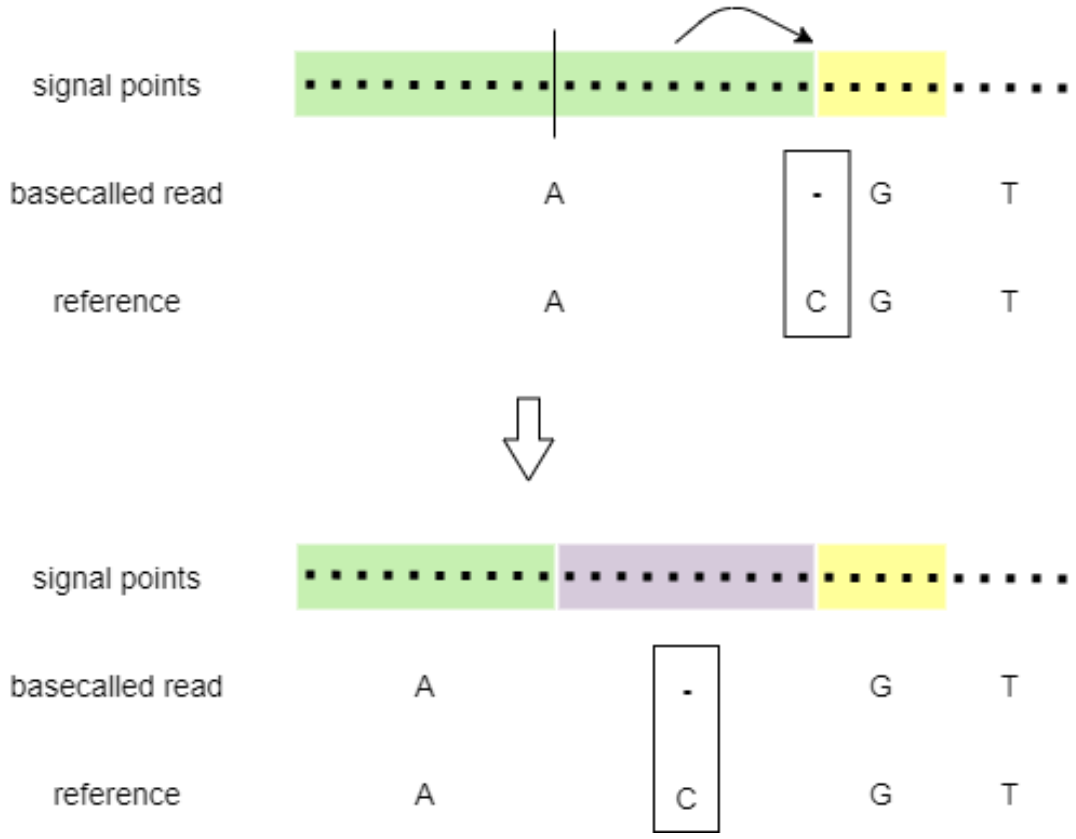


**Figure 4.5:** Longer neighbour method for resolving deletions

The lines 7 to 16 in Algorithm 2 are key if we choose the Longer Neighbour approach. The number of signal points of left and right neighbour are compared. If one neighbour is larger than the other, additionally we need to check if its number of signal points is larger than the number of deletions, i.e. if it has enough points to cover all of them. If both conditions are met, signal start and signal end are adjusted according to start and end of that neighbour. Otherwise, signal start and signal end are set as before and classic Concatenate and divide method is applied.

## 4.3. Binary writer

Binary writer in the original implementation writes signals of exactly 340 points in a binary file, therefore offsets, used for random access of examples, are known. Now, due to manual remapping, and removing the fixed sampling of 20 signals per base,

signals of variable lengths must be successfully written. In order to do so, a certain overhead shall be introduced, a header containing number of examples and lengths of every example.

Every processor instantiates one object of Binary Writer class, which stores examples, i.e. extracted data for reads that are processed. Method `write_data` inside Binary writer takes as inputs relevant data, label if the dataset is synthetic or BED information if the dataset is native. The examples are structured in the following way:

- **signal** - array of raw signal points of variable length
- **lens** - lengths of the event intervals
- **kmer** - bases in the relevant region extracted from the reference (e.g. 17-mer for motif "CG" and window equal to 8 shown previously in Figure 2.4) written as integers (different integers are assigned to different bases)
- **label** - stored label, 0 or 1, for unmodified or modified example, respectively

If the dataset is synthetic, label is simple stored to the example, whereas if it a native dataset, label is decided based on the modification frequency stored within BED information. If modification frequency is larger than 50% label is 1, 0 otherwise.

Examples are stored one by one in temporary data.bin files, and lengths of every example in bytes are stored in temporary header.bin files. There are as many temporary files, as there are processors running in parallel.

Finally, `on_extraction_finish` method concatenates all temporary data.bin files to one, and all temporary header.bin files to one, with first number in file representing total number of examples. Afterward, header.bin and data.bin are concatenated into an unique data.bin file ready for training.

The data.bin file is structured as described above because first the total number of examples is read determining how many integers representing the lengths of every example must be read next. Then, lengths of examples are read and summed cumulatively in order to obtain offsets necessary for random access of examples whilst training. The examples are read when needed and can be accessed by index from which the corresponding offset is calculated.

# 5. Implementation

This chapter provides an overview of external dependencies used for the implementation presented in Section 5.1. Moreover, detailed code structure and implementation details are given in Section 5.2. At last, Section 5.3 describes training procedure of the deep model.

## 5.1. Dependencies

This section deals with external dependencies used for the final implementation. First, it gives a short introduction to Guppy, method used for basecalling the reads (see Subsection 5.1.1. Next, Mappy, method used for aligning reads to the reference is outlined in Subsection 5.1.2. Furthermore, in Subsection 5.1.3 PyTorch and PyTorch Lightning, libraries commonly used for deep learning, are described. Lastly, Subsection 5.1.4 provides a brief overview of other libraries and tools used in this implementation.

### 5.1.1. Guppy

**Guppy**[1] is a basecaller based on the RNN architecture that transforms raw FAST5 data into canonical bases of DNA or RNA. The Guppy toolkit also performs modified basecalling (5mC, 6mA, and CpG) from the raw signal data, returning an additional FAST5 file of modified base probabilities as the output.

In this implementation, Guppy basecall server is run on a certain port, using basecalling of high accuracy and GPU mode, in order to obtain accurate basecalling at an acceptable speed. Furthermore, **ont-pyguppy-client-lib**[2] is a Python Guppy API, used as a client which connects to the said server and basecalls given FAST5 reads.

Both Guppy basecall server and client shall have compatible versions, and in this implementation version 4.4.2 is used.

[1] `https://community.nanoporetech.com/protocols/Guppy-protocol/`
[2] `https://pypi.org/project/ont-pyguppy-client-lib/`

### 5.1.2. Mappy

**Mappy**[3] is a Python API built on top of the Minimap2 (Li, 2018) implementation. Minimap2 is a sequence alignment program which aligns DNA or mRNA sequences against a large reference genome. In this implementation it is used for mapping Oxford Nanopore genomic reads to the human genome, as well as E. coli reference.

### 5.1.3. PyTorch and PyTorch Lightning

**PyTorch**[4] (Paszke et al., 2019) is an open source deep learning Python library used for tensor computation with strong GPU acceleration, as well as building different deep neural network architectures.

**PyTorch Lightning**[5] is built on top of PyTorch and its main usage is organising the training code, making it more readable, easier to reproduce, less prone to errors, scalable to any hardware without changing the model, etc.
In this implementation, PyTorch is used to implement the deep model and perform certain tensor calculations, whilst PyTorch Ligthning serves as a tool for organising the training code.

### 5.1.4. Other dependencies

**h5py**[6] is a Python interface to the HDF5 binary data format used to store huge amounts of numerical data, and later easily manipulate that data. In the implementation FAST5 reads are stored in HDF5 files. Furthermore, **ont_fast5_api**[7], a simple Python interface to HDF5 files of the Oxford Nanopore. FAST5 file format, is used to handle those types of files directly from Python code.

Other dependencies are listed here briefly, due to their familiarity and widespread usage:

- **NumPy**[8] - fundamental package for scientific computing in Python
- **Biopython**[9] - set of tools for biological computation in Python
- **Matplotlib**[10] - Python library used for visualisation

[3] https://pypi.org/project/mappy/
[4] https://pytorch.org/
[5] https://www.pytorchlightning.ai/
[6] https://www.h5py.org/
[7] https://github.com/nanoporetech/ont_fast5_api
[8] https://numpy.org/
[9] https://biopython.org/
[10] https://matplotlib.org/

– **tqdm**[11] - a fast, extensible progress bar for Python and CLI

## 5.2. Code structure

The whole implementation has been written in Python 3.7.10, with the help of the dependencies described in the previous section. The code is written in an object oriented matter and using multiprocessing to achieve comparable speed with the original implementation. The code is publicly available under the MIT licence at the following link: `https://github.com/sanjadeur/master-thesis`.

The code pipeline is shown in Figure 5.1 where it can be observed that the Tombo framework present in the original Rockfish implementation (see Figure 2.3) is replaced with the new Remapper tool. Besides the sole implementation, Remapper needs to be successfully integrated into Rockfish pipeline. The biggest difference from the original Rockfish code is the fact that signal vectors now have variable lengths, instead of their length being fixated at exactly 340 signal points.

Unlike the original implementation which takes previously basecalled and re-segmented reads as inputs, this implementation simply takes raw FAST5 reads.

Next, feature extraction begins with basecalling, continues with the Remapper tool, and ends in the same way as before. The basecalling process is written completely in Python using tools mentioned in 5.1.1. In order to avoid basecalling to become a bottleneck in the pipeline, it is written using multiprocessing, using the producer-consumer pattern. The number of producers and consumers, called processors in this implementation, can be set as a parameter of the program. Processors are used to basecall the input raw data and put processed reads into the queue. Then, processors take basecalled reads from the queue, and further process them using Remapper, and finally write them in a binary format. The program ends when the queue is empty, meaning there are no more basecalled reads to further process.

[11] `https://github.com/tqdm/tqdm`

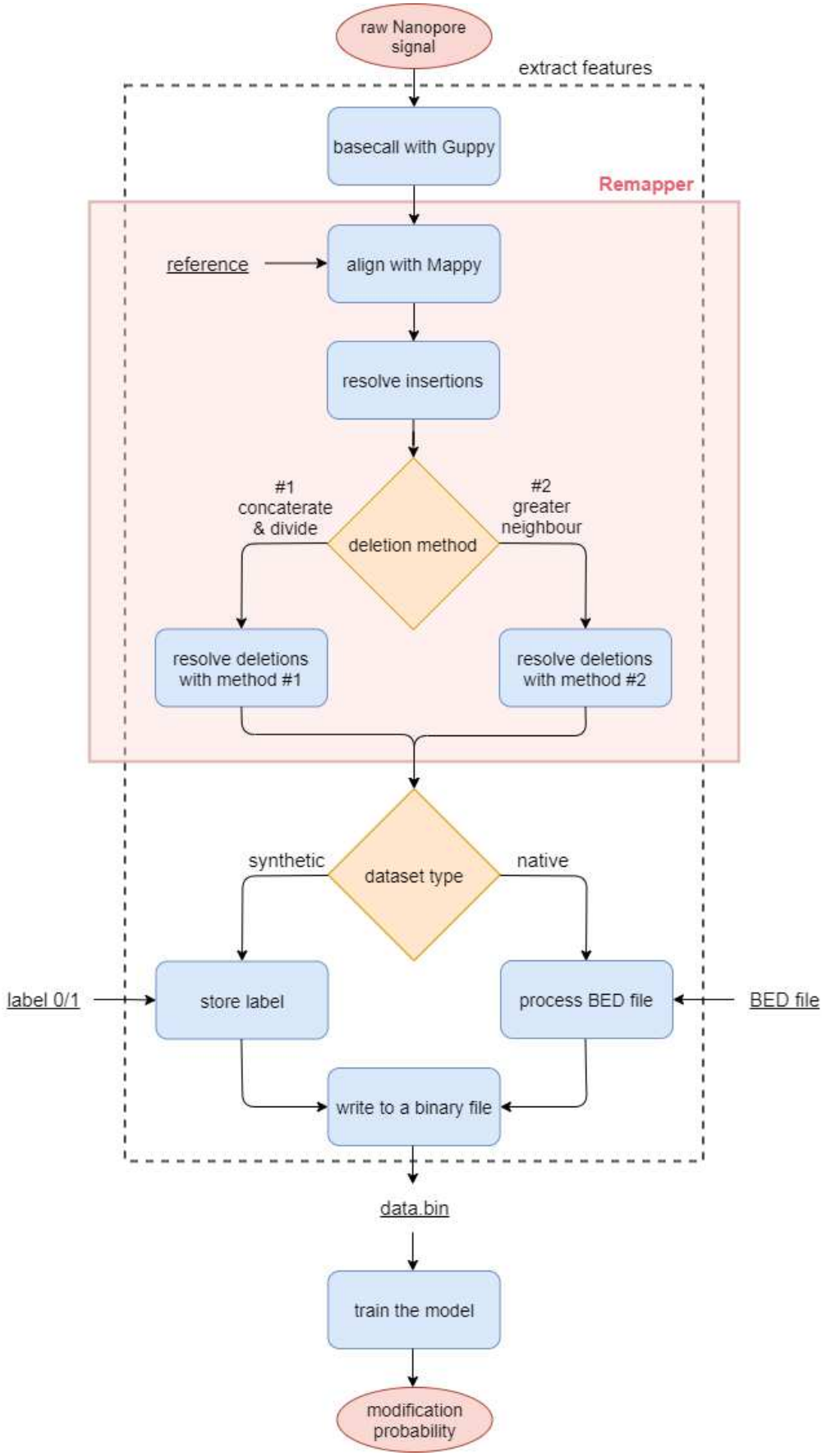


**Figure 5.1:** Rockfish with Remapper pipeline

Subsequently, an object of a class Remapper is instantiated and given as an argument to all the processors. Remapper consists of seven most important steps, as follows:

1. find motif positions on the reference
2. align read to the reference using Mappy
3. convert sequence to raw signal
4. find relevant motif positions
5. resolve insertions
6. resolve deletions
7. extract relevant data

In the Figure 5.1 some of the steps above are omitted for the clarity sake but are going to be thoroughly explained in the following text.

In the Remapper initialisation part Mappy aligner is instantiated, used for aligning the read to the reference later on. Also, the positions of central base in the motif, for instance "C" for the CpG context, are found on the reference, both for forward and reverse alignment strand. If it is dealt with native dataset, then BED positions shall be extracted, together with the alignment strand information. It is also possible to filter the BED positions, leaving only the high confidence ones. The intersection between found motif positions and allowed BED positions shall be made. Afterwards, the final motif positions are stored into a variable in a dictionary format, keys being different chromosomes, and values being a pair of two sets, one for forward strand, and one for the reverse one.

Next step is aligning read to the reference, and getting the relevant motif positions, in a way that it is observed which of the previously found motif positions are covered by the alignment, i.e. also match position on the query.

Usually, after applying Tombo's re-squiggle algorithm an event table is produced, which contains a sequence of k-mers with reference to the interval in the raw signal. With having Tombo removed from the pipeline, this event information must be extracted from basecalled data, output which is returned by Guppy. By applying certain manipulations of the basecalled data[12], raw signal intervals are obtained and given to

[12] `https://community.nanoporetech.com/posts/mapping-of-signal-to-basec`

the resolve insertions method as an input, together with alignment obtained before with Mappy. Alignment contains information about the start and end index of query, which can then be used to retrieve the first and last signal interval corresponding to that nucleotide sequence. Therefore, it is exactly known which signal points belong to which nucleotides.

Steps 5. and 6. are not going to be explained into further details, because they have already been examined in sections 4.1 and 4.2, respectively.

Finally, the program loops through the relevant positions and extracts important information for each of them. The resegmentation data is structured in the following way:

- **position** - central base position on the reference
- **event_intervals** - raw signal intervals mapped to the bases on the reference
- **event_lens** - lengths of the event intervals
- **bases** - bases in the relevant region extracted from the reference (e.g. 17-mer for motif "CG" and window equal to 8 shown previously in Figure 2.4)

Even though, the remaining part of the pipeline looks the same as in the original implementation, quite a lot of changes needed to be done in order for whole pipeline to work correctly. First of all, Binary Writer is implemented as previously described in Section 4.3 and it outputs data.bin, a binary file containing examples, i.e. extracted data for relevant positions.

Training portion of the code can now easily read mentioned binary data file, because offsets are given in the binary header file. Additional changes must be made in the training as well because signals differ in length. First, bases in the 17-mer should be repeated based on event lengths, and not constantly 20 times as before. Secondly, padding containing zeros must be added to the beginning and the end of the signal in order to obtain same tensors. Signals which are too long are cut in a way that we keep the middle of the signal. At last, convolutional layers should be adjusted based on the new lengths (they are not always 340). After all of that successfully implemented, modification probabilities can be again obtained as the final output of the pipeline.

## 5.3. Training procedure

The model is trained for 30 epochs in mini-batches of 256 examples, using the back-propagation algorithm. Binary cross-entropy loss is used in order to measure the performance of the model. For training, AdamW optimizer (Loshchilov i Hutter, 2018),

which implements weight decay regularization for Adam algorithm (Kingma i Ba, 2014), and cyclic learning rate scheduler (Smith, 2015) are utilised. Learning rate is cyclically changed between lower and upper boundary, which equal to $10^{-5}$ and $10^{-4}$, respectively. After each cycle, upper bound is changed to half of the value of previous upper bound. One step size represents half cycle and is equal to $4 \times iterations\_in\_epoch$ .

# 6. Results

This chapter offers an overview of the obtained results divided into two parts, measuring and comparing execution time for different methods, and comparing the accuracy of methods. Later on, a brief commentary on the obtained results is given, as well as the propositions for the future work in the field.

Table 6.1 summarizes specifications of the machine on which all of the tests have been carried out.

**Table 6.1:** Machine specifications

| | |
|---|---|
| Operating system | Ubuntu 4.4.0-124-generic |
| Processor architecture | x86_64 |
| Processor | Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GH |
| Number of cores | 32 |
| Number of GPUs | 1 |
| RAM | 566GiB |

The assumption, and the very goal of replacing the Tombo framework, is that the runtime will be shorter than before. Tombo uses a more complex heuristic than those developed in this thesis, hence the execution time should be greater, but for the same reason, it is expected that the accuracy achieved with Tombo is higher than with the Remapper tool. Nevertheless, we hope that heuristics developed in this thesis will yield good enough accuracy, while at the same time lowering the runtime.

## 6.1. Runtime

Runtime is first measured for the original Rockfish code, including basecalling using Guppy, re-squiggling using Tombo, and finally feature extraction. Secondly, runtime is measured for the Rockfish code including Remapper which only has feature extraction

step. The measurements are conducted three times and the average value is taken, since there have been small differences across the measurements.

The results for E. coli dataset are presented in Table 6.2. Original Rockfish with Tombo, Remapper with the first deletion method (Concatenate and divide), and Remapper with the second deletion method (Longer neighbour) are compared. Guppy runs in the GPU mode and basecalling is done with 28 basecallers (parameter n_producers in Remapper), for all of the methods. Furthermore, 4 processors (parameter workers in Rockfish, and n_processors in Remapper) have been used to process the basecalled reads in all of the cases. Lastly, bed_filter parameter is set to high_confidence, meaning that only positions that have coverage of at least 10 reads and have all reads modified or unmodified (modification frequency of 0% or 100%) are taken into consideration. The remaining parameters are set to the default values. All of the measurements are conducted on the same machine and with the same setting, thus achieving comparable results.

**Table 6.2:** Runtime comparison for Escherichia coli dataset [sec]

| **Model** | **Modification** | **Guppy** | **Tombo** | **Feature extraction** | **Total** |
|---|---|---|---|---|---|
| Rockfish | mod | 19.173 | 53.393 | 81.043 | 153.609 |
| | nomod | 20.323 | 58.520 | 89.270 | 168.113 |
| Remapper - del #1 | mod | - | - | 25.237 | **25.237** |
| | nomod | - | - | 30.757 | 30.757 |
| Remapper - del #2 | mod | - | - | 25.640 | 25.640 |
| | nomod | - | - | 27.287 | **27.287** |

Table 6.3 summarizes runtime results for NA12878 human dataset, which is a native dataset, therefore mod/nomod differentiation cannot be shown. The measurements are obtained for the same methods and in the same conditions as described above for E. coli dataset.

**Table 6.3:** Runtime comparison for NA12878 human dataset [hh:mm:ss]

| Model | Guppy | Tombo | Feature extraction | Total |
|---|---|---|---|---|
| Rockfish | 03:23:25 | 17:03:18 | 00:39:09 | 21:05:52 |
| Remapper - del #1 | - | - | 04:01:08 | **04:01:08** |
| Remapper - del #2 | - | - | 04:01:29 | 04:01:29 |

## 6.2. Accuracy

In classification problem, accuracy means the fraction of predictions the model got right. Or, more formally,

$$Accuracy = \frac{1}{N}\sum_{i}^{N} 1(y_i = \hat{y}_i) \tag{6.1}$$

where $y$ is a tensor of target values, and $\hat{y}$ is a tensor of predicted values.

Table 6.4 presents validation accuracy measured on the native human dataset. Data is split in the way previously shown in 3.1. The accuracy is measured for original Rockfish implementation, for implementation which includes Remapper with Concatenate and divide deletion method, and for the one with Longer neighbour deletion method.

**Table 6.4:** Validation accuracy comparison for NA12878 human dataset [%]

| Model | Accuracy |
|---|---|
| Rockfish | **93.400** |
| Remapper - del #1 | 82.107 |
| Remapper - del #2 | 83.978 |

The training is performed on two data.bin files obtained as outputs of feature extraction method for training and validation data. All of the parameters are set to the default values.

## 6.3. Discussion

The main goal of this thesis is to find a heuristic which would replace Tombo and whose performance is similar to the original model, but the speed is improved. From taking a glance at Table 6.2 and 6.3 it can be concluded that we have suceeded in the latter. Remapper is faster than Rockfish in both of its implementations, regarding the way of resolving deletions. All of the methods are slower for unmodified reads which could possibly be attributed to them having longer signals in average (see Subsection 3.3.1). The difference in runtimes might not seem so drastic on 2000 E. coli reads, but when the program is scaled for a large amount of data (406,821 human reads), the difference in speed is indeed noticeable.

On the one hand, for the original Rockfish implementation data has to be read from files and written back to files three times, once in the basecalling portion, once while re-squiggling with Tombo, and lastly while performing data extraction. These operations are costly, and together with long runtime of the Tombo part, cause long overall runtime. From Table 6.2 containing E. coli data it appears that feature extraction is the slowest part, but in reality, when looking at Table 6.3 comprising a much bigger human dataset, it can be concluded that Tombo is the real problem, confirming the initial statement made in introduction.

On the other hand, this thesis' feature extraction implementation requires only the raw data as the input, performs all of the work, and outputs resulting file ready for training. This approach is faster, easier, and requires less storage, as there exist no intermediate steps where files need to be stored. Moreover, reads which have mapping quality below set threshold or no alignment is produced for them are discarded early on in the program. Reads which do not contain any relevant regions, for instance if all of them are filtered out using the bed_filter parameter, are also discarded.

In conclusion, runtime for the two Remapper variations are quite similar to each other - differences are measured in seconds for both datasets, and may even be contributed to slight deviations across the measurements. However, runtime for Remapper implementation is 5 to 6 times faster than for the Rockfish one across both datasets.

As it has been anticipated, Rockfish with Tombo achieves higher accuracy than both Remapper implementations. Furthermore, Remapper with the Longer neighbour method for resolving deletions performs slightly better than the one with Concatenate and divide method. The reason probably lies in previously explored assumption that the signal points from deleted base are assigned to the neighbour who has more points, therefore points from the longer neighbour should be assigned back to the deleted base.

This theory appears to achieve better results than dividing the points uniformly from both neighbours.

## 6.4. Future work

In future work, the parameter space might be explored more thoroughly, since there is a large number of parameters in the implementation, which leaves a justified doubt that optimal configuration has not yet been found.

Furthermore, methods for resolving insertions and deletions are written in a way that they can easily be modified without affecting the rest of the pipeline. Thus, possible improvements may lie in implementation of a different remapping strategy, for instance looking at more than one neighbour at each side, applying certain statistical tests alike Tombo, etc. However, keeping the rest of implementation is advised, since there has been put a lot of effort to write it optimally, and it achieves remarkable results regarding the runtime.

# 7. Conclusion

The field of epigenetics shows a great potential for helping detect tumors and other diseases early on. DNA modification detection is crucial for discovering possible epigenetic abnormalities, hence, in the last few years, multiple tools have been developed to tackle said detection. The main concern of this thesis is the development of a method for detecting DNA base modifications using DNA Nanopore sequencing and deep learning methods, thus giving a contribution to this exciting and important field.

This thesis' implementation is built on top of Rockfish, a state-of-the-art deep learning model based on transformer architecture. The idea is to replace Tombo, a bottleneck in the Rockfish pipeline, with a faster method. The data is thoroughly analysed in order to devise the most suitable solution, called Remapper. The Remapper method consists of aligning basecalled reads to the reference using Mappy and remapping signal points at indels. To resolve insertions Half-half method is developed, whereas to resolve deletions two approaches are proposed, Concatenate and divide method, and Longer neighbour method.

The solution is implemented in Python with the help of PyTorch library. The runtime of data extraction, including basecalling and Remapper method, is evaluated on both E. coli and NA12878 human dataset, whilst the model is trained solely on the human data.

It is shown that Remapper is 5 to 6 times faster than Rockfish in both of its implementations and on both datasets. However, obtained speed comes with a cost of lowering the modification detection accuracy. Rockfish has the highest accuracy, Remapper with the Longer neighbour method follows, and Remapper with the Concatenate and divide method is the last.

To conclude, remarkable results regarding the runtime are achieved, though at the cost of reduced, but still reasonable, accuracy. The implementation is designed to enable easy improvements, thus different remapping methods could be explored, alongside different parameters, to obtain a higher accuracy.

# BIBLIOGRAPHY


CDC. What is epigenetics?, 2020. URL https://www.cdc.gov/genomics/disease/epigenetics.htm.

ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57, 2012.

Cathérine Dupont, D Randall Armant, i Carol A Brenner. Epigenetics: definition, mechanisms and clinical perspective. U *Seminars in reproductive medicine*, svezak 27, stranica 351. NIH Public Access, 2009.

Laura Elnitski. Epigenetics, 2021. URL https://www.genome.gov/genetics-glossary/Epigenetics.

Ming He, Xu Chi, i Jie Ren. Applications of oxford nanopore sequencing in schizosaccharomyces pombe. U *Yeast Protocols*, stranice 97–116. Springer, 2021.

Dan Hendrycks i Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 2016.

Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345, 2018.

Merck KGaA. Introduction to dna methylation, 2021. URL https://www.sigmaaldrich.com/HR/en/technical-documents/technical-article/genomics/epigenetics/introduction-to-dna-methylation.

Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34 (18):3094–3100, 2018.

Qian Liu, Li Fang, Guoliang Yu, Depeng Wang, Chuan-Le Xiao, i Kai Wang. Detection of dna base modifications by deep recurrent neural network on oxford nanopore sequencing data. *Nature communications*, 10(1):1–11, 2019a.

Qian Liu, Daniela C Georgieva, Dieter Egli, i Kai Wang. Nanomod: a computational tool to detect dna modifications using nanopore long-read sequencing data. *BMC genomics*, 20(1):31–42, 2019b.

Ilya Loshchilov i Frank Hutter. Fixing weight decay regularization in adam. 2018.

Peng Ni, Neng Huang, Zhi Zhang, De-Peng Wang, Fan Liang, Yu Miao, Chuan-Le Xiao, Feng Luo, i Jianxin Wang. Deepsignal: detecting dna methylation state from nanopore sequencing reads using deep-learning. *Bioinformatics*, 35(22):4586–4595, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Arthur C Rand, Miten Jain, Jordan M Eizenga, Audrey Musselman-Brown, Hugh E Olsen, Mark Akeson, i Benedict Paten. Mapping dna methylation with high-throughput nanopore sequencing. *Nature methods*, 14(4):411–413, 2017.

Valerie A Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, Paul A Kitts, Terence D Murphy, Kim D Pruitt, Françoise Thibaud-Nissen, Derek Albracht, et al. Evaluation of grch38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome research*, 27 (5):849–864, 2017.

Danielle Simmons. Epigenetic influences and disease, 2008. URL `https://www.nature.com/scitable/topicpage/epigenetic-influences-and-disease-895/`.

Leslie N Smith. No more pesky learning rate guessing games. *CoRR, abs/1506.01186*, 5, 2015.

Marcus Stoiber, Joshua Quick, Rob Egan, Ji Eun Lee, Susan Celniker, Robert K Neely, Nicholas Loman, Len A Pennacchio, i James Brown. De novo identification of dna modifications enabled by genome-guided nanopore signal processing. *BioRxiv*, stranica 094672, 2016.

Dmitry Ulyanov, Andrea Vedaldi, i Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, i Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Bob Weinhold. Epigenetics: the science of change, 2006.

Ryan R Wick, Louise M Judd, i Kathryn E Holt. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome biology*, 20(1):1–10, 2019.

Zaka Wing-Sze Yuen, Akanksha Srivastava, Runa Daniel, Dennis McNevin, Cameron Jack, i Eduardo Eyras. Systematic benchmarking of tools for cpg methylation detection from nanopore sequencing. *bioRxiv*, 2020.

**Detection of Modified Nucleotides Using Nanopore Sequencing and Deep Learning Methods**


**Abstract**

In this thesis, a method for DNA base modification detection is developed using Nanopore sequencing and deep learning principles. Nanopore sequencing detects different electrical current signals for different nucleotides, whilst a DNA strand passes through a nanopore. Then, basecalling is performed, translating the detected signals into a DNA sequence. After aligning basecalled reads to the reference, signal points at insertions and deletions should be remapped, using the developed Remapper method. The implementation achieves significant results regarding the runtime, though at the cost of reduced accuracy. Source code is available at `https://github.com/sanjadeur/master-thesis/`.



**Keywords:** epigenetics, DNA modification, 5mC methylation, CpG context, Nanopore sequencing, deep learning


**Određivanje modificiranih nukleotida koristeći sekvenciranje nanoporama i duboko učenje**


**Sažetak**

U ovome je radu razvijena metoda za detekciju modifikacija baza u DNA, koristeći sekvenciranje nanoporama i principe dubokog učenja. Sekvenciranje nanoporama detektira različite električne signale za različite nukleotide, za vrijeme prolaska DNA lanca kroz nanoporu. Zatim se provodi pretvorba detektiranih signala u DNA sekvencu. Nakon poravnanja očitanja na referencu, točke signala na mjestima insercija i delecija trebaju se remapirati, koristeći razvijenu metodu Remapper. Implementacija postiže značajne rezultate što se tiče vremena izvođenja, međutim s cijenom smanjenja točnosti. Izvorni kod dostupan je na `https://github.com/sanjadeur/master-thesis/`.



**Ključne riječi:** epigenetika, modifikacija DNA, 5mC metilacija, CpG kontekst, sekvenciranje nanoporama, duboko učenje