RECOGNIZING IRREGULARITIES IN STACKED NANOPORE SIGNALS FROM IN SILICO PERMUTED SEQUENCING DATA

MSc major internship report

Maarten van Elst (5673968)

Supervised by Marc Pagès-Gallego, Jeroen de Ridder (UMC Utrecht)
Second review by Berend Snel (Utrecht University)

Table of Contents

Abstract

Basecalling a nanopore read has been solved to an accurate degree, and this has been done similarly for methylation such as 5mC and 6mA. Other epigenetic modifications exist which are still hard to detect from a single nanopore signal. We present two approaches to solve this problem: Bayesian optimization of expected values used for signal-to-base assignment, and random forest classification of irregular signals. Due to the lack of ground truth data, we use *in silico* mutations in nanopore data to simulate such modifications. Due to a lack of information present in a single signal we use stacked signals to increase performance and for potential use in repeated regions of the genome. Our efforts in recalculating the expected signal values showed promise, but upon further inspection we found that the scoring function was very sensitive to local optima. With this we concluded that Bayesian optimization of the re-squiggle k-mer table is an ineffective method for discovering new reference values from irregularities in a signal. We then explored using a random forest machine learning model on features to classify the *in silico* generated mutations from a stacked signal. We showcase that this model is effective for high stack depth (accuracy of 98.7%), however for lower depth it returns to guessing-level accuracy (~50%). For further research we recommend improvements to the acquisition- and scoring function for k-mer table optimization and the use of deep learning for classification of irregularities.

Plain language summary

Deoxyribonucleic acid (DNA) is a highly important molecule in most living cells, it contains inheritable information in the form of genes and the expression of these genes is how a cell regulates itself. In recent years there have been many technological advancements in the field of DNA sequencing – reading the information contained in a DNA molecule from some biological sample. One technique used for this is called nanopore sequencing. A nanopore sequencer outputs an electrical signal that varies based on the DNA sample that is being sequenced. Using bioinformatic techniques we can distinguish the four DNA nucleotides (bases) from this signal, turning the electrical signal into a sequence of bases consisting of 'A', 'C', 'T', and 'G'. This genetic information can differ between individuals or cells and is relevant for research into biology, disease, and especially the mechanisms behind cancer. However, DNA contains more information than the sequence of these four bases. There are also chemically modified versions of these bases that function differently depending on the specific modification. Such modifications are called epigenetic modifications. They are often quite small, but also influence the regulation within a cell. So far it has been a challenge for scientists to detect such modifications in the electrical signal of a nanopore sequencer.

In this project we try to improve the recognition of epigenetic modifications in nanopore signals. We attempt this in two ways: optimization and machine learning. Both approaches require a dataset that contains known epigenetic modifications. However, as mentioned above it is difficult to know from sequencing whether a modification is present. Because of this we don't have a lot of nanopore data where we reliably know the presence of modifications. To solve this problem, we use data containing computer-simulated irregularities that represent modifications. Given that we do the simulations, we know for sure which parts of the data are "modified" and which are not. In addition to this simulation, we overlay signals corresponding to the same piece of DNA on top of each other in a "stack" to increase the amount of information available to the algorithms we use.

Usually when a nanopore signal is processed and we gain the sequence of bases (ACTG) and some expected signal value corresponding to these bases without considering any possible modifications. In the first experiment we optimize this expected signal from simulated irregular data. Ideally, an optimization algorithm finds an expected value corresponding to a simulated modification that is different from the expected value for a normal base. If this works on simulated data, then we can move on to real-world data to discover the values corresponding to epigenetic modifications. The optimization algorithm works by iteratively guessing new values, and for each guess it gets feedback from a 'scoring function' that decides whether a value approaches a good result. The scoring function we use indicated good results. However, values corresponding to a good score were not in line with the expected (and correct) values of our simulated dataset. This discrepancy shows that this approach is ineffective, and we hypothesize that further research is required on the scoring function.

The second experiment attempted to use machine learning on stacked signals that were either correct or contained irregularities. This machine learning method uses part of the data to learn how to recognize irregularities, and another part for testing the accuracy of what it has learned. The testing part achieved a maximum accuracy of ~98% with a very large stack of signals. The main problem here is that accuracy decreases for learning with smaller stack sizes, while ideally we recognize modifications in a single signal with no stacking at all. From this machine learning method, we can however interpret which properties of a stacked signal help in predicting modifications. And from the properties that work well we hypothesize that measurements of the number of peaks in stacked data might improve the model. Overall, this approach was effective within the context of this experiment but requires additional efforts to be used in real-case scenarios.

Introduction

In the past decades, sequencing technologies have been developing rapidly and with great impact. This has enabled the human reference genome to go through many iterations since 2003 when the first assembly of euchromatic regions was published using Sanger sequencing (International Human Genome Sequencing Consortium, 2004). Sanger sequencing produces reads with a maximum length of approximately 1 kilobase. These reads can be assembled into longer sequences using bioinformatics methods. While this works well enough to assemble most of the human genome, heterochromatic repeat regions are more challenging. This resulted in unresolved regions of the genome (International Human Genome Sequencing Consortium, 2004). Specifically, heterochromatic regions of the genome containing higher order repeats were impossible to assemble. Due to the nature of repeats no unambiguous assembly can be made if reads are shorter than the length of the repeated region. Illumina reduced the sequencing costs providing higher sequencing throughput. However, due to the short nature of their reads (~200 base pairs), the repetitive regions of the genome remained a mystery. Third-generation sequencing technologies, such as Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) allowed the sequencing of longer reads from native DNA (Amarasinghe et al., 2020). By combining long and short read technologies researchers have recently made a big step towards 'completing' the telomere-to-telomere human reference genome by including heterochromatic repeat regions (Jain et al., 2018; Nurk et al., 2022).

The human genome contains many different repetitive regions, the most well-known of which are the telomere and centromere regions. To illustrate the repetitive nature of this region, it contains higher order repeats (HORs) of the alpha-satellite monomers of approximately 171bp length. The centromere contains different HORs of lengths varying between 2 and dozens of repeats of the alpha-satellite, and HORs in turn can repeat in HOR arrays reaching megabase length (Hartley & O'Neill, 2019). The total length was impossible to determine through older sequencing methods, but they are biologically relevant due to the centromere's role in cell division. Compared to alpha-satellites, the vertebrate telomeric repeat sequence is relatively short (TTAGGG). It is rich in guanine and therefore potentially rich in epigenetically modified guanine, which can lead to structural and maintenance problems (Fouquerel et al., 2019). Sequencing developments are now helping with understanding these regions of the human genome, and being able to adequately map the repeats and their epigenetic modifications could help in diagnostics or further our understanding of telomere dynamics.

In addition to the length of the sequence helping with genome assembly, nanopore sequencing allows for detection of epigenetic modifications that are present on the native DNA. This is because it works by translocating a single stranded DNA (or RNA) molecule through an artificial pore. This pore is embedded on a membrane with a membrane potential in an ionic solution. Ions passing through the pore will be influenced by the molecular composition of the DNA; this results in a changing, measurable current that is used to distinguish sets of different nucleotides passing through the pore (Figure 1).

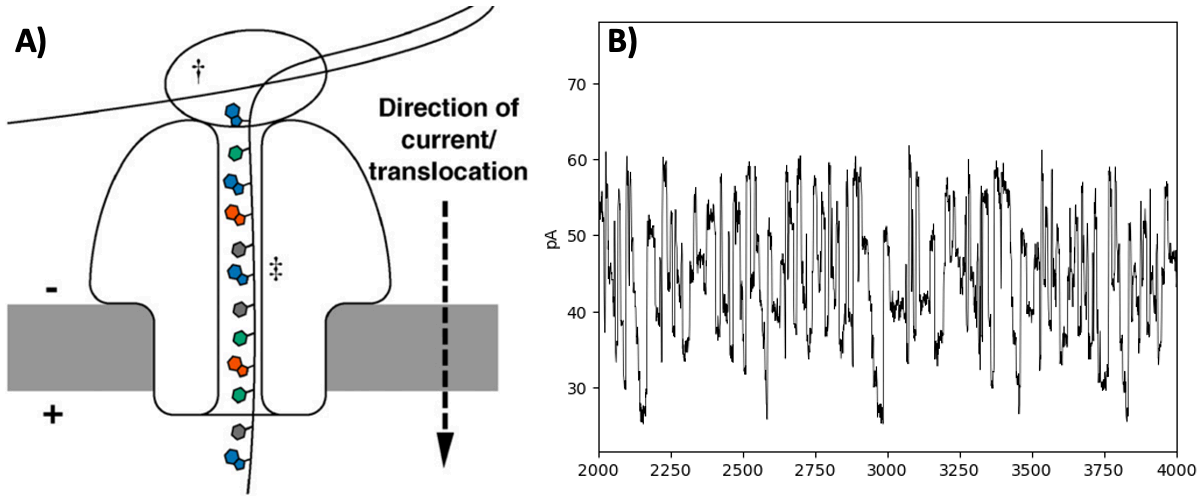


Figure 1 - A) Schematic view of nanopore DNA sequencing. DNA is unwound by a motor protein (†), and a single strand moves through the nanopore (‡) (Adapted from Heather & Chain, 2016). B) Example of a truncated nanopore signal produced by a single DNA with the first 2000 and last 8000 datapoints omitted. The y axis indicates electrical current in pA, the x axis indicates samples with a sampling frequency of 4000 Hz.

Translating the measured electric signal back to the original DNA sequence is a challenge. Mainly because of noise in the signal levels and varying speed at which the DNA strand moves through the pore. This is a sequence-to-sequence problem and has similarities with speech recognition where audio is translated to text. In DNA sequencing of canonical bases this task is performed by algorithms called basecallers. Modern nanopore basecallers are based on deep learning models (Pagès-Gallego & de Ridder, 2023). Covalently modified bases and oxidized bases such as 5mC and 8-oxodG respectively, will also pass through the pore and can theoretically be measured. Several tools have already been developed to achieve what is called methylation-calling with 5mC and 6mA (Yuen et al., 2021).

Many nanopore-detectable epigenetic modifications exist, of which 5-methylcytosine (5mC) is the most abundant and can serve as an example of the importance of epigenetic modifications. 4% of cytosines are 5mC, making it the most frequent epigenetic modification in vertebrate cell lines (Liyanage et al., 2014). Because of the high occurrence, scientists oftentimes refer to 5mC as the fifth base (Breiling & Lyko, 2015; Lister & Ecker, 2009). 5mC usually occurs at CpG sites, which are often concentrated together in the genome in so-called CpG islands. Genes with 5mC in the promotor generally have downregulated transcription. Unusual 5mC levels are often found in different types of cancer, where promoters of tumor suppressor genes are hypermethylated and many other genes hypomethylated (Ehrlich, 2019; Nguyen et al., 2021). Other examples of native DNA modifications include 6-methyladenine (6mA, existing among several other methyladenines), a group of methylguanines, and 8-oxoguanine (8-oxoG) (Sood et al., 2019). Due to its omnipresence and importance, many tools have been developed to achieve accurate methylation-calling with raw nanopore signals.

In general, methylation callers only require a raw or base called file of the nanopore signal (FAST5 format). These signals are then processed by differing algorithms. Nanopolish is a tool that can achieve this and uses a Hidden Markov Model (HMM) to recognize 5mC from base called FAST5. Guppy is also used for base calling, but able to detect methylation with a Recurrent Neural Net (RNN) model. The same model type is used in Megalodon. Other examples include DeepSignal and DeepMod which use RNN and LSTM (Long-Short Term Memory) layers in a neural net, or DeepMP which uses a convolutional neural network. Then there is also METEORE which achieves a consensus from some of these models with a random forest approach. Several of the above require deep learning methods

that have to be trained to recognize a specific DNA modification (Y. Liu et al., 2021). Methylation detection is demonstrably effective for 5mC (and 5hmC and 6mA, other types of native DNA methylation). For preprocessing some of these tools (DeepMP, DeepSignal) also rely on the *re-squiggle* algorithm which will be discussed in more depth later.

Table 1 – Overview of several epigenetic modification callers used for calling 5mC and the different types of algorithms involved. Adapted from (Y. Liu et al., 2021).

Tool	**Algorithm**	**Ref.**
Nanopolish	Hidden Markov Model	(Simpson et al., 2017)
Tombo	Mann-Whitney and Fisher's exact test (statistical testing)	(Stoiber et al., 2017)
SignalAlign	HMM with Hierarchical Dirichlet Process (HMM-HDP), a clustering method	(Rand et al., 2017)
Guppy	Recurrent Neural Network (RNN)	(Y. Liu et al., 2021)
NanoMod	Kolmogorov-Smirnov test (statistical testing)	(Q. Liu, Georgieva, et al., 2019)
DeepMod	Bidirectional RNN with Long Short-Term Memory (LSTM)	(Q. Liu, Fang, et al., 2019)
DeepMP	Convolutional Neural Network (CNN) and RNN	(Bonet et al., 2021)
METEORE	Combines 1+ other algorithm's predictions, random forest (RF) and multiple linear regression are used for integration.	(Yuen et al., 2021)

The tombo toolkit

Tombo is another one of these toolkits that can also detect modified bases from nanopore data. It relies on a mapping of each signal point to a nucleotide in the basecalled sequence. This mapping is provided by the *re-squiggle* algorithm in the Tombo package and does not rely on deep learning models like several of the tools mentioned in Table 1. Given a reference sequence (basecalls or a known sequence), this algorithm scores an alignment between the raw signal and the reference bases with dynamic programming. The scoring of individual steps in this algorithm is based on predetermined reference values for all possible 6-mers that could go through the nanopore. These values are aligned to the reference sequence to get an expected signal, and the distance between this expectation and the actual signal is minimized to get approximate boundaries for each base. This results in a segmented view of the signal (Figure 2).

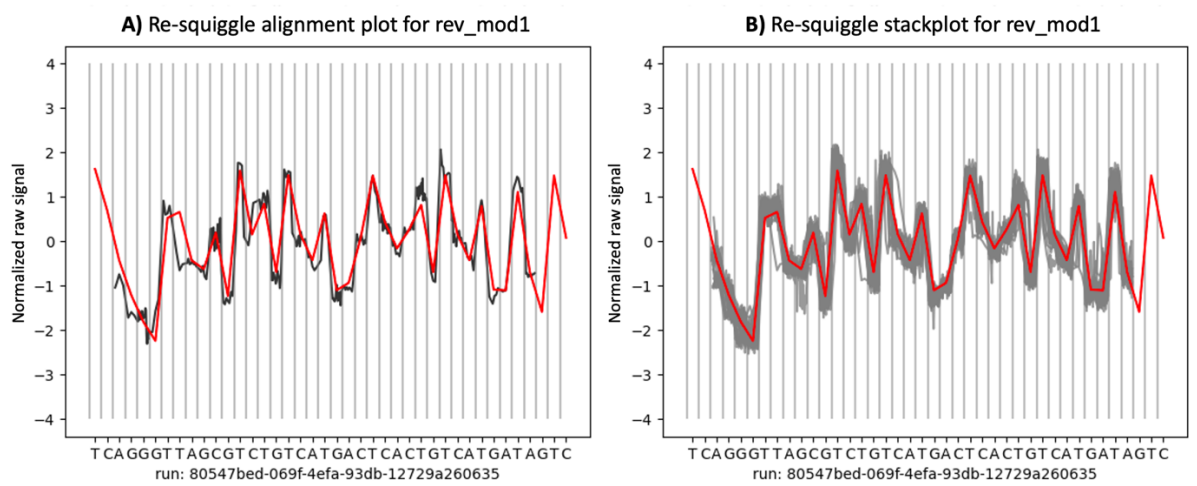


Figure 2 - Illustration of a single (panel A) and stacked (panel B) normalized nanopore signal with the boundaries between bases as found by the tombo re-squiggle algorithm. The stacked data results several repeats for the given read ID (run). The y-axis shows the normalized raw signal value, x-axis contains the corresponding bases according to the re-squiggle algorithm for reference sequence rev_mod1. The red line indicates the expected signal by the tombo model that is used in re-squiggling.

Next to the detection of 5mC by tombo and other tools, there is a plethora of other epigenetic modifications that can occur on a native DNA strand that have potential to be found in a nanopore sequencing signal (Sood et al., 2019). One of these is 8-oxodeoxyguanine (8-oxodG), a biomarker for and the result of oxidative DNA damage caused by exogenous factors or inherent cell processes (Poetsch, 2020). This damage often occurs due to the presence of reactive oxygen species (ROS) such as hydroxyl radicals, hydrogen peroxide or singlet oxygen. ROS are continuously generated as a byproduct of aerobic cellular respiration (Hahm et al., 2022). Guanine is most frequently oxidized by ROS due to its low reduction potential compared to other nucleosides in the DNA, making 8-oxodG is the most common oxidated nucleoside in DNA.

8-oxodG and DNA damage

Due to the modification, 8-oxodG in its *syn* conformation can form a Hoogsteen base pair with adenine instead of a Watson-Crick base pair with cytosine (Hahm et al., 2022). This incorrect pairing eventually results in G>T (and corresponding C>A) transversions (Figure 2). Healthy cells defend against this via base excision repair, with specific 8-oxodG detection by DNA glycosylase OGG1. Higher degrees of 8-oxodG are indicative of cell age, inflammation or age-related diseases such as cancer or neurodegenerative disorders (Poetsch, 2020). In addition to destructive effects, 8-oxodG also has epigenetic side effects by affecting transcription through allowing TET1 to initiate demethylation (Hahm et al., 2022) and plays a role in synaptic plasticity and memory formation (Beckhauser et al., 2016).

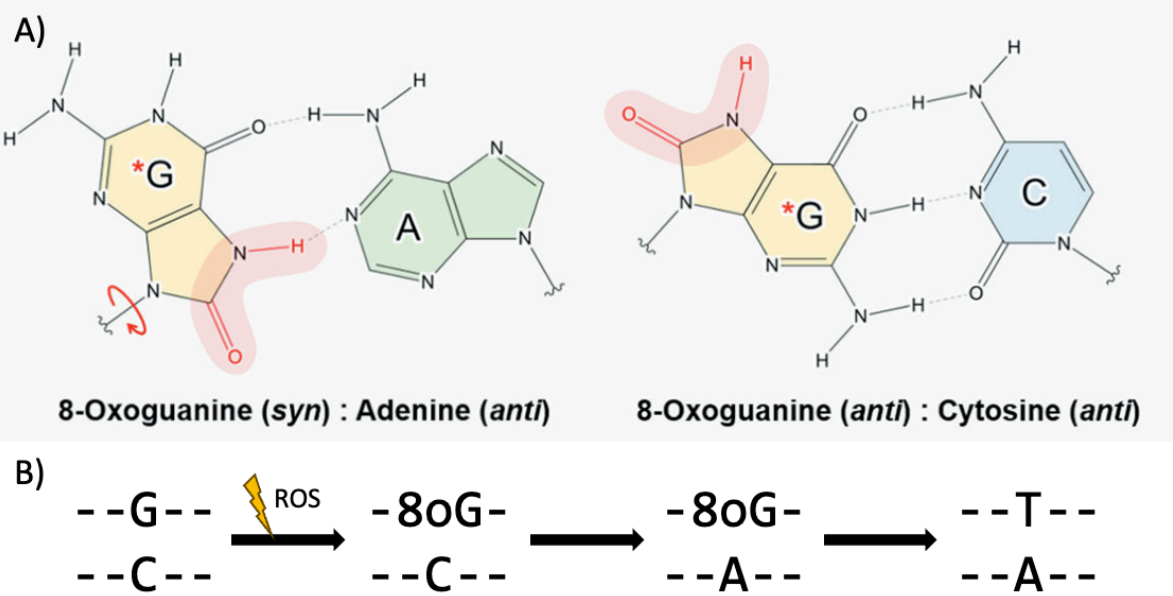


Figure 3 – Mutagenesis from 8-oxodG. A) Hoogsteen base pair of 8-oxoguanine in syn conformation with adenine (left) and Watson-Crick base pair formation of 8-oxoguanine in anti conformation with cytosine (right). Result of oxidation site highlighted in red. Adapted from (Hahm et al., 2022). B) Mutagenesis cascade resulting in G>T and C>A transversion after damage by ROS followed by 2 rounds of replication.

This project attempts to find epigenetic modifications, specifically 8-oxodG, in raw nanopore data by processing the repeats in a repetitive region using different algorithmic approaches. By recognizing the epigenetic differences among repeated sequences, we hypothesize that we can improve modification base calling of these regions and hopefully open avenues toward a better understanding of their biological function. To achieve this, we use a dataset from oligomers with known repetitive motifs and known *in silico* generated DNA modifications. These repetitive motifs are stacked using sequence to sequence alignment algorithm based on Dynamic Time Warping.

Dynamic Time Warping

Given a DNA molecule that is passing through a nanopore, tombo uses expected signal levels which are known for all possible 6-mers of the four canonical bases (A, C, G, T). This known 'artificial' signal is built upon the basecalls and helps in estimating where any modifications in the sequence might exist. Once generated, the artificial signal must be aligned to the raw nanopore signal to give a segmented view of the raw signal, with each segment correlating to 6-mer. Different versions of nanopore use different sampling frequencies, our data is sampled at 4KHz. The DNA moves at approximately 450 bases per second through the pore, therefore the average base has approximately 10 datapoints in the raw signal. Combining the stretched artificial signal to the raw signal is now a signal alignment problem. Signal alignment problems are well-known from other fields and can be solved using a continuous dynamic programing approach: Dynamic Time Warping (DTW) algorithm. This algorithm forms the base of the more specialized re-squiggle algorithm.

DTW was developed to align two sequences of non-discrete values of different lengths. This is the case for raw nanopore data because the DNA moves through the pore at varying speeds. The algorithm is constrained by several conditions: *monotonicity* and *continuity* conditions guarantee that the signals stay aligned to each other and no loops or gaps occur. Sometimes there is also a *boundary* condition, where the signals must be globally aligned – thus the warping path goes from the bottom left corner to the top right corner in Figure 4. This is however a manual example, tombo uses an open end and determines the beginning separately to get the optimal alignment. Specialized DTW implementations

often contain extra constraints to prevent erroneous warping paths such as the Sakoe-Chiba band that keeps the path constrained to a diagonal region (Müller, 2007). An ideal path should not deviate too much from the main diagonal or be very steep; this is why we tweak the artificial signal (Figure 4, panel A y-axis).

To achieve the DTW alignment, a distance matrix and subsequent cost matrix between the two sequences are calculated. The distance matrix is generated by calculating the all-to-all distance between the signal values. To find the warping path a cost matrix is recursively calculated on the distance matrix from the start to end (bottom left to top right in Figure 4A) by selection of the minimum cost value from previous points in the alignment and adding the cost of that step. There are three options for how a path develops from point to point: moving one point on the index of the reference sequence, the query sequence, or both. This ensures monotonicity and continuity, preventing loops, breaks, or reversing of a signal. The alignment results from tracing the lowest values in this cost matrix from the end of the sequences to the start; the so-called 'warping path' (Figure 4A). Infinite values around the edges of the cost matrix satisfy the boundary condition. DTW cannot be parallelized due to dependency on previously calculated values. However, by using dynamic programming techniques this calculation is made more efficient. Dynamic programming combines recursion and memoization, speeding up the distance calculations by keeping track of intermediate results.

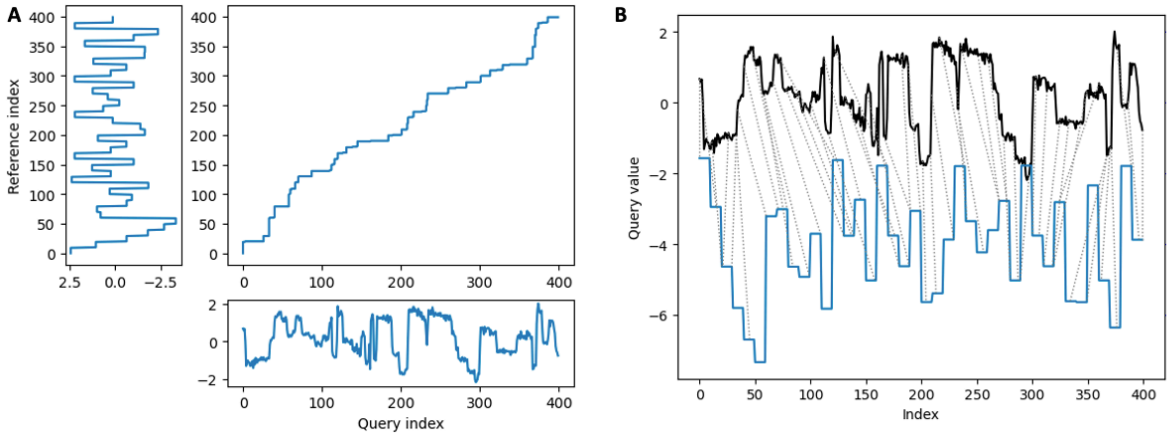


Figure 4 - Example DTW alignment between artificial reference signal as it is generated by the tombo model (Reference) and a raw nanopore signal (Query). A) "Threeway" plot showing the reference (y-axis) and query (x-axis) signals as they are aligned, the warping path is shown in the middle. B) "Twoway" plot showing point-to-point translation of the warping path. The expected signal is shifted down for visualization purposes.

The re-squiggle algorithm

The re-squiggle algorithm included in the tombo package takes a DTW-like approach but is more attuned to the noisy data of nanopore sequencing. Before the warping stage, several steps are taken to prepare the data. Base calls from the read are mapped to a reference genome, giving a known reference sequence. The raw signal is normalized twice, using median shift and MAD for a first pass (Equation 1) and using a shift and scale parameter that are calculated after initial matching to a reference sequence (Equation 2). Additionally, top and bottom 5% of datapoints are winsorized per event before calculating the Z-score to make the algorithm more robust to outliers.

$$Normalized\ Raw\ (first\ pass) = \frac{(Raw\ Signal - Median)}{Median\ Absolute\ Deviation}$$

Equation 1 – Signal normalization on the first pass. Data is shifted by the median of the raw signal and scaled to the Median Absolute Deviation (MAD) of the raw signal for robustness to outliers. (Re-Squiggle Algorithm — Tombo 1.5.1 Documentation, n.d.)

$$Normalized\ Raw\ (consecutive\ passes) = \frac{(Raw\ Signal - Shift)}{Scale}$$

Equation 2 – Signal normalization after the first pass. The scale parameter is calculated by correcting the MAD with the Theil-Sen estimator between expected (From the reference genome) and observed signal levels in the read. The shift parameter is also corrected, using the median of intercepts over Theil-Sen estimators within the read. (Re-Squiggle Algorithm — Tombo 1.5.1 Documentation, n.d.)

The signal is segmented via a running-window approach that recognizes large jumps and is referred to as event-level data. In the warping step (also known as sequence to signal assignment), the normalized event-level signal is warped against the reference genomic sequence. This is initially done with a wide first part of the signal, to find the start of actual read data in the raw signal. Because the event-level data is used, this step is faster than it would be on signal level data but does not contain the nuanced mapping required. Once the start of the sequence is found, the algorithm utilizes a smaller, adaptive band like the Sakoe-Chiba band to align the segmented signal to the genomic sequence.

The tombo re-squiggle algorithm relies on an expected distribution to model a given k-mer of the DNA strand. This is represented by a mean and standard deviation, which are used in calculating the Z-scores used in the warping algorithm. However, this model is incomplete as it lacks entries for any non-canonical DNA bases or modifications. Re-squiggle is good at segmenting the bases, which allows us to create a stack of several signals. Tombo provides methods of recognizing methylation on such a stack, but these are known to be inaccurate. We hypothesize that the stack of signals or the process of generating it can be of value in recognizing modifications present on the DNA strand. We investigate this in two ways: optimization of the tombo k-mer table and classification of errors in the re-squiggle algorithm.

Our first approach tries to recognize epigenetic modifications differently by learning new values through permutation of the old model and potentially finding a new model value in the process - re-optimizing the k-mer table after each iteration. Then a scoring method is used to find out whether a re-squiggle result is accurate. The second approach uses a random forest for classification of irregularities in the re-squiggle stack based on features calculated from the stack data. Due to the lack of known truth nanopore signals for epigenetic modifications, both approaches use *in silico* generated mutations as a proxy for epigenetic modifications (Figure 5, 9).

Results

Bayesian optimization of tombo models

Here we attempt to recognize potential modified bases by optimizing the values of the expected canonical DNA sequence. Bayesian optimization is used to search for optimal new values because it is designed to optimize hard to evaluate functions, as is the case with multiple re-squiggles.

The approach is illustrated in Figure 5. A discrepancy between the model and reference sequence is simulated by changing the reference sequence (Figure 5B). This emulates the presence of a non-canonical base; the goal is to show that optimization can find the value corresponding to the original base (Figure 5C). Due to re-squiggle using a moving window of 6 nucleotides, 6 entries in the reference table are changed. Bayesian optimization from the scikit-optimize python package is used to perform gaussian process regression for these 6 dimensions. The Mean Squared Error (MSE) is calculated on event-level data, using the error between the model and re-squiggled datapoints. In Figure 5, thymine on position 5 is changed to adenine, indicating a large step in signal values which should be relatively easy to recognize by an optimizer. As seen in the MSE panels, it is difficult to quantify errors in the model – the emulated change is only visible in an increase in MSE on Thymine on position 6.

The tombo package offers three methods for modified base detection, two of which rely on either known k-mer values or comparison of multiple samples. Ideally, we should be able to discover *de novo* modifications, which they provide in a third method that uses a hypothesis test against the reference model. This has a high error rate and does not give the precise location (*Modified Base Detection — Tombo 1.5.1 Documentation*).

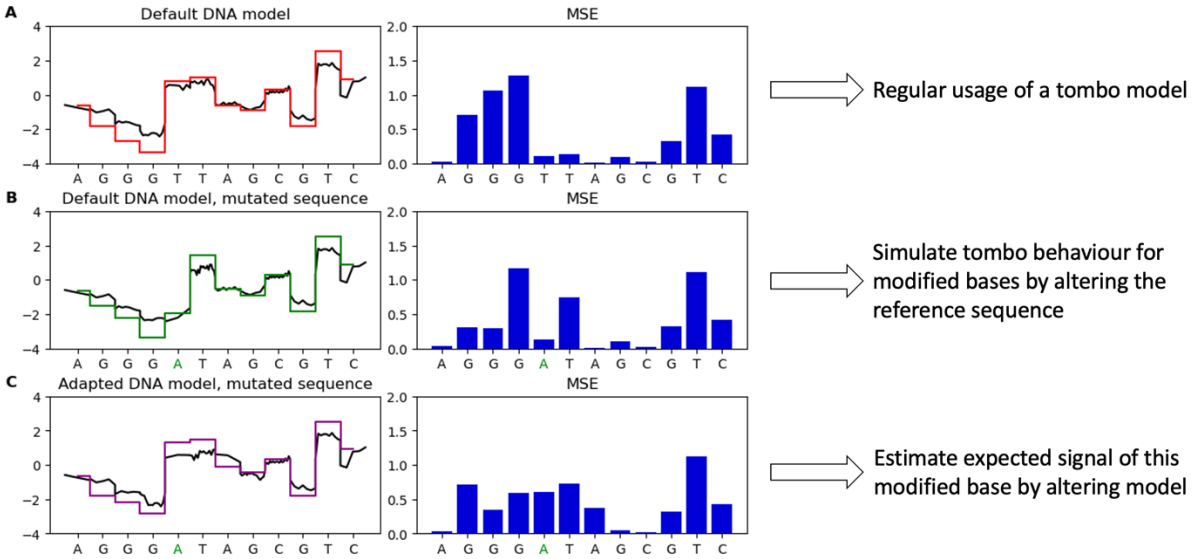


Figure 5 – Illustration of the effects of changing a reference sequence and tombo DNA model. A) Tombo's default DNA signal model (red line) is used to re-squiggle a read to the correct corresponding reference sequence. B) The default DNA model (green) is used with an incorrect reference sequence with adenine on position 5. Simulating behavior of the algorithm when values are not as would be expected by the DNA model. Red dashes on the MSE plot indicate MSE values for the reference sequence in panel A. C) An adapted model (purple) can be optimized to decrease MSE and find potential new signal levels that are different from the 4 canonical nucleotides of DNA. Panels on the right show the corresponding Mean Squared Error per base between datapoints and the model signals on the left.

Bayesian optimization was executed for most bases of all reference sequences mentioned in appendix 2. The oligos have a length of 40 bp, 30 of which are used to optimize for. For the 4 oligo's this results in 360 total permuted bases, each of which a new tombo model is optimized. Each permutation 'experiment' has 6 k-mers that are optimized and have the corresponding MSE scores calculated (Figure 6). All optimizer results are available in the project repository (appendix 4).

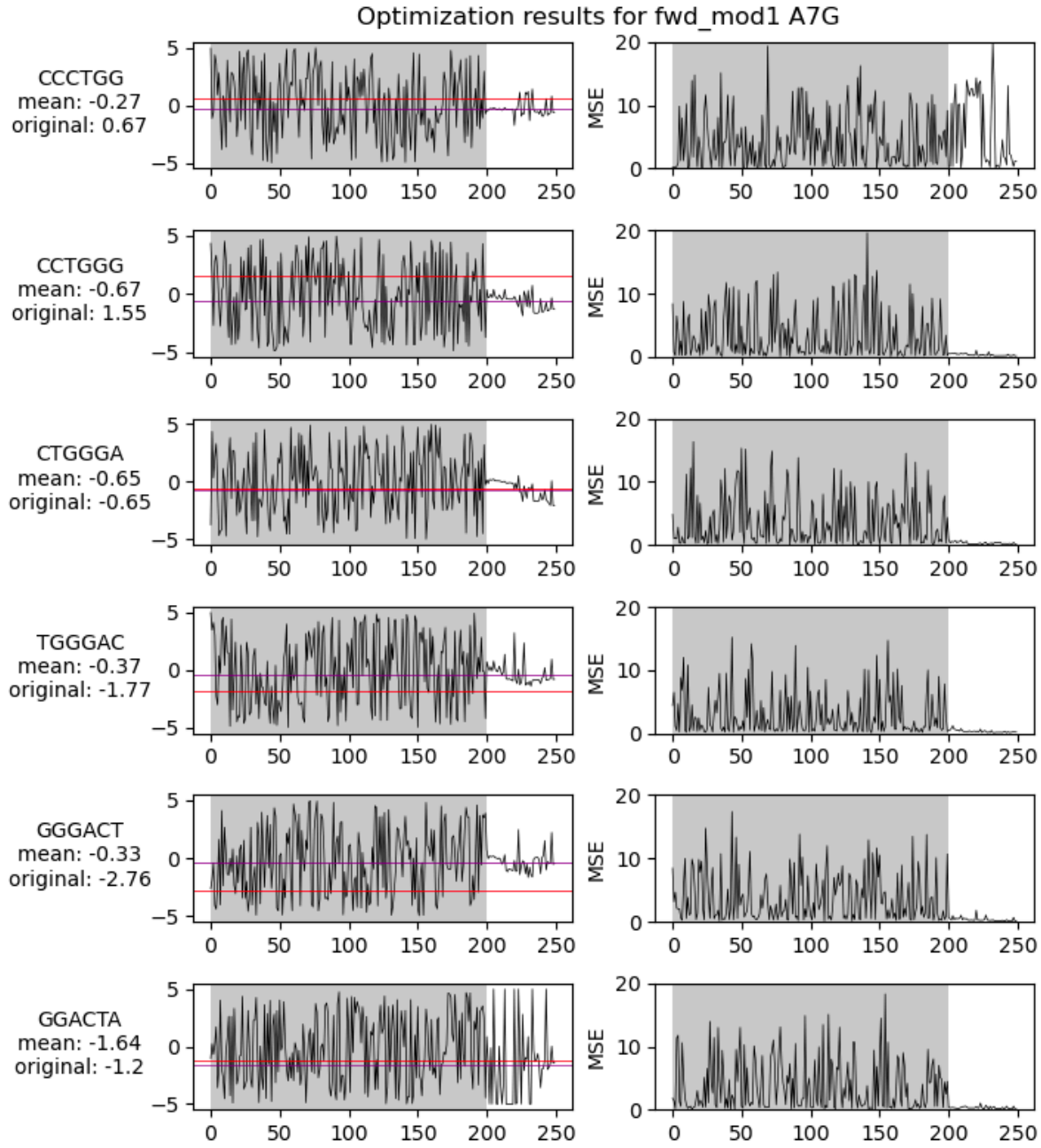


Figure 6 – Results for Bayesian optimization of a tombo model on a single permutation of a reference sequence. 6-dimensinal optimization values are shown on the left, with the original k-mer's sequence on the y-axis and the corresponding default tombo DNA value ("original") shown in red. Purple indicates the mean value of the optimizer between steps 200-250 ("mean"). The grey region shows grid search iterations. MSE-scores corresponding to the re-squiggle result are shown in the panels on the right.

The MSE score for some k-mers converges quickly, with a low MSE score and a predicted and original value closely together (Figure 6, k-mer CTGGGA). This is the desired behavior and several other bases come close to the desired original model value and show a low MSE (GGACTA, CCCTGG). Sometimes

we see high peaks in MSE score happening after the grid search phase (Figure 6, top panel). These peaks are due to re-squiggle error, which is punished with a MSE value increase of 10. There are also cases (CCTGGG, TGGGAC, GGGACT) where the MSE and discovered mean converge to some value but the original value is far from the discovered mean.

Looking at the bottom value (GGACTA) in Figure 6, there seems to be less certainty as the optimizer still attempts 'extreme' values. This occurs often involving the first or last k-mer and could be contributed to this base not being of much importance to the nanopore signal and thus unrelated to optimizer outcome. A similar effect can be seen for CCCTGG, but only in the MSE score. The corresponding optimizer value seems to converge quite well. It could be that a single k-mer's MSE penalty does not influence the sum of MSE's enough compared to the improvement of the others and therefore the optimizer seems sure about the value.

The optimized values converge to a point that is probably a local optimum which corresponds to a low MSE score. This can be seen by the differences between the purple and red lines in the optimization values and on the labels of Figure 6. From the tombo default DNA model, we know that ~1300 unique values for unique k-mers fit in the range of -4 to 4. The differences we see here are clearly larger than the range where a single k-mer should fall (8/1300 suggests ~2 to 3 decimals accuracy required to be within a reasonable range). This means that we see a local optimum, which makes this method not successful in recognizing wrong bases in the reference sequence.

Summed MSE values

The value that is passed to the optimizer is the sum of these 6 MSE's (Figure 7). To interpret this score, we compare it to the summed MSE for the same k-mers for a non-modified or optimized re-squiggle. Based on the summed MSE score the optimizer performs well, because the sum of the MSE is lower compared to a 'regular' re-squiggle model. However, the discrepancies between model values seen in Figure 6 indicate that there are still issues with the optimization process.

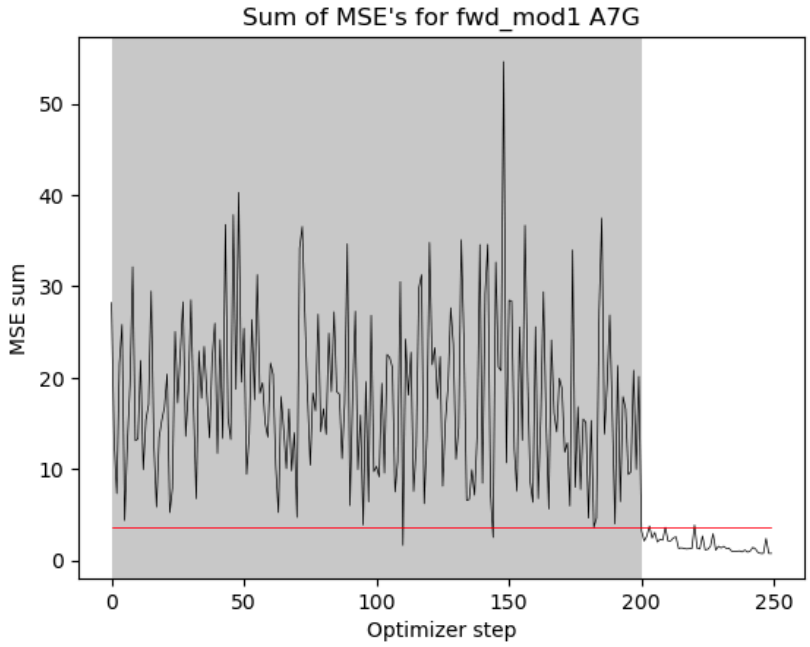


Figure 7 – Sum of MSE-scores shown in Figure 6. The red line indicates the summed MSE score of the relevant k-mers for a re-squiggle of this data and the default tombo DNA model. The grey area indicates grid search steps.

To investigate this discrepancy further, we look at the optimizations done per k-mer to see whether there is a general trend in optimization (Figure 8). The regression line in both plots shows no clear correlation between optimizer predictions and actual values, while ideally all datapoints would lie on the diagonal. The lack of correlation indicates that the earlier conclusion about local optima goes for all optimizations in the dataset, probably due to the summed MSE poorly reflecting whether an optimization is correct.

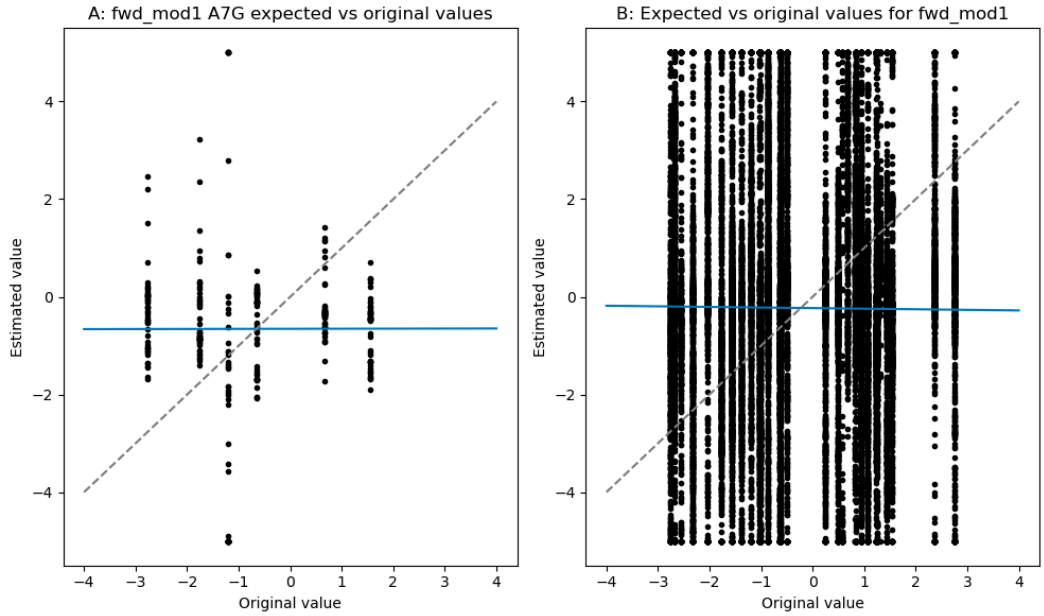


Figure 8 – Overview of estimated values for the A7G permutation on fwd_mod1 (A) and all in silico permutations on this reference sequence (B). The blue line indicates a regression on the plot's data points, the dashed grey line indicates the diagonal between estimated and tombo default DNA model values (identity function).

Random forest modification calling

We then investigated whether irregularities in the re-squiggle can be detected without any alterations to the tombo model. A re-squiggle result on a correct reference sequence could contain hints as to the location of DNA modifications. When non-canonical bases are sequenced and re-squiggled, they could cause misalignment of the re-squiggle results due to the signal levels being different from what would be expected from canonical bases. Therefore, descriptive features of this re-squiggled data can possibly be used to train a machine learning model to recognize native DNA modifications. To train and test a machine learning model for this purpose, artificial errors are induced in a re-squiggle dataset with known reference sequences by mutating these reference sequences *in silico*. This *in silico* mutation data gives complete certainty of labels being correct, allowing for a reliable accuracy measure within this context. Here we evaluate the performance of a random forest model for recognizing these *in silico* mutations based on their re-squiggle results.

Inducing re-squiggle errors

Using the oligo dataset, re-squiggle errors are introduced for each location on every reference sequence (*in silico* mutations). This is done by passing an incorrect reference sequence to the re-squiggle algorithm alongside the default (assumed correct) tombo model. An example is shown in Figure 9, where G11 of rev_mod1 is re-squiggled correctly once (Figure 9) and three times for incorrect reference sequences (Figure 9B-D). Some of these erroneous re-squiggles are easily spotted: *in silico* mutations to cytosine and thymine clearly leave artefacts by shifting parts of the signal to other bases. *In silico* mutation to adenine is less clear, with only some slight noise visible that is hard to distinguish by eye.

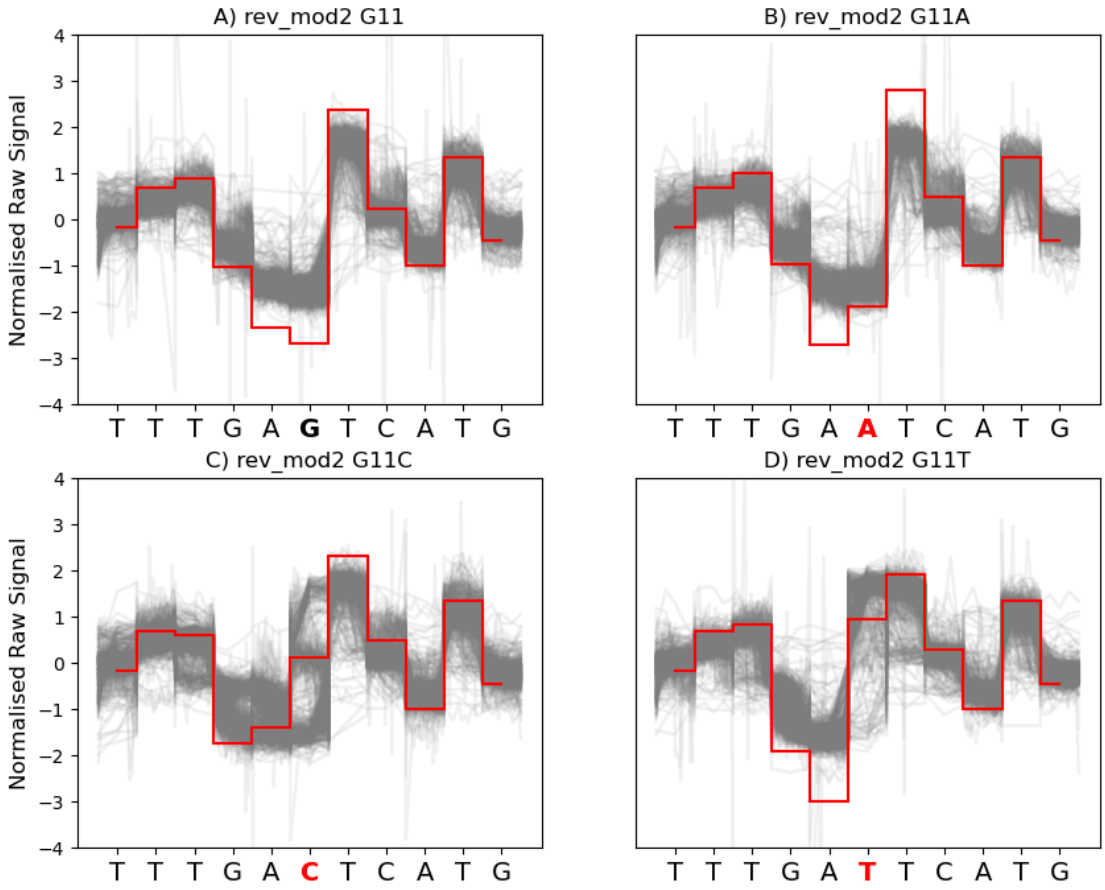


Figure 9 – Stack plot examples of re-squiggle mistakes due to reference sequence modifications on position 11 (originally guanine). Figures of all other in silico permutations are available in the GitHub repository (appendix 4). These are normalized signals re-squiggled to the A) correct reference sequence, B) G>A modified sequence, C) G>C modified sequence, D) G>T modified sequence.

Feature calculations, data splitting

To quantify these re-squiggles, several features are calculated for each base. These are descriptive features for the 'base of interest' and preceding and succeeding bases. Because nanopore sequencing signals (and the re-squiggle algorithm) works with 6-mers. Features are calculated on the segmented data for all 6 bases that would be included in the tombo 6-mer and labeled individually. The 'base of interest' is located on position 3 of this 6-mer. These features are the mean, median, standard deviation (stdev), number of datapoints as a fraction of the whole 6-mer (len), and slope of linear regression. Several signals can be stacked to reduce noise, this is referred to as depth of the signal.

Feature tables were generated based on 2 parameters: depth and sub-sampling. Depth refers to the number of squiggles used in a stack, and the sub-sampling factor decides how often this stack is built (From a different sample of squiggles) and features are calculated. Three main experiments were done with these datasets investigating the overall accuracy and the effect of depth, accuracy between oligo's, and accuracy per base (an overview of these experiments can be found in appendix B, Figure B3).

These experiments require different splits of the data. For the general accuracy experiment data was balanced from the original 75/25 split for *in silico* modifications to 50/50 and then split in a train and test fold. For the other experiments it was important to isolate the reference sequences to prevent learning a representation of the reference sequence and since balancing steps are done through down sampling splitting and balancing steps were as a result reversed to maximize the number of samples

used for model training and testing. In all experiments random forests were fitted with 1000 estimators and a minimal sample split size of 2.

Overall accuracy

The maximum accuracy (98.7%) was obtained when using the complete depth of the dataset (varying from 3822 to 6477, Table 3), indicated as the dashed line in Figure 10. This utilizes the full dataset to get a view of what features of a stacked signal look like. Next, we evaluated the performance of the random forest model as a function of depth (Figure 10), showing that higher depth improves model performance towards the maximum achievable score with this data (ranging from 36% to 97%).

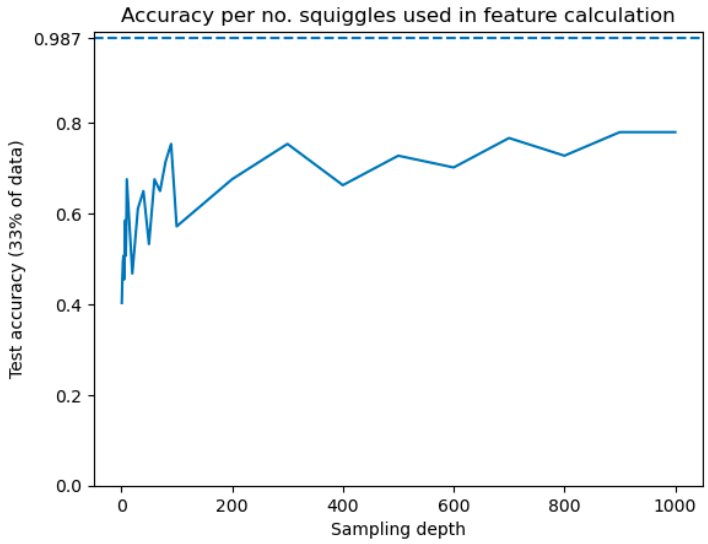


Figure 10 – Accuracy for different sampling depths. The dashed line at the top provides accuracy when the maximum sampling depth is used i.e., the max number of re-squiggle hits available for the train reference sequence. Training and testing were done on a single sample of every base of every oligo with a 66%/33% split.

For the best performing model, we then investigated the features used (Figure 11). Features are visibly grouped on feature type rather than on 6-mer location. The slope of regression is clearly the most important feature and mean the least important, possibly because these are better and worse measures of the visible bimodality in Figure 9 respectively. As one can imagine, a re-squiggle mistake is identified by horizontal shifts of datapoints to other parts of the stack. These datapoints are then at the beginning or end of the next base and can therefore more easily influence the slope than the mean.

The re-squiggle algorithm also determines where signal points are placed based on some expected mean value, and due to the high density of tombo model values within the signal range it is nearly impossible to find a meaningful mean value that indicates some specific *in silico* mutation. This density of tombo model values also proved to be an issue when using Bayesian optimization for discovery of new model values.

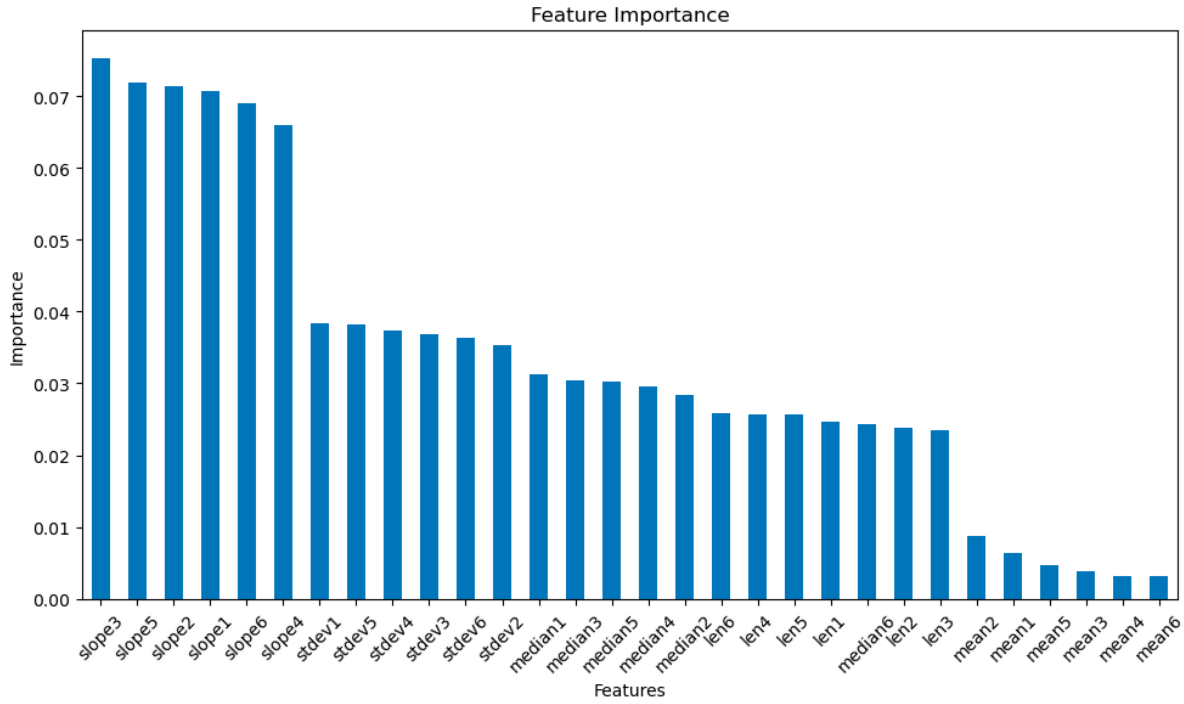


Figure 11 – Feature importance for random forest with highest performance. Relative importance is shown on the y-axis, all used features are shown. The same model is used to calculate the dashed line in Figure 10.

Accuracy between Oligo's

To investigate whether the models overfits to certain DNA sequences, we performed cross-validation between the different reference sequences, combinations perform better than others due to overfitting, their combinations were trained and tested (including to self) (Figure 12).This experiment again shows an increase in accuracy at different depths, with depth 1 ranging from 80%-90% accuracy, and maximum depth getting close to perfect accuracy for all dataset combinations. Interestingly, the difference in performance between testing on the same DNA sequence as trained, or a different one is small (max performance difference of 7% over all depths, Figure 12), suggesting that the model has generalized beyond the DNA sequence.

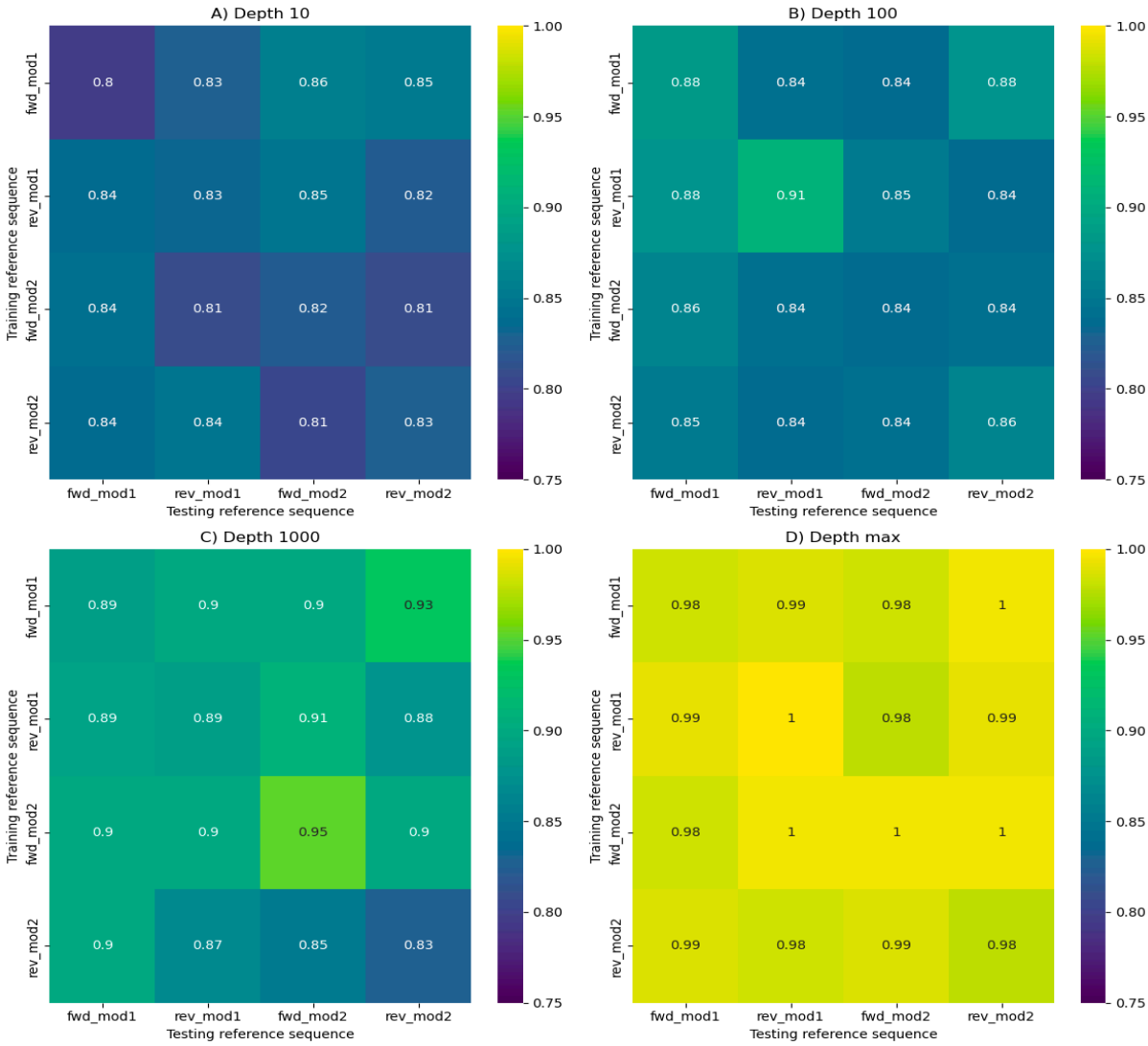


Figure 12 – Accuracy between different training/testing combinations of reference sequences plotted for different depths in panels A-D. Generated with a sub-sampling factor of 1.

Per-base accuracy

To calculate per-base accuracy, multiple samples of the feature data for a given base were used and train/test data was split based on headers instead of the mixed approach used above (note appendix B3). With multiple samples for a given permutation it is now possible to achieve an accuracy per base that is more nuanced than a binary accuracy score when classifying only a single sample. When we do this there seems to be an increase in accuracy when a higher re-sampling number is used (Figure 13). However, sub-samplings are compute intensive and therefore limited to a factor 10 for this Figure. The increase in accuracy for different depths is less visible in this Figure compared to Figure 10. In the previous experiment the model started at accuracy around 0,5 and here it starts around 0,8. An explanation for this could be that learning from a single reference sequence and testing on another one is more effective and there is some overfitting going on between reference sequences.

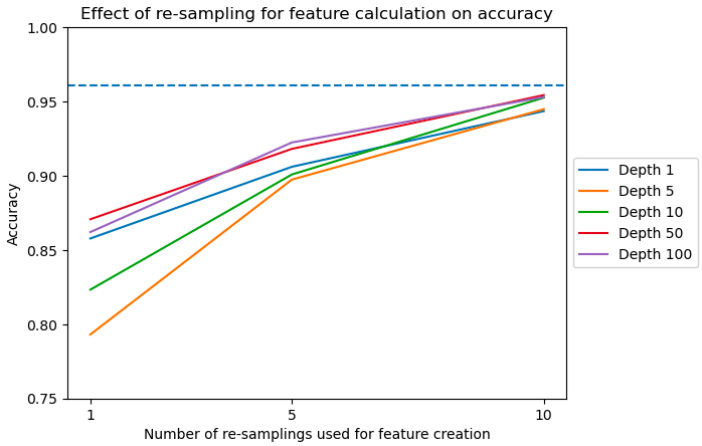


Figure 13 – Effect of re-sampling in feature calculation on accuracy, note the y-axis starting at 0.75 for better visibility of the difference between depths. The models were trained on rev_mod1 and tested on rev_mod2. The blue dashed line indicates maximum performance from Figure 10.

Accuracy by nucleotide and *in silico* mutation

To investigate whether the accuracy is uniform throughout the reference sequences, we evaluated the model results on a per mutation per reference base. This is displayed in Figure 14 for reference sequence fwd_mod1. The random forest used to generate this data was trained on the other three reference sequences with depth 80 and re-sampling factor 50. The plot indicates that for some bases the accuracy is lower (< 0.4) than for others, such as A10, T12, G18, A19, and C28. This same plot can be found for the other reference sequences in appendix C. This plot raises the question whether certain *in silico* mutation types are harder to detect than others, therefore corresponding accuracies are visualized in Figure 15. Interestingly changes from cytosine to thymine (e.g., C28 and C33 in Figure 14) are most difficult to detect while cytosine to other bases has an accuracy of 1. Another bottleneck in detection seems adenine to guanine, with a near 50/50 chance of correct identification. Other mutations range around 0.8, which is more sensible for a depth around 80, and corresponds to the results obtained on the whole sequence.

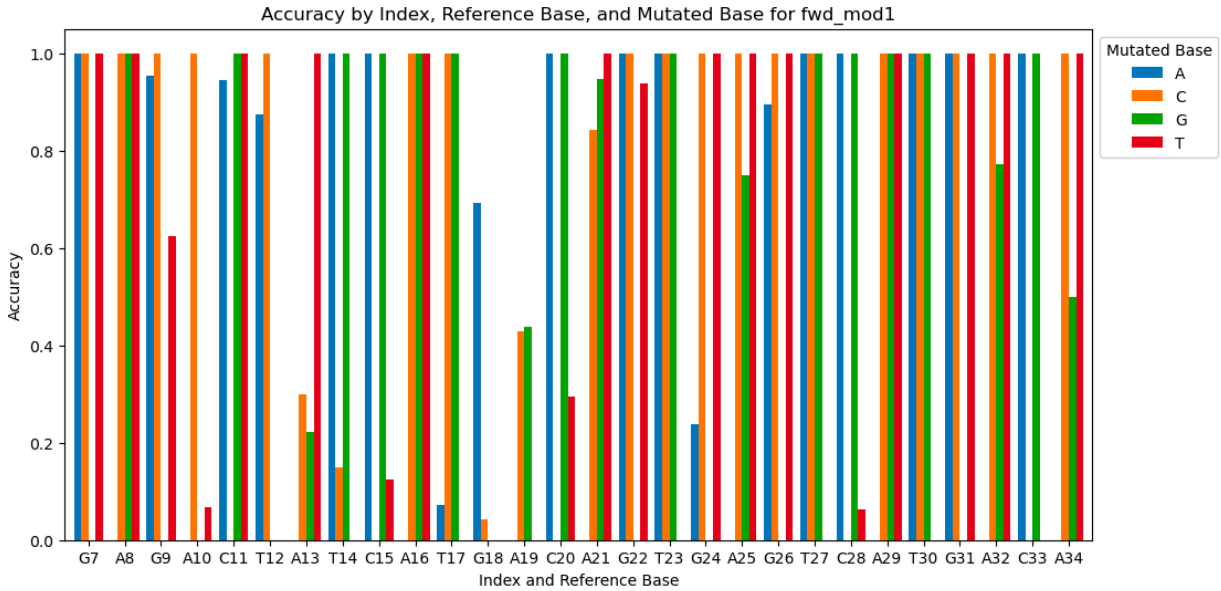


Figure 14 – Accuracy per index of fwd_mod1, shown for the 3 corresponding mutations for a given base at a given index (x-axis). This test performance was generated with 50 re-samples of data with depth 80. Training was done on the other three reference sequences. This graph for the other reference sequences can be found in appendix C.

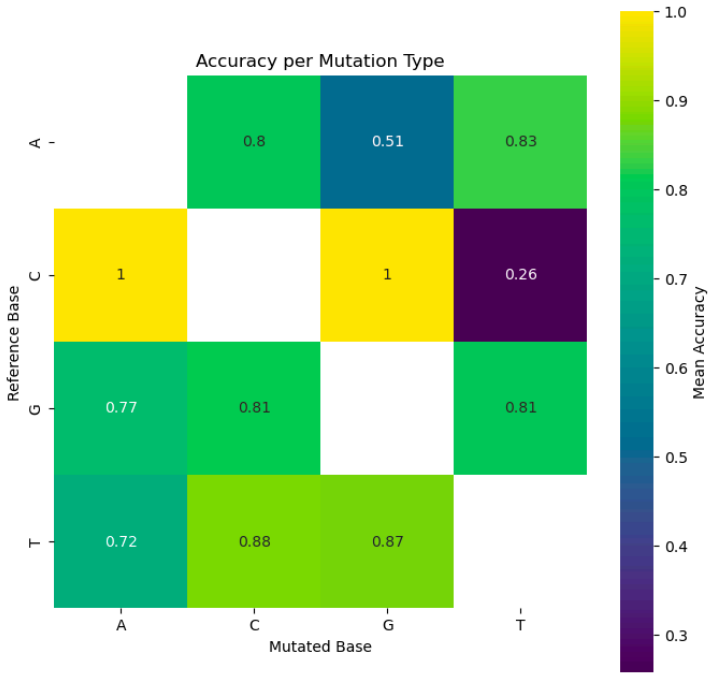


Figure 15 – Heatmap of accuracy per mutation type for the plot of fwd_mod1 in Figure 14.

Attempted prediction of oxidized guanine

For actual recognition of 8-oxodG or methylation we will eventually need to step away from large and artificially tweaked datasets. Given that two of our stacks contain 8-oxodG, we can however test the accuracy on a stacked squiggle. To locate one of the 8-oxodG modifications, the dataset for fwd_mod1 with depth 1000 and re-sampling factor of 10 was used to train a random forest and predict the modification of 8-oxodG on position 18. This model performed with an accuracy of 0.35 for detecting 8-oxodG on location 18 of fwd_mod2 and 0.98 on the *in silico* modified locations of this sequence. However, no method to validate this outcome is possible with the given dataset due to a risk of the model learning a representation of the sequence when fitting to training data.

Discussion

In this project we attempted to recognize irregularities in the re-squiggle workflow by either optimizing values of the k-mer table used for the re-squiggle algorithm or by recognizing issues in a re-squiggle with machine learning (the random forest approach). The former experiment was ineffective, while the latter seemed to work to a reasonable degree with improvements to be made. The process yielded several possible improvements and opportunities for further research. There are also some general considerations related to the essence of this problem to be discussed.

Bayesian optimization of tombo models

In the attempt to re-optimize tombo's k-mer table using Bayesian optimization it became clear that this was not an effective approach to recognize irregularities in raw stacked sequencing signals. There are several explanations and points of improvements for this. These are either related to optimizer hyperparameters (acquisition function, number of iterations used in grid search and optimization), or to the scoring method used (penalizing failed squiggles as an improvement over MSE).

Optimization relies on a score calculated from a successful re-squiggle of the raw signal to its expected counterpart, which is constructed from the k-mer table. However, sometimes the re-squiggle algorithm fails due to large incompatibilities between the alignment of the two components. In these cases, we used a penalizing MSE score of 10; however, successful re-squiggles with relatively bad results can already achieve MSE scores around 5-8. We hypothesize that a score of 10 might not be stringent enough to indicate a relatively large error to the optimizer. A larger and clear indication of re-squiggle errors should be used here to indicate that most parts of the search space are not relevant. In addition to tweaking the penalty, the MSE compared to stack mean as a scoring method might not incorporate enough information for this specific problem. A low MSE score was achieved by the optimizer, but this low score does not correspond to accurate k-mer values, which could indicate that the optimizer gets stuck on local optima.

There are multiple ways of solving this local optimum problem, one of which would be using a different scoring function. The used scoring function only considers the error between expected and measured, disregarding the amount of assigned datapoints. An alternative method could be adding a penalty to the MSE calculation that increases as the number of datapoints per base deviates from some expected range. A pre-calculated distribution with the expected number of datapoints per k-mer (or just one general distribution) could be used to generate a penalty value. The k-mer MSE score can then be weighted by this penalty, for example by how many standard deviations the number of bases in a squiggle deviates from the expected value. This would incorporate the penalty on a per-base level, making the method easy to adjust for different contexts.

A downside of this scoring method is that it does not consider the DNA speed changes that occur while sequencing. When a re-squiggle model is penalized for assigning 'too many' or 'too few' points to a segment there is not a clear definition of many or few when the speed is stochastic. Therefore, the distribution of DNA speeds needs to be investigated to help define how stringent the proposed scoring method should be. There is a tradeoff to be considered between accuracy of model values and accuracy in correctly recognizing segments (which in turn influences model values through mis-assignment), ideally this tradeoff can be solved by investigating squiggle data for different contexts.

Another way to approach the local optimum problem is via the acquisition function used in the optimizer. The function used here is "GP Hedge", the default function in the scikit optimize package. This package provides several other acquisition functions: lower confidence bound, expected improvement, and probability of improvement. GP Hedge uses a portfolio strategy approach where a

hedge algorithm decides which of the aforementioned acquisition functions is used in every iteration, giving better performance overall than individual acquisition functions (Brochu et al., 2011). Given that the results of our optimization experiment have many local optima, it could be that in this case an acquisition function which is strictly less exploitative (since the hedge algorithm controls this balance for us) might be useful in our case. An example of this would be to use entropy search, where the goal is to minimize the uncertainty of the location of the optimal value by exploring other regions of the search space (Wang & Jegelka, 2018), however entropy search is known to be computationally intensive and would require more time to test and compare to the GP Hedge method.

Furthermore, hyperparameter tweaking could also benefit the results obtained here. In our experiments, we used 200 points for grid search and 50 subsequent iterations of optimization. This was chosen due to a combination of calculation time constraint and the seemingly converging MSE scores. Several tests on a single mutation with more grid search points and optimization iterations showed that the MSE converged similarly to lower value parameters at the cost of more CPU time. For further research it might be useful to run the optimizer for more values, however the time per iteration steeply increased when adding optimizer steps so this is a tradeoff to keep in mind with further experimentation.

In following research, multiple scoring methods should be tested with the used *in silico* mutation method to see which is most effective for finding the optimum corresponding to a base or irregularity. With a perfect scoring method, it should at least be possible to find a range closely corresponding to the original value (as these are defined by a mean and standard deviation both for the k-mer table and a gaussian process), and then move on to detection of novel ranges for unknown irregularities in the raw signal. Our results so far indicated that this approach is not valid, however aforementioned issues and tweaks indicate some of the many unexplored avenues towards optimizing k-mer tables for the tombo algorithm to recognize squiggle irregularities.

Random forest modification calling

To test whether irregularities can be classified using machine learning model, the *in silico* approach for making erroneous squiggles used above was further to generate labeled feature data. This data was used to train and test a random forest model for recognizing mutations. The random forest model works well within the context of the experiment. However, the model depends on high depth or subsampling for training which is unachievable for a single read. For practical application in recognizing irregularities in a signal with depth 1 improvements are required. Several issues with the model should also be addressed; there seem to be irregular performances due to subsampling, the data only covers a small part of the k-mer space, and there are other types of models available (and well-known) for dealing with nanopore read data.

A method of addressing the issue of low depth corresponding to low accuracy could be to train a model with a high depth dataset measure its accuracy on lower-depth data. The accuracy here could be different from models trained on equal depth data and it might be worth investigating the relationship between train data depth and test data depth. Like the experiments done here this will require correct splitting for overfitting of the model and possibly training on more k-mer contexts for generalization to any other read dataset.

The observed performance difference between low depth without subsampling and low depth with subsampling shows that sampling increases the information from reads available to the model. This makes sense as the features do not contain a lot of information and could be resulting from many different distributions. An extensive search to an effective combination of subsampling factor and depth factor might allow for a model with better performance on lower density data.

In addition to this, the information contained in the features might be improved if some measure of modality is added to these calculations. As was seen in the graphs of Figure 9, a stack with high depth can contain multiple distinct peaks of erroneous re-squiggles. Further research into new features such as the number of peaks can be achieved with gaussian mixture models, kernel density estimation, or statistical tests for unimodality of the data. Random forests allow for easy feature engineering with access to feature importance for every model. So varying statistics can be tested, and the best performing should be used for validation and improvement of the model.

A different and very popular model type that can be used to solve these kinds of problems are deep learning models. Deep learning models perform well in base calling because of the high noise and high quantity of sequencing data. Given many genetic contexts for a known irregularity, a deep learning model could be trained to recognize a modification in a base caller-like fashion. An example of this would be the base caller Guppy, which is now able to detect 5mC, a prevalent epigenetic marker. As most basecallers are, this is developed for single signals. The experiments we attempted are aimed at repetitive data that allows for stacking of signals and resulting extra ability of recognizing irregularities. Therefore, it might be an interesting avenue to adapt or develop a basecaller method to work with stacked signals.

Further research into irregular nanopore signals

For further experimentation with epigenetic modifications, it is useful to consider there are many nucleotide modifications to consider and these all have different properties and frequency of occurrence – which can also vary in health versus disease (Sood et al., 2019). The oligonucleotides used here contained 8-oxodG on 2 locations. A final accuracy test on this modification was performed to see whether the most accurate random forest model could distinguish them. This test worked to some degree (accuracy of 0.35), which is a hopeful result given that the model was only trained on *in silico* mutated data. If this approach is further developed, it can be useful to keep in mind any tradeoffs unique to the irregularity that should be detected. For example, a major problem with 8-oxodG (and several other epigenetic mutations/modifications) remains unaddressed; due to a low frequency of occurrence and the possibly tiny changes in signal level corresponding to them it remains difficult to achieve high true positive and true negative rates when detecting modifications in human genetic material. Due to the biological and clinical relevance of 8-oxodG, 5mC, and other epigenetic markers, it remains important to develop methods for detecting these from nanopore sequencing data despite their individual difficulties.

Methods

Data used

Four oligonucleotides (oligos) were sequenced to obtain a dataset that can be used for re-squiggling evaluation. Two oligos contain oxidized guanine (8-oxodG), the other two are free of modifications or damage (appendix B1). For each oligo we have approximately between 500 and 1500 sequenced reads (appendix B2), containing repetitions of the reference sequence. In total the dataset contains 4000 reads.

Reads were assigned to a reference sequence using the scikit-bio SSW (Smith-Waterman) alignment implementation. This allows local sequence alignment between a read's base calls and the reference sequence. The read is assigned to the reference sequence with the highest score, with a cutoff of any scores below 50. Some reads were therefore not assigned to a reference sequence; these are categorized as 'None'. These reads do however contain regions that are recognized by the re-squiggle algorithm (76 regions in 33 reads) that are used in stacking (appendix B2).

Raw signal to base assignment

The raw signals were re-squiggled using tombo version 1.4, via a custom script that interfaces with the tombo API (link in appendix D). The re-squiggle algorithm outputs segmentation values that map the raw signal values to a reference sequence). These segmentation values describe the start position in a read, and the number of raw values corresponding to each base. Due to DNA speed changes the number of datapoints per base differ, thus the segmentation values allow us to stack squiggles and use them for further experimentation.

A reference sequence (or: genomic sequence) is required to align a squiggle. Raw reads are only aligned from the 3rd to the 3rd to last location of this sequence due to 6-mers in the re-squiggle table giving the value corresponding to their 3rd base. Given that the reference sequences are repeated in a read, reference sequences are padded with the 2 last bases of the repeat before it and the 3 first bases of the repeat after it. This allows for a stacked squiggle for the complete reference sequence instead of just base 3 through 37.

Raw signal normalization

To prevent irregularities between reads each raw signal is normalized by its mean and standard deviation (Equation 3). After normalization and given the re-squiggle segments, reads are stacked (or: placed in 'bins') by their segments.

$$\text{normalized}(\mathrm{X}) = \frac{(\mathbf{x} - \mu)}{\sigma}$$

Equation 3 – Normalization of read data by the mean and standard deviation of a read.

Reference sequence cropping

For the two following mutation experiments, mutations were made from the sixth until the sixth to last position of reference sequences. This is to prevent any signal effects from outside of a known re-squiggle result window. Re-squiggles are organized by the start and end values in the raw signal that corresponds to the reference sequence, a small buffer ensures that effects of changes to the k-mer table that occur earlier/later in the sequence than the mutated base are found in the region of a stack.

Bayesian optimization of tombo models

In silico mutation data generation

For the optimization experiment a single nucleotide of a reference sequence was chosen and mutated to a different base. This was then assumed as the new reference sequence and all reads in the dataset were re-squiggled again to this new read. This re-squiggle result is scored, and the process is repeated using varying permutations of the tombo k-mer table as decided by the optimizer. The optimization process was then repeated for each nucleotide in the reference sequence and its corresponding possible mutations, resulting in 360 mutated re-squiggle datasets and 4 wildtype re-squiggle datasets containing all optimizer suggestions.

K-mer table modification

The tombo k-mer table (model) contains a standard deviation and a mean for each k-mer. The 6 k-mer values corresponding to a mutation are modified in this model during optimization. We did not re-calculate a standard deviation, therefore only the mean values for relevant k-mers are replaced for a new optimizer iteration and this model is then passed to the re-squiggle algorithm. This was done using the default DNA model found in tombo version 1.4.

Scoring function

Each iteration a new k-mer model was generated from the six-dimensional optimizer suggestion, this model was used to re-squiggle the dataset, and the summed MSE score (Mean Squared Error relative to the stack mean, equation 4) was calculated and returned to the optimizer. To calculate the MSE score the reads are stacked, to combine all the raw values of the same base. These values are then used to calculate the MSE for this given base relative to the expected value in the suggested tombo model. The summed MSE score is the sum of the 6 relevant MSE scores surrounding the mutated base; 2 bases in upstream, the base itself, and 3 bases downstream in the reference sequence.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_{mean})^2$$

Equation 4 – Mean-Squared-Error calculation relative to the mean of the stacked data. Y_i indicates the datapoint within a stacked squiggle, y_{mean} indicates the mean of all datapoints in a stack of squiggles for a given base. The MSE scores are calculated per base, and the sum of 6 relevant bases is used for the summed MSE.

Optimizer hyperparameters

The optimizer used is the default Optimizer from scikit-optimize version 0.9.0, conFigured to search for float values ranging from -5.0 to 5.0 in six dimensions. For reproducibility the random state was initialized to 1. The base estimator is set to 'GP', the gaussian process estimator. 200 initial grid search points were used after which optimization was done for 50 iterations. This number of iterations was chosen after some experimentation with up to 1000 points of grid search and 100 points of optimizer steps for a single mutation. This resulted in similar convergence of the resulting MSE score but with significantly higher computational time involved, therefore it was decided that 200 grid search points and 50 iterations of optimization suffice for the full experiment with all possible mutations.

Prediction selection

To get the best possible model estimation from the iterations, they are ranked by the total MSE score and 3-fold validation was used to determine the top-N optimizer suggestions to use in calculating an accurate mean value that best reflects a low total MSE. This 3-fold validation uses the possible permutations of a base in each fold and calculate the Euclidean norm between the mean of two permutations against the third permutation for a range of n values (Equation 5). These values are aggregated by summing them for a given N, and the N value with the lowest distance was selected

(Equation 6). This suggested that N=50 (highest possible, since there are 50 iterations) is most accurate to determine a vector that most closely resembles other permutation results on the same site. This value is used to calculate the average predicted model values for the k-mers (purple line in Figure 1).

$$Euclidean\ norm(P_C) = \ ||\frac{(P_G - P_T)}{2} - P_A||$$

Equation 5 – Distance between 2 groups and the third group as used in fold validation of n. Different P's represent the vector of suggestions for a given base of a given permutation. In this example a cytosine is mutated and A is left out of the results for fold validation.

$$\min_{[1,50]} (\sum_{i=1}^{Folds} Euclidean\ norm\ (P_C))$$

Equation 6 – Minimal of the sum Euclidean norm to determine the optimal n value. The range [1,50] is determined by the number of optimization iterations. The number of folds is decided by the number of optimization experiments included, for this validation 30 optimization experiments (90 groupings) were used.

Random forest modification calling

In silico data generation

In silico mutations are induced in the reference sequence, then the raw data is re-squiggled against this modified reference sequence using the default tombo model. From this re-squiggled data, features are generated which are then used to train a random forest model.

Features for the random forest are calculated for the *in silico* mutated base and for relevant neighboring bases. Relevant neighboring bases are the two bases before and three bases following the *in silico* mutated base. This total of 6 bases is selected because these are part of the 6-mer present in the nanopore. 5 features are calculated per base (a total of 30 features) and defined as follows:

- *Mean*: The mean value of all datapoints in a stacked signal.
- *Median:* The median value of all datapoints in a stacked signal.
- *Relative length:* The number of datapoints in a stacked signal of a given base, divided by the total number of datapoints for the 6 bases of interest.
- *Standard deviation:* The standard deviation of all datapoints in a stacked signal.
- *Slope of regression:* The slope of regression after the datapoints for each raw signal in the stack are evenly spaced over an x-axis range of 0 to 1.

Feature calculation hyperparameters

To simulate training data, we create *in silico* re-squiggle mistakes by inserting an incorrect base in the used reference sequence. Re-squiggle and feature gathering steps are repeated for all mutations. This results in a dataset with 25% correct bases' features and 75% incorrect bases' features.

Feature values can be calculated at different depths (number of squiggles used in the calculation). The depths used are 1-10 with increments of 1, 10-100 with increments of 10, and 100-1000 with increments of 100. We also calculate the features corresponding to maximum depth, which depends on the number of available re-squiggles results and thus varies per reference sequence from 3882 to 6477 (Table 3).

In addition to this, other feature tables were generated with subsampling. This was done by randomly selecting the squiggles used and adding their results to the feature table, resulting in more entries to

train/test on. Different combinations of sub-sampling and depth are used in different experiments and are further described below.

Cross-validation and balancing

Three experiments were done with different data splits and balancing steps (Figure B3). First, to establish the maximum achievable accuracy, the feature dataset was first label balanced by down sampling. Afterwards it was split using 66% samples for training and 33% for testing. This was done for all available depths and without any sub-sampling.

To calculate the accuracy between oligonucleotides the data was first split on reference sequence; then, combinations of two reference sequences were selected for training and testing data. Finally, label balancing was applied. This is done for all combinations of reference sequences and with subsampling factor 5 and with depth 1, 10, 100, 1000 and maximum depth.

For the accuracy per nucleotide first a test oligo is selected, and then the dataset is balanced by sampling from a mix of the three remaining (train) oligos. This was done with a depth of 80 and re-sampling of 50 to maximize the information available per-base. The same random forest was used to determine the accuracy per mutation type (For each oligo to another, Figure [X]).

Random forest hyperparameters

The random forests used are built from the RandomForestClassifier class from the scikit-learn package, version 1.2.2. All forests are conFigured with 1000 estimators, no maximum tree depth, and a minimum sample split of 2.

References

Amarasinghe, S. L., Su, S., Dong, X., Zappia, L., Ritchie, M. E., & Gouil, Q. (2020). Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, *21*(1), 30. https://doi.org/10.1186/s13059-020-1935-5

Beckhauser, T. F., Francis-Oliveira, J., & De Pasquale, R. (2016). Reactive Oxygen Species: Physiological and Physiopathological Effects on Synaptic Plasticity. *Journal of Experimental Neuroscience*, *10*(Suppl 1), 23–48. https://doi.org/10.4137/JEN.S39887

Bonet, J., Chen, M., Dabad, M., Heath, S., Gonzalez-Perez, A., Lopez-Bigas, N., & Lagergren, J. (2021). *DeepMP: A deep learning tool to detect DNA base modifications on Nanopore sequencing data* (p. 2021.06.28.450135). bioRxiv. https://doi.org/10.1101/2021.06.28.450135

Breiling, A., & Lyko, F. (2015). Epigenetic regulatory functions of DNA modifications: 5-methylcytosine and beyond. *Epigenetics & Chromatin*, *8*(1), 24. https://doi.org/10.1186/s13072-015-0016-6

Brochu, E., Hoffman, M. W., & de Freitas, N. (2011). *Portfolio Allocation for Bayesian Optimization* (arXiv:1009.5419). arXiv. http://arxiv.org/abs/1009.5419

Ehrlich, M. (2019). DNA hypermethylation in disease: Mechanisms and clinical relevance. *Epigenetics*, *14*(12), 1141–1163. https://doi.org/10.1080/15592294.2019.1638701

Fouquerel, E., Barnes, R. P., Uttam, S., Watkins, S. C., Bruchez, M. P., & Opresko, P. L. (2019). Targeted and Persistent 8-Oxoguanine Base Damage at Telomeres Promotes Telomere Loss and Crisis. *Molecular Cell*, *75*(1), 117-130.e6. https://doi.org/10.1016/j.molcel.2019.04.024

Hahm, J. Y., Park, J., Jang, E.-S., & Chi, S. W. (2022). 8-Oxoguanine: From oxidative damage to epigenetic and epitranscriptional modification. *Experimental & Molecular Medicine*, *54*(10), Article 10. https://doi.org/10.1038/s12276-022-00822-z

Hartley, G., & O'Neill, R. J. (2019). Centromere Repeats: Hidden Gems of the Genome. *Genes*, *10*(3), 223. https://doi.org/10.3390/genes10030223

International Human Genome Sequencing Consortium. (2004). Finishing the euchromatic sequence of the human genome. *Nature*, *431*(7011), Article 7011. https://doi.org/10.1038/nature03001

Jain, M., Koren, S., Miga, K. H., Quick, J., Rand, A. C., Sasani, T. A., Tyson, J. R., Beggs, A. D., Dilthey, A. T., Fiddes, I. T., Malla, S., Marriott, H., Nieto, T., O'Grady, J., Olsen, H. E., Pedersen, B. S., Rhie, A., Richardson, H., Quinlan, A. R., … Loose, M. (2018). Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, *36*(4), Article 4. https://doi.org/10.1038/nbt.4060

Lister, R., & Ecker, J. R. (2009). Finding the fifth base: Genome-wide sequencing of cytosine methylation. *Genome Research*, *19*(6), 959–966. https://doi.org/10.1101/gr.083451.108

Liu, Q., Fang, L., Yu, G., Wang, D., Xiao, C.-L., & Wang, K. (2019). Detection of DNA base modifications by deep recurrent neural network on Oxford Nanopore sequencing data. *Nature Communications*, *10*(1), Article 1. https://doi.org/10.1038/s41467-019-10168-2

Liu, Q., Georgieva, D. C., Egli, D., & Wang, K. (2019). NanoMod: A computational tool to detect DNA modifications using Nanopore long-read sequencing data. *BMC Genomics*, *20*(1), 78. https://doi.org/10.1186/s12864-018-5372-8

Liu, Y., Rosikiewicz, W., Pan, Z., Jillette, N., Wang, P., Taghbalout, A., Foox, J., Mason, C., Carroll, M., Cheng, A., & Li, S. (2021). DNA methylation-calling tools for Oxford Nanopore sequencing: A survey and human epigenome-wide evaluation. *Genome Biology*, *22*(1), 295. https://doi.org/10.1186/s13059-021-02510-z

Liyanage, V. R. B., Jarmasz, J. S., Murugeshan, N., Del Bigio, M. R., Rastegar, M., & Davie, J. R. (2014). DNA Modifications: Function and Applications in Normal and Disease States. *Biology*, *3*(4), 670–723. https://doi.org/10.3390/biology3040670

Modified Base Detection—Tombo 1.5.1 documentation. (n.d.). Retrieved October 31, 2023, from https://nanoporetech.github.io/tombo/modified_base_detection.html

Müller, M. (2007). Dynamic time warping. *Information Retrieval for Music and Motion*, *2*, 69–84. https://doi.org/10.1007/978-3-540-74048-3_4

Nguyen, D., Tran, T.-A., Nguyen Quoc Khanh, L., Pham, D.-M., & Ou, Y.-Y. (2021). An extensive examination of discovering 5-Methylcytosine Sites in Genome-Wide DNA Promoters using machine learning based approaches. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1–1. https://doi.org/10.1109/TCBB.2021.3082184

Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A. V., Mikheenko, A., Vollger, M. R., Altemose, N., Uralsky, L., Gershman, A., Aganezov, S., Hoyt, S. J., Diekhans, M., Logsdon, G. A., Alonge, M., Antonarakis, S. E., Borchers, M., Bouffard, G. G., Brooks, S. Y., … Phillippy, A. M. (2022). The complete sequence of a human genome. *Science*, *376*(6588), 44–53. https://doi.org/10.1126/science.abj6987

Pagès-Gallego, M., & de Ridder, J. (2023). Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling. *Genome Biology*, *24*(1), 71. https://doi.org/10.1186/s13059-023-02903-2

Poetsch, A. R. (2020). The genomics of oxidative DNA damage, repair, and resulting mutagenesis. *Computational and Structural Biotechnology Journal*, *18*, 207–219. https://doi.org/10.1016/j.csbj.2019.12.013

Rand, A. C., Jain, M., Eizenga, J. M., Musselman-Brown, A., Olsen, H. E., Akeson, M., & Paten, B. (2017). Mapping DNA methylation with high-throughput nanopore sequencing. *Nature Methods*, *14*(4), Article 4. https://doi.org/10.1038/nmeth.4189

Re-squiggle Algorithm—Tombo 1.5.1 documentation. (n.d.). Retrieved June 15, 2023, from https://nanoporetech.github.io/tombo/resquiggle.html

Simpson, J. T., Workman, R. E., Zuzarte, P. C., David, M., Dursi, L. J., & Timp, W. (2017). Detecting DNA cytosine methylation using nanopore sequencing. *Nature Methods*, *14*(4), Article 4. https://doi.org/10.1038/nmeth.4184

Sood, A. J., Viner, C., & Hoffman, M. M. (2019). DNAmod: The DNA modification database. *Journal of Cheminformatics*, *11*(1), 30. https://doi.org/10.1186/s13321-019-0349-4

Stoiber, M., Quick, J., Egan, R., Lee, J. E., Celniker, S., Neely, R. K., Loman, N., Pennacchio, L. A., & Brown, J. (2017). *De novo Identification of DNA Modifications Enabled by Genome-Guided Nanopore Signal Processing* (p. 094672). bioRxiv. https://doi.org/10.1101/094672

Wang, Z., & Jegelka, S. (2018). *Max-value Entropy Search for Efficient Bayesian Optimization* (arXiv:1703.01968). arXiv. https://doi.org/10.48550/arXiv.1703.01968

Yuen, Z. W.-S., Srivastava, A., Daniel, R., McNevin, D., Jack, C., & Eyras, E. (2021). Systematic benchmarking of tools for CpG methylation detection from nanopore sequencing. *Nature Communications*, *12*(1), Article 1. https://doi.org/10.1038/s41467-021-23778-6

Appendices

A. Software used

Package	Version
ont-tombo	1.4
parasail	1.2.4
scikit-bio	0.5.6

This is not an extensive list, .yml files containing conda environments can be found in the GitHub repository for this project. This repository is hosted at: https://github.com/maartenvanelst/nano.

B. Data used

B1. Reference sequences

Table B1 – Reference sequences used. 8-oxodG modifications are present for two sequences at position 18, bold and in red.

Header	Sequence
fwd_mod1	AACCCTGAGACTATCAT**G**ACAGTGAGTCATGACAGACGCT
fwd_mod2	AACCCTGACTCATCTGA**G**CATGTGAGCTGAGCATGACTCA
rev_mod1	TCAGGGTTAGCGTCTGTCATGACTCACTGTCATGATAGTC
rev_mod2	TCAGGGTTTGAGTCATGCTCAGCTCACATGCTCAGATGAG

B2. Read and re-squiggle distribution over reference sequences

Table B2 – Read and re-squiggle counts of each oligonucleotide sequence.

Header	Associated number of reads	Re-squiggle results
fwd_mod1	942	5395
fwd_mod2	1409	6477
rev_mod1	957	5621
rev_mod2	659	3882
None	33	76
Total	4000	21375

B3. Feature validation and balancing graph per experiment

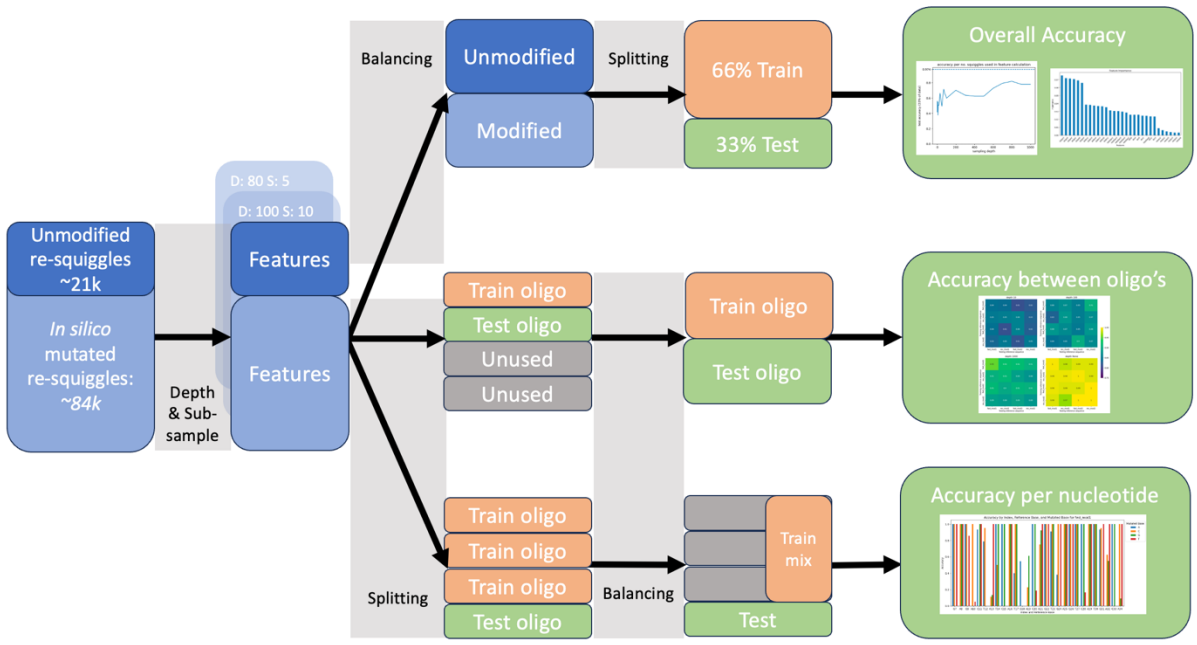
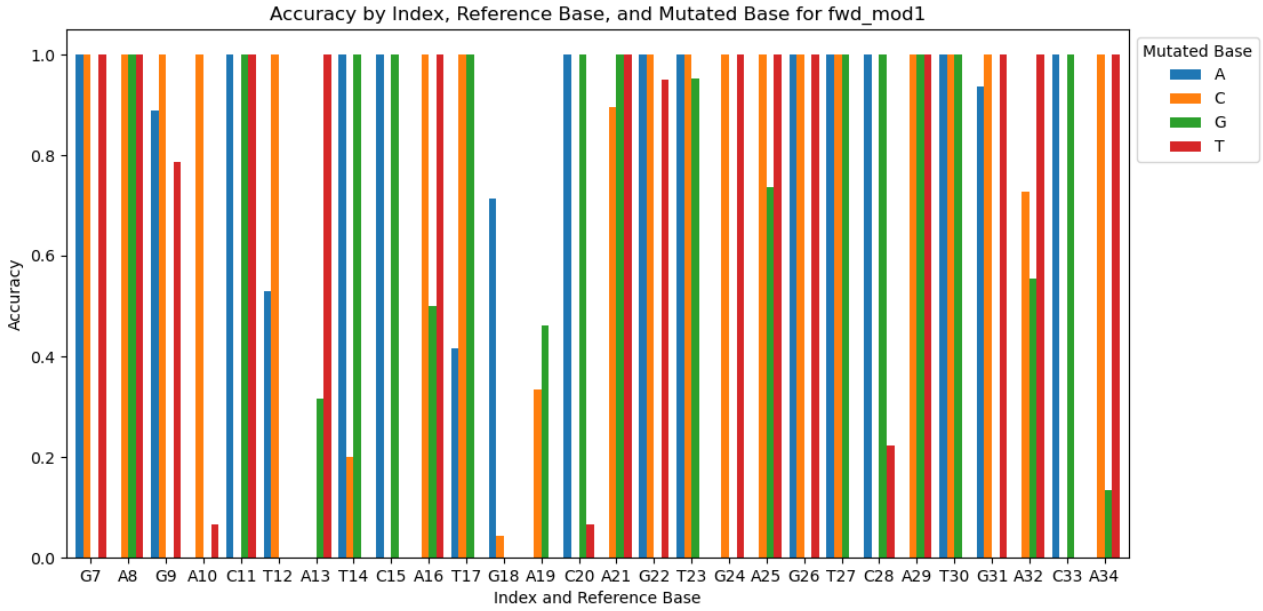
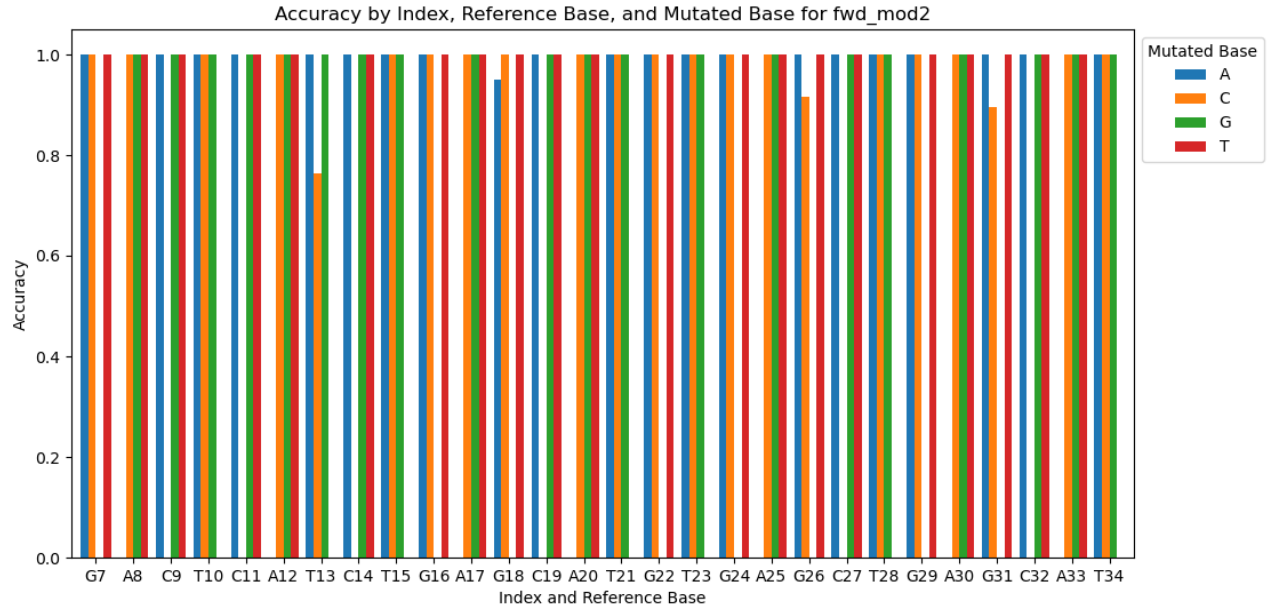
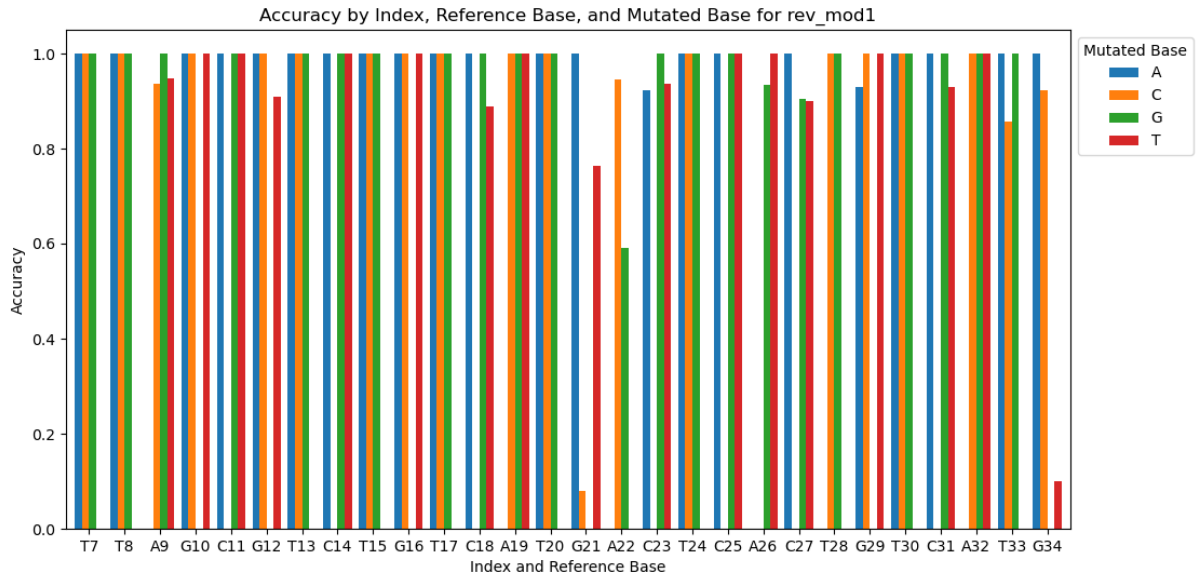


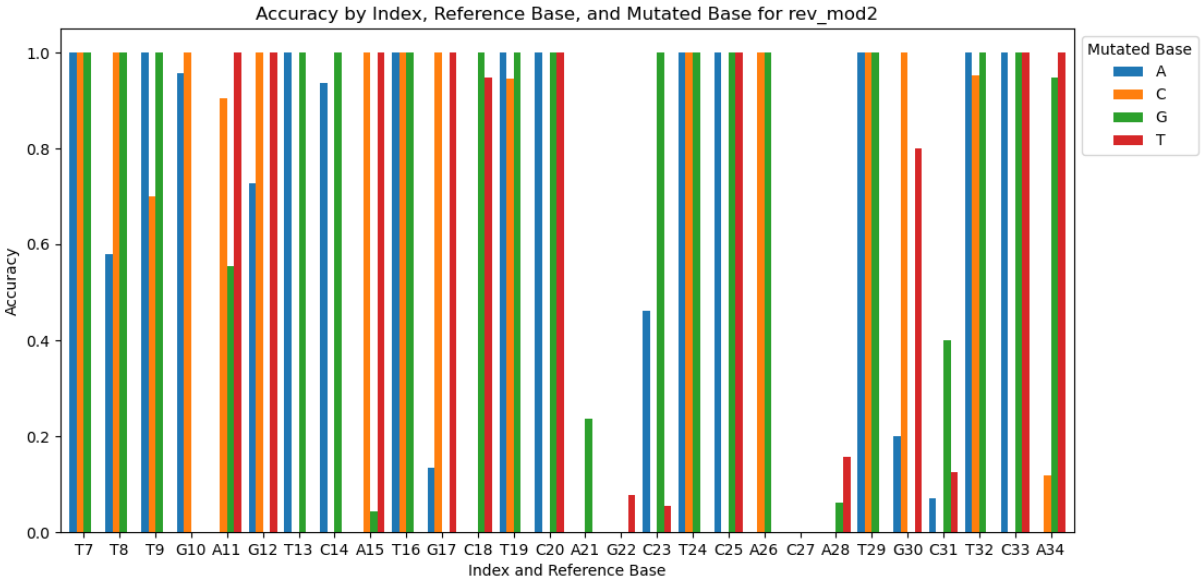
Figure B3 – Visualization of data flow for three random forest experiments. Feature datasets are generated based on 2 parameters; depth defined the number of squiggles used for feature calculations, and sub-sampling how often these squiggles are sampled and features calculated. Balancing steps always indicate a 50/50 balance of modified and unmodified data, splitting steps differ based on the experimental setup.

C. Accuracy per base plot for each oligo






Accuracy by Index, Reference Base, and Mutated Base for rev_mod1
Mutated Base
A
C
G
T
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
T7 T8 A9 G10 C11 G12 T13 C14 T15 G16 T17 C18 A19 T20 G21 A22 C23 T24 C25 A26 C27 T28 G29 T30 C31 A32 T33 G34
Index and Reference Base


Accuracy by Index, Reference Base, and Mutated Base for rev_mod2
Mutated Base
A
C
G
T
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
T7 T8 T9 G10 A11 G12 T13 C14 A15 T16 G17 C18 T19 C20 A21 G22 C23 T24 C25 A26 C27 A28 T29 G30 C31 T32 C33 A34
Index and Reference Base

D. Result and code URLs

Results and code are available in the GitHub repository hosted here: https://github.com/maartenvanelst/nano

Optimizer Figures can be found here:
https://github.com/maartenvanelst/nano/tree/master/optimize_tombo/Figures/sixplot

Custom script for re-squiggle using the tombo API:
nano/src/resquiggle_tombo.py at master · maartenvanelst/nano (github.com)