Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Monitoring and Controlling Nanopore Sequencing Runs

Master's Thesis

2023
Bc. Matej Fedor

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Monitoring and Controlling Nanopore Sequencing Runs

Master's Thesis

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	FMFI.KAI - Katedra aplikovanej informatiky
Školiteľ:	doc. Mgr. Tomáš Vinař, PhD.

Bratislava, 2023
Bc. Matej Fedor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Matej Fedor
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Monitoring and Controlling Nanopore Sequencing Runs
Monitorovanie a riadenie nanopórových sekvenačných behov

Anotácia: Cieľom tejto práce je vyvinúť nástroje, ktoré umožnia monitorovať priebeh nanopórového sekvenovania v reálnom čase. Ďalším cieľom je preskúmať možnosti riadenia sekvenovania prostredníctvom selektívneho sekvenovania. Vyvíjané nástroje budeme aplikovať v kontexte sekvenovania patogénov (ako napr. COVID-19).

Vedúci: doc. Mgr. Tomáš Vinař, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 13.10.2020

Dátum schválenia: 28.04.2023 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Matej Fedor
Study programme: Computer Science (Single degree study, master II. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Diploma Thesis
Language of Thesis: English
Secondary language: Slovak

Title: Monitoring and Controlling Nanopore Sequencing Runs

Annotation: The goal of this thesis is to develop real-time analysis tools, that will allow to monitor nanopore sequencing runs and explore the possibility of controlling nanopore sequencing runs through ReadUntil framework. The tools will be developed in the context of pathogen sequencing (such as COVID-19).

Supervisor: doc. Mgr. Tomáš Vinař, PhD.
Department: FMFI.KAI - Department of Applied Informatics
Head of department: doc. RNDr. Tatiana Jajcayová, PhD.

Assigned: 13.10.2020

Approved: 28.04.2023 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

I would like to thank my supervisor doc. Mgr. Tomáš Vinař, PhD.
and doc. Mgr. Bronislava Brejová, PhD. for their guidance and support
while working on this thesis.

Abstrakt

Moderné nanopórové sekvenátory ponúkajú užívateľovi možnosť rozhodnúť sa, či sa reťazec DNA odmietne alebo bude sekvenovaný. Rozhodnutia sú založené na surovom nanopórovom signále v reálnom čase. Táto funkčnosť nanopórových sekvenátorov sa nazýva selektívne sekvenovanie. Sekvenovacie pokrytie možno navýšiť adaptívnym vzorkovaním požadovaných reťazcov DNA a odmietaním ostatných. V súčasnosti prebieha výskum viacerých metód na adaptívne vzorkovanie.

Našou prácou uľahčujeme vývoj a testovanie nástrojov na adaptívne vzorkovanie. Predstavujeme sekvenovací emulátor schopný emulovať selektívne sekvenovanie. Opisujeme jeho vývoj a demonštrujeme jeho využitie v kombinácii so známym nástrojom na adaptívne vzorkovanie. Taktiež skúmame možnosti využitia strojového učenia pre účely adaptívneho vzorkovania. Navrhujeme klasifikátor založený na konvolučnej neurónovej sieti, ktorý robí rýchle rozhodnutia o sekvenovanej DNA. Klasifikátor porovnávame s iným nástrojom na adaptívne vzorkovanie a prezentujeme naše zistenia.

Kľúčové slová: sekvenovanie, adaptívne, emulátor, vzorkovanie

Abstract

Modern nanopore sequencers provide users with the option to decide if a DNA sequence is rejected or sequenced. Decisions are made based on the raw nanopore signal in real time. This feature of nanopore sequencers is called selective sequencing. One can increase the sequencing coverage by adaptively sampling the desired DNA sequences and rejecting the undesired ones. The potential coverage gain achieved by adaptive sampling increases with the decision speed. Various methods to perform adaptive sampling are currently being researched.

We facilitate the development and testing of adaptive sampling tools by introducing a sequencing emulator capable of emulating the selective sequencing. We describe in detail the development of the emulator and demonstrate its use in combination with a well-known adaptive sampling tool. We also study the potential use of machine learning for adaptive sampling. We propose a convolutional neural network classifier that makes fast decisions about sequenced DNA. We compare the classifier with another adaptive sampling tool and present our findings.

Keywords: sequencing, adaptive, emulator, sampling

Contents

List of Figures

List of Tables

Introduction

Recent sequencing devices come with an exciting new feature. The user is allowed to intervene during the sequencing run based on the sequencing data distributed in real time. One can decide if the sequencing of a read continues or if the read is rejected and another one is sequenced instead. A sequencing strategy can be chosen based on the objectives of the sequencing run. The new capability of sequencing devices has potential applications in many areas utilizing sequencing technology. However, the research into techniques to realize the feature's full potential is still ongoing. One must be able to make rapid decisions about individual sequenced reads in order to gain an advantage during the sequencing run. Typically, the similarity between reads and desired reference genomes is evaluated to make a decision. The need for fast decisions about a read's biological origin based on its small portion constitutes a difficult problem in the field of bioinformatics. Current approaches often run into scalability issues. Decision-making algorithms are computationally intensive. The difficulty of the problem grows with the size of reference genome. For a long time, deciding about the nature of reads based on their similarity to human genome sized sequences seemed like an unattainable goal. The recent use of GPU hardware helped accelerate the decision-making algorithms. However, reported results keep falling short of the scientific community's original expectations.

In our work, we study the area of controlling sequencing runs. After reviewing the current research, we find that advancements in the area might be hampered by the required expertise in both the fields of biology and informatics. One must setup a sequencing run using a physical sequencing device in order to observe the impact of a new algorithm on a sequencing run. We address the issue by developing a realistic sequencing emulator. The emulator facilitates the development of decision-making algorithms and reduces calibration costs when deploying the algorithm in diverse conditions.

We demonstrate the use of the emulator throughout this thesis when it facilitates the development of our own decision-making algorithm. We study a potential of machine learning to develop an algorithm specialized for use with a single reference genome. In mission-critical epidemiological applications, which utilize sequencing technologies on a mass scale, the flexibility of the algorithm can be sacrificed for a performance gain. We propose an approach to accelerate the sequencing of *SARS-CoV-2* clinical samples.

In Chapter 1, we review the current research in the field and further elaborate on notable results. In Chapter 2, we describe the design and development of the sequencing emulator. We reserve a space to describe in detail the notable issues that we encountered during the development. Finally, we demonstrate the use of the emulator by fine-tuning the configuration of a third-party decision-making tool well known by the community using the emulated sequencing runs. In Chapter 3, we propose a machine learning model to make decisions about read's biological origin. We describe its development and test it in realistic conditions using the emulator. Finally, we discuss the results.

Chapter 1

Research Overview

The *DNA sequencing* is the process of determining the order of nucleotides in a DNA sequence. The device performing sequencing is called *sequencer*. In this thesis, we study the sequencers manufactured by *Oxford Nanopore Technologies*, which use *nanopore sequencing* technology. The nanopore sequencer contains a disposable component called *flowcell*, on which *nanopore channels* are located. A nanopore channel is a protein structure whose shape resembles a tunnel or channel. The number of channels depends on the sequencer variant; it ranges between 128 and thousands of channels. To every DNA sequence in a sequenced sample, the *adapter* is attached as a part of sequencing library preparation. An adapter enables a DNA sequence to attach to the nanopore channel and insert into it. The part of the adapter is a *motor protein*, which maintains a stable speed of DNA sequence when passing through the channel. A carrier electric current is passing through the nanopore channel. It controls the function of the motor protein. Individual nucleotides of the DNA sequence passing through the channel obstruct the stream of electrons, producing a modulated output *signal*. Each context of several nucleotides creates a unique signal signature. The output signal produced by a DNA sequence passing through the nanopore channel is called a *read* and is distributed to the sequencer control software in real time. The order of nucleotides in DNA sequences is determined by the analysis of reads yielded during the *sequencing run*.

Nanopore sequencing greatly facilitated the spread of a sequencing technology among its possible applications. Some of the nanopore sequencing benefits are compact form factor of nanopore sequencers and high potential length of produced reads. Experiments confirm that DNA sequences hundreds of kilobases (kb) long can be sequenced in single pieces[12]. That is a vast improvement over the previous generation of sequencing technology capable of producing reads of up to hundreds of bases long. Even though the sequencing accuracy of the previous generation sequencers is still superior to nanopore sequencing, the increased potential read length allows a better extraction

of information about the structure of a genome, e.g., the discovery of its new structural variants. Long repeating regions in a genome are notoriously difficult to assemble when only short reads unable to cover them are available. Nanopore sequencing technology helps to tackle this problem by being able to cover many of those regions in a single read, thus identifying the structure of the regions and the number of its repetitions. Other beneficial characteristics of nanopore sequencers are their low energy demand, ability to sequence RNA sequences, avoiding the need for their transcription to cDNA and output information being available in real time. Said benefits play a part in the spread of the sequencing technology into fields such as cancer research, food safety check or customized medicine.

The sequencing technology also plays a key role in modern epidemiology[15]. The analysis of viral and bacterial genomes provides means for scientists to determine the nature of their interaction with host organisms, the way they spread among hosts, or their resistance towards various external conditions. The proper medical treatment can be chosen for a patient based on the antibiotic resistance analysis driven by the sequencing of clinical samples contaning the bacteria. A continuous monitoring of the viral genome after an epidemic outbreak allows one to keep track of the genome's evolution based on time and specific climate conditions. Gathered information can be used to better understand the function of the virus and to implement restrictive measures designed to effectively limit the spread of the virus. The gathered information also plays a key role in a potential vaccine design. Viral genome sequencing has already proven itself to be an effective method for the analysis of *Ebola*, *Zika* and *SARS-CoV-2* viruses[15].

The field of epidemiology presents additional challenges for modern sequencing technology. Viral genome expression in a clinical sample is typically too low, which makes it difficult to gather sufficient read coverage of the genome for its consequent assembly by directly sequencing the sample. In theory, it is possible to sequence the sample long enough to ensure that proper viral genome coverage is reached. However, the process is time-consuming. Such long sequencing runs would easily wear out the disposable parts of the nanopore sequencer, making the whole process expensive on a mass scale. In some scenarios, the sequencing duration necessary to achieve sufficient read coverage would even surpass the operational lifetime of those sequencer components. In addition, an enormous number of reads that are not of interest would be produced during the run. These data need to be stored and analyzed in order to determine their significance in the context of the viral genome analysis. It has been documented that samples from patients with an acute *Ebola* infection contained a sufficient number of viral copies to obtain the desired genome coverage in a reasonable time by directly sequencing the clinical sample[15]. However, in general, a bioinformatic protocol may have to be designed for application to a clinical sample of the virus as a part of the

sequencing run preparation process. The protocol aims to artificially amplify the viral genome expression in a sample. It performs *targeted enrichment* of the viral genome. It increases the absolute number of copies of the viral genome in the sample and removes host background DNA, thus further increasing the relative representation of the viral genome in the sample. A bioinformatic protocol for targeted enrichment is commonly used for routine sequencing of *Zika* and *SARS-CoV-2* viruses[15].

1.1 Selective Sequencing

Oxford Nanopore Technologies is an established manufacturer of nanopore sequencers. In addition to the benefits described in the beginning of this chapter, Oxford Nanopore's sequencers introduce a feature that enables a user to actively alter the sequencing run at any moment based on a real-time distributed raw output signal from the sequencer. The feature is called the *selective sequencing*. A bi-directional communication can be established between a user software and the nanopore sequencer. The user software is provided with the raw sequencing signal gathered during the fixed time period for each nanopore channel. The sequencer receives commands specific to a single nanopore channel ordered by the user software based on the provided data. Using commands, the user software can interrupt the sequencing of a DNA sequence that is currently in a nanopore channel. The sequencer reverses the polarity of electric current passing through the channel, thus reversing the function of a motor protein that is otherwise inserting the DNA sequence deeper into the channel. In the reverse setting, the motor protein ejects the DNA sequence out of the channel. The process is called *unblocking* of the nanopore channel. After the process is finished, the nanopore channel is no longer *blocked* by a DNA sequence in it, and another DNA sequence can start sequencing. The user software performs *adaptive sampling* using commands to control the selective sequencing run and is often referred to as the *adaptive sampling tool*[17]. The feature is exposed to adaptive sampling tool through a programming interface called *Read Until API*[28]. The interface executes the communication with the sequencer. It exposes the current sequencing data and communicates commands to the sequencer.

Rejective commands ordered by the adaptive sampling tool are supposed to ensure that only DNA sequences relevant for future data analysis are sequenced. Several possible applications can be found for the feature. The biological background that is not subject to future analysis can be suppressed in sequencing data. Achieving background suppression can decrease the amount of data processed during the genome assembly, saving time and computational resources. When performing a sequencing of various clinical samples distinguished by unique markers called *barcodes*, i.e., performing the *multiplex sequencing*, selective sequencing based on barcodes can be used to maintain

a balanced coverage among all of the sequenced samples. Once a sufficient coverage of an individual clinical sample is reached, further reads belonging to that sample can be rejected in favor of less covered ones. Selective sequencing can be used to enrich the desired reads in sequencing data. The sequencing speed of the current *MinION* sequencer nanopore channel is 450 bases per second. It is therefore not uncommon that a single DNA sequence occupies a nanopore channel for more than 20 seconds. Early detection of undesired read and consequent unblocking of the nanopore channel can save a significant amount of time that could be spent sequencing a desired read instead. Using selective sequencing, it is possible to achieve a higher expression of desired reads in sequencing data, relative to undesired ones, than expected based on the expression of a desired genome in the sequenced sample. The increased expression is called *relative enrichment*. If the desired genome is enriched to such an extent that an absolute number of desired bases is higher than a number of desired bases potentially achievable during a non-selective sequencing run, the effect is called *absolute enrichment*.

To achieve relative enrichment of a desired genome, a performant and precise adaptive sampling tool is needed. High performance is necessary to keep up with the sequencing speed of nanopore channels while filtering out undesired reads. The precision ensures that desired reads are not unblocked incorrectly. Achieving absolute enrichment is even more challenging task. It requires the adaptive sampling tool to increase the throughput of nanopore channels for desired reads in comparison with a non-selective sequencing run. The task is made more challenging by the fact that selective sequencing, in general, decreases the throughput of nanopore channels. The reasons are multiple. Part of a DNA sequence needs to be sequenced for an informed selective decision to take place. It is not uncommon that more than 450 bases (1 second of sequencing time) are necessary for the decision to be made[23]. Also, the decision-making process consumes a non-trivial amount of time, while the DNA sequence is being sequenced and further inserted into a nanopore channel. After an unblocking decision is made, the DNA sequence needs to be ejected from a nanopore channel. Throughout the duration of the unblocking process, only a short read is produced, and the yield of a nanopore channel is decreased. Another reason for a lower throughput of nanopore channels during the selective sequencing run is their increased failure rate[23]. Frequent polarity changes of the electric current undergone by nanopore channels increase the chance of them losing their structural integrity. Such changes render nanopore channels unable to attach a DNA sequence or let a DNA sequence pass through. Therefore, after several hours of selective sequencing, a decreased throughput of nanopore channels is observed. The failure rate increases with the frequency of received unblocking decisions[23]. If the sequenced sample consists of short enough DNA sequences, almost no advantage can be gained using selective sequencing. By the time the read is ejected from the nanopore channel, it would have already been sequenced

in a non-selective sequencing run anyway. Therefore, the potential gain from selective sequencing increases with the length of DNA sequences in the sample.

The scientific community is currently inventing and developing methods to facilitate selective sequencing execution. While high relative enrichment is successfully being achieved since earlier published experiments[25], the levels of absolute enrichment remain rather modest, not surpassing 5-fold enrichment[23]. At such low levels, selective sequencing can at best supplement the commonly used bioinformatic protocols for targeted enrichment, which are used to amplify the number of copies of the desired genome in the sample much more significantly. Such assistance of selective sequencing accompanied by prior targeted enrichment can help shorten the time necessary to obtain the desired coverage of the target genome if absolute enrichment is consistently achieved during the sequencing run. The time necessary to accumulate a specified read coverage of a genome is called *time to answer*[23] and its decrease is considered a valuable improvement even in settings where absolute enrichment achieved by selective sequencing alone is not sufficient and targeted enrichment methods need to be deployed.

There are two major approaches to adaptive sampling execution. In the first approach, the unblocking decisions are made based on the raw sequencing signal, often referred to as a *squiggle*. Therefore, we say that the decisions are made in a *squiggle space*. In the second approach, the unblocking decisions are made based on a sequence of nucleotides represented in text form. These sequences are extracted from the raw signal by a transformative process called *base calling*. We say that the decisions are made in a *sequence space*. In the following sections, we describe both approaches and their notable results.

1.2 Squiggle Space Selection

To our knowledge, the first experiments with selective sequencing were published by Loose et al.[21]. Back then, the sequencing speed of nanopore sequencers used to be 70 bases per second. Overall requirements for adaptive sampling tools were therefore less demanding. A comparison to a *reference sequence* was used to determine if a read is desired or should be unblocked. A *reference squiggle* was syntehsized from the reference sequence using a *Hidden Markov Model*. The model is designed, trained and made publicly available by Oxford Nanopore Technologies. The adaptive sampling tool aligned chunks of the raw signal directly to the reference squiggle using the *Dynamic Time Warping* (DTW) algorithm[31]. The reached alignment score was used as a criterion based on which the unblocking decision was made. A server with 22 CPU cores was used for computations. The adaptive sampling tool was able to sample

a 5kb region of a reference genome and normalize a genome coverage when divided into 2kb regions. DTW is a dynamic programming-based algorithm. Its worst case time complexity is $O(nm)$ such that n is the length of a reference squiggle, and m is the length of an obtained and aligned squiggle. As demonstrated by Loose et al., the straight-forward use of the DTW algorithm in an adaptive sampling tool requires substantial computational power, even when aligning to a relatively short reference squiggle corresponding to a few kilobases. Scaling adaptive sampling up to human chromosome-like reference sequence lengths is highly unrealistic. Limited results of the work could not be replicated once the nanopore channel sequencing speed has been increased to the current 450 bases per second.

Masutani et al.[14] improved on the existing approach. They introduced a statistical model of selective sequencing. The model is based on prior knowledge of probability distributions D_0, D_1, such that D_0 models the alignment score distribution of undesired squiggles, and D_1 models the alignment score distribution of desired ones. Using the statistical model and distributions D_0, D_1, a score threshold Θ is determined. Threshold Θ is used as an unblocking criterion for a scores of the alignments of live squiggles to reference squiggle during the selective sequencing run. The authors showed that the dependency of distributions D_0, D_1 on the sequenced sample properties, such as GC content or a fraction of DNA sequences being desired, is negligible as long as the sample background consisting of undesired DNA sequences can be considered statistically random.

To speed up the DTW algorithm, three heuristical optimizations were introduced. Firstly, the live squiggle is compacted by a process that authors call *re-chunking*. In an average squiggle, more than 8 discrete signal values correspond to a single nucleotide. Squiggle re-chunking eliminates some of the redundant information contained in it. The squiggle is divided into non-overlapping regions. Every chunking region is sequentially processed. A current average signal value is computed and updated as algorithm iterates over the discrete signal values. If the distance between the current average value and a processed value is lower than a threshold τ, the processed value is considered redundant, it is removed and the average value is updated. The reference squiggle undergoes the equivalent transformation, resulting in shorter squiggles being aligned. Secondly, *seeding* is used in the alignment process. Only a sub-squiggle s-times shorter than the original one is aligned to the reference squiggle. Consequently, k candidates with the best alignment score are selected in $O(n)$ time complexity using *Floyd-Rivest* algorithm. Knowing the alignment starting positions of all k candidates, their alignments are extended to the full length, and the one with the best score is selected. The time complexity of the alignment using the seeding method is $O(nm/s)$, due to the fact that k is negligibly small compared to the length of the reference squiggle. Lastly, an alignment *prunning* is added, based on the monotonicity of scores in the DTW

	Total kb	Total reads	Target kb	Target reads
Repl.1	175 467	97 704	1698	161
Control	1 319 785	73 779	1660	120
Repl.2	129 430	65 949	1277	121
Control	1 283 144	68 119	1401	102

Table 1.1: Results of selective sequencing using S.cerevisiae and lambda phage[14]

dynamic programming table 1.1:

$$j \leq j\prime \implies min(D[i][j]) \leq min(D[i][j\prime]) \tag{1.1}$$

Once the alignment score surpasses the classifier threshold Θ, the alignment computation can be interrupted prematurely without a loss of information. Using the parameters $\tau = 0,36, s = 3, k = 14$, an accuracy of 80% was achieved in comparison with the original DTW algorithm. At the same time, approximately 6-fold alignment acceleration was reached. Consequent experiments were conducted on a MacBook Pro with an Intel®Core i5 2GHz CPU and an 8GB of RAM which is a significant improvement over a server appliance due to its portability, allowing the conduction of field experiments. During the experiment, even nanopore channels were performing the selective sequencing, while the odd ones served as a control. The amplified region of the reference sequence was 200kb long, representing 0.12% of the sequenced sample. For illustration, we include the results of the experiment in Table 1.1.

A significant decrease in throughput is observed for nanopore channels participating in selective sequencing, demonstrating the difficulties in increasing the throughput for desired reads in order to achieve absolute enrichment. This is mainly due to the high number of signals necessary for the alignment to make an unblocking decision about a live squiggle. While practically no absolute enrichment was reached, a 34-fold relative enrichment of desired reads was achieved. However, the experiments demonstrated that selective sequencing can be performed using the increased sequencing speed of 450 bases per second and a reference sequence hundreds of kilobases long. It still applies on the submission day of this thesis that adaptive sampling methods evaluating the raw sequencing signal in squiggle space do not scale well when increasing the reference sequence length to human chromosome-like sizes[17].

1.3 Sequence Space Selection

In recent years, the base calling process has been significantly sped up, mainly due to a new base caller designs based on a recurrent neural networks (RNN). Due to their

nature, RNN models can be efficiently simulated using GPU hardware, thus increasing the base caller throughput and creating new potential base caller applications, such as low latency live base calling. More recently, base callers have been further optimized and achieve sufficient performance even when simulated on CPUs[32]. Not only is the base calling step currently rapid enough for its application in adaptive sampling, the base called sequence of nucleotides proves to be a more compact representation of a DNA sequence. Well-established tools using advanced heuristics, such as *minimap2*, can be used to align a sequence to the reference sequence.

To our knowledge, the first attempt to perform adaptive sampling in the sequence space was a *Read Until with Basecall and Reference-Informed Criteria (RUBRIC)* published by Edwards et al.[25]. The adaptive sampling hardware infrastructure consisted of a notebook controlling the MinION sequencer and communicating the live sequencing data to a desktop computer in the form of queries over the network. The desktop PC was a Dell Optilex 9020 with an Intel®Core i7 3.6GHz CPU and 16GB of RAM. The unblocking decision is based on the alignment score of the live sequence to reference sequence alignment. Experiments did not show promising results. Base calling process turned out to be computationally intensive enough for the rather powerful desktop PC to only partially keep up with the base calling requests from the notebook. The absolute enrichment reached throughout the experiments was $< 2\%$. However, the undesired background of the sequenced sample was successfully suppressed when 330-fold relative enrichment of desired read coverage was achieved. The length of enriched region rose to 4.6 Mb. Edwards et al. demonstrated the unavoidable complexity of the adaptive sampling implementation. The nanopore channel live data were obtained from the Read Until API even in moments when no DNA sequences were actually sequenced. The raw signal produced by the nanopore channel while no DNA sequence is being sequenced or a DNA sequence is stuck and not moving in the nanopore channel is called the *stall signal*. Edwards et al. report that 89% of the raw sequencing signal obtained from the sequencer was the stall signal, which held no valuable information about a DNA sequence, yet kept loading the adaptive sampling pipeline with data. The authors had to design a statistical method for stall signal recognition and filtering. However, even with the filtering in place, authors report a decrease in the throughput of the adaptive sampling pipeline due to the excessive amount of stall signal being received.

Loose et al. improved on the existing approach by utilizing a GPU hardware to perform the base calling transformation in their adaptive sampling tool *Readfish*. For the first time, the authors scaled the adaptive sampling to human chromosome-like reference sequence length[23]. A ONT GridION MK-1 platform with integrated GPU is used for adaptive sampling execution. At the same time, the improved Read Until API that recognizes and filters out the stall signal is used, thus significantly decreasing

the load put on the pipeline. Loose et al. demostrated significantly lower unblocking decision latency due to the use of GPU for base calling. Authors achieved absolute enrichment of single human chromosome when sequencing long DNA sequences of a human sample. However, 1.2 seconds of sequencing corresponding to approximately 540 bases was necessary to successfully align read chunks to the reference sequence, slowing down execution of unblocking decisions. Another metagenomic sequencing experiment tested the ability to enrich the Saccharomyces cerevisiae representing 2% of the sequenced sample. Authors report 1.6-fold absolute enrichment, thus shortening the time necessary to achieve a desired genome coverage by 40%.

Recently, Ulrich et al.[17] pointed out the fact that *minimap2* is not well-suited for the alignment of short reads. The tool was primarily designed for efficient long-read alignment and reaches lower sensitivity and specificity when used for adaptive sampling. The authors developed adaptive sampling tool *ReadBouncer*[18]. They proposed the use of *interleaved bloom filter* data structure as a reference sequence index. Instead of actual alignment of the live sequence to the reference sequence, a similarity to the reference sequence is estimated using a k-mer hashing method. The authors report consistently higher classification accuracy compared to *minimap2* using 360 bases long sequences. Also, a lower average decision response time is achieved. These improvements should theoretically lead to higher achievable enrichments. However, the tools were only tested using a sequencing emulation that provides limited information about the effects of the improvements.

The results suggest that even though the invention of new adaptive sampling methods capable of making more rapid unblocking decisions would be beneficial, their impact on the selective sequencing may be limited due to the long chunks of sequenced raw data needed for classification. Currently, long sequencing of a DNA sequence is required before making the unblocking decision. This causes a significant decrease in the nanopore channel throughput, making it difficult to achieve substantial absolute enrichment. The longest phase of adaptive sampling decision-making being waiting for a sufficient amount of the raw data to be produced, reducing the amount of data necessary for the classification could have a large impact on potential adaptive sampling performance. Currently, there is no consensus in the scientific community about the preferred approach optimizing adaptive sampling performance. Both adaptive sampling in the squiggle space and the sequence space are being studied and considered viable options. Our work follows up on the current research and mentioned ideas in order to design an improved adaptive sampling tool.

Chapter 2

Virtual Sequencing

The development of a filtering pipeline performing real-time adaptive sampling often requires numerous experiments in a realistic setup. While some of the pipeline components can be developed and fine-tuned as standalone tools, it is their interconnection into a single filtering mechanism and actual deployment that often require additional calibration. One must take into account factors such as hardware limitations of their particular setup and properties of the sequenced sample. Both factors are highly variable and may require different pipeline configuration in order to make it work efficiently. Currently, it is rather difficult and costly to test and observe the impact that the specific pipeline configuration has on adaptive sampling performance during a real sequencing run. As we explain in detail in this chapter, actual sequencing of the sample using the physical device is often necessary in order to evaluate a filtering pipeline under realistic conditions and set it up properly. This increases the cost and difficulty of the adaptive sampling technology deployment by the community and also makes the research in the field less accessible as both knowledge in the fields of information technology and biology is necessary.

In this chapter, we the describe design and implementation of the tool for emulation of sequencing runs, which we call the *virtual sequencer*. It emulates the real MinION nanopore sequencer together with its selective sequencing capabilities using stored data from previous sequencing runs. It provides the means to test and configure a filtering pipeline based on its performance relative to the specific hardware available and DNA sample properties, if those are known beforehand. In addition, we demonstrate such pipeline configuration process. We connect the Readfish[23] tool for adaptive sampling to the virtual sequencer and attempt to enrich the *Saccharomyces cerevisiae* in the ZymoBIOMICS sample[8] using emulated sequencing data. We discuss some of the configuration details that proved to be crucial for adaptive sampling performance and present the results.

2.1 Available Sequencing Emulators

To our knowledge, currently the only available emulator of nanopore sequencing runs is the playback feature of Oxford Nanopore's MinKNOW software. Ulrich et al.[17] experimented with the feature while trying to compare Readfish to their own tool for adaptive sampling. The MinKNOW playback feature uses specialized bulk fast5 files to replay the stored sequencing run exactly as it happened in the past. Additional information stored in the bulk fast5 allows for emulation detail that would not have been possible to achieve using standard fast5 files containing the raw sequencing signals. However, its support for the emulation of selective sequencing capabilities of the physical device is very limited. Once the filtering pipeline, such as Readfish, decides to eject the DNA sequence out of the nanopore channel, MinKNOW emulation breaks the currently sequenced read into two reads, generates a new unique read identifier for the read and otherwise continues to emulate the sequencing of the original read without any change.

As demonstrated by Ulrich et al., resulting statistics obtained from such emulated sequencing runs are misleading and limited. First, the number of on-target and off-target bases obtained during the emulated run is fixed regardless of adaptive sampling use and its performance. This is because the ejection of a DNA sequence causes no change in the nature of the currently sequenced sequence and does not affect the sequencing run itself. Thus, the only property of the emulated runs that can be changed by adaptive sampling is the number of on-target and off-target reads. Typically, the increased number of off-target reads is expected. A well-performing adaptive sampling tool will reject many off-target reads quickly, which in the context of MinKNOW emulation means breaking off-target reads into multiple reads. Similarly, the number of on-target reads is expected to increase as little as possible, indicating high sensitivity of an adaptive sampling tool. Thus, the use of well-performing adaptive sampling tool leads to quite counterintuitive behavior of the emulator.

While the described metric can provide some insight into the comparison of adaptive sampling tools, it is not sufficient for fine-tuning the tool configuration for optimal performance. The available result data provide no information about relative enrichment nor absolute enrichment of the on-target bases. Therefore, it is not possible to observe the actual impact of the improved tool or pipeline component on the achievement of adaptive sampling's main objective without using the physical sequencing device.

MinKNOW playback feature also provides information about the length distribution of rejected reads. The information is crucial for verification that an adaptive sampling tool is able to keep up with the amount of sequencing data presented to it at every moment. It might be helphul while configuring the adaptive sampling tool to better utilize the GPU resources, if those are being used. However, the emulation of

the computational load itself that is put on a filtering pipeline is not necessarily realistic and can be exaggerated. Long off-target reads seem to be the biggest source of inaccuracies. In MinKNOW emulation, ejecting such a read is not accompanied by the actual read ejection and a wait for another DNA sequence to load into the nanopore channel. In practice, such a pause helps to relieve the pressure on the computing resources used by the filering pipeline. Instead, the sequencing of a new read starts instantly in the emulation, and the read is again an off-target read with certainty. This unchanging nature of the new read compared to the previously ejected read may be realistic enough, if an on-target fraction in a sequenced sample is very small. However, the bigger is the fraction of the on-target genome, the more the distribution of the sequenced reads skewed by the emulation, therefore putting additional unrealistic pressure on the filtering pipeline. That may cause the length distribution of rejected reads to be too pessimistic as the adaptive sampling was operating under an unrealistic load.

By creating the virtual sequencer, we try to address these inaccuracies and provide a more realistic way to emulate the sequencing runs with adaptive sampling. While the MinKNOW playback feature can be useful for troubleshooting and can provide some insight into the performance of an adaptive sampling tool, the virtual sequencer is intended as a complete replacement for a sequencing device, facilitating research and development of adaptive sampling tools without the need for any prior detailed knowledge of biology and sequencing technologies.

2.2 Virtual Sequencer Design

In order to discuss the design of our tool, we need to first describe the design and components participating in the actual selective sequencing process. The architecture is visualized in Figure 2.1. The sequencer device is controlled by MinKNOW control software. MinKNOW comes with a graphical interface that allows the user to control any of the operational aspects of the sequencing run. Settings and commands are communicated to the sequencer by the MinKNOW software. MinKNOW is also responsible for processing the sequencer output, a stream of discrete values produced by each nanopore channel in real time, called the *raw signal*. Apart from storing the raw signals along with their other attributes constituting *reads* in fast5 output files, MinKNOW is also responsible for presenting the currently sequenced data for selective sequencing purposes. For every nanopore channel, a signal sequenced in user-defined time period is made available through the gRPC framework[28]. The obtained portion of the raw signal is also called *data chunk*. The Read Until API communicates with the MinKNOW software using this framework. It obtains the data chunks from the past

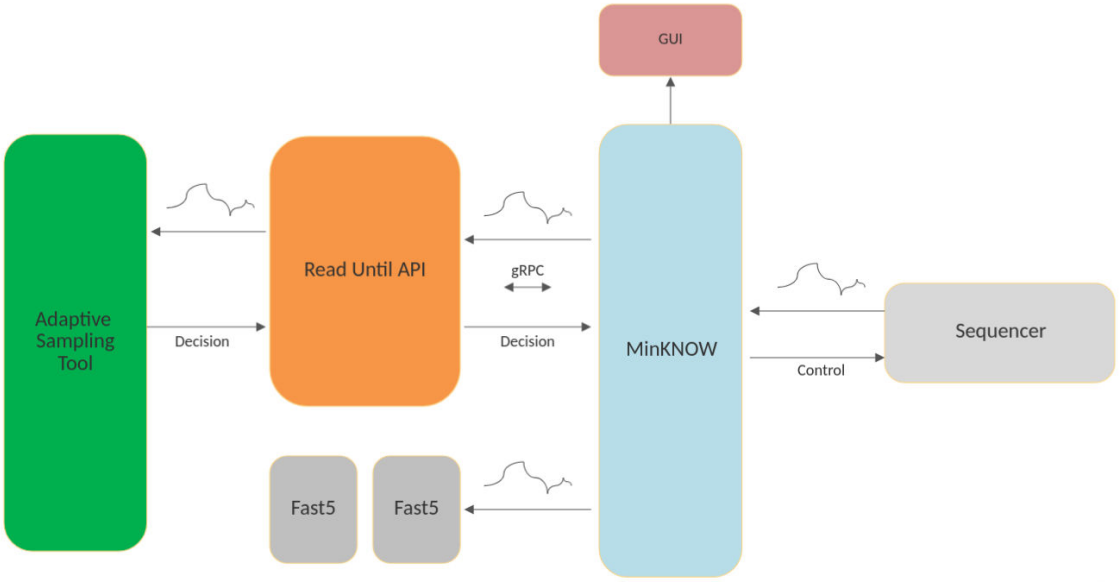

Figure 2.1: Components participating in selective sequencing

time period and sends the decisions of the filtering pipeline to MinKNOW. The read can either be rejected using the *unblock* decision, or it can continue to be sequenced using the *stop receiving* decision, in which case no more data chunks will be received for that particular read in future time periods. Decisions are further communicated to the sequencer by the MinKNOW software. Implementation of the Read Until API is provided by Oxford Nanopore Technologies. However, in principle, it is a task for the filtering pipeline developers to develop an API that suits their needs. The Read Until API can cache sequencing data from multiple time periods or process them in any way imaginable for adaptive sampling purposes. Multiple implementations exist today [28][20].

The virtual sequencer tool simplifies that design. Its architecture is visualized in Figure 2.2. The virtual sequencer core collapses some of the physical sequencer and MinKNOW features into a single virtual component. The virtual sequencer core uses the raw signal from a previous finished sequencing run to mimic MinKNOW's feature of presenting the sequencing signals for selective sequencing purposes. The virtual sequencer core also produces its own sequencing summary output which is discussed in detail in Section 2.3.2. The Read Until API keeps an instance of the virtual sequencer core and controls it directly. Similarly to the previous architecture, it obtains the sequencing data from past time period and directly controls the propagation of the filtering pipeline decisions. The virtual sequencer core stops presenting the raw data for a read, if the *stop receiving* decision is received. When the *unblock* decision is received, the virtual sequencer core emulates the DNA sequence ejection from the nanopore channel, followed by the emulation of loading a next sequenced DNA into a nanopore channel. The sequencing of the read that would otherwise have started on

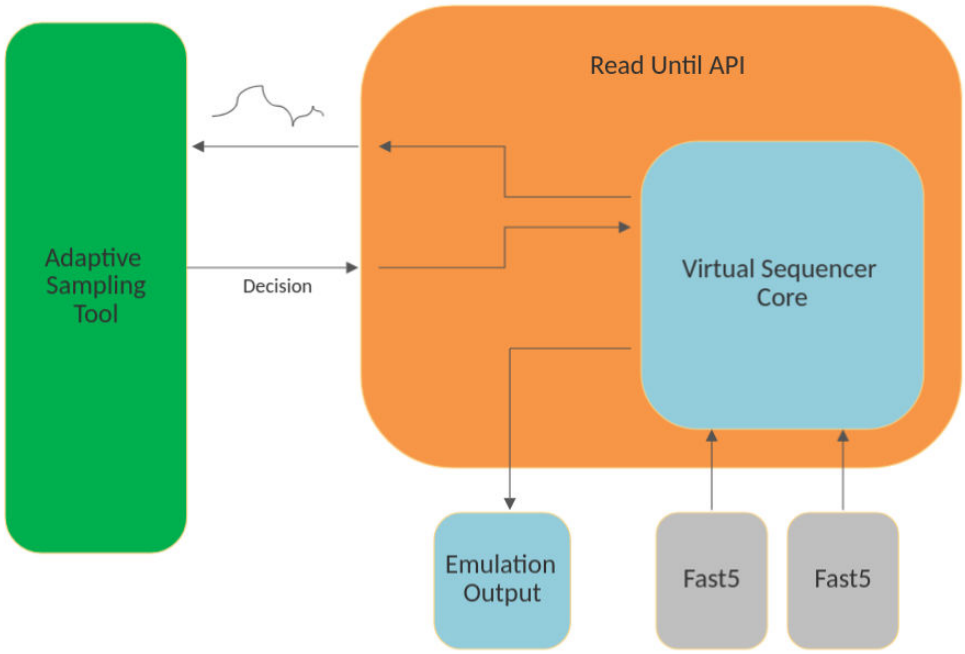

Figure 2.2: Architecture of the virtual sequencer

that nanopore channel after the sequencing of the recently ejected read has finished, is started earlier in the emulation. Filtering pipeline can therefore effectively modify the sequencing run that is being replayed. Reads from the future provide the source of the data in the emulation, thus preserving many properties of the real sequencing run. As an example, we can state the distribution of the DNA sequences loaded into a nanopore channel after the sequence ejection based on the filtering pipeline decision. Also, the time necessary to load another DNA sequence into the nanopore channel is preserved; it is drawn directly from the data being replayed. If hours-long emulations with intensive adaptive sampling are desirable, tens of hours-long sequencing runs are necessary for playback to provide enough future data. If no adaptive sampling-related decision is made by the filtering pipeline during the entire emulated run, the differences between the actual sequencing run and its emulation will be negligible. The filtering pipeline component remains unchanged in the design.

We consider the necessity to record a bulk fast5 file during the sequencing run in order to replay it an unnecessary limitation. Therefore, the virtual sequencer is designed to replay fast5 files, which are a standard data product of a sequencing run. This decision greatly increases its usability but also brings some inherent limitations to the design. We discuss those in the following section. Sequencers from Oxford Nanopore Technologies are also capable of sequencing RNA sequences. Since the output fast5 format of such sequencing runs remains unchanged, the virtual sequencer can also emulate those runs with simple parameter changes.

The virtual sequencer uses a precomputed index for an emulation of the sequencing

run. The purpose of the index is to hold the information about the order of individual reads for each nanopore channel, in which they are supposed to be emulated. Reads are sorted based on the starting time, of their sequencing during the original sequencing run. Index data for each nanopore channel is loaded in batches continuously during the entire emulation. Original fast5 files still need to be accessible for an emulation to take place. This is because the index does not keep any more information about the reads except their order and a pointer to fast5 files where further information can be found and loaded. The approach also makes the index format more flexible. If we decided to use additional read metadata from fast5 files in the future, the change would not require changes in the index format. This allows the user to avoid the potential recomputation of several indices each time we introduce a new feature.

Besides the objectives that we describe in detail in the following sections, our overall implementation objective is to make the virtual sequencer feel familiar to an end user - that is, a filtering pipeline developer. That endeavor starts with the design components resembling the actual adaptive sampling setup, as shown in Figure 2.2, and goes as deep as mimicking relevant interfaces wherever possible and providing as much of the real life functionality as possible. Thus a filtering pipeline tuned to work with the virtual sequencer requires only minimal changes for use in a real adaptive sampling setup. The modular architecture of the virtual sequencer allows us to potentially mimic multiple well-known Read Until APIs if necessary in the future.

2.3 Implementation Details

We chose Oxford Nanopore's Read Until API[28] as a feature reference and provided the functionality presented on its public interface in the Virtual Sequencer. We study adaptive sampling-related literature and Oxford Nanopore's materials in order to gain further insight into the inner workings of selective sequencing capabilities in a physical sequencer[23][28]. We implemented the tool using *Python* programming language. The choice simplifies the interface bindings to the Readfish tool and provides us with access to Oxford Nanopore's *ont_fast5_api* tool[27] for enumeration of fast5 files instead of implementing and maintaining our own. Our python interfaces can still be bound to adaptive sampling tools implemented in other programming languages, such as ReadBouncer[18] implemented in *C++*. In the following subsections, we describe various aspects of the virtual sequencer implementation.

2.3.1 Read Indexer

The read indexer computes an emulation index holding information about the sequencing order of reads during the emulation and a pointer to fast5 files, where more read

	Peak memory usage	computation time
Regular sort	1.83 GB	18m 42s
Online sort	0.30 GB	19m 37s

Table 2.1: Comparison of regular sort and online sort

metadata can be found for each read. The Read Indexer enumerates all of the reads in all of the fast5 files that form the real sequencing run output. The index stores as little information as possible in order to avoid unnecessary data redundancy. In particular, we store the *read identifier*, *channel identifier* and *starting time*. The fast5 file format allows random access to read metadata using *read identifier*. The index entry consists of *read identifier* and a *file pointer*, which is simply an order of the fast5 file in the sequencing run folder when sorted numerically according to unique part of the fast5 file name. Both the read indexer and virtual sequencer sort the fast5 file names the same way to obtain an identical mapping of file orders to file descriptors. Index entries are sorted based on the *starting time* and output to the read queue file identified by *channel identifier*. To reduce the index size, a binary format is used.

With index entries being compact, it is possible to simply extract index entries from the entire sequencing run and subsequently sort them for every queue corresponding to an individual nanopore channel. However, this approach requires storing all of the index entries at once, thus increasing peak memory usage to an extend that might not be acceptable for some applications. In addition, reads in fast5 files are not completely out of order either. Even though we do not understand the exact read ordering in fast5 files produced by MinKNOW, our experience suggests that the ordering of reads is close enough to our desired order that a naive online sorting algorithm is beneficial to employ instead of a regular sort. The online sorting algorithm allows the reads to be continuously stored in output index files throughout the indexing process. Thus reducing peak memory usage when indexing large sequencing data. The algorithm assumes that the reads are loaded in the proper order and tries to append the read at the end of the queue. If that is not possible, it moves one element towards the beginning of the queue and tries again. That repeats until the read is successfully inserted in the sorted queue. We call the number of steps taken towards the beginning of the queue during the sorting insert a *sorting distance*. We always keep a minimum number of sorted reads in the cache and check if the sorting distance did not exceed that number, which would mean that we might not have been able to insert the read in a proper position in the queue due to the lack of past queued data. We use a simple array as a sorted container because this check would not have been possible if we simply used a heap data structure.

We compare on real data using regular sorting and online sorting approaches. The sequencing data were 312 GB in size. We built an index of size 216 MB in less than 20 minutes. Table 2.1 shows that our online sorting algorithm reduces the peak memory usage 6-fold while paying a little time penalty. The overall average sorting distance is 2.41. The low sorting distance also makes the insert to the sorted array in a constant time superior to using the heap data structure or inserting with binary search in $O(log(n))$ time with the number of cached reads.

2.3.2 Virtual Sequencer Core

The core of the virtual sequencer is a multi-threaded application. Loosely coupled components of the core are executed asynchronously in order to minimize internal latencies. While Python's *Global Interpreter Lock*[11] prevents multiple threads of a single process from executing CPU tasks concurrently, this does not constitute a problem for the core, designed specifically to avoid any intensive computation. On the other hand, the core's asynchronous design benefits massively from a concurrent execution of CPU and I/O tasks that take place continuously during the entire emulation. The asynchronous design is depicted in Figure 2.3.

The *read loader* thread loads future reads and their metadata from the index and fast5 files into the cache. It maintains the minimal number of future reads in cache for each nanopore channel to prevent running out of future reads even under heavy load of unblock decisions. The *read scheduler* thread loads future reads from the cache and schedules them for sequencing. Any unblock decision is communicated to the read scheduler, which reschedules the future read for an earlier sequencing. Read scheduler keeps a *saved time* record for each nanopore channel. Every time a read is rejected by a filtering pipeline, the saved time record is updated for the relevant nanopore channel, and a future read is rescheduled accordingly. Once the time to sequence a particular read has come, the *live-read provider* is notified with all the necessary read metadata. The live-read provider is responsible for presenting the new sequenced data chunks for each user-defined time period. It keeps track of currently active nanopore channels, for which selective sequencing capabilities are emulated. When the *stop receiving* command is received or if read sequencing is finished, it updates the active channel list without the need for the read scheduler's intervention. Only newly scheduled reads are obtained from the read scheduler. Raw data chunks themselves are stored in a proper data structure and obtained by the Read Until API's processor thread. Filtering pipeline decisions are pre-processed in the *user thread* that executes the filtering pipeline algorithm. Multiple validation checks are made in order to prevent incorrect behavior and take unnecessary load off the simulation thread. Invalid decisions are the ones that came too late, near the end of the sequencing of the read or even after the

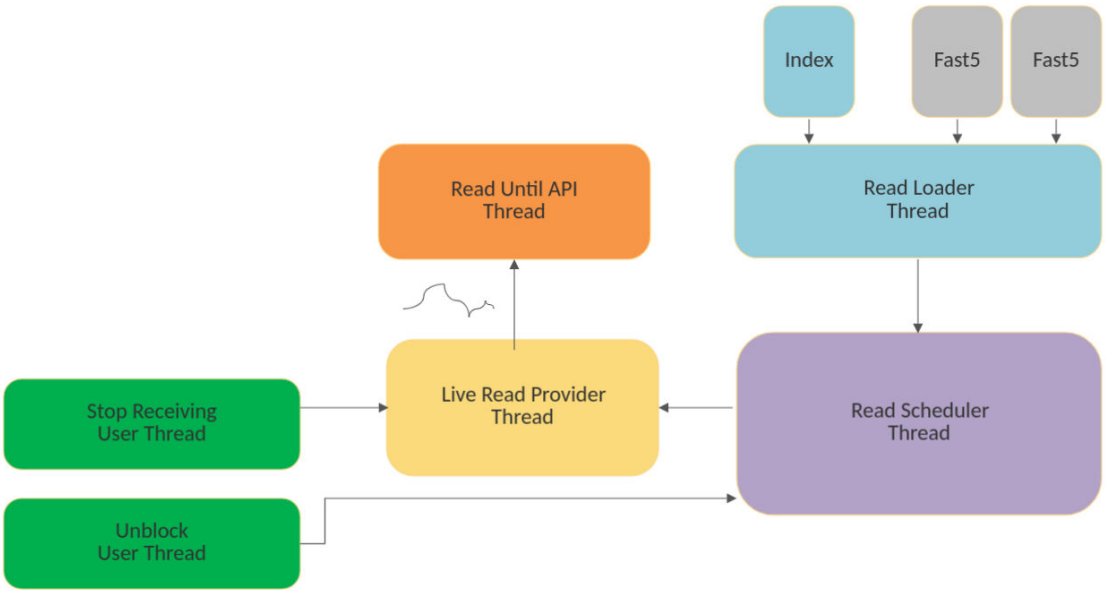

Figure 2.3: Asynchronous design of the virtual sequencer core

read sequencing has ended. Also the repeating repeating decisions; all tend to occur naturally.

The read scheduler maintains one future read per nanopore channel stored in the heap data structure. Once unblock decisions are introduced in the design, choosing the right data structure for read scheduling is important. Frequent rescheduling of the reads would make the maintenance of a sorted array inefficient. Insertion of the freshly rescheduled reads would take $O(n)$ time. Having a sorted array of reads is not even absolutely necessary. Only the information about the read with minimal starting time that needs to be scheduled is needed. Therefore, we use the heap data structure to store the reads. When a future read needs to be rescheduled due to an unblock decision, we insert a fresh version of the read in the heap in $O(log(n))$ time. We avoid the need for the removal of the expired read version by storing the fresh upcoming read for every nanopore channel in a separate data structure. That way, when the next read is pulled from the heap, it can be compared to a fresh read scheduled for the corresponding nanopore channel in order to recognize expired future reads abandoned in the heap data structure.

The virtual sequencer core produces its own sequencing output. It makes no sense for the core to output the authentic sequencing data as they are already present and act as the data source of the emulation. Therefore, only the record of the modifications to the sequencing run caused by unblock decisions is output to enable evaluation of the emulated run later. More specifically, an output entry consists of *read identifier*, a sequenced length and a single-digit *decision* value. We describe the evaluation of the emulated run in Section 2.4.

The virtual sequencer currently supports DNA sequencing at a speed of 450 bases per second. In the fast5 file, we find the sampling rate of 4000 signal samples per second used by the physical sequencer. Similarly, we support RNA sequencing at a speed of 70 bases per second. We found the sampling rate to be slightly lower 3012 signal samples per second. These are the attributes of the most commonly used nanopore channels for DNA and RNA sequencing. Other sequencing speeds and sampling rates are currently not supported.

We encountered timing issues during the development of the virtual sequencer core. Multiple reads scheduled for almost the same time combined with a large number of *unblock* decisions, typically delivered in batches, can put the read scheduler under load, causing non-negligible internal latencies. Unpredictable latency in unblocking and rescheduling of the reads leads to inaccurate reporting of the sequenced length of unblocked reads, because length reporting takes place on a user thread. Solving this issue by assigning the read scheduler with the reporting feature would further overload a critical section of the code. Besides those issues, the physical sequencer seems to be able to unblock reads almost instantly, allowing a user to specify latencies of 0.1s[30]. Therefore, we decided to prioritize low unblock latency over the timely start of new read sequencing. By reducing the time spent in synchronized critical sections of code and by ublocking reads in multiple sections of code, we managed to keep the unblock latency under 0.01s, which is equivalent to a 4.5 sequenced nucleotides, considering the current DNA sequencing speed. Delayed starts of read sequencing caused by ocasional bursts of *unblock* decisions are projected into *saved time* records to keep the emulation consistent. The typical inconsistent behavior can be described as follows. The scheduled read starts to be sequenced with a delay due to the internal latency. Its *starting time* is updated, so the live-read provider presents the correct data chunks to the Read Until API. The read may happen to be unblocked early by the filtering pipeline, saving a lot of sequencing time. However, the *saved time* is too optimistic because the delay is not included in it. Therefore, it is higher than realistically achievable. A *saved time* record is updated with a new saved time value. A series of such inaccuracies can lead to a situation when *saved time* record for a particular nanopore channel is so high that future reads are being scheduled in the past instead of the future. Normalizing such inconsistencies creates an error-prone emulation environment where many other logical errors can be hidden, harming the realism of the emulation. The approach of prioritizing the unblock latency and compensating for delays makes the emulation effectively slow down on less performant CPUs to meet their capabilities. However, the emulation remains consistent, reported lenghts of unblocked reads are accurate and comparisons of the experiments conducted on the same device are relevant.

2.3.3 Known Limitations

The virtual sequencer emulates sequencing runs based on fast5 files. That greatly increases its usability by avoiding the need for recording specialized bulk fast5 files to be able to replay them in the future. However, this decision has its drawbacks and introduces some known limitations that need to be understood. Fast5 files do not necessarily contain the continuous raw signal divided between the read records. DNA sequences may take a variable amount of time to properly attach to a nanopore channel and start to be sequenced. The time may be seconds long. During that time, mostly the carrier signal is recorded by the physical sequencer, also called *stall signal*. MinKNOW software tries to omit the stall signal in fast5 files. Using proprietary algorithms, it tries to detect the stall signal recorded between the sequencing of two DNA sequences. Sometimes, two reads follow each other so closely on a single nanopore channel that no signal is omitted and concatenation of the signals stored in fast5 file produces a continuous signal. This is not very common and often the signal obtained from fast5 files is discontinuous.

The Oxford Nanopore's Read Until API introduces the option to configure MinKNOW software to filter the presented raw signal by various classes[29]. Those are extraordinarily poorly documented. Unintuitive class names seem to have no further explanation available for the integrators of the Read Until API. Our conclusion is later confirmed by Payne et al.[12]. Payne et al. observe the raw signal in bulk fast5 files using the visualizer software that they developed and deduce the meaning of individual classes based on the annotations in the file. Although they admitted that not all classes seemed to have clear meaning, they successfully identified and described most of them.

The virtual sequencer is not able to support the filtering functionality because no signal annotations are present in the fast5 file format. Only a default setting can be used, which is defined by the logic of the stall signal-omitting algorithms of MinKNOW software. The Read Until API default filter setting allows MinKNOW to present the sequencing signal produced by DNA sequences in the nanopore channel and their adapters. Other signals recorded during various transitioning states of the nanopore channels are often considered a burden for a filtering pipeline with limited resources. Fortunately, we are able to support the default and most commonly used Read Until API setting of the filters. All signals corresponding to other classes mentioned in bulk fast5 files is effectively filtered out by the virtual sequencer as it is simply not present in fast5 files.

Nanopore channels commonly fail during sequencing run. The DNA sequence may get stuck in the nanopore channel, or the structural integrity of the protein forming the channel may be broken. Both cause the nanopore channel to stop working correctly. Payne et al.[23] observe an increased rate of lost nanopore channels on flowcells

performing the selective sequencing. The more intensive the channel unblocking, the greater the rate at which the channels were lost. We do not emulate this behavior in the virtual sequencer. We expect that may cause the emulation to allow for a little optimistic enrichment to be achieved. However, the effect is compensated to some extent. Nanopore channels are being lost even during the non-selective sequencing run. Reads drawn further and further from the future in order to emulate the selective sequencing capabilities also brings the point of nanopore channel failure closer from the future to the present, therefore also increasing the rate at which channels are being lost. It even applies to the emulation that the more intensive the channel unblocking, the greater the failure rate becomes, just like observed during real sequencing runs. However, we did not examine in detail the impact of this design simplification on the emulated run.

As of the submission day of this thesis, we have not managed to find out about the speed at which DNA sequences are ejected from the nanopore channel once the *unblock* decision is communicated to the physical sequencer. We also have no knowledge regarding the difference between the ejection speed of DNA and RNA sequence. Therefore, for the demonstration, we chose the ejection speed of 10-times the DNA sequencing speed, approximately 4500 bases per second. This is a guess on our side, and it might undermine the ability of the virtual sequencer to plausibly predict the results of adaptive sampling experiments using a real sequencing device. However, the sequence ejection speed is not the most impactful factor throughout an adaptive sampling experiment, and its impact decreases with the data chunk length necessary for the filtering pipeline to make an unblocking decision.

2.3.4 Integration

The filtering pipeline interacts with the virtual sequencer core through the Read Until API implementation. We implement a *Read Until Simulator*, which directly controls an instance of the virtual sequencer core. The read until simulator design is intended to resemble the architecture of Oxford Nanopore's Read Until API. Its public interface mimics the Read Until API wherever possible to ensure a minimum effort is needed the filtering pipeline to be used with the Read Until API. The initialization aspect of the interface has to be modified as read until simulator no longer connects to the MinKNOW software using the gRPC framework. We were not able to obtain a full design of the read chunk data structure obtained from the MinKNOW software and consequently presented on the Read Until API public interface. Therefore, we mimic at least those data members of the structure that are used in the Read Until API source code. Other aspects of the Read Until API interface remain unchanged. The read until simulator actually inherits most of its implementation from the Read Until API to minimize maintenance costs once Read Until API updates are released.

The Readfish adaptive sampling tool is known to use a slightly modified version of the Read Until API called *Read Until API v2*. The modifications adjust the API to use a *Python 3* programming language. Authors also add a new cache implementation that concatenates the read chunks obtained from the MinKNOW software that belong to a single read. Thus allowing Readfish to base its decisions on multiple data chunks if one chunk is not sufficient. The public interface of the Read Until API v2 undergoes only a few minor changes.

We connect the virtual sequencer to the Readfish tool. The Readfish has been shown on multiple occasions to work reliably in combination with the MinKNOW and a physical sequencer[23][17]. Our communication in the Nanopore Community also suggests that it is well adopted by the community. Therefore, the Readfish functioning without any issues when combined with the virtual sequencer is an important sign of the credibility of our emulations. In addition, several of Readfish's scripts designed for different types of adaptive sampling analyses, along with its extensive possible configuration using TOML files with comprehensive output prove to be a useful testing tool for the virtual sequencer. We modify the *unblock_all* and *ru_gen* scripts from the Readfish project and connect them to the virtual sequencer. The *unblock_all* script unblocks all obtained reads after a fixed specified period of time. The *ru_gen* script performs an analysis on the obtained data chunks. It uses Python bindings for Oxford Nanopore's Guppy base caller to base call the data chunks. Then the Python binding for minimap2 is used to align the base called sequences to the reference sequence. The unblocking decision is made based on the alignment and a TOML configuration file. A decision may be defined for various alignment results. The reference sequence may even be divided into multiple regions if amplifying or depleting specific regions of the reference sequence is desired. We also include Readfish's *validate* script without any changes to allow a convenient way to validate a TOML configuration file. The modifications necessary for the integration are subtle. Mainly, we change the initialization part of the Read Until API, which is replaced by the read until simulator, and add several new terminal parameters to facilitate the initialization.

The Readfish integration helped us identify some of the virtual sequencer's weaknesses. Emulated runs helped us spot the timing issues described in the previous section. We also experienced problems with using a Python binding for the Guppy base caller. It turns out that the Guppy' output produced for short data chunks only dozens of values long is problematic. Short signals are notoriously difficult to base call and the quality of the base called sequence is poor. Guppy can not base call such short data chunks properly. The data structure produced by base calling short data chunks is not just less populated with data. It is a whole different data structure where many fields are omitted. The program that tries to access them runs into a runtime error when executed. This unoptimal software design helped us find that such short data

chunks are not being obtained from MinKNOW software in reality. We find empirically that data chunks at least 100 samples long do not cause such problems. Aiming to keep the Readfish unchanged during the process of connecting to the virtual sequencer, we set a threshold for the minimum data chunk length presented by the virtual sequencer core.

2.4 Results

We perform a series of experiments using the Readfish connected to the virtual sequencer. We emulate the sequencing of the ZymoBIOMICS standard sample based on a fast5 data[8]. ZymoBIOMICS Microbial Community Standards[24] define the composition and other attributes of metagenomic DNA samples to facilitate the reproducibility of metagenomic sequencing experiments. The sequencing run that we emulate is approximately 48 hours long. Therefore, it has a substantial number of future data to draw from. Much like Payne et al.[23] did in their own experiment, we attempt to enrich the *Saccharomyces cerevisiae*, which is represented in the sample at approximately 2%. We demonstrate the use of the virtual sequencer for fine-tuning the Readfish tool configuration in order to maximize its adaptive sampling performance.

We use the virtual sequencer output to evaluate the results. The output entries consist of *read identifier*, a sequenced length of the read, and a *desicion* value. This information effectively documents the changes made to the sequencing run by the adaptive sampling tool. In order to evaluate the sequencing run, we base call the original fast5 files using the *Guppy* base caller. The virtual sequencer provides information about the the highest ranking fast5 file that was actually used during the emulation. This helps reduce the amount of data that needs to be base called. Then we align the base called sequences to a reference sequence using *minimap2*. The *minimap2* alignments serve as the ground truth for our evaluation. During the evaluation, we use *read identifier* to determine if the read is aligned to the reference sequence. This way, we calculate the number of on-target and off-target bases, the number of on-target and off-taget reads and the mean on-target and off-target read length.

We perform a series of 10 minute-long sequencing experiments to properly calibrate the adaptive sampling setup. We use a desktop PC with AMD®Ryzen 7 5700G 3.8GHz CPU and 32 GB of RAM accompanied by NVIDIA®GeForce RTX 3060 Ti GPU. We computed an emulation index and observe the virtual sequencer scaling the sequencing emulation to 312 GB of data. At first, we emulate the sequencing run using a primitive testing application for a filtering pipeline. The application waits for 10 minutes and then finishes the emulation. Without a single interaction, the virtual sequencer replays the sequencing run much like it originally happened, creating a baseline for future

comparisons.

Then we attempt to enrich the *Saccharomyces cerevisiae*. We downloaded the FASTA file with the consensus genome of *Saccharomyces cerevisiae*[6]. We precomputed a minimizer index using *minimap2* and configured it as a reference index for Readfish tool. We configured all 16 chromosomes in the FASTA file as target reference regions. Firstly, we chose Readfish reference configuration publicly available on Github[19]. We configure Readfish to send *stop_receiving* command if the data chunk aligns to the reference and *proceed* otherwise. We set the maximum number of data chunks received per read to 12. If the raw signal consisting of 12 concatenated data chunks can not be aligned to the reference sequence, the read will be unblocked by default. We set the data chunk length to 0.4 seconds of sequencing like Payne et al. This equals approximately 180 sequenced bases. On *Guppy server*, we choose *fast* model variant to minimize the decision latency. We emulate the sequencing for 10 minutes.

Table 2.2 shows that the average length of on-target reads slightly increased, indicating small number of incorrectly unblocked reads. On average, almost 2200 bases were sequenced before unblocking an off-target read, which limited the potential enrichment. We still observe a 1.5-fold absolute enrichment of the target genome coverage.

In order to lower the number of sequenced off-target bases, we decrease the maximum number of data chunks allowed to be aligned before the read is unblocked by default. We allow at most 3 data chunks. Table 2.2 shows an increase in number of sequenced on-target bases and a significant decrease in the average length of off-target reads. However, we also notice a decrease in the average length of on-target reads. This is because 3 data chunks were not always sufficient for read to properly align to the reference sequence.

During the emulation, we notice that the GPU utilization is extraordinarily low - almost never surpassing the 8% threshold. Therefore, we use the *HAC* (high accuracy) base calling model in the *Guppy server*. We expect that the higher base calling accuracy will allow shorter data chunks to align to the reference sequence. Table 2.2 shows that the average length of on-target reads increased, almost compensating for the decreased maximum data chunks setting. The average length of off-target reads is practically unchanged, indicating that the GPU's performance is not a bottleneck for adaptive sampling performance.

We attempt to further improve the absolute enrichment of *Saccharomyces cerevisiae*. The sequenced sample is a standardized ZymoBiomics metagenomic sample. Therefore, we have knowledge about all of the genomes contained in it. We decided to exploit this knowledge to configure the Readfish with a better-informed adaptive sampling strategy. We obtain the consensus genomes of *Pseudomonas aeruginosa*[5], *Escherichia coli*[3], *Salmonella enterica*[7], *Enterococcus faecalis*[2], *Staphylococcus aureus*[10], *Listeria monocytogenes*[4] and *Bacillus subtilis*[1], which are all present in the

	Original Run	Adaptive Max 12 Chunks	Adaptive Max 3 Chunks	Adaptive Max 3 Chunks HAC
On-target Read Count	547	810	2047	2036
Off-target Read Count	27063	38815	90577	90040
On-target Avg. Length	3236.38b	3281.97b	3172.02b	3242.54b
Off-target Avg. Length	3541.05b	2181.77b	677.18b	681.89b
On-target Bases	1.77M	2.66M	6.49M	6.60M
Off-target Bases	95.83M	84.69M	61.34M	61.40M
Absolute Enrichment	1.00x	1.50x	3.67x	3.72x

Table 2.2: Impact of adaptive sampling configuration on sequencing run (1)

sequenced sample. In previous experiments, reads had to be aligned to the reference sequence in order to be sequenced; otherwise, they were rejected once the maximum allowed number of data chunks has been received. In the following experiments, the logic is inverted. The depleted genomes are included in a reference index. A read needs to be aligned to any of the reference sequence target regions in order to be rejected, rather than being rejected by default. The *Saccharomyces cerevisiae* is still present in the reference index. For a read to be sequenced, it has to be aligned to the *Saccharomyces cerevisiae* regions in the reference index. However, the maximum number of allowed data chunks can be increased because the rejection of off-target reads is no longer bound to it. While many off-target reads will be rejected as soon as they are aligned to some of the target regions, the Readfish can obtain a sufficient amount of data for the rest of the reads to make a well-informed decision. The read is still rejected by default once the maximum allowed number of data chunks has been received. We run a series of 10 minute-long emulations and experiment with the values of maximum allowed data chunks.

Tables 2.3 and 2.4 show an increase in the average length of on-target reads as it approaches the average on-target read length of the entire replayed sequencing run. Figure 2.4 shows that some reads can not be aligned to the reference sequence even when 12 data chunks are used. The balance between obtaining enough data chunks to make an informed unblocking decision and wasting sequencing resources on reads that

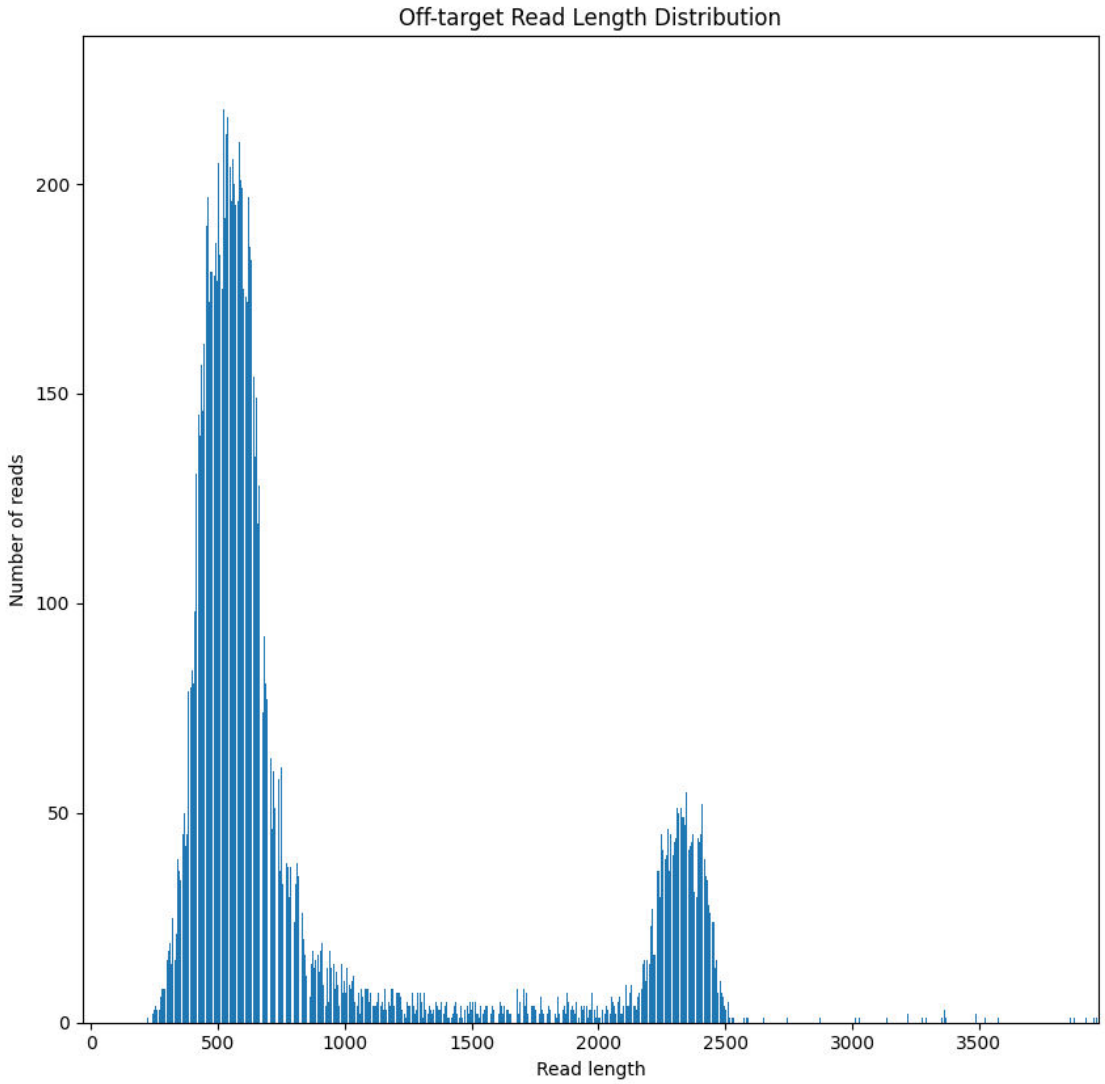

Figure 2.4: Off-target read length distribution when processing at most 12 data chunks

will not align to the reference sequence needs to be found. Figure 2.5 shows the peak in the histogram created by reads that could not be aligned to the reference sequence, shifting to the side as we lower the maximum number of data chunks to 5. Table 2.4 shows an increasing absolute enrichment of the target genome and a decreasing number of sequenced off-target bases as we lower the maximum number data chunks. The loss of *Saccharomyces cerevisiae* reads produced by incorrectly unblocking short data chunks that could not align to the reference sequence is compensated by unblocking off-target reads as soon as they are recognized. The trend reaches its peak when at most 5 data chunks are allowed.

While 10 minute-long experiments are useful for calibration, they may produce too optimistic results. We notice that most nanopore channels are operational during the initial 10 minutes of the sequencing run. Due to the destructive effect that sequencing

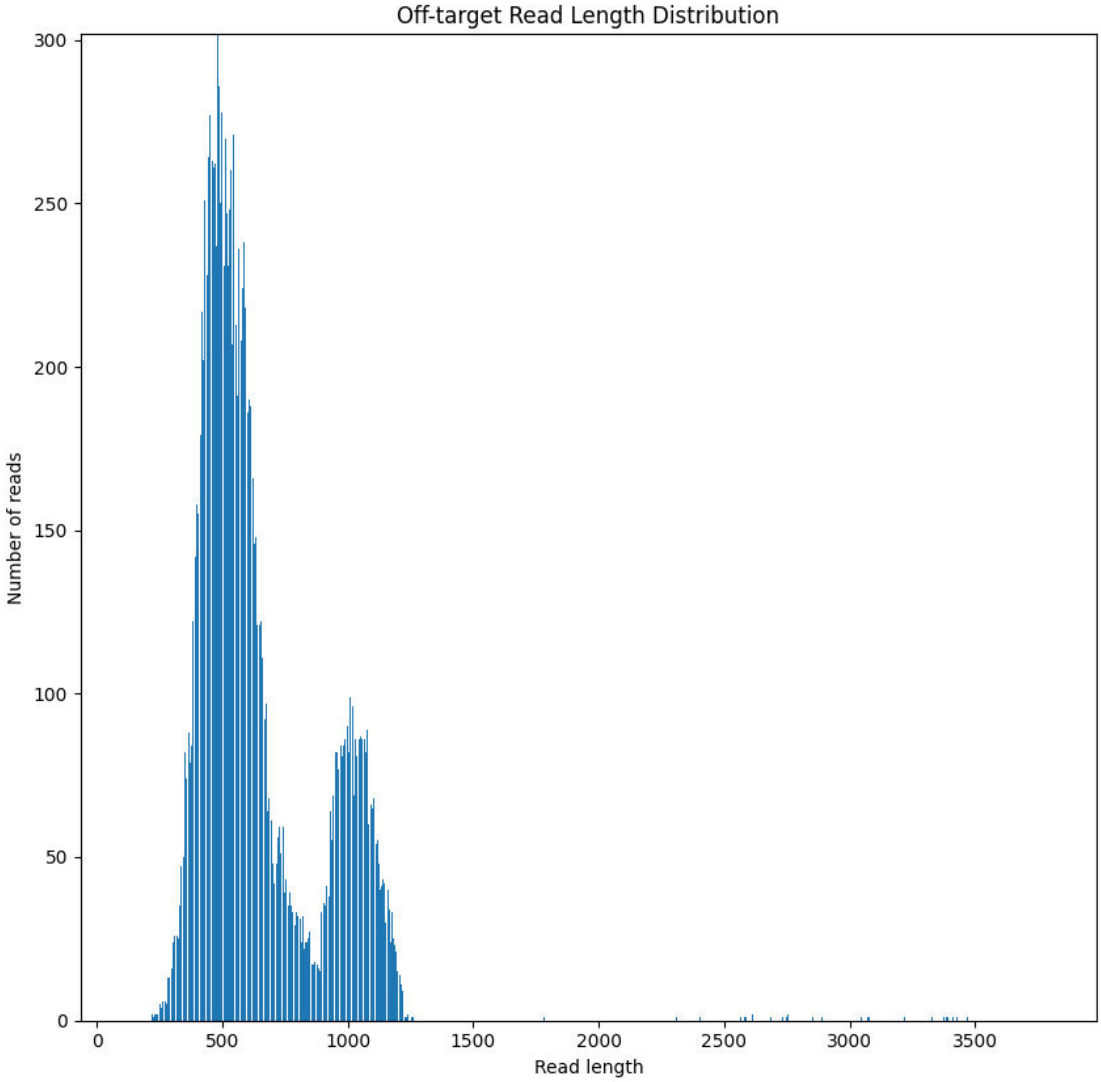

Figure 2.5: Off-target read length distribution when processing at most 5 data chunks

has on some fraction of the channels, the absolute enrichment can become more modest once the sample is sequenced for a long time. We use the inverted adaptive sampling logic from the previous experiment and set the maximum number of data chunks to 5. We emulate the sequencing run for 3 hours; however, we run out of future sequencing data to draw from in slightly less than two hours. We try again, but this time we emulate the sequencing run for 1 hour. Figures 2.6 and 2.7 compare the off-target read length distribution of the original sequencing run and the selective sequencing run.

We compare our experiment to the similar experiment conducted by Payne et al.[23]. We expect absolute enrichments achieved in experiments to differ. The authors used different hardware for their computations. They also conducted their experiment with a slightly different setup. While we attempt to enrich the *Saccharomyces cerevisiae* right from the start of the experiment, Payne et al. only start rejecting reads aligned to a

	Inverted Max 12 Chunks	Inverted Max 10 Chunks	Inverted Max 8 Chunks
On-target Read Count	1736	1827	1922
Off-target Read Count	77274	80521	84540
On-target Avg. Length	3343.34b	3333.71b	3339.79b
Off-target Avg. Length	870.16b	813.47b	753.20b
On-target Bases	5.80M	6.09M	6.42M
Off-target Bases	67.24M	65.5M	63.68M
Absolute Enrichment	3.28x	3.44x	3.62x

Table 2.3: Impact of adaptive sampling configuration on sequencing run (2)

	Inverted Max 6 Chunks	Inverted Max 5 Chunks	Inverted Max 4 Chunks
On-target Read Count	2005	2113	2196
Off-target Read Count	88336	92848	96673
On-target Avg. Length	3322.86b	3322.57b	3302.11b
Off-target Avg. Length	701.74b	647.97b	603.90b
On-target Bases	6.66M	7.02M	7.25M
Off-target Bases	61.99M	60.16M	58.38M
Absolute Enrichment	3.76x	3.97x	4.09x

Table 2.4: Impact of adaptive sampling configuration on sequencing run (3)

	Original Run	Adaptive Sampling Run
On-target Read Count	4310	14358
Off-target Read Count	189113	635277
On-target Avg. Length	3562.47b	3459.40b
Off-target Avg. Length	3725.40b	663.10b
On-target Bases	15.35M	49.67M
Off-target Bases	704.52M	421.26M
Absolute Enrichment	1.00x	3.24x

Table 2.5: Emulated adaptive sampling performance

particular reference genome once the desired read coverage of the genome is reached. This delays the sequencing phase, in which *Saccharomyces cerevisiae* is being enriched alone for approximately 2 hours, thus decreasing the average enrichment. The Readfish configuration used by the authors in the experiment has not been published. Therefore,

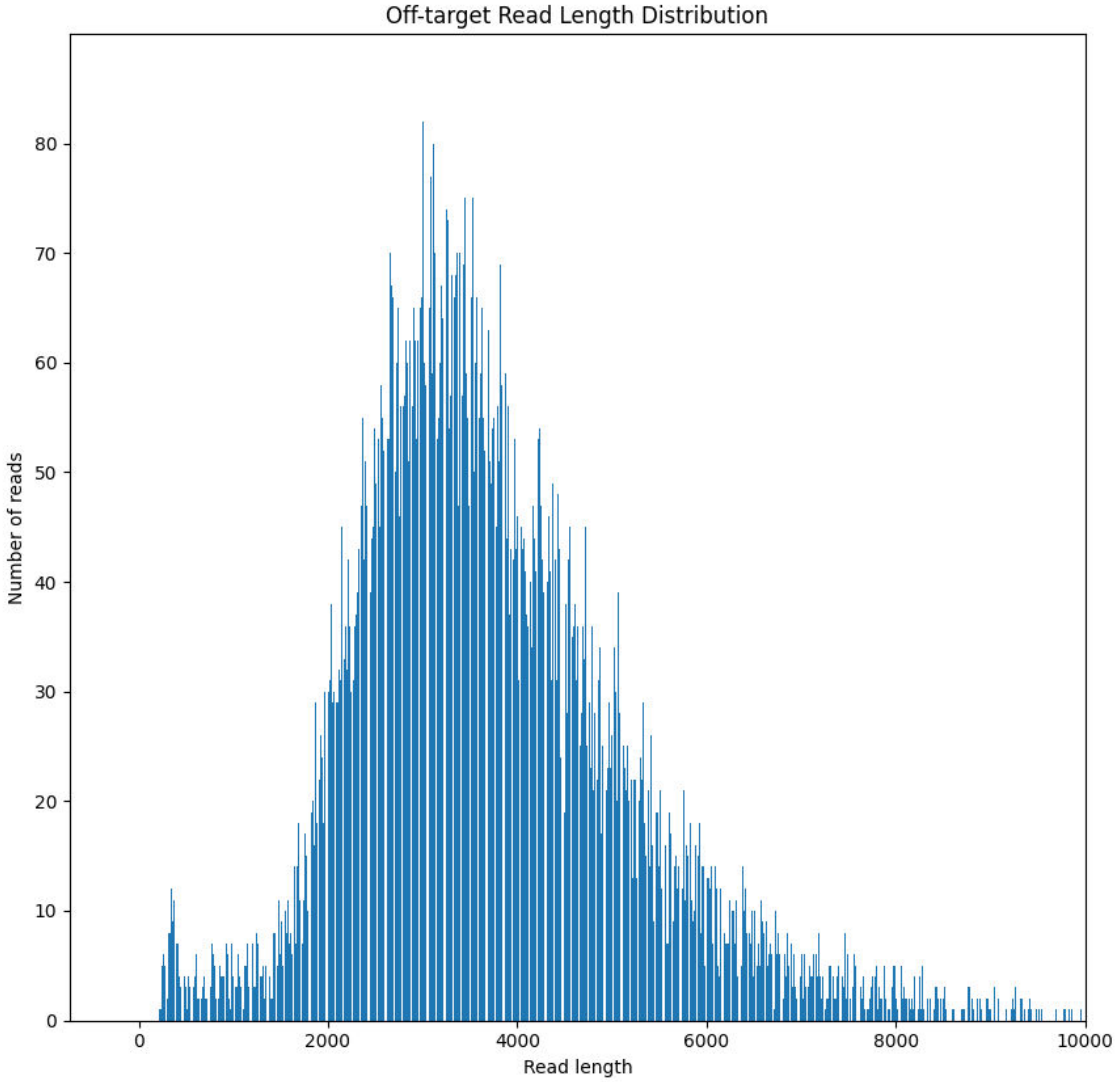

Figure 2.6: Off-target read length distribution in the original sequencing run

it is unknown. Payne et al. sequenced the sample for 16 hours and reported 1.6-fold absolute enrichment of *Saccharomyces cerevisiae*. The enrichment is estimated based on the known sample composition and the amount of sequencing data yielded in non-selective sequencing experiments conducted using similar sequencing samples.

Table 2.5 shows a 3.2-fold enrichment in the experiment. We attribute a partial responsibility for the different results to the likely differing Readfish configurations in the experiments and different setups of the experiments. However, considering these factors, the comparison suggests that the results yielded by the emulation are not fully realistic. This is likely caused by the known limitations described in Section 2.3.3. If more realistic prediction of results is desired, the nanopore channel ejection speed needs to be determined and a model of an increased nanopore channel failure rate needs to be added to the emulation. However, the experiments confirm that the emulations

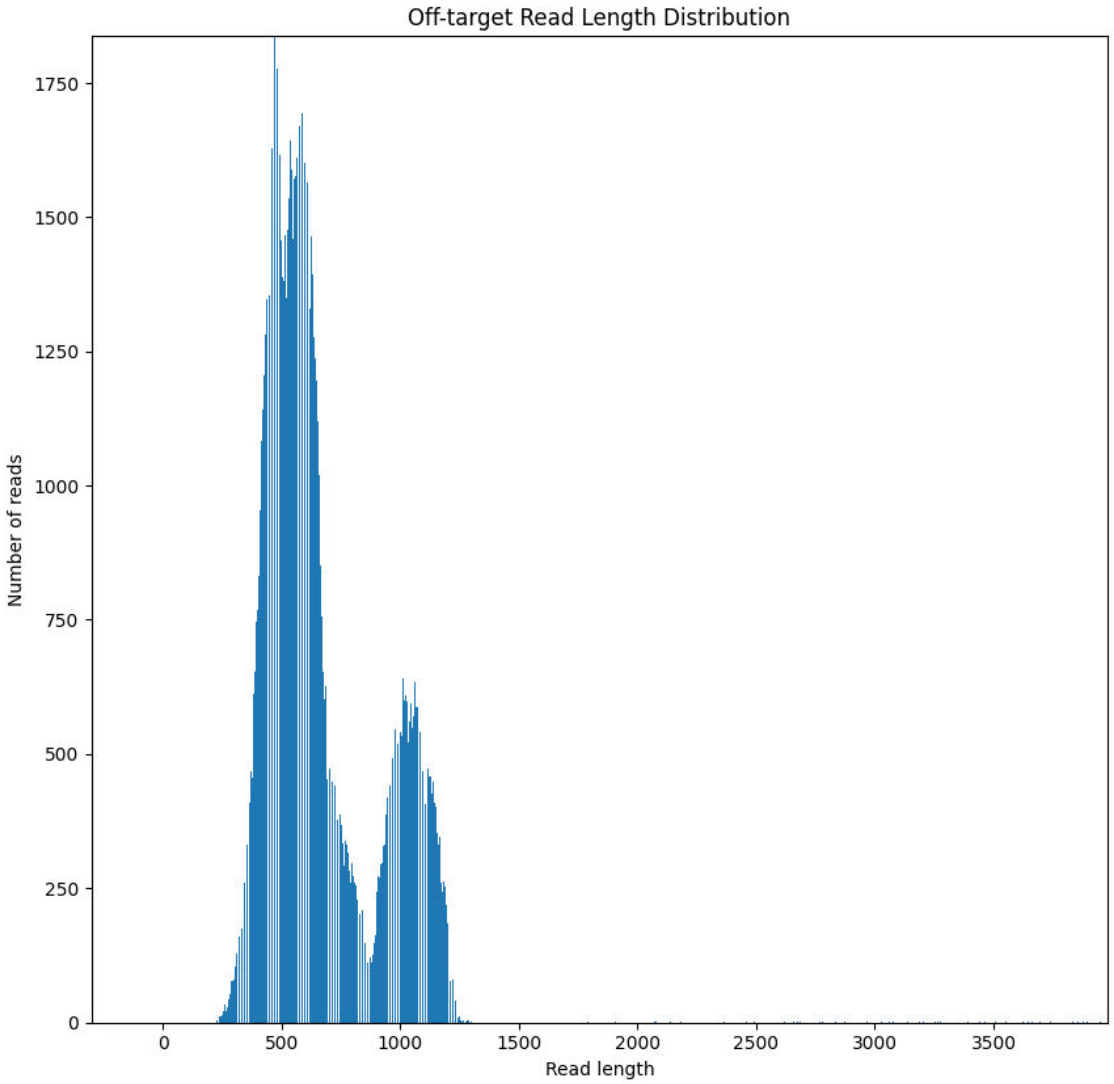

Figure 2.7: Off-target read length distribution in the selective sequencing run

conducted using the virtual sequencer are consistent and allow the different adaptive sampling tool configurations to be evaluated.

Chapter 3

Adaptive Sampling

Considerable effort is currently being invested in the research and development of tools for adaptive sampling execution that provide general sampling capabilities. An arbitrary reference genome can be configured to be enriched or depleted during the selective sequencing run. Methods that determine the similarity of data chunks to the arbitrarily chosen reference genome with high sensitivity and specificity are being researched. They operate with the sequences of nucleotides[23][17] or with the raw signal[26]. Such general sampling capabilities provide clear advantages of simple configuration and deployment in diverse field experiments. However, generality has its cost in these methods as not all of the sequencing data features can be used when determining the similarity to the reference genome. In some applications, the need for single genome sequencing is strong enough that a genome-specific adaptive sampling method may be beneficial. One of such applications is the recent effort to study the *SARS-CoV-2* virus. The number of *SARS-CoV-2* genome copies in the sequencing sample is often amplified using biological methods, such as polymerase chain reaction[15], to achieve satisfactory sequencing coverage. An adaptive sampling method specialized for enriching the *SARS-CoV-2* and exploiting all of the features in the sequencing data could represent the way to achieve a better adaptive sampling performance in a narrowly scoped, mission-critical operations of epidemiology.

We wonder if a higher adaptive sampling performance can be achieved at the cost of the sampling method's generality. We describe our approach to the fast determination of the similarity between chunks of raw sequencing data and the *SARS-CoV-2* genome. We implement the adaptive sampling tool called *selectify*, which integrates the suggested decision-making algorithm. Then we attempt to enrich the *SARS-CoV-2* sequencing data while emulating the sequencing run using the virtual sequencer.

3.1 General Approach

To increase the efficiency of an adaptive sampling method, the decision-making response time needs to be decreased while high decision sensitivity and specificity are preserved. The time necessary for a read rejection to be executed consists of three elements. The dominant element is typically the time needed for sequencing the data chunk, based on which the unblocking decision is made. The second element is the time necessary for an unblocking decision to be made. The final element is the time necessary for ejecting the unblocked read from the nanopore channel. As we observed in the experiments in the previous chapter, fast decision-making process may have limited impact if long data chunks are necessary for the algorithm to decide about their nature. Current state-of-the-art adaptive sampling methods often use base calling in order to transform the raw signal into a sequence of nucleotides. Even though modern base callers based on a reccurent neural networks are considerably faster compared to the previous generation, especially when run on specialized hardware such as a GPU, their use is an added step in a decision-making process. While base calling time may not be an issue most of the time, the emerging issue is a pessimistic prospect of decreasing the data chunk length necessary for a proper base calling. Base calling of short data chunks leads to poor base calling quality because the signal normalization step fails during the base calling process. As a result, more chunks of data need to be obtained from MinKNOW software to retry the base calling. This makes it difficult to reduce the time necessary to obtain the satisfactory data chunk length for further processing, which is a dominant fraction of the time that a decision-making process takes. In the current literature[17], the adaptive sampling tool's sensitivity and specificity are evaluated using a 360 base-long testing data samples, which take at least 0.8 seconds to obtain.

Our goal is to significantly reduce the decision-making response time by decreasing the necessary data chunk length. Because of the limitations mentioned, this is not realistically achievable when deciding based on a base called sequencing data. Therefore, we skip the base calling step and make unblocking decisions based on the raw signal. Current methods for providing general adaptive sampling capabilities without the use of base calling are computationaly intensive and do not scale well when a large reference genome is desired[17]. This is caused by the variable nature of the raw nanopore signal, with multiple discrete variable values representing a single nucleotide. However, in our setting, the general sampling capabilities are not necessary. Only a similarity to a fixed reference signal has to be determined. We use a convolutional neural network model (CNN) to learn to distinguish a *SARS-CoV-2*-related raw signal. We design and train the classifier aiming at its compact design, allowing for short response times while emphasizing on the minimum data chunk length needed to make an unblocking

decision. The trained classifier is an integral part of the selectify adaptive sampling tool.

3.2 Polymerase Chain Reaction

Viral genome expression in a *SARS-CoV-2* clinical sample is low, on average only 10-48 genome copies per microlitre[15]. This constitutes an issue for sequencing methods that often require a bioinformatic protocol for targeted enrichment to amplify the number of viral genome copies in the sample. However, the low viral genome coverage yielded during the sequencing of the original clinical sample also hampers any potential training of a classification model due to the lack of training data. Therefore, we resort to extracting the training data from clinical samples where the viral genome was amplified using a targeted enrichment protocol. Polymerase chain reaction (PCR)-based bioinformatic protocols are commonly used. Such modified clinical samples have specific properties that need to be understood.

PCR is a laboratory technique. It relies on using short synthetic DNA sequences called *primers* and a *DNA polymerase* enzyme. The clinical sample temperature is first increased in order to divide double-stranded DNA sequences in the sample into single strands. Every primer sequence is a complementary sequence to some short region of a divided single strand. After the decrease of clinical sample's temperature, primers ligate on the single strands, marking the beginnings of amplified regions. Primers ligation to a single strand is much more likely compared to a ligation of two strands because compact primer sequences can move in the sample with significantly less resistance. Primers ligated on DNA strands create a *two-stranded initiation* for the polymerase enzyme. In the following chain reaction, the polymerase enzyme elongates the primer sequences, thus efectively synthesizing a complementary strand copy. Consequently, the temperature is risen again. Elongated strands called *amplicons* are divided from their complementary strands. The described process repeats itself in multiple iterations. After the sample temperature decreases, primers ligate on single strands of DNA again. This time, newly synthesized amplicons participate in the chemical reaction. Therefore, the number of synthesized amplicons grows exponentially with the number of iterations.

To efficiently utilize the sequencing resources, multiple clinical samples are sequenced in a single sequencing run. Individual clinical samples, while still separate, are marked with a unique marker called *barcode*. As a step in sequencing sample preparation, a barcode sequence is added at both ends of every DNA sequence in the sample. Uniquely barcoded clinical samples are then merged and sequenced as a single clinical sample. Such a process is called *multiplex sequencing* and uses *barcodes* to assign individual reads to a specific patient.

The design of the targeted enrichment protocol has various consequences for our use of the sequencing data as classifier training data. Firstly, the read beginnings in the stored sequencing data are not randomly distributed around the entire viral genome. Almost all of the reads start at specified primer binding positions. Therefore, extracting a fixed portion of reads for training data set will not result in training data randomly covering the whole genome. Secondly, barcode sequences may pose an issue for a practical use of a trained classifier as multiple barcoding kits are available on the market. In addition, barcodes may not be used at all when performing the sequencing of a single clinical sample.

3.3 Read Classification

3.3.1 Training Datasets

Ongoing efforts focused on monitoring *SARS-CoV-2* spread and evolution provide us with a vital source of training data. We used sequencing data produced in Institute of Virology, Slovak Academy of Sciences. Sequenced sample was prepared using the protocol published by Brejová et al.[13]. Analysis performed after the sequencing run yields statistics of barcode occurrences and viral genome coverage achieved for each barcode. It also yields the list of read identifiers that were successfully aligned to *SARS-CoV-2* genome with respect to each barcode. We use the alignmet scores of full reads as the ground truth for the labeling of the training data. We call reads that could be aligned to *SARS-CoV-2* genome *positive training examples*. On the contrary, reads that could not be aligned to *SARS-CoV-2* genome are called *negative training examples*. Using the statistics, we extract the training dataset from the sequencing data. Firstly, we parse the statistics files and load all read identifiers corresponding to reads that aligned well to *SARS-CoV-2*. Next, we enumerate all of the reads stored in all of the fast5 files produced during the sequencing run. We use loaded read identifiers to determine the positive or negative nature of the reads. Knowing the target number of examples in the training dataset, we keep the number of positive and negative training examples approximately balanced. For each training example, the first 5 000 raw signal values (equivalent to 562 bases) are extracted from a fast5 file. We do not trim the beginnings of the raw signal produced by adapter sequences and barcode sequences passing through the nanopore channel. This information could be lacking while classifying the live raw signals. We normalize the raw signal using a *modified z-score* 3.1 and add binary label at the end.

$$\bar{X} = median(X)$$
$$MAD = median(||X_i - \bar{X}||)$$

$$modified\_z\_score = \frac{0.6745 \times (X_i - \bar{X})}{MAD} \tag{3.1}$$

Reads that are shorter than 675 bases are not included in the dataset. Consequently, a random permutation of the training data is generated, and the data is stored in a binary file.

At first, we extract only one training example per read, we choose the first 5 000 raw signal values. Even though the training dataset does not cover the whole *SARS-CoV-2* genome, its purpose is to enable us to observe the properties of various model designs trained on the dataset. We extract approximately 160 000 positive training examples and a similar number of negative training examples. We designate a tenth of the data to be the testing data; the rest is used for the training of the CNN model.

3.3.2 Classifier Architecture

We use a CNN for the classification of the raw sequencing signal. Reassured by the work of Mostavi et al.[22], we believe that, similarly to a picture, some patterns of a local character can be observed in the sequenced signals. Exploiting this locality using a convolutional neural network helps us reduce the number of parameters in our model. Mostavi et al. already demonstrated that considering larger patterns or patterns spanning multiple discontinuous parts of a genome does not help the model classify a read more accurately. The authors tried to build a two-dimensional picture out of a base called DNA sequence stored in a one-dimensional line. This way, filters of convolutional layer could consider multiple regions of a signal in dot product during the forward pass. No significant statistical correlation was found and the approach did not overcome the one where a one-dimensional kernel is used for one-dimensional data. We follow the approach of using one-dimensional kernels for convolutional layers. However, instead of base called sequences of nucleotides, we are working with the raw sequencing signal. Unlike Mostavi et al., we find that it is beneficial to move the kernel by a smaller stride during the forward pass. The authors were not confident that there were any significant correlations in neighboring gene expressions, since they achieved the best results using the stride of the size of the kernel. In the raw signal, multiple variable values represent a single nucleotide, and even the number of values corresponding to a single nucleotide is variable. Therefore, neighboring regions of the raw signal might be correlated as they might represent various changes in nanopore channel state. As mentioned by the authors, increasing the depth of the CNN does not help to yield better accuracy, which we experimentally confirmed. Adding multiple layers to the model led to increasingly bad results in both time complexity and classification accuracy.

In the following text, we demonstrate the iterative process of designing a CNN model for the classification task. Using multiple experiments, we try to understand the properties of the raw sequencing signal. We use *Keras* framework to build and run CNN models. Model architectures are illustrated using the Keras building blocks in the Python programming language. For each design iteration, we report accuracy, specificity, sensitivity, precision and F1-score. Because of the intended model application, we also report the classification time. We classify testing examples one-by-one to avoid batching optimizations of the Keras framework that affect the performance results.

The initial CNN classifier design M_1 is shown in Figure 3.1. We use a sequence of convolutional layers to explore the local features of an input signal with each layer perceiving the input from a more global perspective than the previous one. The used kernel size and stride are to some extent arbitrarily chosen, however, the choice is based on the experience of Mostavi et al. We use a pooling layer to compact the convolutional layer activation tensors and train a dense layer on the compacted inputs. We use one-hot encoding for the output layer to provide the adaptive sampling tool with access to information about the confidence of the input classification.

```
cnn = Sequential()
cnn.add(Conv1D(filters=32, kernel_size=75, strides=10, activation='relu',
    padding='same',  input_shape=input_shape))
cnn.add(Conv1D(filters=64, kernel_size=75, strides=10, activation='relu',
    padding='same'))
cnn.add(Conv1D(filters=128, kernel_size=75, strides=10,
    activation='relu',  padding='same'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Flatten())
cnn.add(Dense(128, activation='tanh'))
cnn.add(Dense(2, activation='softmax'))
```

Listing 3.1: M_1 design

We find out that less than 5 000 raw signal values are necessary for correct classification. Minimizing the necessary length, we find that 3 000 values, equivalent to approximately 337 bases, are sufficient for a model to identify a read's biological origin with an accuracy of >91%. We also find that due to the classification of smaller inputs containing fewer features for a model to learn, M_1 can be further simplified. We modify M_1 in multiple iterations. We remove the dense layer. We move the last convolutional layer below the pooling layer, allowing it to explore more global features of the input signal. Finally, we decrease the number of filters for the last convolutional layer to

avoid overfitting and decrease classification time. We illustrate model M_2 in Figure below.

Listing 3.2: M_2 design

```
cnn = Sequential()
cnn.add(Conv1D(filters=32, kernel_size=75, strides=10, activation='relu',
    padding='same',  input_shape=input_shape))
cnn.add(Conv1D(filters=64, kernel_size=75, strides=10, activation='relu',
    padding='same'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Conv1D(filters=64, kernel_size=75, strides=10, activation='relu',
     padding='same'))
cnn.add(Flatten())
cnn.add(Dense(2, activation='softmax'))
```

Table 3.1 shows that M_2 achieves similar testing accuracy to M_1, while Table 3.2 shows decrease in classification time due to the more compact design. Even though both experimental models proved the ability to classify the raw signal with an accuracy of $> 91\%$, they have various shortcomings that need to be addressed. Both models exhibit symptoms of overfitting during the training. The validation accuracy of $> 96\%$ surpasses the testing accuracy by a large margin. The lack of any learning rate scheduling mechanism makes the learning progress unstable towards the end of the training. Finally, some of the model hyperparameters, such as kernel size and strides, are chosen arbitrarily without proper research.

We set most of the neural network hyperparameters, such as layer layout, empirically using published knowledge[22] and our own experiments. However, the kernel size and strides hyperparameters depend heavily on the raw signal properties, that we do not understand. Therefore, we search for a better combination using a limited grid search. Using a GPU to accelerate the training, we encounter issues with the reproducibility of the training. With testing accuracy deviation being potentially higher than an improvement achieved by the fine-tuning of the hyperparameters, we could come to false conclusions. We solve the issue by seeding all of the random generators with a fixed seed. We randomly initialize all neural network layers using the fixed seed. We also limit the internal parallelism capabilities of the GPU, thus achieving higher reproducibility of GPU computations at the cost of training speed. We find a more optimal hyperparameter combination. Model M_3, illustrated in Figure 3.3, consistently achieves testing accuracy of $>93\%$.

Listing 3.3: M_3 design

```
cnn = Sequential()
cnn.add(Conv1D(filters=32, kernel_size=60, strides=7, activation='relu',
    padding='same', input_shape=input_shape))
cnn.add(Conv1D(filters=64, kernel_size=60, strides=7, activation='relu',
    padding='same'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Conv1D(filters=64, kernel_size=60, strides=7, activation='relu',
    padding='same'))
cnn.add(Flatten())
cnn.add(Dense(2, activation='softmax'))
```

We further modify M_3 in multiple iterations to solve the remaining issues. We find that a higher number of trainable parameters helps model to achieve higher testing accuracy, if the overfitting effect can be limited. We double the number of filters for all convolutional layers to increase their feature extraction capacity. We add dropout layers to limit the potential overfitting. The dropout layer rate is set to mimic the relatively high error rate of nanopore reads. We also schedule the learning rate by decreasing it exponentially as the training progresses. We illustrate the changes in Figure 3.4. Table 3.1 shows an improvement in the testing accuracy, which is >93.8%. At the same time, the overfitting symptoms persist during training. At this point, the most likely cause for the model to overfit is the compact training dataset. Later, we report the testing results using larger datasets.

The yield of PCR amplification in the clinical sample varies greatly. Unsuccessful PCR amplification has various potential causes. To name at least one, viral genomes evolve over time. If a viral region designated for primer sequences to ligate to mutates during the evolution of the virus, primer sequence ligation on the strand sequence may be weak, or primers may be completely unable to ligate to the strand sequence. Such changes of the amplified viral genome typically require a redesign of the primer scheme. Poor primer ligation may lower the quality of amplicons or prevent some regions of the target sequence from being amplified completely. The described training dataset is deliberately extracted from the sequencing data produced from a clinical sample where PCR amplification yield was high. For most of patient samples, approximately 90% of the reads could be aligned to a *SARS-CoV-2* genome, indicating both high amplicon yield and reasonable amplicon quality. For the next experiment, we extracted the training data from a sequencing run with a lower PCR yield. The data were also produced by the Institute of Virology, Slovak Academy of Sciences. We chose the reads from patient samples, where approximately 30% of the reads could be aligned to the *SARS-CoV-2* genome. We expect aligned amplicons to have lower quality and want to

observe how it affects the testing accuracy. We extract approximately 150 000 positive training examples and a similar number of negative training examples.

Listing 3.4: M_4 design

```
cnn = Sequential()
cnn.add(Conv1D(filters=64, kernel_size=60, strides=7, activation='relu',
    padding='same', input_shape=input_shape))
cnn.add(Dropout(rate=0.1, seed=SEED))
cnn.add(Conv1D(filters=128, kernel_size=60, strides=7, activation='relu',
    padding='same'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Dropout(rate=0.1, seed=SEED))
cnn.add(Conv1D(filters=128, kernel_size=60, strides=7, activation='relu',
    padding='same'))
cnn.add(Flatten())
cnn.add(Dense(2, activation='softmax'))

learning_rate_schedule =
    keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=0.0005,
    decay_steps=10_000, decay_rate=0.96, staircase=True)
```

Table 3.1 shows significantly decreased testing accuracy and M_4 being unable to fit the training data. The training accuracy was only 92% compared to previous >98%. The data labeling method using full read alignment to *SARS-CoV-2* genome might provide us with an incorrect ground truth when using a training dataset extracted from a low-quality sequencing data. It is possible that the training dataset contains large numbers of *SARS-CoV-2* reads with poor quality. If those reads can not be aligned to the target sequence, our labeling method considers them negative training examples. Thus forcing the CNN model to distinguish between low-quality *SARS-CoV-2* reads and well-aligned ones. To prove our hypothesis, we train M_4 on another training dataset. We use positive training examples extracted from low-quality sequencing data. We expect a greater quality-related variation compared to the original training dataset. We combine the positive examples with the negative training examples from the original dataset, expecting them to contain a low number of *SARS-CoV-2* reads that could not be aligned to the target sequence. Table 3.1 shows the testing accuracy of >95%, which is comparable to the previous experiments with M_4.

In the experiments conducted so far, we have only used the first 3 000 raw signal values for training examples. Even though the results seem promising, they may be just a consequence of the limited training dataset. The training examples do not cover the

Model Name	Sensitivity	Specificity	Precision	Accuracy	F1-score
M_1	93.49%	89.85%	89.45%	91.60%	91.43%
M_2	93.67%	90.82%	90.39%	92.18%	91.99%
M_3	94.35%	92.49%	92.05%	93.38%	93.18%
M_4	94.03%	93.77%	93.29%	93.90%	93.65%
M_4*	86.15%	87.43%	85.44%	86.84%	85.79%
M_4**	95.57%	95.23%	94.54%	95.39%	95.05%
M_5	95.71%	96.61%	94.86%	96.26%	95.28%
M_6	95.98%	97.85%	96.67%	97.09%	96.32%
M_7	96.14%	96.38%	94.54%	96.30%	95.33%
M_8	95.31%	96.07%	94.06%	95.78%	94.68%

Table 3.1: Classification measurements of the proposed classifiers

* - measurements on low-quality sequencing data

** - measurements on high-quality sequencing data using combined training dataset

Model Name	Average time	Maximum time	Minimum time
M_1	2.69ms	26.49ms	2.56ms
M_2	2.55ms	18.21ms	2.35ms
M_3	2.50ms	10.03ms	2.37ms
M_4	2.87ms	17.87ms	2.67ms
M_5	2.87ms	76.91ms	2.62ms
M_6	2.85ms	102.03ms	2.62ms
M_7	2.79ms	71.92ms	2.64ms
M_8	2.76ms	65.90ms	2.52ms

Table 3.2: Classification times of the proposed classifiers

entire *SARS-CoV-2* genome. The coverage is limited due to the fixed set of amplicon starting positions coming from the primer scheme of a PCR amplification protocol. To train a CNN model that can be applied to an adaptive sampling task, we conducted a series of experiments increasing the demands for the generalization of the raw signal features.

First, we extract a new training dataset from the sequencing data. Instead of one window of 5 000 raw signal values, multiple windows of the raw signal are extracted from fast5 files. Each signal window forms an individual training example, and we extract as many of them as can be fit into the read. This greatly increases the training data coverage of the target sequence and the size of the training dataset. We extract approximately 600 000 positive training examples and a similar number of negative training examples. We designate a tenth of the dataset as testing data and use the rest for training. We train model M_5 and examine its ability to learn the features from a richer dataset. Again, the first 3 000 raw signal values of each training example are used during the training. Table 3.1 shows that M_5 achieved a testing accuracy of $>95\%$. An increased size of the training dataset helps M_5 avoid overfitting during the training.

In a real adaptive sampling scenario, an arbitrary portion of the raw signal can form a classification input. We demonstrated that the increased training dataset coverage of both target and non-tartget genomes does not harm the model's ability to classify the raw signal. However, M_5 was still trained and tested using fixed regions of the target genome. To mimic classification inputs during a real adaptive sampling scenario, we further modify the training process. Again, we use an input window of 3 000 raw signal values from each training example, but this time its position within the 5 000 value-long training example is chosen randomly each time. This way, raw signal motives can be located in multiple regions of the inputs during training. We use a uniform probability distribution to position the input window within the training example. We expect the model to learn more general input features in order to classify the signal. We train model M_6 using the described technique. During the evaluation of the model, we implement the same random positioning of the testing inputs. Table 3.1 shows even higher testing accuracy of $>97\%$.

Since the achieved results are optimistic, we aim to shorten the input length necessary for the raw signal classification. We train a model M_7 using the inputs of 2500 raw signal values, which is equivalent to approximately 281 bases. We also train a model M_8 using the inputs of 2000 raw signal values, approximately 225 bases. Table 3.1 shows an acceptable decrease in testing accuracy and a decrease in classification time as we lower the input length.

3.3.3 Selectify

We implemented a simple adaptive sampling tool. Its integral part is the trained classification model described in the previous section. Data chunks obtained using the Read Until API are filtered before the classification step. Even if the Read Until API is configured with a chunk length sufficient to form a classifier input, shorter chunks produced by DNA sequences that just started sequencing can be received from the API. Only data chunks of sufficient length are normalized and classified. The *proceed* decision is made for the rest of the chunks. A minimum classification confidence can be set for selectify to make *ublock* or *stop receiving* decisions. If a data chunk is not classified with satisfactory confidence, *proceed* decision is made by default. At most two data chunks per read are classified. If a confident enough classification can not be achieved, *stop receiving* decision is made by default as a conservative sampling strategy.

3.4 Results

We connect selectify to the virtual sequencer and test the classifier in emulated conditions. We emulate the sequencing run used for the training of the classifier. We compare statistics from a 10 minute-long non-selective sequencing run and selective sequencing runs using Readfish and selectify adaptive sampling tools. We configure Readfish with a consensus genome of SARS-CoV-2[9] as a reference sequence. The *stop receiving* decision is sent if the data chunk aligns to the reference sequence. We set the maximum number of processed data chunks to 3 and use the HAC base calling model for *Guppy server*. We test selectify in two configurations. Firstly, a 90% data chunk classification confidence is required for selectify to make the decision. In the second experiment, 75% classification confidence is sufficient.

During emulated runs, we find that selectify is unable to correctly classify regions of the genome that are not covered by the training dataset. We overestimated the level of raw signal generalization that the model is capable of. Instead, non-overlapping training example windows caused the model to poorly classify data chunks positioned over the boundary of two training examples. Therefore, we generated a new dataset, but this time we overlapped training examples and made sure that the ends of the reads were also covered. We also ensure that all individual barcoded samples are represented in the dataset. We extracted approximately 5 training examples per read. Training dataset consists of examples extracted from 1 500 reads per each of the 96 barcoded samples in the merged sequenced sample. Overall, the training dataset contains approximately 720 000 positive training examples and a similar number of negative training examples. We train M_8 using the new dataset for 2 hours and achieve testing accuracy of 95.67%.

	Original Run	Readfish	Selectify - 90%	Selectify - 75%
On-target Read Count	15592	16193	16597	16720
Off-target Read Count	3704	4061	3887	3920
On-target Avg. Length	1936.54b	1917.21b	1791.99b	1758.27b
Off-target Avg. Length	1290.07b	898.60b	1210.10b	1192.99b
On-target Bases	30.19M	31.05M	29.74M	29.40M
Off-target Bases	4.78M	3.65M	4.70M	4.68M
Absolute Enrichment	1.00x	1.03x	0.99x	0.97x

Table 3.3: Comparison of the Readfish and selectify in the emulated run

Table 3.3 shows, that Readfish was able to achieve negligible absolute enrichment. The potential gain of unblocking off-target reads was limited by the low average length of off-target reads in the sequenced sample. Selectify decreases the average length of on-target reads more significantly than Readfish and is unable to deplete the biological background of the sample. This is because the diverse nature of host background DNA was not fully covered by the training dataset. Even when requiring 90% classification confidence for *unblock* decisions, on-target reads are being unblocked during the emulation. This is likely caused by the potential differences in 96 different clinical samples merged into a single sample that are not covered by the training dataset.

Table 3.4 shows a decrease in selectify's sensitivity compared to Readfish and its tragic specificity of >8%. The results demonstrate that our proposed adaptive sampling method has potential use only when sequencing samples whose composition is precisely known and can be covered by a training dataset. Figure 3.1 shows a distribution of read alignment starting positions of on-target reads unblocked by selectify. The distribution is normalized by the total number of reads whose alignment to the reference sequence starts at the particular position during the emulated run. We observe that multiple critical areas of the target genome are consistently being depleted due to incorrect classifications. The genome variation occurring in these areas is not properly expressed in the training dataset. Figure 3.2 shows the reversed distribution of alignment starting positions of on-target reads sequenced by selectify. A large portion of the target genome was consistently classified correctly. Selectify's specificity slightly increases with the decreasing classification confidence threshold, but this comes at the cost of its decreased sensitivity. We measured the average time needed for data chunk classification in both Readfish and selectify. Table 3.5 shows that selectify accelerates data chunk classification by 1.6-fold compared to Readfish.

Selectify's faster decisions combined with potentially shorter data chunks needed for the classification could lead to an increase in adaptive sampling performance when applied to a sequencing sample with known composition. The classification speed is

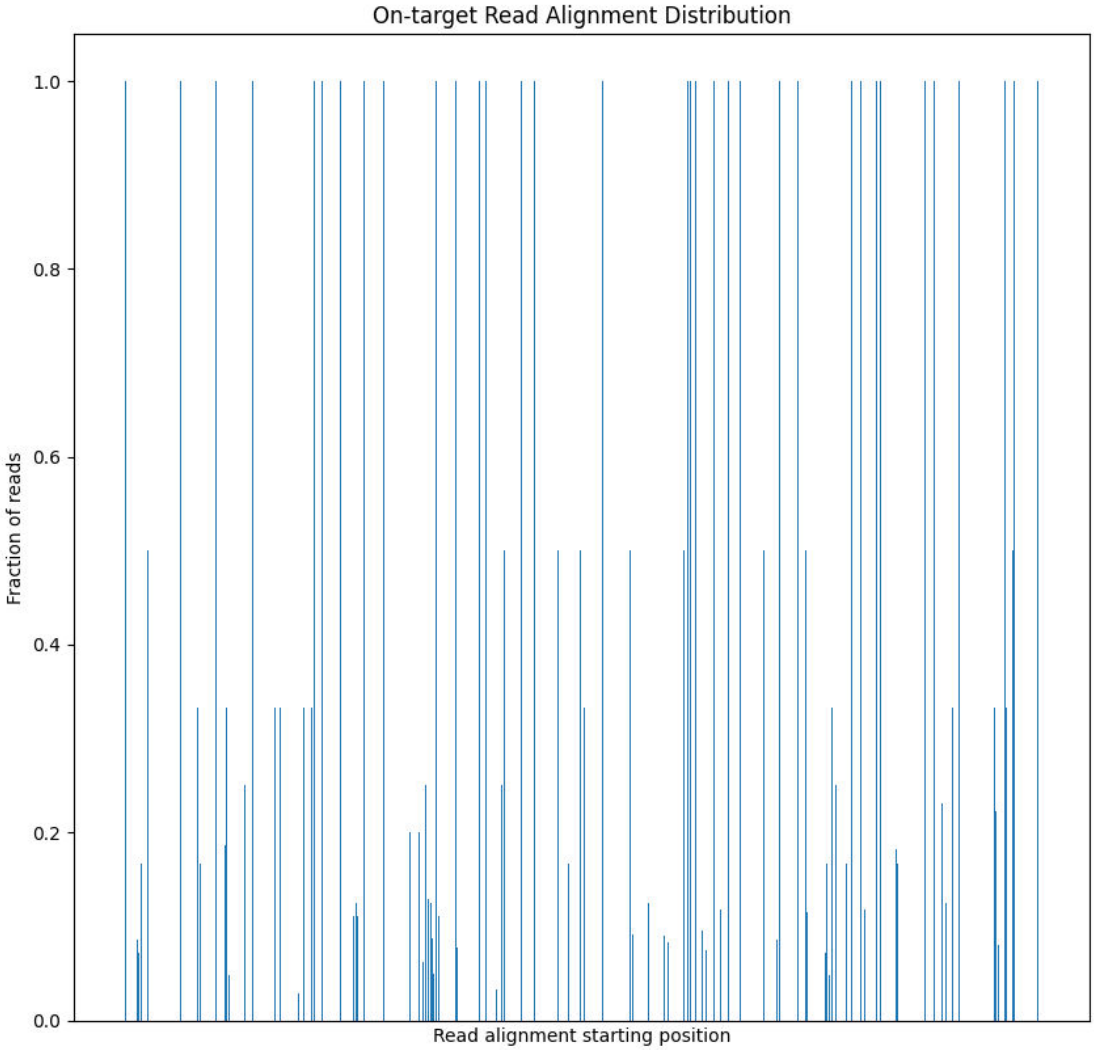

Figure 3.1: Distribution of unblocked on-target read alignment positions

increased at the cost of the model being unable to extract more general features from the training dataset. Therefore, the classifier's accuracy is sensitive to sequencing data deviating from the training data. However, clinical samples contain rapidly evolving viral DNA and a diverse host background DNA. Selectify's current design leads to the depletion of newly introduced variants of viral DNA, while these variants are often of the greatest interest. The strict laboratory conditions that are required for selectify's high adaptive sampling performance are therefore incompatible with its application in the field of epidemiology.

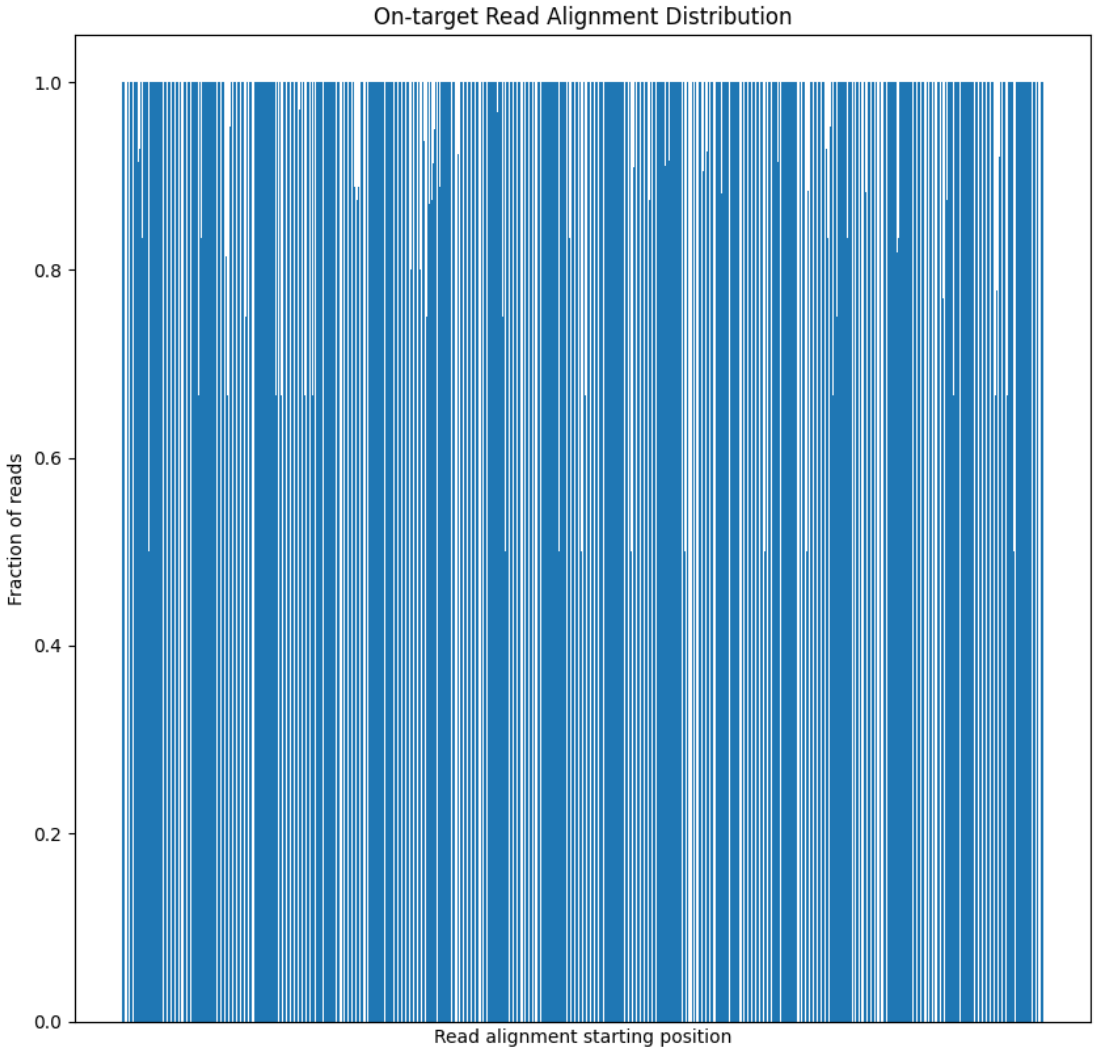

Figure 3.2: Distribution of sequenced on-target read alignment positions

	Readfish	Selectify - 90%	Selectify - 75%
Accuracy	86.58%	75.27%	73.93%
Sensitivity	98.78%	91.00%	88.87%
Specificity	37.90%	8.13%	10.23%
Precission	86.38%	80.88%	80.85%

Table 3.4: Classification measurements of the Readfish and selectify

	Readfish	Selectify - 90%
Time per data chunk	4.45ms	2.68ms
Acceleration	1.00x	1.66x

Table 3.5: Average classification times per data chunk in the emulated run

Discussion

In our work, we aim to develop a tool that will make the research of adaptive sampling methods more accessible and cost-efficient. We introduce the virtual sequencer, a piece of software able to emulate selective sequencing using a previously finished sequencing run. Unlike MinKNOW's playback feature, the virtual sequencer is designed with the intention of replacing the role of the physical sequencer during the development of an adaptive sampling tool. We demonstrate its capabilities on multiple occasions. First, we fine-tune the Readfish configuration in the emulated environment in order to maximize its adaptive sampling performance. Next, we utilize the virtual sequencer in the development of our own adaptive sampling tool. Finally, we use the emulated sequencing runs to compare the two adaptive sampling tools in identical conditions and report extensive statistics. The virtual sequencer already provides an emulated environment for unbiased comparisons of various adaptive sampling methods. However, its ability to predict the coverage achieved during a real sequencing run is limited due to its several design simplifications. As a result, the reported coverage of emulated runs tends to be higher than is achievable in real conditions.

In order to bring the emulation closer to reality, various parameters of the physical sequencer need to be accessed and incorporated in the virtual sequencer. The future development might include direct comparisons with the physical sequencer, which could help identify residual differences. Also, the analysis of real selective sequencing runs recorded in bulk fast5 files would provide us with the raw signal annotated by MinKNOW software. Examination of the annotations could help us identify the speed, at which DNA sequences are ejected from the nanopore channel. As far as we know, the ejection speed of nanopore channels remains the most significant unknown parameter in our emulations.

We study potential applications of machine learning in adaptive sampling. In our proposed adaptive sampling method, adaptive sampling generality is sacrificed in favor of increased performance. We design a convolutional neural network classifier trained specifically to adaptively sample *SARS-CoV-2* viral DNA sequences while sequencing clinical sample. We integrate the classifier into the adaptive sampling tool selectify. Selectify proves itself in terms of decision speed. It requires significantly shorter data chunks to make decisions, and the average time to classify a data chunk

is lower compared to Readfish. However, we were unable to adaptively sample target DNA sequences from a clinical sample. The design of the classifier makes it unable to generalize features of the training dataset in such a way that viral DNA variants not expressed in the dataset can be correctly classified. Therefore, the diverse and rapidly evolving nature of viral clinical samples makes it unsuitable for use in the field of epidemiology.

However, less ambitious experiments should be conducted with selectify to further study its properties. Unlike viral clinical samples, ZymoBIOMICS Microbial Community Standard samples have a known composition. Therefore, the genomes in the sample can be well covered by the training dataset. In such a relaxed setting, selectify might be able to leverage its design to adaptively sample underrepresented DNA sequences from the sample with notable performance.

Bibliography

[1] Bacillus subtilis. https://www.ncbi.nlm.nih.gov/genome/?term=Bacillus+subtilis. Last accessed 2023-05-08.

[2] Enterococcus faecalis. https://www.ncbi.nlm.nih.gov/genome/?term=Enterococcus+faecalis. Last accessed 2023-05-08.

[3] Escherichia coli. https://www.ncbi.nlm.nih.gov/genome/?term=Escherichia+coli. Last accessed 2023-05-08.

[4] Listeria monocytogenes. https://www.ncbi.nlm.nih.gov/genome/?term=Listeria+monocytogenes. Last accessed 2023-05-08.

[5] Pseudomonas aeruginosa. https://www.ncbi.nlm.nih.gov/genome/?term=Pseudomonas+aeruginosa. Last accessed 2023-05-08.

[6] Saccharomyces cerevisiae (baker's yeast). https://www.ncbi.nlm.nih.gov/genome/?term=Saccharomyces%20cerevisiae%5BOrganism%5D&cmd=DetailsSearch. Last accessed 2023-05-08.

[7] Salmonella enterica. https://www.ncbi.nlm.nih.gov/genome/?term=Salmonella+enterica. Last accessed 2023-05-08.

[8] Sequencing data produced by ZymoBIOMICS standard sample. https://github.com/nanoporetech/zymo-data. Last accessed 2023-01-11.

[9] Severe acute respiratory syndrome-related coronavirus. https://www.ncbi.nlm.nih.gov/genome/?term=Severe+acute+respiratory+syndrome-related+coronavirus. Last accessed 2023-05-08.

[10] Staphylococcus aureus. https://www.ncbi.nlm.nih.gov/genome/?term=Staphylococcus+aureus. Last accessed 2023-05-08.

[11] Ajitsaria A. What Is the Python Global Interpreter Lock (GIL)? https://realpython.com/python-gil/. Last accessed 2023-05-08.

[12] Payne A., Holmes N., Rakyan V., and M. Loose. Whale watching with BulkVis: A graphical viewer for Oxford Nanopore bulk fast5 files. https://www.biorxiv.org/content/10.1101/312256v1.full.pdf. Last accessed 2023-05-08.

[13] Brejová B., Boršová K., Hodorová V., Čabanová V., Gafurov A., Fričová D., Neboháčová M., Vinař T., Klempa B., and Nosek J. Nanopore sequencing of SARS-CoV-2: Comparison of short and long PCR-tiling amplicon protocols. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0259277. Last accessed 2023-05-08.

[14] Masutani B. and Morishita S. A framework and an algorithm to detect low-abundance DNA by a handy sequencer and a palm-sized computer. https://doi.org/10.1093/bioinformatics/bty663. Last accessed 2021-01-08.

[15] Quick J. et al. Multiplex PCR method for MinION and Illumina sequencing of Zika and other virus genomes directly from clinical samples. https://doi.org/10.1038/nprot.2017.066. Last accessed 2023-05-06.

[16] Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics 34, 3094-3100 (2018). Last accessed 2021-01-08.

[17] Ulrich J., Lutfi A., Rutzen K., and Renard B. Y. Readbouncer: precise and scalable adaptive sampling for nanopore sequencing. https://academic.oup.com/bioinformatics/article/38/Supplement_1/i153/6617484. Last accessed 2023-05-08.

[18] JensUweUlrich. Readbouncer. https://github.com/JensUweUlrich/ReadBouncer. Last accessed 2023-05-08.

[19] LooseLab. Readfish. https://github.com/LooseLab/readfish. Last accessed 2023-05-08.

[20] LooseLab. read_until_api_v2. https://github.com/LooseLab/read_until_api_v2. Last accessed 2023-05-08.

[21] Loose M., Malla, S., and Stout M. Real-time selective sequencing using nanopore technology. https://doi.org/10.1038/nmeth.3930. Last accessed 2021-01-08.

[22] Mostavi M., Chiu Y., Huang Y., and Chen Y. Convolutional neural network models for cancer type prediction based on gene expression. https://bmcmedgenomics.biomedcentral.com/articles/10.1186/s12920-020-0677-2. Last accessed 2022-12-31.

[23] Alexander Payne, Nadine Holmes, Thomas Clarke, Rory Munro, Bisrat Debebe, and Matthew Loose. Nanopore adaptive sequencing for mixed samples, whole exome capture and targeted panels. https://www.biorxiv.org/content/10.1101/2020.02.03.926956v2.abstract. Last accessed 2023-05-08.

[24] Zymo Research. ZymoBIOMICS Microbial Community Standards. https://zymoresearch.eu/collections/zymobiomics-microbial-community-standards. Last accessed 2023-05-08.

[25] Edwards H. S., Krishnakumar R., Sinha A., Bird S. W., Patel K. D., and Bartsch M. S. Real-Time Selective Sequencing with RUBRIC: Read Until with Basecall and Reference-Informed Criteria. https://www.nature.com/articles/s41598-019-47857-3. Last accessed 2021-01-08.

[26] Kovaka S., Fan Y., Ni B., Timp W., and Schatz M. C. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. https://www.biorxiv.org/content/10.1101/2020.02.03.931923v1.full. Last accessed 2023-05-08.

[27] Oxford Nanopore Technologies. ont_fast5_api. https://github.com/JensUweUlrich/ReadBouncer. Last accessed 2023-05-08.

[28] Oxford Nanopore Technologies. Read until. https://github.com/nanoporetech/read_until_api. Last accessed 2023-05-08.

[29] Oxford Nanopore Technologies. Read until api code snippet demonstrating filtering data chunks based on classes. https://github.com/nanoporetech/read_until_api/blob/release/read_until/base.py#L168. Last accessed 2023-05-08.

[30] Oxford Nanopore Technologies. Read until api code snippet demonstrating real unblock latency. https://github.com/nanoporetech/read_until_api/blob/release/read_until/base.py#L359. Last accessed 2023-05-08.

[31] Vintsyuk T.K. Speech discrimination by dynamic programming. Cybernetics, 4, 52–57. Last accessed 2021-01-08.

[32] Boža V., Perešíni P., Brejová B., and Vinař T. DeepNano-blitz: a fast base caller for MinION nanopore sequencers. https://pubmed.ncbi.nlm.nih.gov/32374816/. Last accessed 2021-01-08.