Improving Cancer Prediction by Optimizing cfDNA Error Model

Forbedring af Kræft Prædiktion Ved Optimering af cfDNA-Fejlmodel

Marcus Kjærgaard Olesen & Sofie Schubert Elving

Supervisors: Jakob Skou Pedersen
Mathilde Diekema

Department of Mathematics
Aarhus University

This thesis is submitted for the degree of
Bachelor (BSc) in Data Science

June 2024

Acknowledgements

We would like to express our gratitude to all those who have contributed to the completion of this project.

First and foremost, we thank our advisors, Jakob Skou Pedersen and Mathilde Diekema, for their guidance, patience, and support throughout the project. Their insightful feedback and encouragement have been invaluable in shaping this thesis.

We extend our sincere thanks to Mathilde Diekema and the anonymous reviewers whose detailed comments and suggestions have significantly improved our work. Your thorough and thoughtful reviews were immensely helpful.

Special thanks to Simon Opstrup Drue for providing the code used for creation of plots in the DREAMS paper, and Mathilde Diekema for providing technical assistance with cancer calling.

Finally, we thank everyone who has directly or indirectly contributed to this project. Your support, whether big or small, has been crucial, and we are deeply appreciative of all your efforts.

Abstract

Cirkulerende tumor-DNA (ctDNA) frigives fra kræftceller til blodet og udgør en lille del af den totale mængde cirkulerende cellefrit DNA hos patienter med kræft. ctDNA-målinger kan derved benyttes til at idetificere og karakterisere tumorer. På grund af en kort halveringstid er ctDNA også oplagt til monitorering af behandlingsrespons og opfølgning hos kræftpatienter. Dog er fraktionen af ctDNA i blodet ofte meget lille, hvilket gør det vanskeligt at skelne kræftmutationer fra somatiske mutationer uden klinisk betydning.
I dette projekt arbejder vi derfor på at optimere en fejlmodel, i form af et neuralt netværk, der kan skelne kræftspecifikke mutationer fra støj. Vores arbejde bygger videre på DREAMS metoden, der anvender et neuralt netværk til lavfrekvent variantkald og detektion af ctDNA fra sekventeringsdata. Specifikt har vi fokus på modellering af segmenter af nukleotider, der indsættes (insertion) eller fjernes (deletion) fra DNA. Vi lykkedes med at forbedre sensitiviteten af kræftforudsigelser med 4.75%, ved en specificitet på 95%, svarende til 21.6% af de detekterbare tilfælde. Dette understreger potetialet i at udarbejde præcise fejlmodeller til brug i DREAMS opsætningen for effektivt at kunne opspore kræft.

Table of contents

Abbreviations

AUC area under the curve.

bp base pair.

cfDNA circulating free DNA.

CHIP clonal hematopoiesis of indetermined potential.

CRC colorectal cancer.

ctDNA circulating tumor DNA.

CV cross-validation.

DNN deep neural network.

DREAMS Deep Read-level Modelling of Sequencing-errors.

DREAMS-*cc* DREAMS cancer calling.

DREAMS-*vc* DREAMS variant calling.

FNN feedforward neural network.

INDEL insertion and deletion.

LOCO “leave-one-covariate-out”.

MLP multilayer perceptron.

MNV multi-nucleotide variant.

MRD minimal residual disease.

NGS next-generation sequencing.

NN neural network.

PBMC peripheral blood mononuclear cell.

pre-OP preoperative.

RFE Recursive Feature Elimination.

ROC receiver operating characteristic.

SNV single nucleotide variant.

TNC trinucleotide context.

UMI unique molecular identifier.

UMIseq ultra-deep mutation-integrated sequencing.

Background & Review

1.1 Overview & Scope

In recent years, liquid biopsy has become a promising tool for early cancer diagnosis and monitoring [2, 17]. In particular, circulating tumor DNA (ctDNA) is of special interest as a biomarker that can aid in detecting, identifying and characterizing tumors [4, 15, 45].

Circulating free DNA (cfDNA) consists of DNA fragments released into the bloodstream through apoptosis, necrosis, and active secretion from various cell types [26]. The portion of cfDNA that originates from tumor cells is known as ctDNA. A positive correlation between the concentration of ctDNA and the tumor burden of a patient has been reported [1, 39]. Since the half-life of cfDNA is shorter than two hours [10], ctDNA measurements essentially provide real-time insights into the tumor burden at a given moment. Moreover, compared to conventional tissue biopsy procedures, liquid biopsies, such as plasma from blood samples, are both more cost-effective and less invasive. Hence, effective techniques for ctDNA detection will prove invaluable for an array of applications. These include real-time monitoring of disease progression, targeted treatment in precision cancer medicine, and assessment of treatment response and resistance development [6, 11, 15]. This enables detection of minimal residual disease (MRD) in patients who have undergone curative surgery, and ultimately early detection of cancer in screening programs [9, 19, 23, 37].

However, the primary challenge in ctDNA applications is the weak tumor signal, often comprising less than 0.1% of the cfDNA, reflecting the low tumor burden. Since most widely used sequencing techniques observe error rates on the same order of magnitude as the levels of ctDNA, it becomes difficult to discern sequencing errors from true tumor-derived mutations [36, 48].

This thesis addresses the challenge by adapting and improving the modelling of expected errors using the DREAMS approach [9], that outperforms state-of-the-art methods Mutect2 [3] and Shearwater [21]. Specifically, we aim to improve the modelling of INDEL variants occurring in cfDNA by optimizing a neural network (NN).

1.2 Errors in cfDNA

In humans, genomes consist of the 23 pairs of chromosomes that are located in our cells. The chromosomes hold our DNA, which is made up of four different nucleotides; adenine (A), thymine (T), guanine (G) and cytosine (C). These attach to each other in a pairwise manner, with A and T, and G and C, respectively creating a genomic base pair (bp). However, when these pairings do not match the expected reference genome, it is unclear whether the variant is a tumor-derived mutation or an error from other types of damage. Several variants that may occur: A single nucleotide variant (SNV) is when a single nucleotide substitutes another, resulting in different pairings than the aforementioned base pairs. Insertion and deletions (INDELs), on the other hand, refers to a small length of DNA that has either been inserted to or deleted from the genome [20]. Other variants include phased variants and multi-nucleotide variants (MNVs) [20]. In Fig. 1.1 below are examples of some of these mismatches.

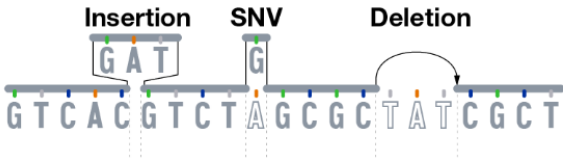


Fig. 1.1: Examples of a SNV and insertion and deletions (INDELs). Figure by [46]

SNVs are more common than the other variants. Phased variants, MNVs, and genrally INDELs, are charactarized by a higher complexity compared to SNVs [18]. This complexity often correlates with a low error rate, comparatively, making them attractive targets for the detection of low-frequency variants [30]. Furthermore, it is reported that INDELs can sometimes have a larger impact on health and disease [46]. In this thesis we will be mainly focusing on detection of INDEL variants.

The challenge of distinguishing between tumor-derived mutations and other errors is further increased since cfDNA can accumulate errors at several stages between its initial release into the bloodstream and sequencing (Fig. 1.2). The cfDNA in the blood may be damaged both in vivo and in vitro. In vivo, this damage could be due to various factors such as circulatory shear stress, degradation by nucleases in the blood or immune response, among others [7, 32]. In vitro, errors could be induced by collection technique, handling, and processing, ect. [7, 32]. Errors may also be introduced at each PCR duplication during amplification of the sequencing library, as well as from artifacts introduced either during the PCR amplification process or the sequencing itself [32].

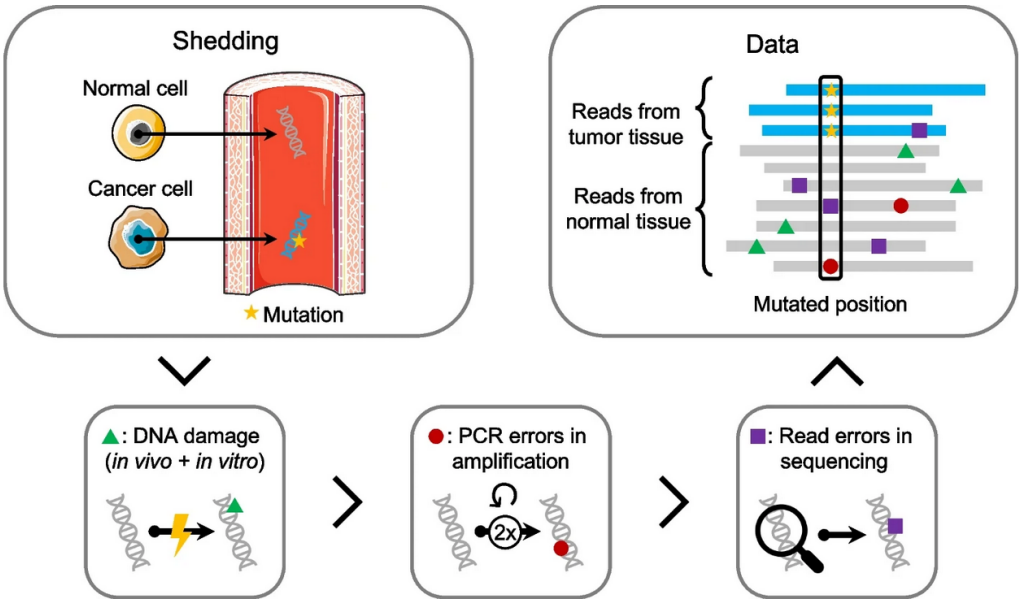


Fig. 1.2:
Occurrence of errors in next-generation sequencing (NGS) data. Normal cells and cancer cells shed DNA into the bloodstream, as described in section 1.1. The cfDNA from normal cells and cancer cells are represented as gray and blue, respectively. The shapes indicate different variations, such as cancer mutation (yellow star), damage in vivo + in vitro (green triangle), PCR errors in amplification (red circle) and errors accumulated during sequencing and mapping (purple squares). The final data contains mapped reads, where some mismatches are errors, and others are mutations from tumor cells. Figure from [9]

1.3 UMIs

Using unique molecular identifiers (UMIs), some of the errors that occur during PCR amplification and sequencing can be rectified (Fig. 1.3). By using these UMIs, each DNA fragment is uniquely labeled prior to the PCR amplification. This allows replicates from the same fragments to be grouped together, enabling systematic removal of rare errors, i.e. errors that are not found in most replicates and are thus likely to have occurred during PCR amplification or sequencing.

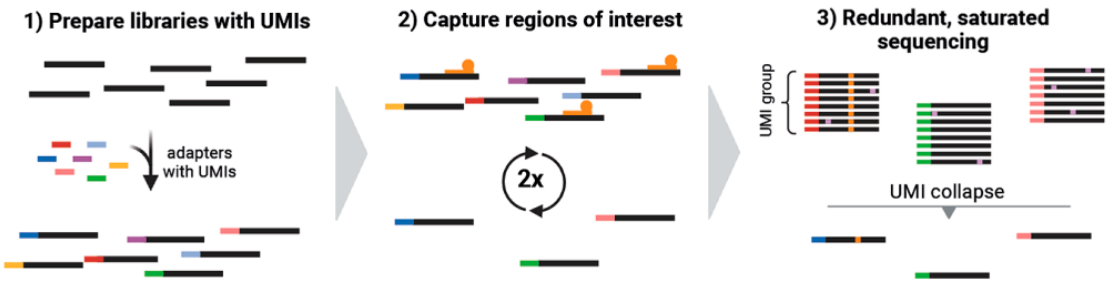


Fig. 1.3: Illustration of a workflow using UMIs. UMI-containing sequencing adapters are introduced during library preparation. Hereafter, genomic regions frequently mutated in colorectal cancer (CRC) are enriched using hybridization-based capture. Finally, redundant, saturated sequencing is applied to guarantee the creation of UMI-families with at least 3 reads. Each UMI-family is collapsed into a consensus read, thereby eliminating random sequencing errors. Figure adapted from [18]

However, errors that occur prior to the UMI-labeling, e.g., damage by deamination or oxidation, are not eliminated. It is these errors we aim to model. A model that approximates general read-level error behaviors in cfDNA (null model) would enable evaluation of whether an individual error is likely to have occurred as a consequence of pre-UMI-labeling damage, or as a result of a cancer mutation. Since the aim is to model general read-level error behaviors in cfDNA, there is no need for curated data from reads known to belong to ctDNA. I.e., data from any region outside established mutated positions, or from independent normal samples, can be utilized to train the error model. The only caveat being that having similar, or better yet standardized, test- and training-data will yield the best results.

1.4 DREAMS

When it comes to distinguishing errors from true mutations in cfDNA data, several viable approaches exist, including mutation-based approaches able to detect variants specific to cancer. One such approach is coined DREAMS-*vc*, and is a method for variant calling based on DREAMS [9]. As suggested by the acronym, DREAMS employs a deep neural network (DNN) to establish a detailed background model for the expected frequency of sequencing errors at individual read positions. In extension to this method is DREAMS cancer calling (DREAMS-*cc*), whereby it is possible to accurately determine the tumor burden—hence also the overall cancer status—by aggregating the signal from DREAMS-*vc* across a catalog of mutations. The method can be used in both a tumor-informed and tumor-agnostic setting.

By training a DNN on NGS data that incorporates both read-level features and information about the sequence specific context, the DREAMS approach makes it possible to accurately estimate positional error rates. Thereby, the mutational evidence can be evaluated with a statistical test based on the expected frequency of errors. In the sense that a read alignment mismatch, e.g. an INDEL, with a low predicted error rate can provide more evidence of a tumor-derived mutation than a mismatch with a high error rate. In principle, DREAMS is applicable to any NGS data. However, for shallow whole-genome sequencing—rather than deep sequencing of panels—the use of read group context features, that is, using UMI-based error correction, is not applicable.

By leveraging the large amounts of data often available in the setting of non-curated cfDNA data without ctDNA labeling, the DNN can identify subtle, yet significant, predictors of errors that may not ordinarily be captured in simpler models. Moreover, NNs have great flexibility in handling various types of input, making them adaptable even to severe changes in laboratory protocols and conditions. Although, in the case of such changes, the model can, and should, be re-trained to ensure a consistent and precise performance. Another aspect of flexibility becomes especially apparent in that the error model is applicable across all genomic regions—even where training data is sparse to none—since DREAMS aims to model the general read-level error behavior of cfDNA.

Christensen et al., like many others, focused solely on calling SNVs [9]. Consequently, the error model was optimized for that task. Since then, the code has been adapted to also train a model for approximating general positional errors for INDELs, allowing these two error models to work in tandem when using DREAMS-*cc*. However, the overall structure of the DNN responsible for modelling INDEL variants was ported without further optimization. In this thesis, we aim to optimize the model to more accurately model the expected errors of INDELs in cfDNA. To do this, we reconsider various aspects such as the model architecture (i.e., layers and layer sizes), training procedures and training data, feature handling and selection, hyper-parameter tuning, and evaluation.

After optimization, the INDEL model will be applied with the SNV model, using DREAMS-*cc*, to determine if the changes improve cancer calling.

1.5 Neural Networks

Artificial neural networks are a sub-field of machine learning inspired by the human brain's architecture [41]. These networks are typically organized into layers of nodes, or neurons, which communicate via connections. Similar to how synapses in the brain vary in strength, these connections in NNs have associated weights that determine the strength of the signals transmitted between nodes. Information can then be propagated through the network and used for learning the general structure of the data, which can be used on other unseen data.

A NN is well-suited for the task of modelling the expected errors in cfDNA due to its ability to capture complex, non-linear relationships between multiple input features and the target outcome. Specifically, the universal approximation theorems state that feedforward neural networks (FNNs) can learn any continuous function—and any Borel measurable function—*arbitrarily* well given sufficient width [12, 25], and depth [31]. The function estimated (or learned) in this case is the one describing the distribution of

$$P(X_{i,j} = x|\ D_{i,j}),\ \ x \in \{I, D, N\},$$

where $X_{i,j}$ represents the observed outcome in read j at position i—either an insertion (I), a deletion (D), or unchanged nucleotide/ no change (N)—and $D_{i,j}$ is the set of observed features for that read position. Interpreting this as a random event, $X_{i,j}$ can then be considered the outcome of a three-dimensional multinomial distribution with one trial. For a non-mutated, homozygote position, the observed nucleotide should in most cases match the reference nucleotide [9]. Any instances of non-reference nucleotide would be considered errors. In this setup, $P(X_{i,j} = N|\ D_{i,j})$ should be close to 1 if the observed allele matches the reference allele for read position (i, j), and $P(X_{i,j} = v|\ D_{i,j}),\ \ v \in \{I, D\}$ would indicate the error rates for the two variants.

In the paper by Christensen et al. [9], a multilayer perceptron (MLP) is used as the model for predicting the error rate at a given read positions. MLP is the name of a type of modern FNN, consisting of multiple ($\geq$3) dense layers with nonlinear activation functions. Let $\mathscr{M}_{MLP}$ be a MLP with L layers, using the activation functions f^l at layer l. Then for input $z = (z_1, \cdots, z_K)$ of any layer l, where K is the number of nodes,

$$f_k^l(z_k) = \text{ReLU}(z_k) = z_k^+ = a_k^l \qquad \text{for } l \in \{1, \cdots, L-1\}$$

is the activation for the k'th node of layer l and

$$f_k^l(z) = \sigma(z) = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}} = a_k^l \qquad \text{for } l = L$$

is the activation at the k'th node of the output layer, L. All hidden layers use ReLU for activations—to mitigate the vanishing gradient problem when back-propagating—while the last layer employs the SoftMax function to ensure that the output is probabilistic. Moreover, let $W^l = (w_{p,k}^l)$ represent the vector of weights connecting layer $l-1$ and l, such that $w_{p,k}^l$ is the weight between the p'th neuron at layer $l-1$ and the k'th neuron at layer l. Then the output of the network for read position (i, j) is given by

$$\mathscr{M}_{MLP}(D_{i,j}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 D_{i,j}) \cdots))$$

This model learns using two key components: The first being backpropagation, first introduced in [42], which is used to find the gradient of a loss function C, w.r.t the weights of a NN, $\partial C / \partial w_{i,j}^l$. This loss function, also known as cost function, is a measure of the distance between the predicted output $\mathscr{M}_{MLP}(D_{i,j})$ and the target output $Y_{i,j}$, i.e, the observed outcome $X_{i,j}$ when one-hot encoded:

$$C(Y_{i,j}, \mathscr{M}_{MLP}(D_{i,j}))$$

The second component is the optimizing algorithm, that is used to minimize the loss by iteratively updating the weights of the network. In the case of this project the loss function and optimizer are Categorical Cross Entropy (log loss) and ADAM, respectively, following the approach in [9]. Both are common and well-established choices for multi-class classification tasks.

Methods

2.1 Data Characteristics

The data used in this project aims to mitigate several of the challenges mentioned in section 1.2, and was collected using a tumor-informed fixed-panel strategy—ultra-deep mutation-integrated sequencing (UMIseq)—specifically for CRC patients [18]. The data stems from a cohort involving 488 patients, comprising a total of 381 CRC patients and 107 healthy controls (Fig. 2.1). UMIseq has been applied to next-generation sequencing (NGS) data from samples including cfDNA isolated from blood plasma, normal DNA from peripheral blood mononuclear cells (PBMCs), and tumor DNA from biopsies. The data is divided into training, validation, and control subsets. The training set includes 37 healthy controls and 126 stage I-IV CRC patients. The validation set consists of 24 healthy controls, 209 stage I-III CRC patients, 29 patients with polyp-derived cancer, and 17 patients with adenomas. The additional control group includes 46 healthy donors.

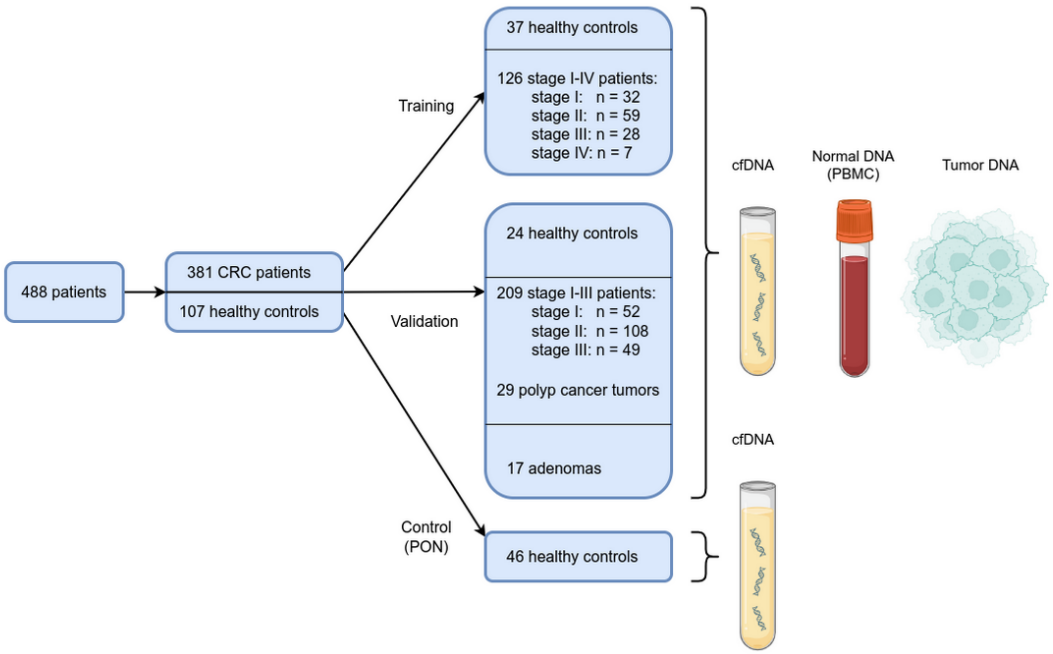


Fig. 2.1: Cohort distribution and sample collection for CRC data

To train the error model, we use the control group, consisting of 46 healthy donors. Specifically, we use a subset of the data which has been sampled in a biased manner such that 50% of the observations (reads) match the reference genome, while the remaining half

of the observations consist of INDELs variants. This 50/50 distribution between matches and mismatches has been done on a donor level, i.e., for each mismatch, a match from the corresponding donor has been sampled. This can be seen in figure 2.2. Moreover, the figure also shows that there is a wide range of sequencing depths between donors. Overall, the biased data comprises 3,300,320 rows and 25 variables (features). Each row corresponds to an observation from the sequenced blood samples of the 46 healthy donors, i.e., the control group. Of the 25 features, 21 consist of observation-specific information; a mix of local sequence-context features and read-level features. A description of each of these features can be seen in Table 2.1 below. The remaining four features are general information about the dataset, such as the total number of matches, total number mismatches, total coverage, and a value for error rate scaling to fit the original mismatch distribution (β), i.e., the probability that a non-error is included in the down-sampled data.

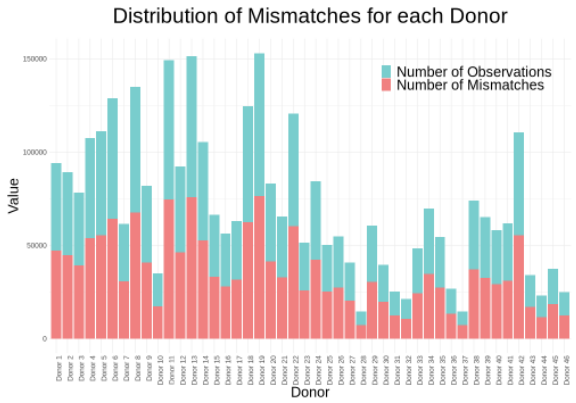


Fig. 2.2: Count of total observations (blue) and count of mismatches (red) for each donor (anonymized). For each donor, there is a 50/50 split of matches and mismatches.

Variable:		**Description:**
qname	*(chr)*	Sample identification.
chr	*(chr)*	Which chromosome read originates from.
genomic_pos	*(dbl)*	Genomic position in the DNA.
obs	*(chr)*	Observation: Observed as insertion (I), deletion (D) or same nucleotide (N) compared to reference genome.
ref	*(chr)*	Corresponding nucleotide on reference genome.
strand	*(chr)*	If the strand is either forward (fwd) or reverse (rev).
first_in_pair	*(dbl)*	Whether strand is first (1) or second (0) in pair.
read_index	*(dbl)*	Base position in the read.
fragment_size	*(dbl)*	Length of the underlying fragment.
ctx_minus1	*(chr)*	Nucleotide preceding the genomic position.
ctx_plus1	*(chr)*	Nucleotide following the genomic position.
trinucleotide_ctx	*(chr)*	Trinucleotide context (TNC) of the genomic position: Consists of the nucleotide at the genomic position as well as the immediately preceding and following nucleotides. Aka. the 3-mer context of the reference genome.
context11	*(chr)*	Similar to TNC, but with an 11 base context rather than 3 base context. This is what is referred to as the local region of the read.
local_complexity_1	*(dbl)*	Shannon entropy of nucleotide, 1-mer frequency: The entropy value indicates the level of uncertainty or randomness in the distribution of nucleotides at that position in the sequence.
local_complexity_2	*(dbl)*	Shannon entropy of dinucleotide, 2-mer frequency: The entropy value indicates the level of uncertainty or randomness in the distribution of paired nucleotides at adjacent positions along the DNA sequence.
local_GC	*(dbl)*	Proportion of nucleotides G (guanine) and C (cytosine) in DNA sequence in local region.
n_insertions_in_read	*(dbl)*	Number of insertions in the read.
n_deletions_in_read	*(dbl)*	Number of deletions in the read.
seq_length	*(dbl)*	Length of sequence.
umi_count	*(dbl)*	UMI group size (number of UMIs making up a consensus read).
umi_errors	*(dbl)*	Number of errors in the UMI group.

Table 2.1: Description of variables contained in the used dataset.
(chr) defines character variables and *(dbl)* defines doubles (floating-point numbers) variables

The *qname* variable includes a donor number, which could, when combined with other information, potentially result in patient identification. Since the data also contains genomic information, it is person sensitive data. Hence, we have taken necessary precautions to ensure compliance with GDPR law, thus securing the registered individuals' privacy rights. As such, all work with the data, has been done through the GenomeDK cluster to keep the donors' information as safe as possible.

Other studies have shown that some read-level and local sequence-context features can influence the rate of errors at individual read positions. These features include length of the fragment (*fragment_size*) [45], position of the read (*read_index*) [38], UMI group size (*umi_count*) [36], GC-content (*local_GC*) [27], proximity to fragment ends (*fragment_size* + *read_index*) [7], and the trinucleotide context (*trinucleotide_ctx*) [33].

Throughout this project, particularly during data exploration, we will use the error rate (ER) to discern the impact of different features on explainability and prediction. The error

rate is calculated as the number of mismatches (INDELs) divided by the total number of observations, as shown below:

$$ER = \frac{\text{Number of Mistmatches (INDELs)}}{\text{Total Number of Observations (Mismatches and Matches)}}$$

However, the dataset is heavily skewed because it was intentionally sampled to have an equal ratio of matches to mismatches. Therefore, to determine the actual error rate observed in the original dataset, from which the data was sampled, we need to correct the error rate by scaling it accordingly:

$$\tilde{ER} = \frac{\beta \cdot ER}{\beta \cdot ER - ER + 1}$$

where β is the aforementioned scaling factor (*info.beta*).

2.2 Cross-Validation

Although we have a large number of observations, they come from a few number of donors ($n = 46$). Moreover, it is wise to assume donor-specific bias in these data points. Therefore, to make the best use of our limited data and simultaneously reduce the probability of overfitting while optimizing our NN model, we decided to employ cross-validation (CV). Specifically, we use K-fold cross-validation, which—although more computationally expensive—generally results in a less biased model compared to other CV methods[14]. K-fold CV provides a more reliable estimate of how well the NN model will generalize to new, unseen data compared to a single train-test split, allowing for better comparison between different models.

In the K-fold method, data is randomly divided into K folds. For each of $k \in [1 : K]$ iterations, the model is trained on the $K-1$ other folds and tested on the last K'th hold-out fold. As such, each time the model is trained, a new fold is used as the test data and the remaining folds as the training data, ensuring that every data-point gets to be in the test set exactly once and in the training data the other $K-1$ times. An example of this procedure can be seen below in figure 2.3:

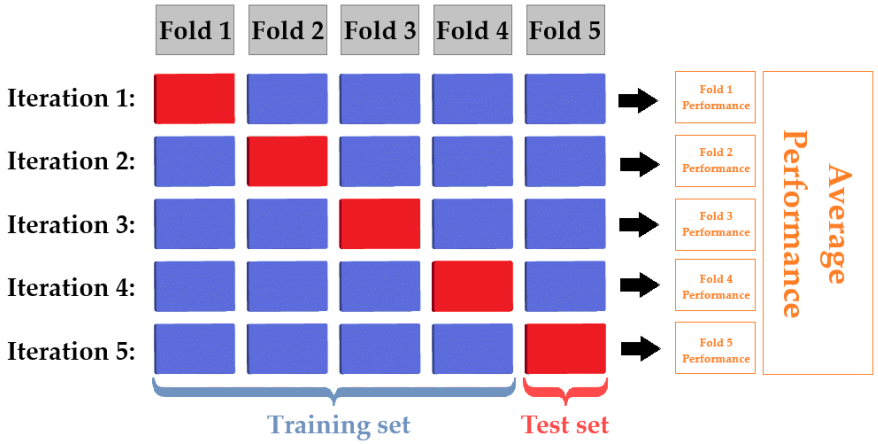


Fig. 2.3: Visualization of 5-fold cross-validation

The choice of K is important in balancing the bias-variance trade-off; higher values of K lead to less biased models but result in higher variance, which may increase the risk of overfitting. Commonly, the choice is between $K = 5$ and $K = 10$, as these values have been shown to perform well without suffering from excessively high bias or very high variance [29]. Considering the available data and the expected batch size, we have decided to use $K = 5$ folds. Moreover, to reduce the possibility of a model learning the characteristics of individual donors, the data has been randomly split into the 5 folds at the donor level. This means that any given donor has observations in only a single fold, ensuring that a model is not trained and tested on data from the same donor. Consequently, the 5 folds are of similar sizes—ranging from $569,512$ to $776,655$ observations per fold—not equal sizes.

2.3 Feature Engineering

We have decided how the data should be used for training the DNN. However, to train the network we also need to provide it with features that may explain the variance in the data. Therefore, we must consider which features to include and how to encode them so we can apply machine learning.

2.3.1 Feature Handling

As mentioned in Table 2.1, our data consists of features that are either doubles (*dbl*) or characters (*chr*), which will be treated as numeric or categorical, respectively. The only exception is *first_in_pair*, which acts as a boolean and should therefore not be considered numeric.

For the categorical features we apply one-hot encoding. In this method, each unique value in the categorical feature is represented by a binary vector. The vector has the same length as the number of unique items, and any such item may be represented by a 1 at the corresponding index of the vector while the rest are 0.
The trinucleotide context (TNC) is somewhat of an outlier and, as such, can be handled in various ways. We could simply employ one-hot encoding, as done with the other categorical features. Four possible bases in a context window of size three could be represented as a 64 (4^3) one-hot vector. However, this representation fails to capture the similarity of certain contexts; for example, ATG and ACG have the same neighboring bases, but the model would not be able to discern this similarity. Alternatively, since TNC is essentially the combination of three categorical features $(ctx\_minus1, ref, ctx\_plus1)$—the reference nucleotide and the neighboring bases—we could apply one-hot encoding to these individual features, thus representing the TNC as three 4-dimensional vectors, using 12 (3×4) input nodes. This implementation allows the model to understand how similar certain combinations are. One critical issue we still face, however, is that the model cannot tell how different contexts might affect the error rate. To address this, the TNC is first one-hot encoded using the former scheme and then embedded into a continuous 3-dimensional vector via learnable weights. This approach is identical to that used in [9], and enables the model to learn to represent the relationships between contexts. This allows it to cluster contexts that have a similar effect on the error rate close together and distinguish those that do not.

Akin to the approach used by Christian et al. [9], we apply Batch Normalization to the numerical features. This is a regularization method used in the context of training DNNs to improve training by reducing internal covariate shift as coined by Ioffe et al. [28]. Later papers suggest that the true value of Batch Normalization lies not in the distributional stability of layer inputs but rather in smoothing the terrain of the loss function [43]. Regardless of the cause, the method helps to generalize the data and avoid overfitting. It also allows for the use of higher learning rates—yielding faster convergence—and requires less careful initialization [5, 28, 43]. The main idea of Batch Normalization is to apply Z-score normalization to the inputs of each hidden layer, i.e., to standardize the scalar features of the current training (*mini*)-batch for each layer [8]. The setup is as follows:
Consider a batch $\mathscr{B}$ of size m. We want to apply Batch Normalization at l layers of the network, each with a d-dimensional input $X = (x^{(1)} \cdots x^{(d)})$; that is, d nodes (or

activations) at layer l. For a given layer we normalize the inputs for each neuron

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathscr{B}}^{(k)}}{\sqrt{\sigma_{\mathscr{B}}^{2\,(k)} + \varepsilon}} \qquad \text{for } i \in [1:m],\ k \in [1:d]$$

where $\varepsilon > 0$ is a small constant, assuring numerical stability by omitting the possibility of division by zero, and $\mu_{\mathscr{B}}^{(k)}$ and $\sigma_{\mathscr{B}}^{2\,(k)}$ are the batch mean and -variance for the k'th activation, respectively, calculated as:

$$\mu_{\mathscr{B}}^{(k)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)}$$
$$\sigma_{\mathscr{B}}^{2\,(k)} = \frac{1}{m}\sum_{i=1}^{m}\left(x_i^{(k)} - \mu_{\mathscr{B}}^{(k)}\right)^2$$

If left as is, this normalization may constrain what the layer can represent. To address this, we have to make sure that the transformation inserted in the network can represent the identity transform [28]. This is done by introducing a pair of learnable parameters, for each activation $x^{(k)}$, responsible for scaling ($\gamma^{(k)}$) and shifting ($\beta^{(k)}$) the normalized values:

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)}$$

γ and β are all initialized as ones and zeros, respectively, thus not re-scaling, nor offsetting the normalization. However, since they are learnable parameters, the model has flexibility to *undo* some (or all) of the normalization at each activation, if that is the optimal thing to do. We see that by setting $\gamma^{(k)} = \sigma_{\mathscr{B}}^{(k)}$ and $\beta^{(k)} = \mu_{\mathscr{B}}^{(k)}$, we could recover the original activations if that was the most suitable setting for the learning process.

2.3.2 Feature Selection

To reduce noise and model complexity, we should ideally only use features that are informative to the task at hand. There are multiple ways to achieve this; however, since this thesis is an extension of the framework of Christian et al., we decided—for ease of comparison—to employ the same methods for feature selection [9]. Specifically, we used a "leave-one-covariate-out" (LOCO) scheme to evaluate the importance of features based on their impact on the model's predictive ability. Subsequently, an optimal subset of features was chosen using a method akin to Recursive Feature Elimination (RFE) [22].

LOCO involves training a full model $\mathscr{M}_{baseline}$ with all n features of interest. Then, for each individual feature of interest, a restricted model $\mathscr{M}_j$ is trained where the j'th feature is removed from the feature set. As described in section 2.2, the available data is split into training data $D_{train}^{(k)}$ and test data $D_{test}^{(k)}$ for k of $K = 5$ iterations. To evaluate the importance of each feature, we measure the drop in performance between the restricted models and the baseline model. This is done by first calculating the change in categorical cross-entropy (loss) for each data point x_i in the test data $D_{test}^{(k)}$:

$$\Delta_i^{(k,j)} = l(p_j^{(k)}(x_i), y_i) - l(p_{baseline}^{(k)}(x_i), y_i), \quad i \in D_{test}^{(k)}$$

Where $l(p_{model}, y)$ is the loss given the predicted probability p_{model}—as a function of x—and the corresponding ground truth y. Then, for each restricted model $\mathscr{M}_j$ and each test data set $D_{test}^{(k)}$, we calculate the feature importance score as the mean change in predictive ability:

$$\bar{\Delta}^{(k,j)} = \frac{1}{\left|D_{test}^{(k)}\right|} \sum_{i \in D_{test}^{(k)}} \Delta_i^{(k,j)}$$

Again, where $k \in [1, K]$ and $j \in [1, n]$.
Using a gaussian approximation, we can construct confidence intervals for the feature importance score of each restricted model $\mathscr{M}_j$ and each test data set $D_{test}^{(k)}$ as follows:

$$\left[\bar{\Delta}^{(k,j)} \pm \Phi^{-1}\left(1 - \frac{\alpha}{2 \cdot n \cdot K}\right) \cdot \frac{s^j}{\sqrt{\left|D_{test}^{(k)}\right|}} \right], \quad s^j = \sqrt{\frac{\sum_{i \in D_{test}^{(k)}} \left(\bar{\Delta}^{(k,j)} - \bar{\Delta}_i^{(k,j)}\right)^2}{\left|D_{test}^{(k)}\right| - 1}}$$

Where the Bonferroni correction has been applied to maintain an overall confidence level of $1 - \alpha$ [16].

Finally, we calculate the average of the means for each restricted model $\mathscr{M}_j$:

$$\bar{\Delta}_{\cdot}^{(j)} = \frac{1}{K} \sum_{k=1}^{K} \bar{\Delta}^{(k,j)}$$

This is the metric used as an overall feature importance score.

After determining the feature importance ranking of the variables of interest, we move on to the next step. Similarly to RFE, presented in [22], we use an instance of backward feature elimination to determine the optimal subset of features: Based on the importance ranking calculated earlier, we recursively eliminate features in increasing order of importance,

with the least important feature being removed first. At each step, just like above, we utilize 5-fold CV and calculate the performance of a model as the mean difference in cross-entropy between the model and the baseline model. Although a little naive, a greedy approach like this gives us an estimate of which features can be left out of the model without any major loss in performance. It also covers for a major weakness of the feature importance ranking: Mainly, if there is high collinearity between some features, i.e., some features convey a lot of the same information, removing any one such feature would not reduce a model's predictive ability a lot. This means they would individually receive a low importance score since any redundant information would be compensated by the remaining correlated features. Overall, the full procedure we will employ for feature selection goes as follows:

Procedure 1 Feature selection

```
s = [1,2,...,n]                # Initialize feature list
M = []                         # Initialize model list
FM = train_model(s)            # Train full model (FM)
r = feature_ranking(FM, s)     # Compute feature importance
                                 (decreasing order)

while r:                       # Until no features are left
    r.pop()                    # Remove the least important feature
    M.append(train_model(r))   # Train model with remaining features

for m in M:                    # For each model
    out = m                    # Final model so far
    for fold in m:             # For each test fold
        if fold < FM:          # If perfomance worse than FM,
            continue           # Then continue
        return out             # Else, return final model
```

This way, the final model is the smallest model that performs similarly to the baseline model. We again apply the Bonferroni correction to compensate for the number of tests when conducting significance testing.

2.4 Regularization

We have already covered the regularization technique known as Batch Normalization. In general, regularization in neural networks consists of useful tools to prevent overfitting and improve generalization of the data. Sometimes, like with Batch Normalization, regularization can indirectly contribute to faster convergence by stabilizing the training process,

thus allowing the model to converge faster. Common regularization techniques include methods such as Early Stopping, Dropout, Weight Initialization, Batch Normalization, and L1- and L2-Regularization (i.e., Lasso and Ridge). Overall, Batch Normalization provides similar regularization benefits to Dropout and often reduces, if not eliminates, the need for Dropout [28]. Regardless, including other forms of regularization could still be useful to further reduce the probability of overfitting and to help with generalization.

Due to long training times, Early Stopping was of key interest among regularization techniques, acting to save computational resources and reduce training time. It involves monitoring the performance of a model during training using a validation set and ending the training process when the performance starts to degrade. Recall from section 2.2 that we incorporated an 80/20 (training/test) split of data using 5-fold CV. To create the validation set, we further split the data used for training into 90% training data and 10% validation data, once again split randomly, but separated on donor level.
The implementation works by adding two additional hyperparameters, *patience* and *min*_Δ [8]. *Patience* determines the number of consecutive epochs the model needs to wait for an improvement in the validation performance before halting the training. This helps alleviate premature stopping due to short-term fluctuations. *Min*_Δ, on the other hand, defines the minimum change in validation performance needed to qualify as an improvement [8]. This ensures a more robust stopping criterion, preventing the training from continuing due to insignificant changes.
The validation data is used after each epoch to evaluate the performance of the new weights. Every time the validation performance is worse than the best performance observed so far, the patience counter is decremented by one. If patience reaches zero, the model training is stopped. If at any point there is a performance increase of at least *min*_Δ, the best validation performance is updated, and the patience counter is reset. Hence, this method helps to further prevent overfitting and significantly reduces training time. Moreover, it removes the need to settle on a specific number of epochs when training and generally provides an approximate idea of how many epochs are necessary for an optimal result.

2.5 Tools & Reproducibility

All models were trained using the Keras library (2.15.0) in R, which is an interface that builds on Tensorflow. Code is entirely written in R (4.3.2), besides the python scripts used for our workflow tool of choice, *gwf* (2.0.5). Code written in this project can be found on the project's GitHub page (Apendix A.1), and the branch of DREAMS used for training INDEL models can be found on the DREAMS GitHub (Apendix A.2).

Results

3.1 Exploratory Analysis of Features

Exploratory data analysis was performed to identify features that might be informative for modelling INDELs in cfDNA. This was done by examining the relationship between various features and the error rate. To this end, multiple plots were created to visualize the interaction between the adjusted error rate (see Section 2.1) and the individual features.

One such visualization, showing the error rate on UMI-group sizes, can be seen in Figure 3.1. As expected, a significant decrease in the error rate is observed in consensus reads formed by larger UMI groups compared to smaller groups. Visually, the distribution of error rates by UMI-group size is similar to that seen in the DREAMS paper by Christensen et al. [9]; however, in this case, much higher error rates are observed for smaller UMI group sizes. Besides using data from another clinical trial, this difference likely stems from Christensen et al. excluding mutations and variants found in germline samples, whereas this project did not.

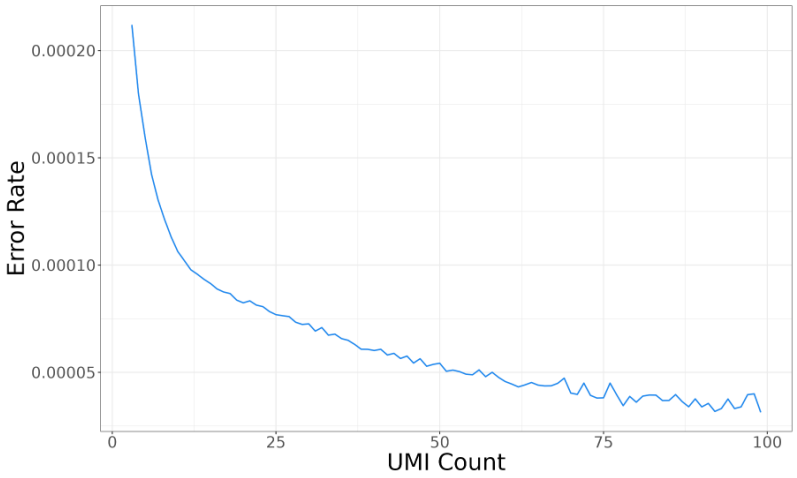


Fig. 3.1: Error rate on UMI-group size (under 100).

Another two plots can be seen in Figure 3.2, where local GC content is inspected. The plot of error rates for different concentrations of guanine and cytosine in local regions of 11 bp (Fig. 3.2b) shows a bimodal distribution of errors, with peaks for both high and low local concentration of GC. To examine whether this distribution of errors can be attributed to the distribution of local GC fractions available in the data, the counts of

local GC concentrations were plotted (Fig. 3.2a). The latter illustrates that the observed fractions of GC are generally evenly distributed, tending to a slight right skew. As such, the count of certain local GC fractions does not seem to be the driving factor in the higher error rate observed in smaller and larger fractions of local GC.

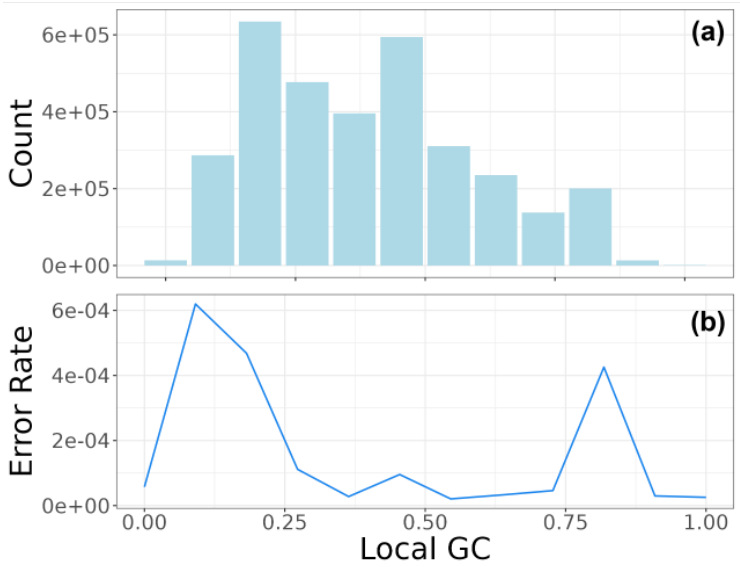


Fig. 3.2: Count of Local GC fractions **(a)** and error rate of Local GC fractions **(b)**.

Although referred to as circulating free DNA, a proportion of cfDNA is likely bound to protein molecules, as nucleosomes or chromatosomes, rather than circulating as free DNA [24, 45]. In eukaryotes, DNA wraps around histone proteins to form nucleosomes, which are connected to each other by linker DNA, creating a bead-like string. It has been shown that the fragment lengths of cfDNA correlate with these binding patterns [9, 17, 45]. Therefore, it is not surprising to observe peaks in the distribution of fragment lengths (Fig. 3.3a) at approximately 167 bp and 321 bp, corresponding roughly to mono-chromatosomal and di-chromatosomal lengths.

Figure 3.3b highlights that the error rate tends to be minimized in fragments of these sizes and that the observed error rate increases significantly for smaller fragments, almost reaching 2.0×10^{-3}.

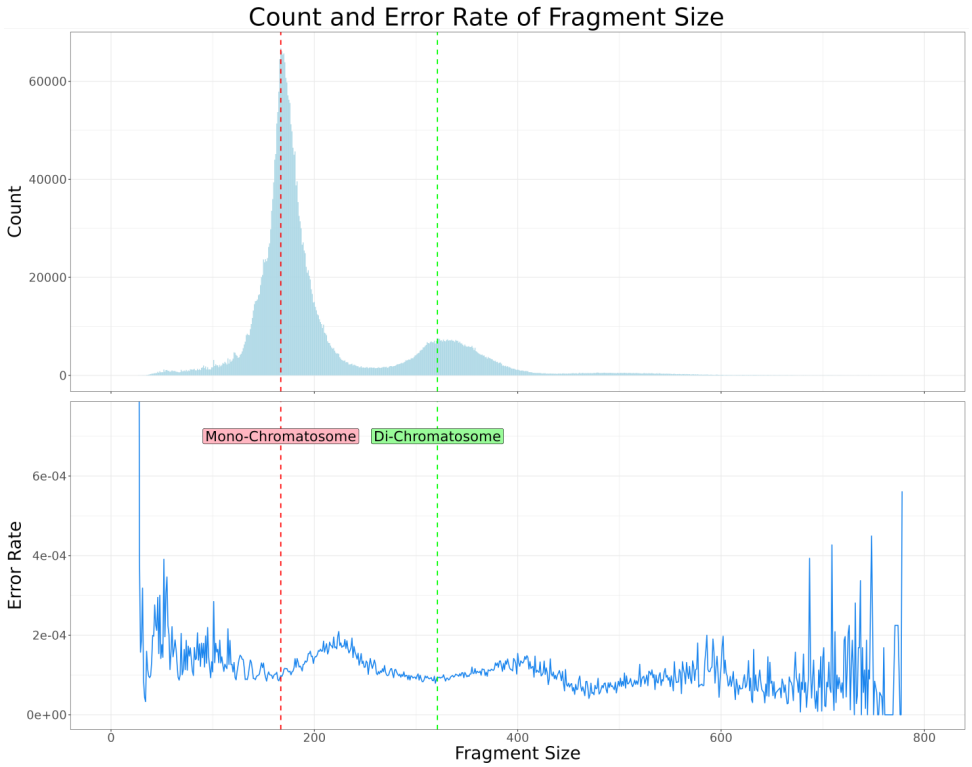


Fig. 3.3: Distribution of fragment sizes **(a)** and error rate by fragment size **(b)**. The red line at a fragment size of 167 and the green line at a fragment size of 321 correspond approximately to the lengths of mono-chromatosomes and di-chromatosomes, respectively.

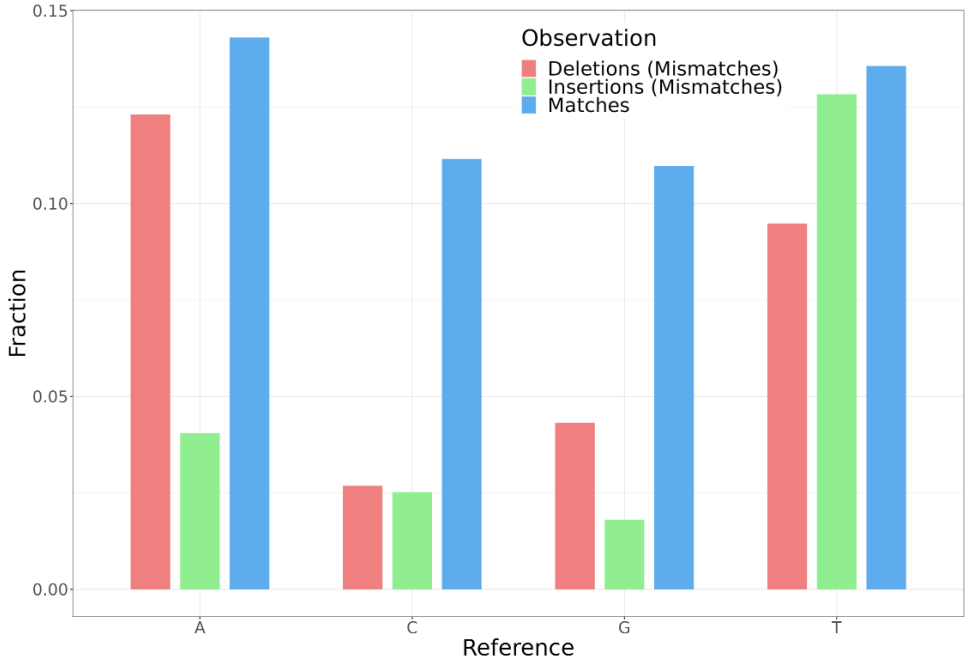


Fig. 3.4: Fraction of the full, downsampled data of different types of observations for each reference.

The eight (4×2) possible alterations within our framework (Nucleotide (N) $\rightarrow$ Insertion (I)/Deletion (D)) showed an unequal distribution (Fig. 3.4). The figure illustrates that mismatches in the data are mostly made up of A$\rightarrow$D and T$\rightarrow$I/D alterations, with approximately two-thirds of the T reference bases exhibiting mismatches. For additional exploration of error rate on features refer to Appendix A.3. Overall, read-level features showed variations in error rate, meaning that for a given genomic position, different reads may have different error rates. In the following, we adapt the DREAMS error rate model for INDELs, to effectively capture this variation to potentially improve detection of ctDNA.

3.2 Model Configuration

The hyperparameters, network architecture, and features from the SNV model, which were ported to the initial non-optimized INDEL model, were made available. Thus, the performance of this initial non-optimized model will serve as a benchmark, since the overarching goal of this thesis is to improve upon the work of Christensen et al.
Recall from Section 1.5 that categorical cross-entropy was used as the loss function. As such, log loss was utilized as a means of evaluating the performance of different models. Analogously to subsection 2.3.2, the individual models were evaluated by first calculating the cross-entropy for each data point in each hold-out fold of our 5-fold CV. Thereafter, the mean loss is computed for each fold. Finally, these means are averaged over all folds to obtain a reliable estimate of a model's performance. To improve the modelling of expected INDELs in cfDNA—thereby improving cancer detection—optimizations will be made in three areas: the architecture of the NN, other hyperparameter tuning, and limiting the input to features that effectively explain the variance in the data.

3.2.1 Network Architecture

The architecture of a NN is crucial for its ability to learn from and generalize data. Moreover, the selected features and hyperparameters of the initial model have been carefully chosen using a mix of domain knowledge and algorithmic exploration. As such, the architecture of the model, i.e., the number and size of the layers, may have the biggest impact on performance. Therefore, the shape of the model will be the first aspect optimized, holding all other factors constant.

Although it is well-established that the architecture of a network significantly influences performance [40], and despite extensive active research in this field, certain aspects remain under-explored and not fully understood; the optimal architecture often varies depending on the specific data and task. Therefore, an exploratory approach was employed to search

Model Name	Network Architecture	# Parameters	Performance (≈)
initial	(128, 64, 32)	12,301	0.1552
initx2	(256, 128, 64)	44,877	0.1504
initx4	(512, 256, 128)	71,469	0.1517
initx8	(1024, 512, 256)	670,413	0.1491
4layer_cone	(256, 128, 64, 32)	46,861	0.1496
5layer_cone	(512, 256, 128, 64, 32)	181,517	0.1473
6layer_cone	(1024, 512, 256, 128, 64, 32)	712,973	0.1493
3layer_cylinder	(128, 128, 128)	35,277	0.1488
4layer_cylinder	(128, 128, 128, 128)	51,789	0.1474
5layer_cylinder	(128, 128, 128, 128, 128)	68,301	0.1555
6layer_diamond	(32, 64, 128, 64, 32)	21,485	0.1592
5layer_point	(128, 64, 64, 64, 32)	20,621	0.1557

Table 3.1: The names, architectures, and number of trainable parameters of the different models that were part of the exploratory search, along with their average cross-entropy performance. The best performing model, marked in green, has an average log loss that is 0.0079 lower compared to the initial model.

for architectures with better performance. The results of this search are presented in Table 3.1. The initial non-optimized model served as the baseline: The network architecture of this model consists of 3 dense hidden layers of 128, 64 and 32 neurons, respectively, and has an average log loss of ≈ 0.155. Drawing from the foundational work on universal approximation in neural networks by Cybenko [12] and Hornik [25], the first exploratory step involved increasing the number of nodes in the layers—for better approximation of the underlying biological function—while maintaining a constant number of layers. The outcome of this change was an initial improvement, as seen in *initx2* (Table 3.1), however the subsequent two models either exhibited poorer generalization (*initx4*) or achieved only a slight improvement at the cost of an excessive number of trainable parameters (*initx8*). Inspired by more recent, and contextually relevant, advances in the domain of universal approximation theorems by Lu et al. [31], the next step in the exploratory search was to concurrently increase the depth of the networks. Keeping with the general shape of the initial network, additional layers were added, resulting the *cone* series of models. Among these were *5layer_cone*, the overall best performing architecture, achieving an average decrease in cross entropy of ≈ 0.0079 compared to the initial model. The observed trend for this type of architecture showed increased performance, but with diminishing returns as the number of trainable parameters increased. By the 6th layer (*6layer_cone*), signs of overfitting began to show. Next, a more common MLP architecture was evaluated: In the *cylinder* series, all layers are uniform in size, each having 128 nodes, with only the network depth varying. With these models, a similar trend and performance to the *cone* series was observed—with *4layer_cylinder* performing marginally worse than *5layer_cone*. Other

architectures preformed worse than the initial model, and were not further explored.

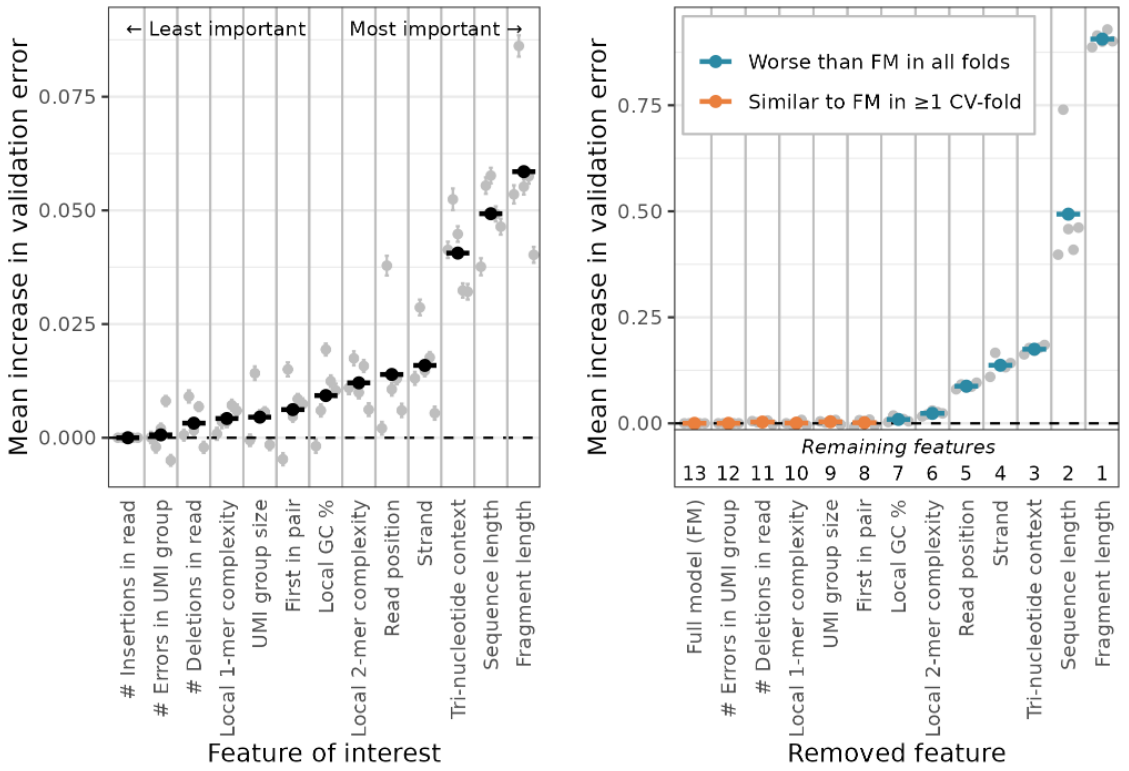
In corroboration with the paper by Ioffe et al. [28], it was found that incorporating Dropout in the model only served to lengthen training time without any increase in performance. Rather, this slightly worsened the model performance, suggesting that the data was underfitted. Architectures that lead to overfitting were fewer than anticipated, indicating that the implemented regularization techniques of Batch Normalization and Early Stopping were working as intended, without any need for additional regularization. Additionally, alternative optimizers were assessed, but the ADAM optimizer remained the best option, as expected.

3.2.2 Key Features

An initial set of features of interest was chosen for the model based on the exploratory analysis of features (see Section 3.1), as well as on the findings of other papers [7, 9, 27, 33, 38, 45]. We assigned an importance ranking to these features using LOCO (Fig. 3.5a)—as described in the first part of subsection 2.3.2. When leaving out the trinucleotide context, instead of removing all context, the feature was replaced with the reference base to deliberately focus on the importance of the neighboring nucleotides.

We found fragment length to have the biggest impact on model performance, suggesting that different sizes of fragments are associated with variations in error rate, as found by others [45]. And similarly to what was observed in [9] and [47], there seems to be a difference in error rate for different contexts, as indicated by the importance of the trinucleotide context. The second, fourth, and fifth most informative features are sequence length, strand, and read position, respectively, revealing that systematic errors might be induced by lab procedures and conditions. Local context features such as the 2-mer complexity and the fraction of guanine and cytosine also contribute to the modelling of errors. The remaining features show diminishing predictive ability, performing better than the baseline model in one or more folds of the CV. In corroboration with [9], we found that read-level features have a significant impact on the accurate modelling of allelic errors and that they seem to be at least as important as sequence-derived local context features.

Exactly as described in the latter part of subsection 2.3.2, we chose an optimal subset of features from the features of interest based on the importance ranking above. The five least important features had little to no impact on the modelling of error rates, and were removed due to performing similarly to the baseline model in one or more folds of the CV. Significance testing using the Bonferroni correction—to compensate for the number of



(a) Feature importance ranking as described in subsection 2.3.2: Features were removed one by one from the full model (baseline) containing all features to measure the increase in validation error. Gray points indicate the mean increase in validation error for each fold of a 5-fold CV. An average of these means is represented by the black points, which indicate the overall importance of a feature. Features are arranged in order of importance, with the right-most being the one with the biggest impact on model performance.

(b) An optimal subset of features was chosen as described in subsection 2.3.2: Based on the importance ranking, the features were cumulatively removed, starting with the least important. At each step, the increase in validation error compared to the full model is calculated for each fold of a 5-fold CV, represented by the gray points. A feature should not be removed if it makes the model significantly worse. Moreover, a feature is only kept if its removal increases validation error in all folds of the CV

Fig. 3.5

tests—found that all but one of the remaining seven features were significant. Specifically, the local CG content might have been included due to random chance. However, we chose to keep this feature in the final set for three different reasons. Firstly, it does not make the model overly complex. Secondly, the restricted model—where it was not included—performed worse than the full model in all folds of the CV. Finally, because the Bonferroni correction is known to come at the cost of increasing type II errors [34, 35]. Therefore, the final feature set includes four read-level features and three local sequence specific context features, namely fragment length, sequence length, strand, and read position and TNC, 2-mer complexity, and the GC fraction, respectively.

3.2.3 Optimal Hyperparameters

For hyperparameter tuning, we decided to conduct a structured hyperparameter search using an adaptive grid (Table 3.2). Due to time constraints, we originally planned to downscale the amount of data used for both training and validation, and speed up the search by training for only a few epochs for each combination of hyperparameters in the grid. This choice was based on [44], which found that models achieving high accuracy typically perform better after a few epochs—a belief that has been reconfirmed in other projects of ours. However, after incorporating Early Stopping in the final stages of the project, we approached it differently. Due to faster training times, we could now apply 5-fold CV on the full dataset, mirroring other parts of the optimization process and exhausting all available information. Although Early Stopping—by proxy of 5-fold CV—enabled more reliable estimates of performance for combinations of hyperparameters, it also introduced two additional hyperparameters (see section 2.4) which we did not have the time to fine-tune. Moreover, the architecture of the DNN was already established, so the hyperparameters we optimized were the learning rate and the batch size, which are still essential for optimizing a neural network.

Hyper parameter	Possible values	Best performing
Learning rate	0.001, 0.005, 0.01, 0.02	0.005
Batch size	16000, 32000, 64000	32000

Table 3.2: Grid used for hyperparameter search, and the result

Values were chosen based on common sense as well as using adopted values from similar papers: In [9], which we are building upon, the values 0.01 and 32000 were used for learning rate and batch size, respectively. We decided to use this as the foundation for an adaptive search. Spanning a search space around these values and then dynamically adjusting the grid based on the results of the initial evaluations.

3.3 Predictive Performance of the Optimized Model

After optimizing the architecture, hyperparameters, and selected features, the final model configuration has been established and is presented in Figure 3.6 along with the DREAMS SNV model from [9].

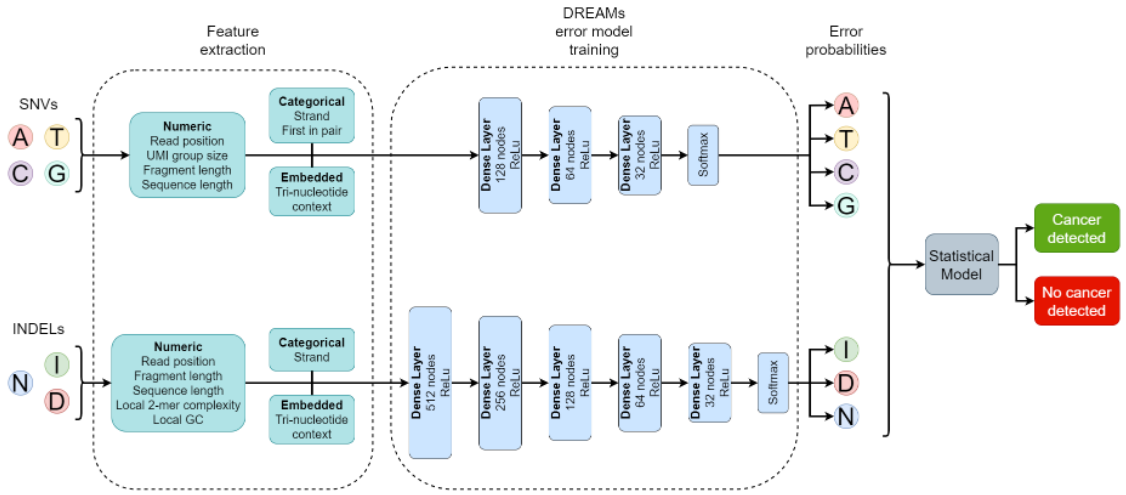


Fig. 3.6: Model architecture of the SNV model and our final INDEL model, along with the final steps of the cancer calling workflow.

3.3.1 Error Estimation on Genomic Positions

To assess our optimized INDEL model's capability to estimate the expected occurrence of errors in cfDNA, we examine the observed error rate against the predicted error rate across all genomic positions. Figure 3.7 below illustrates this for a region with a higher frequency of errors. The predicted error rate generally aligns well with the observed error rate without being overfitted, hence the variation seen throughout.

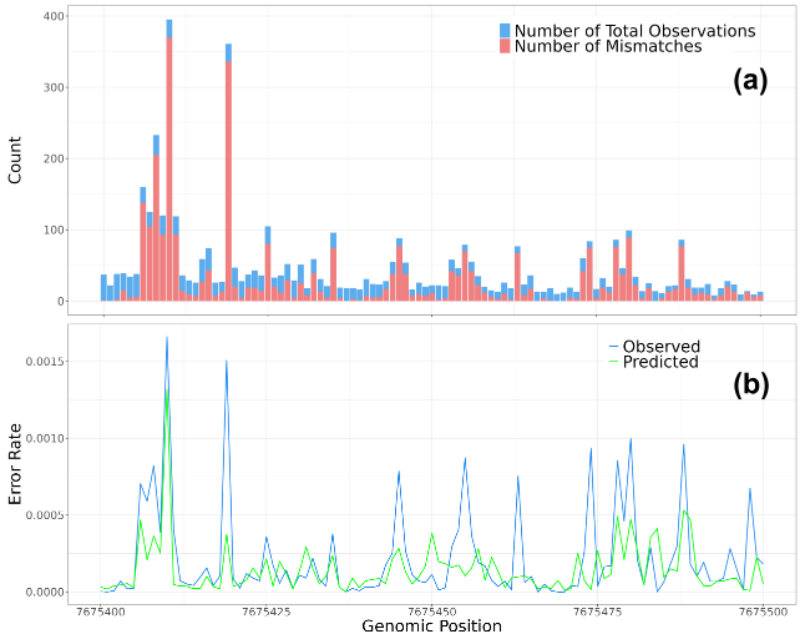
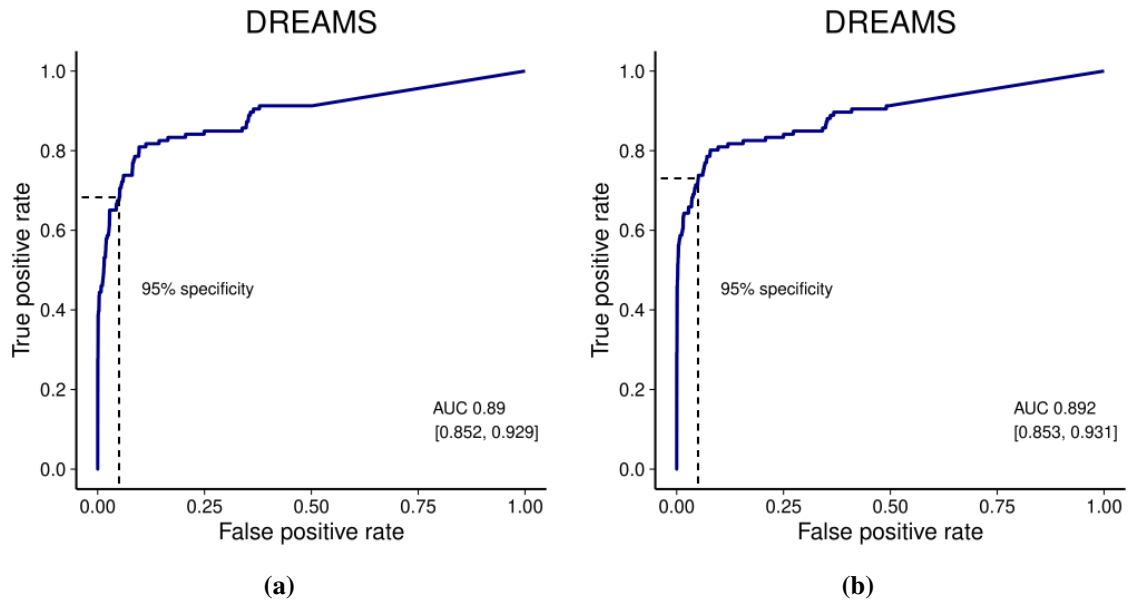


Fig. 3.7: Count of mismatches (INDELs) and observations for genomic position **(a)** and the error rate of observations in the dataset on specific genomic positions, compared with predicted error rate made by the DREAMS INDEL model on certain genomic positions **(b)**.

3.3.2 Tumor Informed Model Prediction

To further assess the performance of our optimized model, we aimed to detect cancer in preoperative (pre-OP) plasma samples from 126 stage I-IV CRC patients using DREAMS-*cc*, following the general procedure outlined in Figure 3.6. Recall from section 1.4 that this cancer-calling method uses a mutation-based approach; DREAMS-*cc* aggregates the mutational evidence across a catalog of known mutations to estimate the tumor fraction with an accompanying p-value for the presence of cancer. These p-values were calculated for the 126 CRC patients—using their respective tumor catalog—and used as positive labels. As negative controls, 1260 (120×10) p-values—representing non-cancer samples—were calculated by applying in silico generated mutation catalogs to randomly selected healthy control cfDNA samples. These samples come from 37 healthy controls that are part of the same training cohort as the 126 stage I-IV CRC patients, as illustrated in Figure 2.1. Each in silico generated mutation catalog was created by sampling n mutations, without replacement, from the 276 unique mutations present in the training cohort, using probabilities equal to a mutation's relative frequency. The number of mutations sampled in each generated catalog (n) follows the same distribution as mutations per patient in the training cohort. Using this setup, receiver operating characteristic (ROC) statistics were computed for the original DREAMS SNV error model, first in tandem with the non-optimized error model for INDELs, then in tandem with the optimized INDEL model; the respective ROC curves for each can be seen in Fig. 3.8a and 3.8a below.



(a) **(b)**

Fig. 3.8: ROC curves for the non-optimized model setup **(a)** and for the optimized model setup **(b)**. Created with the pROC package in R

Using a specificity of 95%, i.e., while limiting the false-positive rate to 5%, the SNV model with the unaltered INDEL model (Fig. 3.8a) achieves a sensitivity of 68.3%. When used together with our optimized model, a 4.8% increase in sensitivity, corresponding to 73.0% was achieved (Fig. 3.8b). The area under the curve (AUC) is similar for both models, being 0.890 and 0.892, respectively, indicating high discriminatory ability for both models. Another iteration, achieved using a smaller learning rate, resulted in a sensitivity of 73,8%, suggesting that additional optimization is possible. Additionally, another iteration achieved a sensitivity of 74.6% using our 5-layer network architecture for both the SNV and INDEL model, while keeping other factors constant (see Appendix A.7). This was explored since the network architecture had the largest impact on model performance for INDELs, and it was important to determine if this also translated to the SNV model.
Of the mutations found in the plasma samples, approximately 22% were INDELs. Therefore, any optimization on the INDEL model can only yield improvements for these mutations. The 4.75% increase in sensitivity—achieved by replacing the initial INDEL model with our optimized version—corresponds to a total increase of 21.6% $((4.75/22) \times 100\%)$ in sensitivity relative to what is possible to detect with the model.

In [9], they report having a true positive rate of 36.6% and 72.1% for stage I- and II CRC patients, respectively, still at a specificity of 95%. This result was produced while only modelling the expected SNVs in cfDNA—without using either the non-optimized INDEL model nor our optimized model—using data from another training cohort from a different clinical trail. However, this result was obtained after excluding variants found in the germline samples. Similarly, we can expect an increase in performance by reducing the signal of clonal hematopoiesis of indetermined potential (CHIP) found in PBMC samples.

Model setups, logs from training, and seed can be found in Appendix A.4, A.5 and A.6, respectively.

Discussion

In this thesis, the aim was to improve the modelling of expected errors in cfDNA to help mitigate the issue of low signals of ctDNA in settings with a low tumor burden. Specifically, to enhance the modelling of INDEL variants by optimizing various aspects of the DREAMS model—a specific approach to modelling sequencing errors using a DNN first introduced by Christensen et al. [9]. The NN error model was assessed by first predicting errors for specific genomic positions and comparing them with the observed error rate. Subsequently, the model was applied for tumor-informed prediction of cancer using an unseen part of the data cohort. An increase in predictive performance was observed when the optimized model was implemented. However, although, the optimized model outperforms the initial model, various obstacles and limitations were encountered throughout the project. These challenges, spanning various aspects of the project, may have influenced the results in different ways.

One such limitation was caused by the method used to sample our data: As briefly mentioned in Section 2.1, the data used for model optimization, was sampled in a biased manner, resulting in an equal split of matches and mismatches for all donors. This approach was taken to address two issues: The issue of having an impractically large number of data points in the full dataset and the issue of having a heavily skewed dataset, where non-errors far exceed the number of errors. Such an imbalance is undesirable for learning the behaviors of non-errors and errors alike. However, because errors become relatively more prominent in the sampled data, this method introduces substantial bias in the estimated class probabilities. To counteract this, Christensen et al. adopted a method of re-scaling the learned class probabilities based on a paper by Dal et al. [13]. This approach of addressing the aforementioned issues has a huge impact on the model's ability to discern between errors and non-errors. Thus, changing the sampling ratio may be an interesting avenue to explore if coupled with measures to prevent the decline of the model's ability to discern. Alternatively, it could be interesting to explore other ways of mitigating these issues, and whether they could impact model-performance positively.

Although a large number of data points would pose an issue in terms of required time and computational resources, more observations may improve the model's generalizability. Due to the large domain of possible feature combinations, some variable combinations might be underrepresented, leading to poorer predictions for similar feature sets. It would be desirable not only to increase the available data but also to reduce bias by introducing more

donors. Having observations from only 46 different donors likely introduced donor-specific bias. This becomes further exacerbated since approximately nine of the 46 donors were used as validation in each CV fold, possibly restricting both the learning and evaluation of the model. During data exploration, it became increasingly clear that the employed sampling scheme induced donor-specific bias. In the sense that, certain feature-values were dominated by matches, mismatches, or no observations due to the values stemming from a single donor. This would not be a problem if that donor is representative for the population, however this is unlikely. Sampling the dataset in a more careful way, as well as increasing the number of donors and their observations, would be a way to mitigate this, since it would reduce the effect of these *outliers* on the learning.

Another obstacle was the time limitation: Between implementing code and getting our bearings, the long training times of the models—especially before incorporating early stopping—forced us to cut the model architecture exploration short. The available time was further reduced due to an implementation mistake in the code for evaluating the models. This error rendered our initial search for architectures obsolete, as the evaluations caused us to grossly overfit the training data. To our dismay, our evaluations led us to believe that performance was continually increasing, even with increasingly large networks and an inflated number of epochs. Overall, this made the model architecture optimization process very slow and tedious. Early stopping was implemented late in the process to save time and simultaneously explore the effects of other regularization techniques, as we were concerned about overfitting despite the evaluation results. This helped speed up training times significantly, which, combined with the library of code written at this point, enabled another exploratory search for high performing architectures after the error was discovered. Had Early Stopping been implemented from the beginning it would have permitted a more thorough search of network architectures in tandem with a more extensive hyperparameter search.

The depth of a network and the number of nodes are hyperparameters that significantly influence the model's performance. Changing these hyperparameters affects the optimal values for other hyperparameters. This brings us to the challenge of determining the order of hyperparameter optimization and the potential bias introduced by this order. Section 3.2.1 explains the reasoning behind starting the initial parameter optimization by testing different model architectures. During the search for optimal network architectures, the same parameters and features as in the SNV model were used. This likely impacted the performance of various model architectures and influenced the final model choice. In hindsight, including all relevant features from the beginning would have been a more disciplined approach. However, the silver lining is that the difference in performance

between the optimal and previous feature sets was subtle, suggesting the induced bias is minimal. Regardless of the hyperparameter optimized first, the possibility of one optimized hyperparameter influencing the next cannot be excluded. A broader search grid of hyperparameters, including the depth and width of the network, could have mitigated this. Unfortunately, this only became feasible after incorporating Early Stopping, at which point we had to move on to other parts of the project to ensure we had a finished model by the end.

The final network architecture, *5layer_cone* (see Table 3.1), was the best performing, albeit only marginally. We found that *4layer_cylinder* performed similarly while being only about a fourth of the size of *5layer_cone* in terms of trainable parameters. This also implies much faster training times. Although CV was used to provide more reliable performance measures, there is still some randomness associated with the initialization of weights and the batch-wise shuffling of data, which could cause performance to slightly vary. Therefore, one could argue that *4layer_cylinder* would be a more suitable network architecture, achieving similar results with less computation.

Our grid search resulted in an optimal learning rate of 0.005. However, a typographical error led to using a learning rate of 0.0005 when applying the optimized model to cancer calling. This mistake showed that a learning rate of 0.0005 resulted in better sensitivity (73.0% vs. 73.8%), as mentioned in subsection 3.3.2. One reason this was not captured in our hyperparameter tuning (as both learning rates gave similar results) could be due to the limitations of the dataset, such as size and bias. It is not surprising that smaller learning rates can produce good results. However, when coupled with an enormous batch size, one would expect a larger learning rate to produce similar results but with faster convergence, since the large batches provide accurate estimates of the gradient.

The employed feature selection method is the same as the one used for the SNV model [9], namely LOCO. While most of the important features remain the same for both models, there are some differences. One reason for these discrepancies may be that some features, or interactions between features, were too complex to be captured in a shallow DNN, like the SNV model. Since the biggest difference between the two models is the model architecture, this is not entirely implausible. However, it is equally likely that these differences are simply the result of natural variation in the occurrence of SNVs and INDELs. Alternatively, it could be due to using different sets of data, which may include various biases, such as cohort bias or bias from differing lab protocols.

When applying cancer calling, an additional increase in sensitivity was observed when the *5layer_cone* architecture was used for the SNV error model, as mentioned in subsection

3.3.2. This suggests that the performance of the SNV error model could be improved using a larger NN, as was found for the INDEL model. One reason for this could be that larger networks are better at modelling the behaviour of errors in cfDNA. Alternatively, this could be uniquely caused by our dataset. This finding implies that further optimization may be possible for the SNV model.

In this project, it was found that an optimized DREAMS model for INDELs, together with a DREAMS model for SNVs, improved the detection of ctDNA in cfDNA at a specificity of 95%, as clearly indicated by the increase in sensitivity when applied to cancer calling. As mentioned in 1.2, INDELs may occur less often than SNVs but are characterized by higher complexity and can have a larger impact on health issues. This may be reflected in the approximately 5% increase in sensitivity. The signal for an INDEL could potentially be stronger than that of an SNV, whereby having one INDEL in a mutation catalog with many SNVs would be enough evidence to call cancer. So although there are fewer INDEL mutations overall, a good model for INDELs seems to be impactful, since the mutational signal may be stronger. Similarly, cancer calling may be increased by also modeling the expected errors of other genomic variants, such as MNVs. Given curated data with these errors, the DREAMS approach of using a DNN could easily be extended, so making additional models for other variants would be interesting for further work.

Conclusion

In this project, we optimized a DNN for modeling expected errors. Our work extends the DREAMS error identification model developed by Christensen et al. [9], by focusing on the INDEL model, which had not been previously optimized. By employing machine learning techniques such as Early Stopping, LOCO, grid search, and adaptive exploratory approaches, we effectively tuned the hyperparameters of the INDEL model.

The initial implementation of the DREAMS setup achieved a cancer call sensitivity of 68.3% at a specificity of 95%. By incorporating our optimized INDEL model, sensitivity improved by approximately 4.8% (reaching a total of 73.0%) at the same specificity level, representing a 21.6% relative increase in detectable cases. Additionally, applying our 5-layer network architecture for both the SNV and INDEL models further increased sensitivity to 74.6%, still at the 95% specificity level. These improvements are significant, and removing known germline variants could further enhance the results.

The findings underscore the potential of optimizing DNNs error models to improve the detection of ctDNA in cfDNA, at a 95% specificity, using the DREAMS approach. Our optimized DREAMS model demonstrated increased sensitivity in cancer detection, seemingly highlighting importance of accurately modeling INDEL errors. This suggests that developing additional models for other genomic variants may further enhance the accuracy and effectiveness of ctDNA-based cancer detection methods.

References

[1] Abbosh, C., Birkbak, N. J., Wilson, G. A., Jamal-Hanjani, M., Constantin, T., Salari, R., Le Quesne, J., Moore, D. A., Veeriah, S., Rosenthal, R., et al. (2017). Phylogenetic ctdna analysis depicts early-stage lung cancer evolution. *Nature*, 545(7655):446–451.

[2] Alix-Panabières, C. and Pantel, K. (2021). Liquid biopsy: from discovery to clinical application. *Cancer discovery*, 11(4):858–873.

[3] Benjamin, D., Sato, T., Cibulskis, K., Getz, G., Stewart, C., and Lichtenstein, L. (2019). Calling somatic snvs and indels with mutect2. *BioRxiv*, page 861054.

[4] Bettegowda, C., Sausen, M., Leary, R. J., Kinde, I., Wang, Y., Agrawal, N., Bartlett, B. R., Wang, H., Luber, B., Alani, R. M., et al. (2014). Detection of circulating tumor dna in early-and late-stage human malignancies. *Science translational medicine*, 6(224):224ra24–224ra24.

[5] Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. (2018). Understanding batch normalization. *Advances in neural information processing systems*, 31.

[6] Cescon, D. W., Bratman, S. V., Chan, S. M., and Siu, L. L. (2020). Circulating tumor dna and liquid biopsy in oncology. *Nature Cancer*, 1(3):276–290.

[7] Chen, L., Liu, P., Evans Jr, T. C., and Ettwiller, L. M. (2017). Dna damage is a pervasive cause of sequencing errors, directly confounding variant identification. *Science*, 355(6326):752–756.

[8] Chollet, F. et al. (2015). Keras. https://keras.io.

[9] Christensen, M. H., Drue, S. O., Rasmussen, M. H., Frydendahl, A., Lyskjær, I., Demuth, C., Nors, J., Gotschalck, K. A., Iversen, L. H., Andersen, C. L., et al. (2023). Dreams: deep read-level error model for sequencing data applied to low-frequency variant calling and circulating tumor dna detection. *Genome Biology*, 24(1):99.

[10] Coakley, M., Garcia-Murillas, I., and Turner, N. C. (2019). Molecular residual disease and adjuvant trial design in solid tumors. *Clinical Cancer Research*, 25(20):6026–6034.

[11] Corcoran, R. B. and Chabner, B. A. (2018). Application of cell-free dna analysis to cancer treatment. *New England Journal of Medicine*, 379(18):1754–1765.

[12] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[13] Dal Pozzolo, A., Caelen, O., Johnson, R. A., and Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE symposium series on computational intelligence*, pages 159–166. IEEE.

[14] DataSciencetest (2023). The importance of cross validation. https://datascientest.com/en/the-importance-of-cross-validation. Accessed: 2024-06-11.

[15] Diehl, F., Schmidt, K., Choti, M. A., Romans, K., Goodman, S., Li, M., Thornton, K., Agrawal, N., Sokoll, L., Szabo, S. A., et al. (2008). Circulating mutant dna to assess tumor dynamics. *Nature medicine*, 14(9):985–990.

[16] Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American statistical association*, 56(293):52–64.

[17] Fan, H. C., Blumenfeld, Y. J., Chitkara, U., Hudgins, L., and Quake, S. R. (2008). Noninvasive diagnosis of fetal aneuploidy by shotgun sequencing dna from maternal blood. *Proceedings of the National Academy of Sciences*, 105(42):16266–16271.

[18] Frydendahl, A., Rasmussen, M. H., Jensen, S. Ø., Henriksen, T. V., Demuth, C., Diekema, M., Ditzel, H. J., Wen, S. W. C., Pedersen, J. S., Dyrskjoet, L., et al. (2024). Error-corrected deep targeted sequencing of circulating cell-free dna from colorectal cancer patients for sensitive detection of circulating tumor dna. *International Journal of Molecular Sciences*, 25(8):4252.

[19] Garcia-Murillas, I., Schiavon, G., Weigelt, B., Ng, C., Hrebien, S., Cutts, R. J., Cheang, M., Osin, P., Nerurkar, A., Kozarewa, I., et al. (2015). Mutation tracking in circulating tumor dna predicts relapse in early breast cancer. *Science translational medicine*, 7(302):302ra133–302ra133.

[20] Garvan Institute of Medical Research (2024). Genomics explainer: types of genetic variants. https://www.garvan.org.au/news-resources/science-explained/types-of-variants. Accessed: 2024-06-03.

[21] Gerstung, M., Papaemmanuil, E., and Campbell, P. J. (2014). Subclonal variant calling with multiple samples and prior knowledge. *Bioinformatics*, 30(9):1198–1204.

[22] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46:389–422.

[23] Henriksen, T. V., Tarazona, N., Frydendahl, A., Reinert, T., Gimeno-Valiente, F., Carbonell-Asins, J. A., Sharma, S., Renner, D., Hafez, D., Roda, D., et al. (2022). Circulating tumor dna in stage iii colorectal cancer, beyond minimal residual disease detection, toward assessment of adjuvant therapy efficacy and clinical behavior of recurrences. *Clinical Cancer Research*, 28(3):507–517.

[24] Holdenrieder, S., Stieber, P., Chan, L. Y., Geiger, S., Kremer, A., Nagel, D., and Lo, Y. D. (2005). Cell-free dna in serum and plasma: comparison of elisa and quantitative pcr. *Clinical chemistry*, 51(8):1544–1546.

[25] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[26] Hu, Z., Chen, H., Long, Y., Li, P., and Gu, Y. (2021). The main sources of circulating cell-free dna: Apoptosis, necrosis and active secretion. *Critical Reviews in Oncology/Hematology*, 157:103166.

[27] Huptas, C., Scherer, S., and Wenning, M. (2016). Optimized illumina pcr-free library preparation for bacterial whole genome sequencing and analysis of factors influencing de novo assembly. *BMC research notes*, 9:1–14.

[28] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

[29] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[30] Kurtz, D. M., Soo, J., Co Ting Keh, L., Alig, S., Chabon, J. J., Sworder, B. J., Schultz, A., Jin, M. C., Scherer, F., Garofalo, A., et al. (2021). Enhanced detection of minimal residual disease by targeted sequencing of phased variants in circulating tumor dna. *Nature biotechnology*, 39(12):1537–1547.

[31] Lu, Y. and Lu, J. (2020). A universal approximation theorem of deep neural networks for expressing probability distributions. *Advances in neural information processing systems*, 33:3094–3105.

[32] Ma, X., Shao, Y., Tian, L., Flasch, D. A., Mulder, H. L., Edmonson, M. N., Liu, Y., Chen, X., Newman, S., Nakitandwe, J., et al. (2019). Analysis of error profiles in deep next-generation sequencing data. *Genome biology*, 20:1–15.

[33] Meacham, F., Boffelli, D., Dhahbi, J., Martin, D. I., Singer, M., and Pachter, L. (2011). Identification and correction of systematic error in high-throughput sequence data. *BMC bioinformatics*, 12:1–11.

[34] Moran, M. D. (2003). Arguments for rejecting the sequential bonferroni in ecological studies. *Oikos*, 100(2):403–405.

[35] Nakagawa, S. (2004). A farewell to bonferroni: the problems of low statistical power and publication bias. *Behavioral ecology*, 15(6):1044–1045.

[36] Newman, A. M., Lovejoy, A. F., Klass, D. M., Kurtz, D. M., Chabon, J. J., Scherer, F., Stehr, H., Liu, C. L., Bratman, S. V., Say, C., et al. (2016). Integrated digital error suppression for improved detection of circulating tumor dna. *Nature biotechnology*, 34(5):547–555.

[37] Øgaard, N., Reinert, T., Henriksen, T. V., Frydendahl, A., Aagaard, E., Ørntoft, M.-B. W., Larsen, M. Ø., Knudsen, A. R., Mortensen, F. V., and Andersen, C. L. (2022). Tumour-agnostic circulating tumour dna analysis for improved recurrence surveillance after resection of colorectal liver metastases: A prospective cohort study. *European Journal of Cancer*, 163:163–176.

[38] Pfeiffer, F., Gröber, C., Blank, M., Händler, K., Beyer, M., Schultze, J. L., and Mayer, G. (2018). Systematic evaluation of error rates and causes in short samples in next-generation sequencing. *Scientific reports*, 8(1):10950.

[39] Phallen, J., Sausen, M., Adleff, V., Leal, A., Hruban, C., White, J., Anagnostou, V., Fiksel, J., Cristiano, S., Papp, E., et al. (2017). Direct detection of early-stage cancers using circulating tumor dna. *Science translational medicine*, 9(403):eaan2415.

[40] Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34.

[41] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[42] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[43] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? *Advances in neural information processing systems*, 31.

[44] Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.

[45] Snyder, M. W., Kircher, M., Hill, A. J., Daza, R. M., and Shendure, J. (2016). Cell-free dna comprises an in vivo nucleosome footprint that informs its tissues-of-origin. *Cell*, 164(1):57–68.

[46] StoryMD (2024). What are the different types of genomic variants? https://storymd.com/journal/m8kyld41pm-human-genomic-variation/page/dde5eu6ke49-what-are-the-different-types-of-genomic-variants. Accessed: 2024-06-03.

[47] Wan, J. C., Heider, K., Gale, D., Murphy, S., Fisher, E., Mouliere, F., Ruiz-Valdepenas, A., Santonja, A., Morris, J., Chandrananda, D., et al. (2020). ctdna monitoring using patient-specific sequencing and integration of variant reads. *Science translational medicine*, 12(548):eaaz8084.

[48] Zhou, J., Chang, L., Guan, Y., Yang, L., Xia, X., Cui, L., Yi, X., and Lin, G. (2016). Application of circulating tumor dna as a non-invasive tool for monitoring the progression of colorectal cancer. *PLoS ONE*, 11.

Appendix

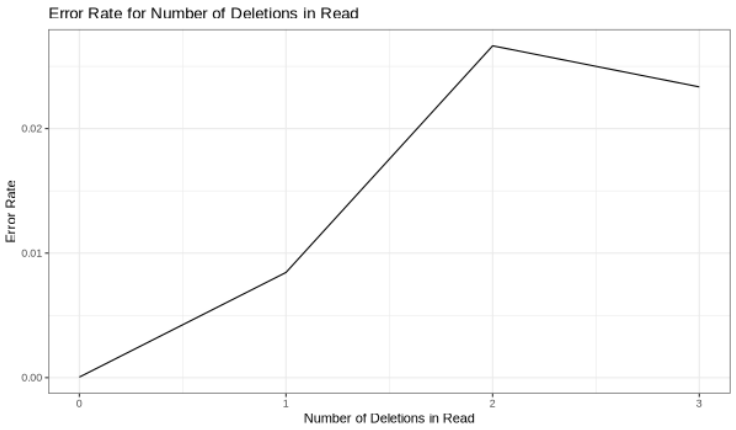
A.1 Project GitHub

GitHub, containing code and scripts made throughout this project:
https://github.com/SofieElving/BachelorProject

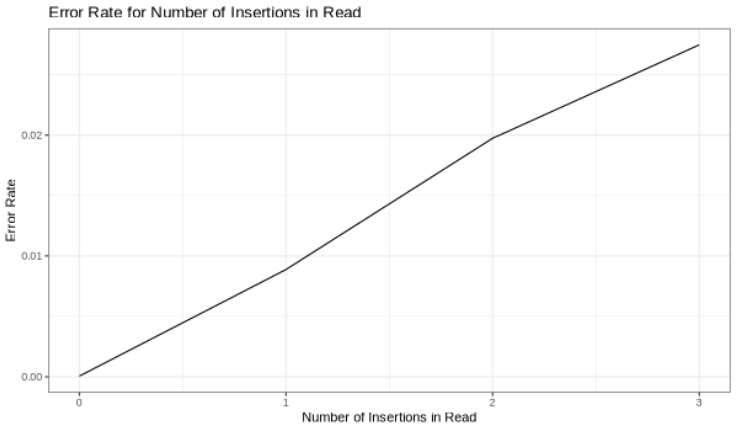
A.2 DREAMS GitHub

Branch of the DREAMS GitHub, used for training the INDEL models ect.:
https://github.com/JakobSkouPedersenLab/dreams/tree/dreams_mathilde

A.3 Additional Feature Exploration

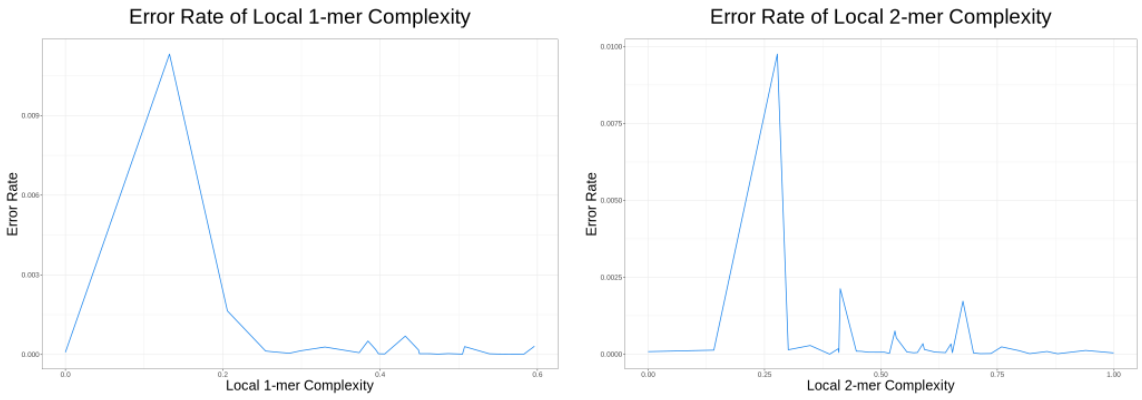


(a)



(b)

Fig. A.1: Error rate on Number of Deletions in Read **(a)** and Number of Insertions in Read **(b)**.



(a) **(b)**

Fig. A.2: Error rate on Local 1-mer Complexity **(a)** and Local 2-mer Complexity **(b)**.

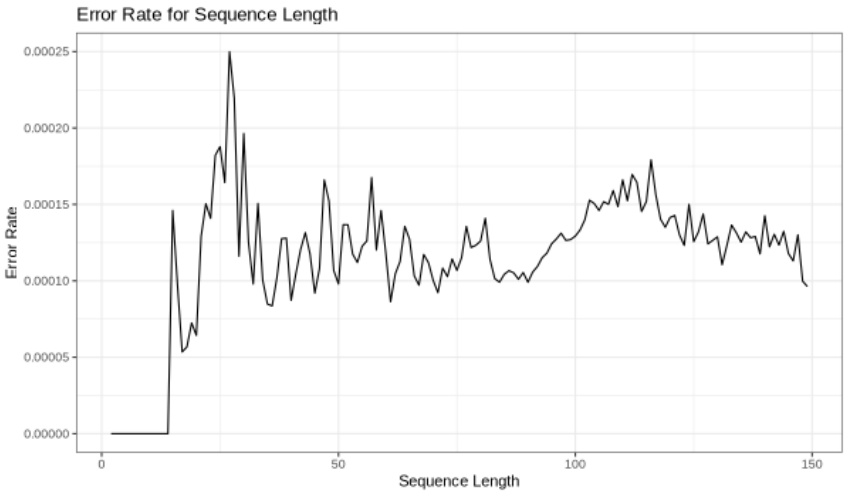


Fig. A.3: Error rate on Sequence Length.

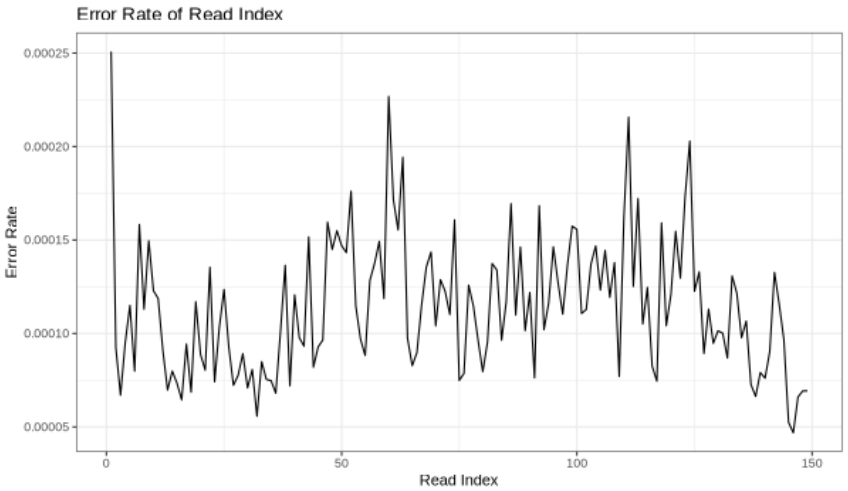


Fig. A.4: Error rate on Read Index (below 150).

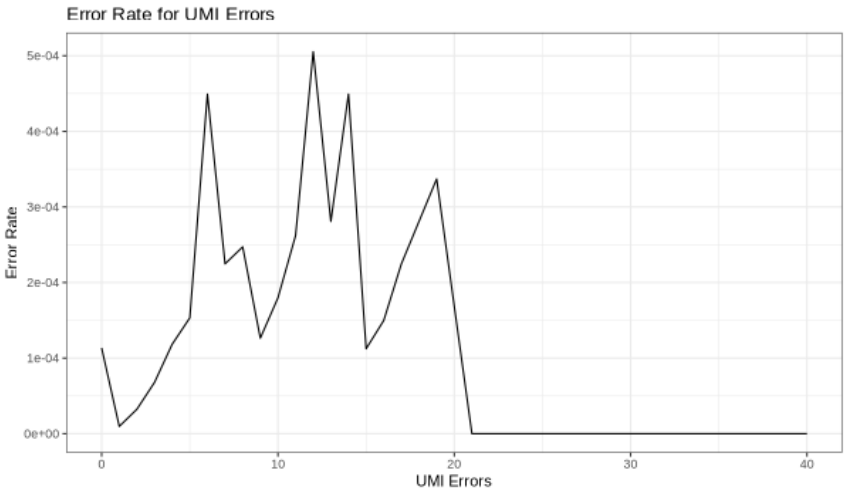


Fig. A.5: Error rate on UMI Errors.

A.4 Model Setup

Indels:

```
model_indel = train_dreams_model_indels(
  train_data_indel,
  validation_data_indel,
  layers = c(128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx", "first_in_pair",
                     "umi_count", "seq_length", "fragment_size", "local_GC"),
  lr = 0.01,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

SNVs:

```
model_snv = train_dreams_model(
  train_data_snv,
  validation_data_snv,
  layers = c(128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx", "first_in_pair",
                     "umi_count", "seq_length", "fragment_size", "n_other_errors",
                     "local_GC"),
  lr = 0.01,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

Fig. A.6: Model setup used for DREAMS-*cc* with initial/original NN structure and parameters.

Indels:

```
model_indel = train_dreams_model_indels(
  train_data_indel,
  validation_data_indel,
  layers = c(512, 256, 128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx",
                     "local_complexity_2", "seq_length", "fragment_size",
                     "local_GC"),
  lr = 0.005,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

SNVs:

```
model_snv = train_dreams_model(
  train_data_snv,
  validation_data_snv,
  layers = c(128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx", "first_in_pair",
                     "umi_count", "seq_length", "fragment_size", "n_other_errors",
                     "local_GC"),
  lr = 0.01,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

Fig. A.7: Model setup used for DREAMS-*cc* with optimized INDELs and original SNV NN structure and parameters.

Indels:

```
model_indel = train_dreams_model_indels(
  train_data_indel,
  validation_data_indel,
  layers = c(512, 256, 128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx",
                     "local_complexity_2", "seq_length", "fragment_size",
                     "local_GC"),
  lr = 0.0005,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

SNVs:

```
model_snv = train_dreams_model(
  train_data_snv,
  validation_data_snv,
  layers = c(128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx", "first_in_pair",
                     "umi_count", "seq_length", "fragment_size", "n_other_errors",
                     "local_GC"),
  lr = 0.01,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

Fig. A.8: Model setup used for DREAMS-*cc* with optimized INDEL (with lower learning rate of 0.0005) and original SNV NN structure and parameters.

Indels:

```
model_indel = train_dreams_model_indels(
  train_data_indel,
  validation_data_indel,
  layers = c(512, 256, 128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx",
                     "local_complexity_2", "seq_length", "fragment_size",
                     "local_GC"),
  lr = 0.005,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

SNVs:

```
# Model SNV
model_snv = train_dreams_model(
  train_data_snv,
  validation_data_snv,
  layers = c(512, 256, 128, 64, 32),
  model_features = c("read_index", "strand", "trinucleotide_ctx", "first_in_pair",
                     "umi_count", "seq_length", "fragment_size", "n_other_errors",
                     "local_GC"),
  lr = 0.01,
  batch_size = 32000,
  epochs = 750,
  log_file_path = args[3],
  min_delta = 0.001,
  patience = 10)
```

Fig. A.9: Model setup used for DREAMS-*cc* with optimized INDEL and SNV (with 5 layers) NN structure and parameters.

A.5 Model Log

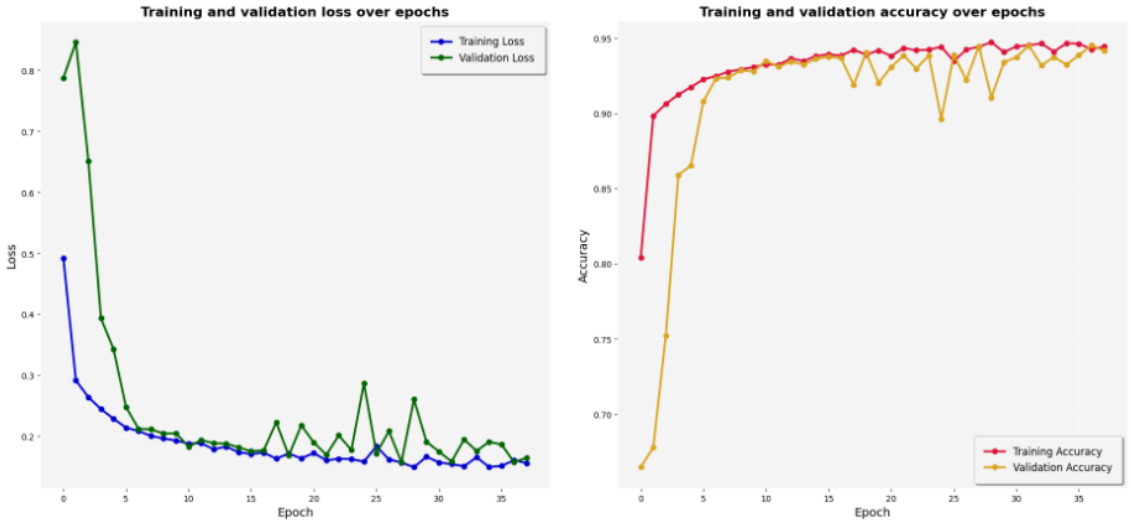


Fig. A.10: Log plots of Loss and Accuracy for each Epoch while training the original INDEL model.

Fig. A.11: Log plots of Loss and Accuracy for each Epoch while training the optimized INDEL model.

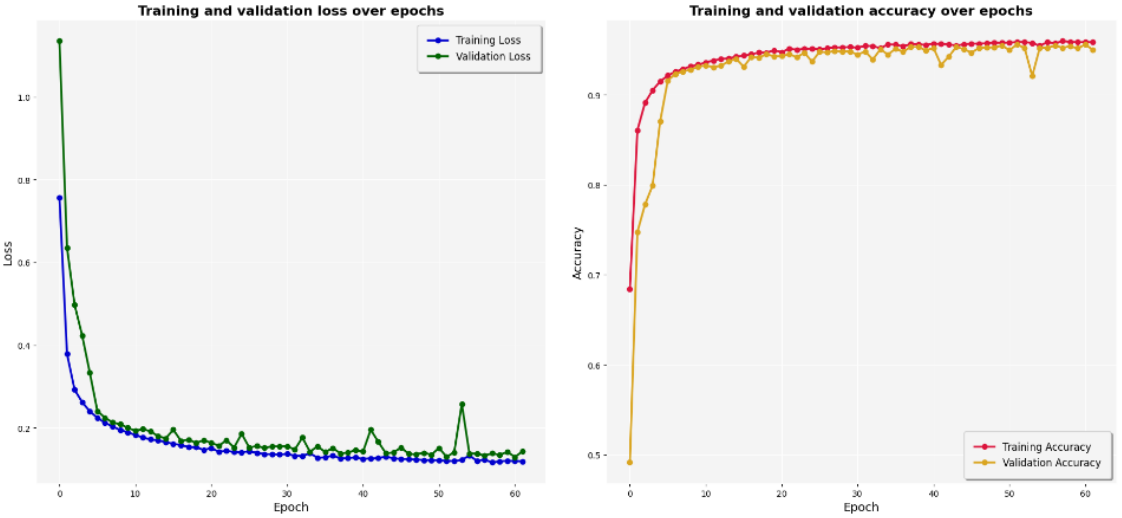


Fig. A.12: Log plots of Loss and Accuracy for each Epoch while training the optimized INDEL model, with lower learning rate of 0.0005.

A.6 Model Seed

```r
args <- commandArgs(trailingOnly=TRUE)

reticulate::use_condaenv(args[1], required = TRUE)

# Set the random seed for R
seed_value <- 1234  # or any integer value of your choice
set.seed(seed_value)


# Loading Libraries

library(dplyr)
library(dreams)
library(keras)


# MODEL TRAINING

# The model can be trained using a neural network and requires basic settings for keras

# Set the random seed for Python (TensorFlow)
reticulate::py_run_string(paste0("
import tensorflow as tf
import numpy as np
import random

seed_value = ", seed_value, "
tf.random.set_seed(seed_value)
np.random.seed(seed_value)
random.seed(seed_value)
"))
```

Fig. A.13: Seed used for DREAMS-*cc*.

A.7 ROC Curves

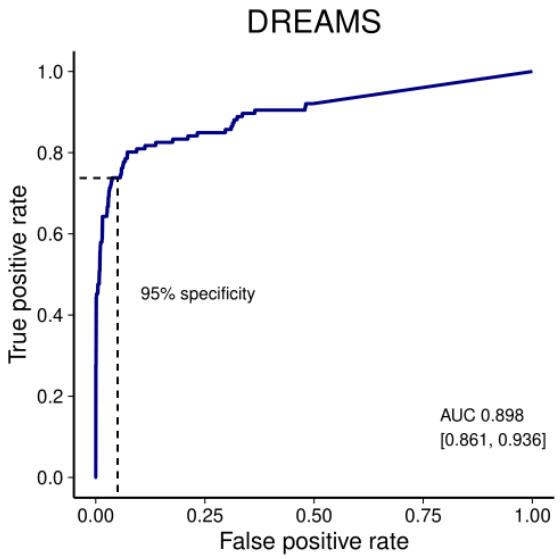


Fig. A.14: ROC curve for optimized INDEL (with lower learning rate of 0.0005) and original SNV NN structure and parameters (model setup in figure A.8).

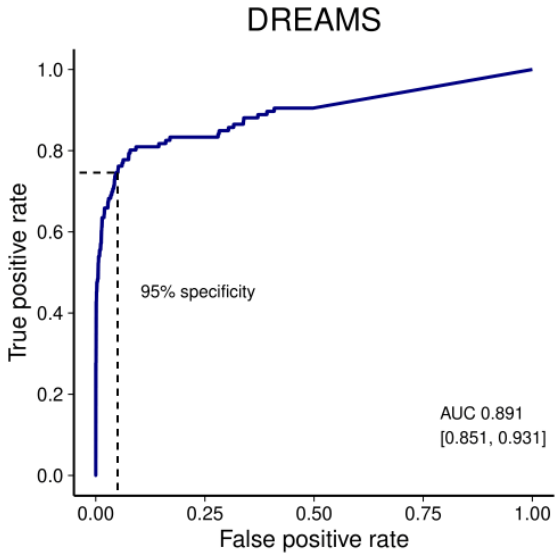


Fig. A.15: ROC curve for optimized INDEL and SNV (with 5 layers) NN structure and parameters (model setup in figure A.9).