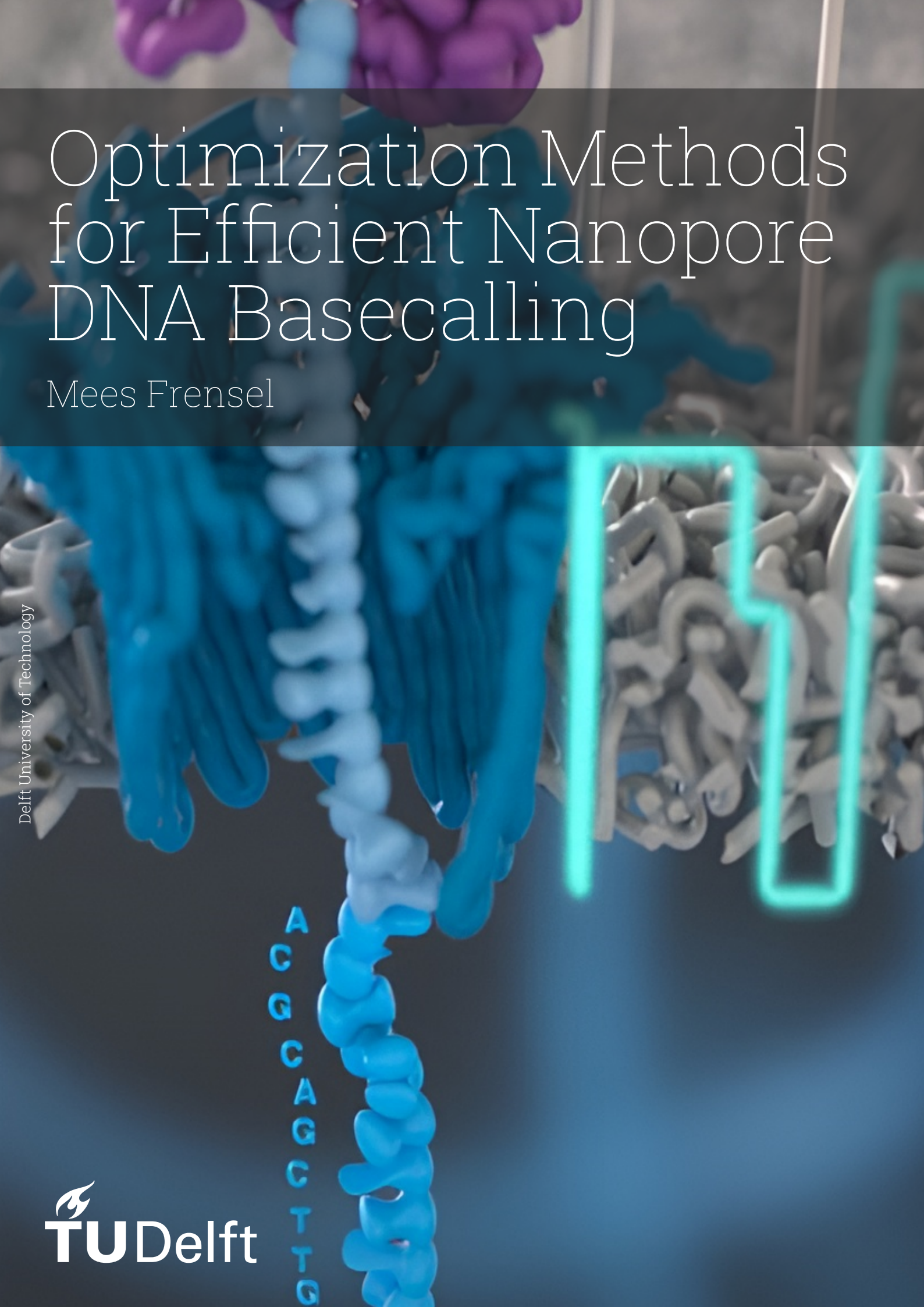
Optimization Methods for Efficient Nanopore DNA Basecalling

Mees Frensel



Delft University of Technology

TU Delft

Optimization Methods for Efficient Nanopore DNA Basecalling

by

Mees Frensel

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday, 12 August 2024, at 10:00.

Student number:	4719778	
Project duration:	November 2023 – August 2024	
Thesis committee:	Dr. ir. Z. Al-Ars,	TU Delft, supervisor
	Prof. dr. H. P. Hofstee,	TU Delft / IBM, supervisor
	Dr. ir. E. van den Akker,	TU Delft / Leiden University Medical Center
	Dr. ir. R. Shirali,	Leiden University Medical Center

This thesis is confidential and cannot be made public until 31 December 2024.

Cover:	A DNA strand passes through a nanopore in a MinION sequencer
Image credit:	Oxford Nanopore Technologies

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

Abstract

Genomics, the study of an organism's complete set of DNA, including all of its genes, has revolutionized our understanding of biological processes and disease mechanisms. The field's rapid advancements have paved the way for personalized medicine, offering targeted therapies and improved healthcare outcomes. These advancements are a result of significant improvements in sequencing technology, bioinformatics, and computational power. Next-generation or long-read sequencing has reduced the cost and time required to sequence entire genomes, and Oxford Nanopore Technologies (ONT) sequencers provide 100–1000× longer contiguous reads, simplifying genome assembly. However, bioinformatics-driven advances in accuracy have come at the cost of high computational requirements because of the dependency on large deep neural networks (DNNs), and the basecalling step now takes 43% of the time in the nanopore sequencing pipeline.

This thesis addresses the large computational demands for high accuracy nanopore basecalling of nanopore reads. Bonito, ONT's research basecaller, and other basecallers use DNNs at their core. The five Long Short-Term Memory (LSTM) layers used by the basecaller are the primary bottleneck to more efficient basecalling, taking almost 90% of the whole model's execution time when basecalling a single read. To alleviate this bottleneck, three approaches are investigated: pruning, model architecture, and quantization. Preliminary results show that pruning is the most impactful approach and has not successfully been used in previous work.

We propose learning structured sparsity using a delayed masking penalty scheduler. By adapting and improving on previous work, each LSTM layer is able to learn its optimal size during training, simultaneously with learning to basecall accurately. The method is optimized for the basecalling application and can be generalized to other tasks. We find that the required number of computations in the LSTM layers can be significantly reduced by up to 21 times with a reduction in match rate of just 1.3% compared to the high accuracy Bonito model. Furthermore, the newly introduced penalty parameter can be tuned to find the optimal trade-off between compute and accuracy for users' requirements.

The results indicate that state-of-the-art basecalling models are overparameterized and that their size can be reduced drastically without significantly affecting accuracy. Future work is suggested to investigate the benefits of pruning the whole model, and to assess the feasibility of combining pruning with advanced quantization methods. This work helps increase the accessibility of nanopore DNA sequencing, broadening the reach and impact of this technology.

The code with the masking mechanism to reproduce the results is available at `https://github.com/meesfrensel/efficient-basecallers`

Preface

After three months of reading papers, I didn't think I had it in me to do research and successfully finish this project. Luckily, I finally saw the light as the dark winter was coming to an end, and each month the project was looking a bit better, with interesting new results and insights every week – right up to this moment, just before submitting the final copy of my thesis. Now, I look back on a great learning experience, both personally and academically, and the conclusion that hard work definitely pays off.

I'd like to thank Zaid and Peter for their great supervision and guidance on this project. Working with supervisors who are genuinely interested and enjoy discovering new insights from my rough results has been a breath of fresh air, and I learned to keep pushing forward even when I didn't really feel like it anymore. It was at these moments that we made the biggest progress, and looking back, one curious thought from Peter and Zaid was often enough to motivate me to keep investigating further. Always asking questions, always thinking outside the box. I should be more curious about my own work as well!

During the past 7 years in Delft, I have made too many friends to thank everyone individually here. I'd like to thank the Compensatie rowing squad, my roommates at Huize Appeltaart, my fellow board 74 members, and everyone else for the coffee, beers, laughs, holidays, festivals and good times in general. Julian, Robin and everyone else studying with us in the office, thanks for keeping me company for 9(!) months. Thanks to my parents for always supporting me in every meaning of the word. Returning to Amersfoort has been a great way to relax and do nothing, although my little brothers won't let me off the hook so easily and there's always some task to be done! Most important of all, so much appreciation to Maaike for keeping up with me when I was stressed, tired or had planned more evening activities than we both would've liked, and for wishing me luck with even the most mundane meetings. 'Do you really have to get up that early?' Nope. I'm done!

Mees Frensel
Delft, August 2024

Contents

Abstract i

Preface ii

1 Introduction 1

2 Background 3

2.1 DNA sequencing pipeline . . . 3

2.1.1 Sample preparation . . . 3

2.1.2 Sequencing . . . 4

2.1.3 Basecalling and alignment . . . 4

2.1.4 Downstream tasks . . . 5

2.2 Basecalling . . . 6

2.2.1 Historical context and evolution . . . 7

2.2.2 Modern basecalling algorithms . . . 7

2.2.3 Challenges in basecalling . . . 8

2.2.4 Impact of basecalling on sequencing quality . . . 8

2.3 Applications of nanopore sequencing . . . 9

2.3.1 Application domains . . . 9

2.3.2 Requirements . . . 10

3 Efficient basecaller architectures 11

3.1 Overview . . . 11

3.2 Previous work . . . 12

3.3 Reducing the computational burden . . . 14

3.3.1 Quantization . . . 14

3.3.2 Model size and architecture . . . 16

3.3.3 Pruning . . . 17

3.3.4 Selecting a method . . . 17

4 Pruning 18

4.1 Methods . . . 18

4.1.1 Unstructured pruning . . . 18

4.1.2 Structured pruning . . . 19

4.2 Results . . . 20

5 Learning structured sparsity **22**
5.1 Introduction . . . 22
5.2 Neuron selection . . . 23
5.2.1 Method . . . 23
5.2.2 Optimizing training time . . . 24
5.2.3 Delayed masking . . . 24
5.2.4 Efficient inference . . . 25
5.3 Evaluation of alternative pruning methods . . . 26

6 Experiments and results **27**
6.1 Experimental setup . . . 27
6.2 Datasets . . . 27
6.3 Basecalling accuracy . . . 29
6.4 Basecalling throughput . . . 29
6.5 Ablation study . . . 31
6.6 Discussion . . . 32

7 Conclusions and recommendations **33**

A Storage requirements **35**

B Comparison of systems **36**

1

Introduction

Genomics is the field of biology that focuses on the structure, function, evolution, mapping, and editing of genomes. A genome is the complete set of DNA, including all of its genes, within an organism. Genomics has revolutionized our understanding of biological processes and disease mechanisms, and personalized medicine can uncover more from a person's DNA than ever. To read the genome of a human or other organism requires sequencing pieces of DNA and assembling the pieces into e.g. chromosomes.

Short-read or second generation sequencing reads strands of DNA that are 100–300 base pairs (bp) in length. Because some repeating structures in the human genome like long terminal repeats span 200–600 bp [45], short-read sequencing traditionally has not been able to assemble complete genomes without fragmented results and missing portions [86]. As a result, assembling the whole genome from short reads requires vast amounts of compute to run the algorithms that are able to perform assembly with many short reads.

Long-read sequencing, also described as third or next-generation sequencing (NGS), can sequence DNA fragments that are two or three orders of magnitude longer at 10–100 kbp, with even longer reads of up to 2 Mbp possible with nanopore sequencing technologies [45]. While historically suffering from low accuracy and high cost, recent advances have nearly closed the gap to short-read sequencing accuracy [86]. Long-read sequencing allows the sequencing and assembly of the entire human genome, which opens up possibilities for genome annotation and single-base as well as structural variation analysis, among others, opening up possibilities for personalized medicine. Combined with significantly reduced time-to-result, long-read sequencing is expected to find applications in clinical settings for diagnosis and genome-based treatments, especially for neurological diseases and cancer.

The basecalling bottleneck

Nanopore sequencing is one form of long-read sequencing, in which DNA strands pass through a nanopore to read the nucleotides. This way of sequencing, developed by Oxford Nanopore Technologies (ONT) competes mostly with PacBio's single molecule real-time sequencing (SMRT), which is not discussed further. A grid of nanopores is situated in a membrane that separates two voltage biases [55]. While DNA strands pass through the nanopore, the individual bases disrupt the ionic current through the nanopore, which is measured and is the output of the sequencer. The noisy current disruption measurements can be traced back to bases by 'basecalling' software. Although basecallers have been much improved over time, now achieving up to 99.5% accuracy, this comes at the cost of much higher computational requirements.

Modern basecallers rely on deep neural networks (DNNs) for their high accuracy. Over the years, these networks have become larger and more complex, requiring large, powerful and expensive graphics processing units (GPUs) to keep up with sequencing output. GPUs can perform many operations simultaneously, which is a good fit for the many matrix multiplications that neural networks have to calculate. However, keeping up with sequencing output in real-time requires powerful hardware and

consumes substantial amounts of energy. In the literature, a number of works address this problem and propose a more efficient neural network or other method to improve the energy consumption and basecalling speed. Nevertheless, none of these works achieve large improvements without significant losses in accuracy.

Objectives and approach

The main problem this thesis attempts to address is the large computational demands for high-accuracy nanopore basecalling. For real-time basecalling, an efficient basecaller requires less powerful hardware to keep up, and re-basecalling previously sequenced data can be done faster and using less energy. This problem can be further subdivided into the following questions:

1. What bottlenecks exist in basecalling models and how can these be alleviated?
2. What improvement avenues are feasible for the basecalling application?
3. How much more efficient can basecalling models become without sacrificing too much accuracy?

Outline

A broad overview of the nanopore sequencing landscape is presented in Chapter 2, with a detailed introduction to basecalling. Chapter 3 discusses previous work on efficient basecalling and introduces three neural network-oriented methods to improve efficiency. Pruning is one method that is discussed in detail in Chapter 4, along with some experiments. Chapter 5 proposes learning structured sparsity by masking the Long Short-Term Memory (LSTM) layers of the model. Using this method, the model learns the optimal size simultaneously with learning to perform basecalling accurately. In Chapter 6, the validity and benefits of the method are tested in several experiments regarding a number of evaluation metrics: accuracy, model size, and throughput. Chapter 7 closes the thesis with conclusions and recommendations for future research directions.

2

Background

Nanopore sequencers measure electric current as DNA molecules pass through nanopores in a membrane, with these measurements being converted into nucleotide sequences by basecalling software. Long-read sequencing technologies, of which nanopore sequencing is one type, address the challenges of short-read sequencing by producing longer contiguous genome reads, significantly simplifying genome assembly. While traditional short reads average 200 base pairs, Oxford Nanopore Technologies (ONT) long reads offer about 100,000 base pairs, reducing the number of reads needed for human genome assembly from 450 million to 900 thousand reads [45]. This simplification aids in aligning long repeat sequences and determining haplotypes, a specific combination of genetic variations on a chromosome inherited as a unit [6]. Early long-read sequencing faced issues of being slow, error-prone, and expensive, but advancements over the past decade have increased accuracy from approximately 60% in 2015 to around 99.5% today with ONT sequencers in the best scenario. Despite some accuracy challenges, long-read sequencing is poised to replace short-read sequencing, with hybrid approaches likely to remain common [6, 86].

Discussing basecallers requires some context of nanopore sequencing. The primary stages of the sequencing pipeline are sample collection and preparation, sequencing, basecalling, and alignment, after which downstream tasks can be performed. Examples of these include variant calling, methylation calling, and (de novo) genome assembly. At this stage, the sequenced data is useful for drawing conclusions about gene mutations, diseases, pathogen detection, cancer identification, etc. Section 2.1 presents an overview of the long read DNA sequencing pipeline, highlighting the importance of accurate and fast basecalling.

Then, Section 2.2 presents a deep dive into (challenges of) the basecalling process, its history and future within the context of the sequencing pipeline. Currently, nanopore sequencing is primarily used ‘in production’ for analysis of viral and bacterial DNA, as well as a research tool for cancer research and diagnosis, wastewater analysis, and remote sequencing. These applications really benefit from portable and low-power sequencing and basecalling and are presented in Section 2.3.

2.1. DNA sequencing pipeline

This section briefly presents a common long-read DNA sequencing pipeline, consisting primarily of sample collection and preparation, sequencing, basecalling, alignment, and finally many different ‘downstream tasks’, which are discussed last.

2.1.1. Sample preparation

Before sequencing starts, the DNA or RNA samples have to be prepared according to one of the three main strategies recommended by ONT: tagmentation, ligitation, and PCR amplification [73]. These so-called library prep strategies heavily influence the sequencing output and picking one over the other doesn’t have clear trade-offs but instead depends on the species, post-sequencing workflow (see Section 2.1.4) and requirements on the output sequences. For example, one strategy might

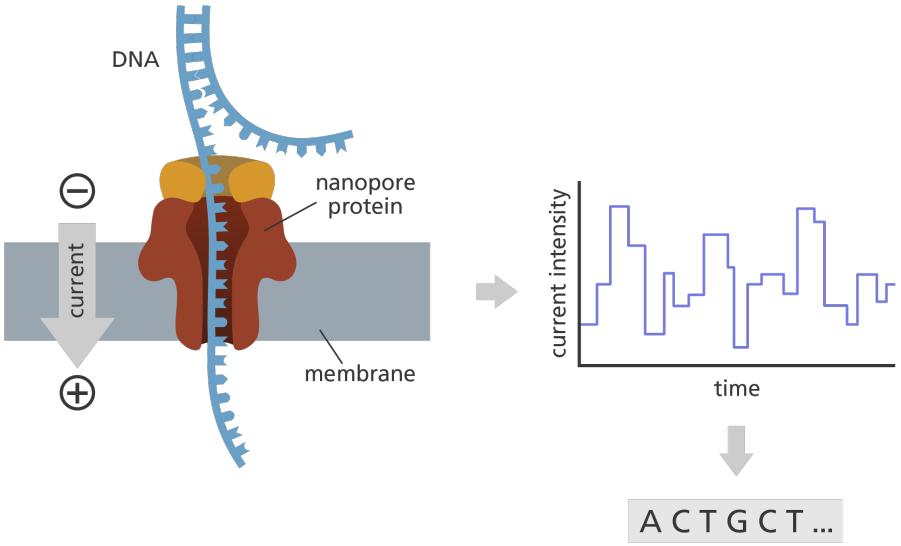


Figure 2.1: Illustration of a DNA strand passing through a nanopore in the membrane. A small piece of electronics, not shown, measures changes in current depicted on the right. Image credit: Laura Olivares Boldú, Wellcome Connecting Science.

produce longer sequences with lower accuracy whereas another might produce shorter sequence with higher accuracy and less homopolymer errors.

One notable sample prep technique is barcoding. By attaching a 'barcode' sequence to the DNA strands, multiple DNA samples can be sequenced in one go by a single sequencer or flowcell [55]. This reduces cost and can increase sequencing throughput if the number of sequencing devices presents a bottleneck. After sequencing, reads can be separated by finding the barcode sequence and grouping reads by barcode. As an example, one paper claims to be able to sequence more than 10,000 specimen samples in one sequencing run at high accuracy [80].

2.1.2. Sequencing

Sequencing is the process of reading the bases of DNA strands in the prepared sample fluid. In the case of nanopore sequencing, DNA strands are fed through a pore in a membrane. The sequencing device measures current flowing through the nanopore, which is disrupted when molecules pass through the pore. This current, which is in the order of around 100 pA, is measured at a rate of 4 or 5 kHz and is the output of the sequencer.

The physical makeup and functioning of nanopores is well described by MacKenzie and Argyropoulos [55], but a short overview, illustrated by Figure 2.1, is given here. The DNA molecules are fed through by a motor protein that separates the double helix structure and feeds them through the nanopore. At any given time, a number of bases are present in the nanopore, which influence the flow of current across the membrane. This short sequence of bases is called a k-mer, and for the nanopore chemistry R9.4.1, which is used in the experiments, $k = 5$. Multiple bases being inside the nanopore at the same time implies that the sequencer cannot measure current by individual bases, but only the change in current when one of the bases is replaced by another one. This is partially why basecalling is so difficult.

2.1.3. Basecalling and alignment

After sequencing the prepared DNA samples, basecalling is the first step in the analysis pipeline. Basecalling is the process of assigning or 'calling' bases, i.e. A, C, G or T (U for RNA), to the signal produced by the sequencing device. This critical first step determines the value of reads in the downstream tasks, since any errors introduced here are propagated throughout the whole pipeline. This might mean that mutations are introduced erroneously or that bases present in the DNA are missed completely. Polishers like Medaka and NanoPolish can find and correct some of these errors

after a draft sequence assembly to remove artifacts of the basecalling and assembly process [46], but in general, the accuracy that a basecaller achieves is an important metric. Basecalling is discussed in-depth in Section 2.2.

The next step is alignment of the reads to the reference sequence. This can be a whole genome, or gene-specific references. Alignment can conceptually be split up into overlap finding, assembly and read mapping. For short reads, alignment can be done with algorithms like Smith-Waterman [79] which finds an optimal alignment of two sequences, but for long reads it is computationally unpractical to use. Instead, heuristic approaches are used with tools like BLAST [5] which provides significantly higher throughput supporting larger datasets. However, for long reads, alignment is commonly done with optimized tools like Minimap2 [47]. Sequence alignment is an ongoing field of study, with new research appearing each year [4].

2.1.4. Downstream tasks

Although many different downstream tasks exist, some predominant ones are discussed here: variant calling, methylation calling, and de novo genome assembly.

Variant calling

Variant calling is a critical downstream task in the genome sequencing process, involving the identification of genetic variations by comparing a sequenced genome to a reference genome or database of genomes. This task is fundamental in genome analysis, as it helps to pinpoint genetic differences that may be associated with diseases, traits, or evolutionary changes.

The primary types of genetic variations identified through variant calling include:

- Single nucleotide variations (SNVs): these are isolated changes in a single nucleotide base within the DNA sequence.
- Insertions and deletions (indels): these variations involve the insertion or deletion of one or more nucleotide bases in the DNA sequence.
- Repeat regions: variations in the length of DNA repeat regions, which can be either shorter or longer than those in the reference genome.

Long-read sequencing offers significant advantages for variant calling over traditional short-read sequencing. Long reads provide a more comprehensive view of the genome, enabling better detection and characterization of genetic variations. In particular, long-read sequencing excels in two areas: long reads can span entire repeat regions, which are often fragmented across multiple short reads in short-read sequencing. This comprehensive coverage allows for more accurate measurement of repeat length and structure [41, 45]. Second, the longer contiguous sequences provided by nanopore sequencing facilitate the accurate identification and characterization of insertions and deletions, which can be challenging to detect with short reads [45].

Traditionally, variant calling has been performed by aligning sequenced reads to a reference genome and identifying discrepancies [67]. This approach, which generally uses dynamic programming algorithms, relies heavily on the accuracy of the reference genome and the quality of the read alignment. This is prone to errors that do not stem from the sequencing experiment, but instead from the reference data.

Recent advancements have introduced the use of deep learning and neural networks in variant calling [67, 3]. These methods can identify genetic variations independent of a reference genome, leveraging patterns learned from large datasets to detect variations directly from the sequencing data. This ‘connectionless’ approach offers increased portability: without the need for extensive reference databases, variant calling can be integrated into more portable sequencing devices, making it feasible to perform genetic analysis in various settings, including remote or low-resource environments. Furthermore, deep learning algorithms can streamline the variant calling process, potentially reducing the time required to identify genetic variations [67].

Methylation calling

Methylation calling is a process in epigenetics that involves identifying and quantifying DNA methylation patterns across the genome [59]. DNA methylation, a biochemical modification where methyl

groups are added to cytosine (C) or adenine (A) nucleotides, plays a pivotal role in gene expression regulation, genomic stability, and development. The primary goal of methylation calling is to accurately determine the methylation status at specific genomic loci, typically CpG sites[1], throughout the genome [51, 59].

Several approaches are employed for methylation calling, each with its strengths and considerations. One common method involves bisulfite sequencing, which selectively converts unmethylated cytosines to uracils, leaving methylated cytosines (5mC) unchanged [48]. By aligning bisulfite-treated sequencing reads to a reference genome and comparing them to the original sequence, researchers can infer methylation status. Another approach utilizes nanopore sequencing technology, which can detect DNA modifications directly as the DNA passes through a nanopore, providing real-time methylation information without the need for bisulfite treatment [51].

Challenges in methylation calling arise from various sources, including sequencing errors, incomplete bisulfite conversion, and biological variability [59]. Computational algorithms must account for these challenges to accurately distinguish methylated from unmethylated cytosines. Furthermore, the interpretation of methylation data often involves statistical modeling and bioinformatics tools to identify differential methylation patterns between different biological conditions or tissues. As methylation patterns are increasingly recognized for their roles in disease mechanisms and environmental responses, advances in methylation calling techniques continue to be vital for understanding epigenetic regulation in health and disease [59]. Nanopore sequencing plays an especially important role in this regard, allowing to perform methylation calling on sequencing data without any modification to the samples [51].

De novo genome assembly
Reconstructing the genome of a species or sample in a population is one of the most important tasks in genomics [60]. De novo – meaning 'from the beginning' or 'from scratch' – genome assembly involves reconstructing a genome from sequenced reads without using a reference genome [15]. Long-read sequencing has significantly simplified this process by reducing the number of reads needed for accurate genome reconstruction. A widely used approach for genome assembly utilizes string graphs, where entire reads are compared using minimizers and MinHash techniques [45].

Recent work demonstrates the use of personalized de novo genome assemblies for the detection of somatic mutations – changes to the DNA that occur after conception – in breast cancer samples [91]. Many cancers are the results of accumulated somatic mutations. Specifically, by using a personal genome reference instead of the standard human genome reference, it is possible to accurately detect somatic mutations that had not been uncovered before. Using a standard human genome reference, there are too many structural variations (SVs) to distinguish genetic differences from somatic mutations. This work is one of a larger body of research that explores the possibilities of personalized medicine, which is only possible with personalized genome references assembled from long reads.

Challenges in this process primarily stem from the accuracy of sequencing and basecalling rather than algorithmic limitations. Currently, a substantial amount of time in de novo assembly is dedicated to correcting errors in the sequenced reads [15]. Achieving error-free, end-to-end assembly of genomes, from telomere to telomere across chromosomes, has recently become feasible by employing novel techniques aimed at enhancing read accuracy [45].

2.2. Basecalling

Basecalling is a critical step in the nanopore sequencing pipeline, translating raw signal data into nucleotide sequences. Understanding the intricacies of basecalling is essential for leveraging the full potential of nanopore sequencing technologies. As illustrated in Figure 2.2, the basecalling step takes the longest amount of time in the sequencing pipeline for long-read technologies and therefore, this thesis focuses on optimizing the throughput and efficiency of this step.

[1] CpG sites refer to CG dinucleotides, i.e. base C followed by G, and not to a C-G base pair.

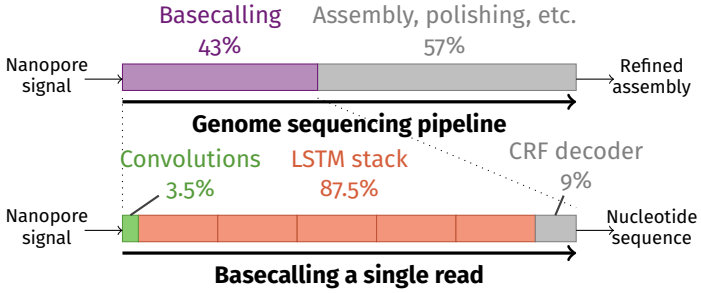


Figure 2.2: Top: basecalling takes 43% of the time in a nanopore genome sequencing pipeline. Bottom: during basecalling, almost 90% of the computing time is spent computing the LSTM layers' outputs. Illustration from [27].

2.2.1. Historical context and evolution

Initially, basecalling in nanopore sequencing relied heavily on simpler algorithms, which often produced high error rates. Early methods, such as Hidden Markov Models (HMMs), provided the groundwork for interpreting ionic current disruptions but were limited by their ability to accurately distinguish between similar signal patterns. While their implementation is relatively simple, the statistical approach is lacking, with match rates of 60–70%. Nanocall [20], a research basecaller, mimicked the HMM architecture which was state-of-the-art at that time, but only achieved a maximum match rate of 68%.

As technology and computational methods advanced, more sophisticated basecalling algorithms emerged using deep learning techniques, significantly improving the accuracy and reliability of sequence data. The first deep neural network (DNN)-based network for basecalling was implemented in Guppy, the graphics processing unit (GPU)-accelerated basecaller that was the production tool for many years. This implementation, by ONT, used only a single layer of recurrent neural networks (RNNs) at first, resulting in still quite low read accuracy of around 88% [89]. This accuracy is too low to perform any useful post-sequencing analysis [45].

Around the same time that this GPU implementation was introduced, independent researchers developed Chiron, a basecaller whose neural network was inspired by speech-to-text models [82]. Chiron combined convolutional layers with RNNs, which meant that the raw nanopore signal could be processed directly, without a separate segmentation preprocessing step. By utilizing a convolutional neural network (CNN), the model extracts useful features of the raw signal instead of feeding just the signal into the RNN directly. Finally, the outputs are classified as bases using a Connectionist Temporal Classification (CTC) decoder. The authors of Chiron state that using both convolutional and recurrent layers is crucial in basecalling networks, as accuracy will otherwise drop significantly [82].

The original RNN-based architectures use Gated Recurrent Unit (GRU) layers, which are a type of RNN. While these are simpler to implement and faster to train, they lack a memory unit which turns out to be instrumental for basecalling. In 2020, ONT introduced Bonito and Dorado, which are open-source basecaller implementations mostly based on five Long Short-Term Memory (LSTM) layers and a mixed CTC-CRF decoder. Combined with improvements in pore technology, this is the most important driver of increasing accuracy and the next section and Section 3.1 provide more detail on this subject.

2.2.2. Modern basecalling algorithms

Today, the most advanced basecalling algorithms use deep learning techniques. These algorithms are trained on large datasets of known sequences and their corresponding raw signal data, enabling them to recognize complex patterns and make more accurate predictions.

Neural networks, particularly CNNs and RNNs, have revolutionized basecalling. CNNs are adept at handling spatial data and can efficiently process the signal data generated by nanopore sequencing. RNNs on the other hand, are designed to handle sequential data, making them ideal for interpreting the continuous stream of signals produced as DNA or RNA passes through the nanopore.

One prominent example is the Dorado basecaller developed by ONT. Dorado employs deep learning models to perform high-accuracy basecalling, reducing error rates and improving the quality of the

sequencing data. Primarily using LSTM layers to handle sequential data dependencies, it is optimized to handle noisy input data through the use of three convolutional layers. It uses a connectionist temporal classification-conditional random field (CTC-CRF) output head, where the CTC part allows the output sequence of k-mers to be shorter than the input sequence, and the Conditional Random Field (CRF) part ensures dependencies between scores at each position. During inference, a beam search approximates the most probable sequence of k-mers, which is then converted to a nucleotide sequence. Dorado reaches 99.5% raw read accuracy and higher when using the newest flowcells and super accuracy (SUP) model [10, 64].

2.2.3. Challenges in basecalling

Despite advancements, basecalling remains a challenging task due to several factors.

Signal noise The raw signal data generated by nanopore sequencing is often noisy, with various sources of error introduced during the sequencing process, such as electrical interference, sensor drift, and biological variability [55]. High-fidelity basecalling must accurately filter out this noise to produce reliable nucleotide sequences.

Homopolymer regions Sequences with long runs of the same nucleotide (e.g. AAAAAA) can produce indistinguishable signals, complicating the basecalling process. The uniform signal produced by these homopolymer regions makes it difficult for basecallers to determine the exact number of repeated nucleotides, leading to potential errors in sequence interpretation. Especially ONT sequencers suffer from low accuracy in homopolymer regions, and this is an active area of research [46, 68, 75].

Speed and throughput Achieving high accuracy in basecalling is computationally intensive. Balancing speed and accuracy is a significant challenge, particularly in high-throughput applications [72]. Real-time basecalling requires efficient algorithms that can process large volumes of data quickly without compromising the accuracy of the results. Additionally, basecalling sometimes needs to be performed in resource-constrained environments, such as in the field or on portable sequencing devices. These scenarios demand basecalling algorithms that are not only accurate and fast but also optimized for low power consumption and minimal computational resources like memory [66].

Data volume The large volume of data generated by long-read sequencing technologies poses a significant challenge. For example, 30 Gbases of sequencing output, not unlikely for a long sequencing run, results in a POD5 file of around 210 GBytes (see Appendix A for details). Efficiently managing and processing this data without overwhelming computational infrastructure requires advanced data handling and storage solutions.

2.2.4. Impact of basecalling on sequencing quality

The accuracy of basecalling directly impacts the quality of sequencing data. The literature has often described how the constantly improving accuracy of nanopore sequencing stems from equal parts biotechnology (nanopore composition among others) and bioinformatics (primarily basecalling) [45, 86]. Errors in basecalling can propagate through subsequent analyses, affecting everything from variant calling to genome assembly. Therefore, improving basecalling algorithms is a critical area of ongoing research and development.

The future of basecalling lies in further integrating machine learning and artificial intelligence. Continuous improvements in training algorithms with larger, more diverse datasets will enhance the ability to accurately decode even the most complex signal patterns. Additionally, developing hybrid models that combine the strengths of different neural network architectures may offer further enhancements in basecalling performance.

Basecalling is a foundational component of the nanopore sequencing pipeline, converting raw signal data into meaningful nucleotide sequences. Advances in deep learning have significantly improved basecalling accuracy, but ongoing challenges remain. Continued innovation in this area will be crucial for fully realizing the potential of nanopore sequencing technology in various scientific and clinical applications.

2.3. Applications of nanopore sequencing

Nanopore sequencing offers distinct advantages for DNA sequencing, particularly in low-resource environments or situations requiring quick results. Compared to the traditional sequencing workflow, requiring freezers and large sequencing devices in a lab, the fast turnaround time of nanopore sequencing enables new applications to use DNA sequencing. This section outlines various applications of nanopore sequencing, emphasizing its adaptability and efficiency in different contexts.

2.3.1. Application domains

Species analysis in off-grid/remote locations

Nanopore sequencing proves invaluable for species analysis in off-grid or remote areas where traditional sequencing methods may be impractical. When doing species analysis in remote locations, it is not always feasible to take samples, prepare them, and then take them back to the lab. This might be a journey of several days, up to a week. In this case, it is critical to be able to perform sequencing on-site, removing the need to transport samples to distant laboratories, which can be time-consuming and challenging.

For instance, one team used nanopore sequencing during expeditions in the Kabobo biodiversity hotspot in the Democratic Republic of Congo [23]. They aimed to discover new species and study wildlife diversity, utilizing the portability of the technology for on-site analysis. Similarly, Ineke Knot used MinION sequencing in the Sumatran jungle to collect molecular data on parasites threatening orangutans, providing essential insights without needing laboratory facilities [43]. Another study [29] demonstrated the use of solar-powered nanopore sequencing to analyze microbial communities on an ice cap, showcasing complete offline capability from sample preparation to basecalling.

Grid power and an internet connection are usually not available in such remote locations, so it is paramount that all principal steps of the sequencing pipeline can be performed offline. When carrying out multiple sequencing experiments, it is infeasible to store all raw sequencing data, so basecalling is an important step to reduce storage requirements. Furthermore, it should ideally be possible to perform basecalling on solar or battery power, and the bioinformatics pipeline should be easy to use, or one has to bring a bioinformatician on the team as well.

Essential requirements for these applications include a lightweight form factor, low power consumption, and user-friendliness, ensuring sequencing can be performed entirely off-grid.

Wastewater analysis for pathogen detection

Wastewater monitoring and analysis has been standard practice to monitor diseases for a long time, starting in the 1940s, and has gained more traction during the Covid-19 pandemic [16]. However, conventional methods require taking samples on-site, and transferring those to a clinical facility. This increases the time to result, and lowering this latency can help react to emergency situations and guide policymakers. For some applications, it is possible to generate the first results within one hour (Ebola) to six hours (pathogens in ICU patients) [62].

Research underscores the potential of nanopore sequencing in wastewater surveillance, enabling quicker responses in public health crises, such as the COVID-19 pandemic [25]. Moreover, the UN Environment Program promotes wastewater surveillance using nanopore sequencing as a cost-effective disease monitoring method, especially in areas lacking formal sewage systems or laboratory facilities [92]. In these cases, being able to do sequencing in the field is even more important than fast results.

Key requirements for wastewater analysis include speed, simplicity, and low power consumption, with a preference for a stand-alone form factor.

Low-cost/volume cancer research

The capability to provide rapid, high-resolution genomic data makes portable sequencing a valuable tool for early cancer detection and personalized treatment. However, rural areas generally lack large hospitals or medical centers with the expertise and funds for traditional sequencing technologies: “a major disadvantage of traditional short-read genome sequencing technology has been the need for high capital investment, which resulted in sequencing infrastructure being located in dedicated

sequencing centres. Consequently, the shipment of samples can become the most time-consuming step of an investigation, rather than the sequencing or analysis work itself. Research shows that such limitations can be overcome by using portable sequencing technology, such as the MinION." [62]

Patients who can benefit most from long read sequencing are rural and remote populations. Research shows that these demographics have shorter life expectancies and higher disease mortality rates [78]. Being able to bring more accurate diagnosis and variant specific treatment to more and more people is a key benefit of portable long-read sequencing. Specific to cancer is the ability to benefit from the long reads that ONT and PacBio can provide, since detecting cancer often relies on mutations in one of multiple repetitions in DNA. Therefore, long read sequencing with devices like the MinION can benefit both remote and poor communities that do not have access to traditionally big and expensive machines [8].

Key requirements for cancer research include speed, accuracy, and ease of use, with a stand-alone form factor preferred for deployment in diverse settings.

2.3.2. Requirements

Successful sequencing experiments depend on many different factors, and the basecalling step depends mostly on storage, compute, and energy requirements. Furthermore, some applications depend on the portability and ease-of-use of equipment.

Energy usage The MinION uses just 5 W of power and has a very small form-factor, making it ideal for experiments outside the lab. To facilitate this end-to-end, the energy use of basecalling must be contained within acceptable levels, which is not the case for the current state-of-the-art.

Throughput In general, users want to perform basecalling simultaneously with sequencing, and this has been a supported (and encouraged) workflow by ONT for quite some time now. The MinION outputs around 1.5–1.6 Msamples/s on average, so the basecaller/computer hardware combination must allow for this. Empirical results show that basecalling with the high accuracy (HAC) model requires the hardware to perform around 2–3 TFLOPS. Appendix B compares several systems ranging from server hardware to mobile phones and NVIDIA Jetsons.

Accuracy Most downstream tasks require high accuracy basecalling to be of any use. In practice, this means that users will want to use the HAC or SUP models, and the Fast model is only useful for preliminary species identification or similar tasks. This requirement, combined with the throughput to keep up with sequencing, currently presents the largest challenge in nanopore sequencing.

Storage Although storage in the form of solid-state drives (SSDs) is becoming cheaper by the year, the storage size of (POD5) sequencing output adds up over time. A single MinION run of 72 hours can add up to 400 GB of raw data and basecalled sequences (see Appendix A for details). When on an expedition, researchers may prefer to perform sequencing and basecalling simultaneously, but keep raw sequencing output for re-basecalling with the SUP model afterward as well. Over multiple experiments, data requirements easily add up to terabytes of storage.

3

Efficient basecaller architectures

3.1. Overview

In the past decade, ONT has created a plethora of tools surrounding the nanopore sequencing pipeline, many of which are basecallers. Since the history of basecallers has been discussed in detail before [55, 65], we will focus on recent developments here. From ONT's side, Bonito is the open-source research basecaller that has been in development since 2020 and is often used as comparison material by independent researchers. Since August 2023, the production basecaller in the sequencing toolkit has changed to Dorado [63], which is a from-scratch C++ implementation of a high-performance basecaller and aligner, among other things [10]. Dorado's neural network architecture is the same as Bonito's.

While products like the MinION are promoted by emphasizing their portability, it is generally not feasible to basecall sequencing data without powerful GPUs [66]. Notably, ONT heavily invests in optimizing the Dorado basecaller for NVIDIA's datacenter GPUs like the A100, exploiting specific architectural improvements that do not apply to older or cheaper GPUs [17]. This is contradictory to the goal of portability, since it is not generally possible to take a bulky and power-hungry server along on expeditions where power and space are scarce [29].

This section discusses the neural network that is at the core of basecallers and why this network is so computationally expensive. Section 3.2 reviews previous work on fast and energy-efficient basecallers, and most importantly, some high-level approaches to reducing the computational burden of basecalling are presented in Section 3.3.

The neural network

The production basecaller Dorado and research and development basecaller Bonito share the same neural networks and architectures to allow training and researching models that can directly be used by Dorado. This section will discuss these networks, referring to Bonito, but this applies to Dorado as well. In general, there are three model architectures that all are trained on different sets of data for different flowcells, sequencing configurations, etc. These are the Fast model, the high accuracy model (HAC), and the super high accuracy model (SUP). Users can choose between these for a different trade-off between basecalling throughput and accuracy.

As of June 2024, the SUP model transitioned from the LSTM-based model to a new transformer-based network, improving on many accuracy metrics within the same compute budget. However, the Fast and HAC models remain LSTM-based. Figure 3.1 illustrates the overall architecture of these networks, consisting of a number of convolutional layers, five LSTM layers, and finally the mixed CTC-CRF decoder. The LSTM stack uses a 'flip-flop' architecture, reversing after each layer. Since the LSTM stack takes around 87.5% of the compute time during basecalling (see Figure 2.2), this research focuses on optimizing the size and throughput of these layers.

Comparing the Fast and HAC models, the only difference is the size of the LSTM layers: for the Fast model, the LSTM input and hidden sizes are both 96, while the HAC sizes are 384 for both. While the

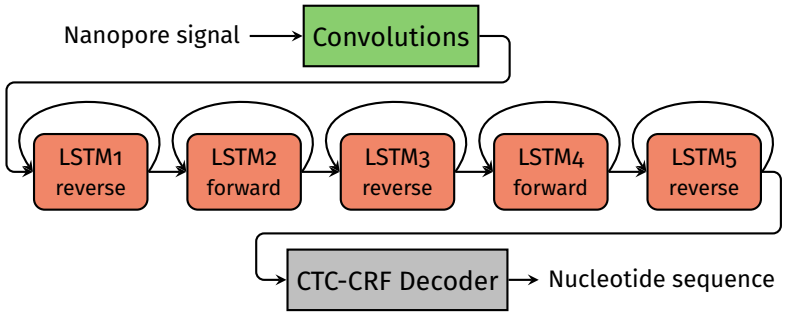


Figure 3.1: Architecture of the Fast and HAC Bonito models

HAC model is more accurate than the Fast model, this brings an 8× increase in compute requirements. As a result, the Fast model is usable on a wide range of hardware, while the HAC model requires a high performance GPU to get even close to real-time basecalling. Because of this, we consider a reduction in the LSTM stack without a large compromise in accuracy a viable way to optimize basecalling.

3.2. Previous work

Basecaller design has been a research topic since the first commercially available nanopore sequencing device, with Nanocall being the first open-source basecaller to be developed [20]. At that time, ONT's basecalling software did not yet use neural networks, but used a HMM instead. Nanocall uses a pore model to estimate the 6-mer that is in the pore at a given time – nowadays this would be a 5-mer with newer pore technology. While this method is highly efficient, being able to keep up with a MinION sequencer using only 4 CPU cores, its 68% match rate is poor compared to the current basecallers that reach match rates of 99% and up.

Most research on nanopore basecallers has focused on using different neural network types and architectures, with the goal of creating a basecaller that is more accurate than ONT's own basecaller, i.e., Bonito or one of its predecessors [40, 44, 54, 94]. Pagès-Gallego and de Ridder [65] provide a great overview of all prominent basecallers developed until 2023, and benchmark them on many different metrics, with the models being trained on the exact same datasets [65]. However, all discussed basecallers focus on accuracy and do not consider basecalling speed to be an important metric.

A few works have nonetheless attempted to develop efficient basecallers that can run on a battery-powered laptop, for example, while keeping the accuracy at an acceptable level when compared to the state-of-the-art. The rest of this section discusses fast or efficient basecallers DeepNano-blitz [12], DeepNano-coral [66], Guppy on Jetson Xavier NX [31], and its hybrid edge/cloud successor [30]. Furthermore, recent research explored the possibilities of pruning basecallers [77]. However, in this work only single-shot pruning and training is tested, and conclude that this approach does not lead to significant savings in compute without sacrificing accuracy. This work is discussed at the end of this section, together with some examples of previous work on accelerating (short-read) genomics algorithms.

DeepNano-blitz

DeepNano-blitz is one of the first basecallers focused on low-power hardware, without a GPU. To get the highest throughput possible, DeepNano-blitz uses a small network consisting of a single convolutional layer and a maxpool layer followed by four 64-cell wide GRU layers in alternating direction, ending with a softmax layer and a CTC decoder to convert the predictions into bases. The GRU layers and CTC decoder perform the bulk of the work and were also used by Bonito at that point. The architecture is much smaller than commonly used networks that stack multiple convolutions and have wider layers. Furthermore, to get the most out of the hardware, the implementation uses cache-aware memory layouts and approximations for the sigmoid and tanh functions.

The paper evaluates multiple different configurations of the model, of which the most accurate model that can keep up with live basecalling from a MinION sequencer is around 2% below Guppy's accuracy. When using a slightly slower configuration, DeepNano-blitz's accuracy is the on par with Guppy's while

still being 3× as fast.

DeepNano-coral
The second basecaller by the same authors is DeepNano-coral, using a CNN-based architecture that is focused on power efficiency through the use of the Core Edge tensor processing unit (TPU). It achieves real-time basecalling of a single MinION device using just 10 W of power. By shrinking and re-engineering the convolutional layers of Bonito and replacing the recurrent layers with a different type of convolution, it is possible to run the entire network, in real-time, on the Coral TPU. This method uses about half the energy that Guppy uses [66].

As is the case with the previous DeepNano basecaller, the model is evaluated in different configurations. The most accurate version that can keep up with sequencing output, defined as 1.5 M samples/s, is on par or better than Guppy with the fast model. However, about 4% fewer reads of the tested human dataset are successfully mapped, which suggests that difficult to basecall reads are skipped and therefore excluded from the accuracy evaluation.

Guppy on Jetson Xavier NX
Grzesik and Mrozek [31] take a different approach, opting to use existing basecallers on low-power edge hardware. Because of its on-board GPU, the Jetson Xavier NX from NVIDIA is one of the few single-board computers (SBCs) capable enough to run basecallers like Guppy. While other devices with an integrated or on-board GPU exist on the market, this is one of the few with enough memory to run the Kraken 2 classification software. This is not as relevant for basecalling, but an important consideration when selecting hardware with the goal of running the whole sequencing pipeline.

Impressively, the Jetson Xavier NX is able to basecall 3.8 M samples/s in its lowest power configuration, easily processing about two MinION sequencing outputs in real-time. However, when the HAC model is required, throughput peaks at 480 k samples/s. It is important to note that by using the stock Guppy basecaller, accuracy is as good as one can get and users benefit from improvements in basecalling accuracy (and speed) that ONT provides in the future.

Hybrid edge/cloud basecalling
The same authors extend their edge analysis in later work with a hybrid edge/cloud setup to offload work to the cloud when network is available [30]. They consider full edge processing to be too expensive, and propose offloading processing parts of the data to a cloud service. While the premise is interesting, at the network speeds tested (up to 512 kB/s), cloud computing cannot solve the problem simply because nanopore sequencers generate so much data.

While the total time to process a dataset is reduced by 15-20% in ideal conditions, the dependence on a stable and sufficiently fast network cannot necessarily compete with the simple workflow of fully offline processing. Although a higher-speed internet connection could significantly reduce the processing time, this requirement does not align with the expected infrastructure at more remote locations. Nowadays, however, a Starlink connection might be a viable option, providing 5–25 Mbps upload bandwidth [81], but this does mean bringing extra equipment on expeditions. Lastly, especially for independent researchers, setting up cloud infrastructure for 15% time savings seems like a skewed trade-off.

Rubicon
The Rubicon framework by Singh et al. addresses the challenge of efficient basecalling through a framework optimizing for hardware. The paper introduces Rubicall, the first hardware-optimized mixed-precision basecaller, outperforming current models in throughput. The framework uses QABAS, a quantization-aware basecalling architecture search framework, and SkipClip, which removes skip connections without accuracy loss [77].

The framework generates a mixed-precision neural network architecture, with bit widths as low as 2 or 3 bits. While the paper reports impressive speedups of 3.8× over previous work like Dorado Fast, it requires the use of specialized hardware that supports these mixed-precision operations. On a GPU, reported basecalling throughput is about 15× slower than Dorado with the Fast model. Furthermore, in the paper, a study into pruning is performed, but for both structured and unstructured pruning,

the authors are not able to obtain a significant speedup without severely impacting accuracy. This is an important point, and is discussed at length in Chapter 4 and we propose a solution to fix this accuracy drop in Chapter 5.

Short-read algorithms
There is also a large body of research that uses dedicated hardware accelerators for genomics algorithms [1, 37, 38]. These algorithms can be effectively accelerated on hardware due to their dataflow nature [2]. However, the accelerated algorithms are classical algorithms commonly based on dynamic programming [69] such as Smith-Waterman or probabilistic analyses [70], which is different from the LSTM-based approach used in Bonito.

Conclusion
The evolution of efficient basecallers highlights the difficult trade-off between model complexity, speed, and accuracy. Basecallers such as DeepNano-blitz and DeepNano-coral demonstrate high efficiency and competitive accuracy on low-power hardware, showcasing implementation approaches like cache-aware memory layouts and the use of the Coral Edge TPU. Despite their simpler architectures, these models achieve high throughput with minimal energy consumption, making them suitable for resource-constrained environments. However, basecalling accuracy lags behind when compared to the state-of-the-art.

Conversely, leveraging existing high-accuracy basecallers on advanced low-power devices, such as using Guppy on the Jetson Xavier NX, presents another effective strategy. This method balances power efficiency with high basecalling accuracy, demonstrating the potential of advanced hardware in improving computational performance. However, this approach does not lead to savings in compute. While hybrid edge/cloud basecalling offers intriguing possibilities, its current limitations due to network constraints highlight the ongoing challenge of optimizing both accuracy and efficiency in basecaller design.

3.3. Reducing the computational burden

Although many strategies exist to reduce the computational burden of basecalling, we argue that optimizing the existing Bonito models is the best way forward, for a number of reasons. First, the convolution-LSTM-CRF architecture is a proven model architecture. Over the years, a large body of research has attempted to improve over the state-of-the-art basecallers, but the LSTM has never disappeared. Second, ONT provides some support for research into basecallers via the Bonito repository, which is up-to-date with the state-of-the-art Dorado basecaller [76]. This makes it easy to keep developments in line with new implementations in Bonito.

In this section, three possible optimization avenues are explored: quantization, to represent weights in fewer bits, changing the model size and architecture, and finally pruning. This is the process of removing (redundant) weights from the model to save space and reduce compute.

3.3.1. Quantization

In machine learning, quantization refers to the process of reducing the precision of the numbers used to represent a model's parameters (weights) and/or inputs and outputs [49]. This is typically done to optimize the performance of models in terms of memory usage and computational efficiency, making them more suitable for deployment on resource-constrained devices like mobile phones and embedded systems. GPUs have increasingly added support for higher throughput compute in lower precision, e.g. INT8 (an 8-bit integer). Recently, a broad group of manufacturers standardized microscaling formats, including an FP4, that will find support in upcoming hardware [71]. Combined with the order(s) of magnitude higher floating-point operations per second (FLOPS) that these devices can perform using tensor cores [61], quantization is a viable way to accelerate basecalling throughput without sacrificing accuracy.

Types and techniques
Quantization can be categorized into several types based on how the precision reduction is applied. Uniform quantization involves mapping continuous values to a fixed number of uniformly spaced

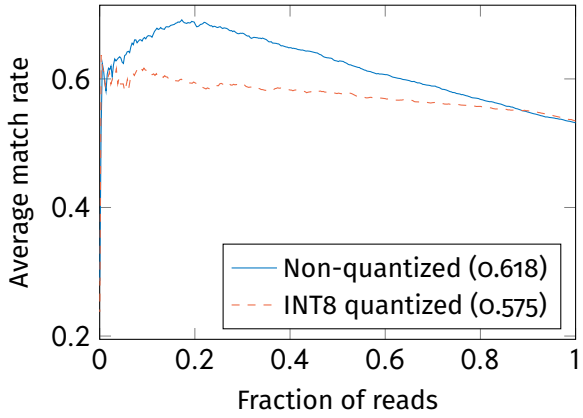


Figure 3.2: Average match rate and AUC of reads sorted by decreasing PhredQ score

levels, making it straightforward to implement and widely used. Non-uniform quantization uses varying step sizes between levels, optimizing precision in value ranges that occur most frequently, thus improving accuracy where it is needed most [11]. Dynamic quantization applies quantization only during inference, allowing the model to maintain higher precision during training and switch to lower precision for deployment. Conversely, static quantization quantizes both weights and activations during training, ensuring the model operates at lower precision throughout both training and inference [11, 49]. Each type has its specific use cases and advantages, tailored to different deployment scenarios and performance requirements.

Second, quantization techniques focus on how and when quantization is applied to a model. Post-training quantization involves quantizing a pre-trained model, which is simpler to implement but can lead to a drop in accuracy due to the abrupt precision reduction [49]. On the other hand, quantization-aware training incorporates quantization during the training process itself, allowing the model to learn and adapt to the lower precision throughout training. This approach generally results in better performance and accuracy retention compared to post-training quantization, as the model parameters are fine-tuned with the quantization constraints in mind from the beginning. Each technique offers different trade-offs in terms of complexity, accuracy, and ease of deployment, and the choice between them depends on the specific requirements and constraints of the application.

Quantizing Bonito

To test the possibilities of quantization for basecalling specifically, we compare baseline Bonito-HAC to the same model, with the LSTM layers dynamically quantized post-training to INT8. Firstly, the match rates are nearly the same at 52.2%, which is unexpected due to the loss of precision. However, this is partially explained by the difference in pass rates: the baseline has a pass rate of 95.8% compared to 86.3% for the quantized model. This indicates that the quantized model might basecall some reads so badly, that they cannot be aligned; then, these reads are not considered when calculating the match rate.

The area under the curve (AUC) results in Figure 3.2 show that while the average match rate over all reads is the same for both models, the downward slop of the non-quantized model indicates that it is able to distinguish (in)correct reads and assign a quality score accordingly. In contrast, the curve of the quantized model is much flatter and shows little decay from the ‘good’ to the ‘bad’ reads – according to the model. For downstream applications, the PhredQ score can be used by algorithms for alignment or other tasks, but if the average score is nearly the same for both correct and incorrect reads, it is not a useful measure.

Post-training quantization is not ideal, as the model is not adjusted to the reduced precision of calculations. This is acknowledged by the data in Figure 3.2, and using quantization-aware training should result in much closer AUC curves. However, since many successful examples of quantization in machine learning and specifically basecalling exist [77], this is not explored any further.

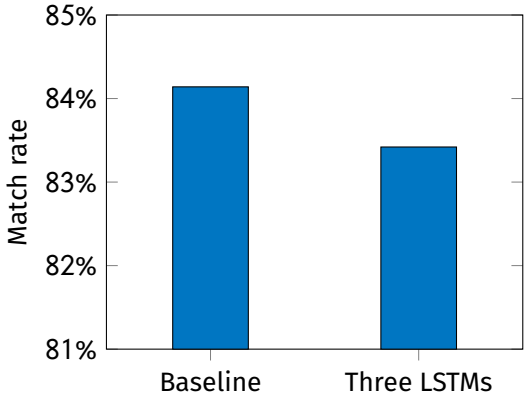


Figure 3.3: The original Bonito model with 5 LSTM layers compared to a smaller 3 LSTM layer version

State-of-the-art
Quantization has been a topic of much research for a long time. These days, many neural networks used in production employ quantization [49]. Dorado is also quantized to a large extent: depending on the input and hidden sizes, the LSTMs are quantized to FP16 or INT8 wherever hardware supports it. This is a form of dynamic quantization because full precision is used during training. Interestingly, the basecalling match rate is not noticeably reduced due to quantization. In fact, it is suspected that quantization in general can improve model accuracy due to better generalization over the dataset, when using static quantization [49]. Because a model is trained in lower precision as well, it learns to classify noisy and less accurate data. Since Dorado can run INT8 quantized LSTMs, and previous work has explored quantization to much lower bit representations [77], this thesis will not go into more detail on quantization.

3.3.2. Model size and architecture
One of the best ways to increase a model's performance, be it in terms of latency or throughput, is to reduce the model's size. By changing the architecture to only three LSTM layers for example, 40% of the compute in the LSTM stack is removed, leading to a 25% reduction in total compute. Additionally, by making the CTC-CRF output decoder less complex, some compute can be saved there as well.

Another approach is to reduce sizes of the layers, as is done for Bonito Fast. Essentially, Fast and HAC are the same model, with one main exception: the final convolution output size and all LSTM input and hidden sizes are reduced from 384 to 96. Since memory requirements are reduced accordingly, the batch size can be much higher as well. This is reflected by an increase in basecalling throughput of 5–6× [33].

To quantify the effect of the aforementioned reduction in LSTM layers, Figure 3.3 illustrates the change in match rate. As can be seen, when trained on the same data for the same number of epochs, the final match rate ends just 0.7% below Bonito HAC for the three layer version. Additionally, the removal of two layers results in an increase in basecalling throughput of around 35%, matching our expectations since the LSTMs are the main bottleneck.

Transformer architecture
With the rise of transformer-based [83] large language models [13], it is only natural to perform some research into adapting the LSTM-based model to use attention layers. Two basecallers that use attention exist: SACall [40] and CATCaller [54]. However, both basecallers cannot keep up with Bonito in terms of accuracy [65] and place no focus on speed and throughput, being orders of magnitude slower than Dorado.

The intuition is that attention can improve basecalling accuracy, allowing for a smaller model to settle at the same accuracy as the LSTM-based model. However, as previous work has shown [40, 54], this is non-trivial. Drawing inspiration from speech-to-text techniques [7, 14, 39] and transformer models aimed at sequence-to-sequence tasks [18, 26], our experiments have been unsuccessful at achieving accuracy results comparable to Bonito.

Since May 2024, Bonito's SUP model uses FlashAttention [19] layers instead of the LSTM stack. ONT claims [21] that using the same amount of compute, the new model achieves higher accuracy across the board: higher match rate, lower homopolymer error rates, etc. However, at lower compute budgets suitable for Fast and HAC models, the LSTMs remain competitive and so these models are unchanged.

3.3.3. Pruning

Pruning is a model optimization technique used to reduce the size and complexity of neural networks by removing parts that are deemed unnecessary. This can be applied to single weights (connections between neurons), neurons, blocks of weights, or even complete layers [36, 49]. In the context of basecalling, pruning can significantly enhance computational efficiency and reduce memory usage without substantially compromising accuracy. By identifying and eliminating redundant or less impactful parts of the network, pruning helps streamline the model, making it faster and more suitable for deployment on resource-constrained hardware [58].

Model compression through pruning not only accelerates inference times but also lowers the overall resource demands, facilitating real-time sequencing applications. The following chapter, Chapter 4, provides an overview of various pruning methods and their specific applications and benefits in basecalling through experiments.

3.3.4. Selecting a method

To reach the goal of reducing the computational burden of basecalling and specifically, the LSTM stack, three methods have been presented in this section. While a first investigation shows promising results in each direction, we attempt to push the boundaries on just a single front, in the interest of time.

Quantization excels at reducing the overparameterization in neural networks: if a neuron can only be in two states, a 1-bit number is able to represent that state. In theory, quantization can provide enormous reductions in the compute requirements, but implementations heavily lean on field-programmable gate array (FPGA) and custom hardware platforms. GPUs do have more and more support for quantization, but bit widths of less than INT8 are not well-supported at the moment in data center GPUs, let alone consumer ones. The lack of GPU support, combined with previous work [77] approaching the limits of low bit width representations, means that quantization is not a viable research direction for this thesis.

Changing the neural network's architecture is the option that most previous work has opted for. However, after about ten different papers each trying something different, the LSTM-based network is still the standard, and this is not expected to change. Furthermore, improving the throughput and accuracy using a different network architecture requires in-depth knowledge of machine learning, training strategies, and access to high-quality training data.

Pruning, on the other hand, offers versatile approaches to optimizing basecallers, particularly in reducing the compute requirements of the LSTM stack that dominates compute. The lack of standardized pruning approaches and research into pruning LSTMs and RNNs in general highlights a scientific gap that can be closed in the process. Unlike quantization, pruning does not rely heavily on specialized hardware, allowing for broader applicability across various platforms, including standard GPUs. Moreover, pruning can be implemented with a relatively shallow understanding of neural network architectures, making it accessible for researchers and engineers with limited machine learning expertise. Given these advantages, pruning emerges as a practical and achievable method for enhancing the efficiency of basecalling neural networks in this thesis.

4

Pruning

Pruning (deep) neural networks is the process of removing redundant weights (connections) or neurons from a given network. Pruning received a lot of interest in the past decade, starting with the research of Han et al. who propose the three step iterative pipeline of training, pruning, then fine-tuning [34]. While their work results in a large reduction in the number of connections, it does not focus on neuron (or structured) pruning, thus preventing effective reduction in computing needs. This is explained in more detail in Section 4.1.1. Furthermore, this and many subsequent works focus on CNNs for computer vision tasks [49], whose concepts in general cannot be directly applied to RNNs which operate on sequences.

This chapter presents an exploration into pruning Bonito's neural network. Section 4.1 discusses the general strategies of unstructured and structured pruning, as well as two specific methods. The results of the latter are presented in Section 4.2.

4.1. Methods

Many different pruning strategies exist for different layer types, network architectures, hardware architectures, etc. Since the LSTM stack takes up the majority of the computation time (87.5% for the HAC model), we focus primarily on reducing the size of the LSTM layers. These can help achieve the highest level of compression but mainly the highest speedup.

For theoretical savings, unstructured magnitude pruning is the simplest strategy available: take the x% smallest weights and remove them. L2 pruning can achieve higher accuracy but requires retraining [34]. However, these methods introduce unstructured sparsity, which is non-trivial to accelerate. Block pruning exploits speedup via block sparsity, which can be hard to achieve but is definitely possible. Neuron pruning does not induce sparsity but instead removes entire rows/columns from the weights matrices because the whole neuron and its connections are removed. This method can highly affect accuracy as not just unimportant weights are removed, but weights with a large magnitude can be pruned as well.

4.1.1. Unstructured pruning

Unstructured pruning is a simple method for reducing the number of nonzero weights in a model [34]. It follows a typical three-step process: first, a large model is trained; second, the weights with the smallest magnitudes are pruned, effectively setting them to zero; and finally, the model undergoes fine-tuning to recover any lost accuracy. The final two steps can be repeated with smaller pruning rates to retain more accuracy. This process essentially removes connections or synapses between neurons from the model, as illustrated by Figure 4.1.

The downside of this method is that it introduces random and unstructured sparsity in the weight matrices, which makes it nontrivial to speed up training and inference on current computer hardware. Conventional math and tensor libraries cannot take advantage of a low degree of unstructured

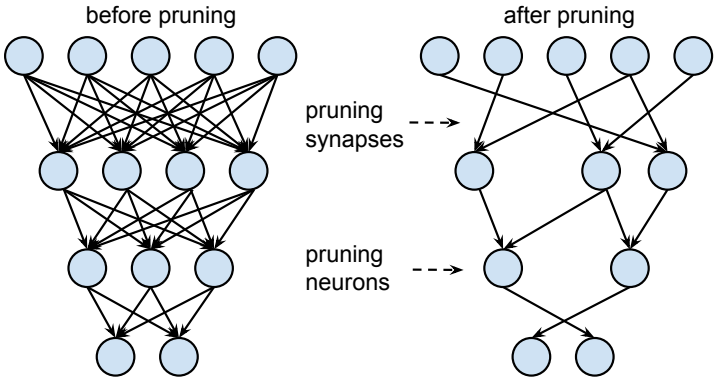


Figure 4.1: Visual explanation of pruning synapses (weights) and neurons. Image from [34].

sparsity: most require sparsity levels of 99% and above to reduce the required computations [36]. At lower sparsity levels, the overhead associated with managing and accessing sparse data can outweigh the benefits, making the approach less feasible for deep neural networks.

Unlike dense matrix-matrix operations, sparse matrices suffer from irregular data access patterns and poor temporal and spatial locality [90]. For neural networks, unstructured sparsity is not well-supported on today's hardware architectures [36]. 90% sparse workloads perform slower than computing zeros in a dense workload in one example on GPU [35] and another example, on a CPU, shows that an 89% sparse neural network performs 25% slower [93]. Implementations for unstructured sparsity at lower levels of sparsity are not mature enough for production use and real-world speedup.

4.1.2. Structured pruning

While unstructured pruning methods that simply prune individual weights can be applied to LSTMs and other RNNs, these are of limited use and a shift towards structured pruning is needed. The process involves evaluating the importance of each neuron which can be based on several indicators, like the magnitude of its associated weights, its effect on the output of a layer or the whole network, and adding some influence on the value of the loss function [36]. Some previous work focuses on mathematically reducing the size of a network by monitoring neuron activity [74] or using new criteria to identify the optimal pruning rate [28]. Wang et al., focusing on an FPGA implementation, prune the columns of weight matrices with the smallest L_1 norm [84]. Another FPGA-based method uses a compression technique with block-circulant matrices instead of sparse matrices [85].

This chapter mainly deals with magnitude-based pruning, and as such, an exploration of magnitude-based columnar pruning methods is presented here. Similarly to unstructured pruning, it is possible to perform an iterative process of training, pruning, and fine-tuning. This approach is easy to implement in frameworks like PyTorch and gives a good indication of the possible savings in model size. Furthermore, columnar pruning, or neuron pruning, directly allows faster and more efficient inference [36].

Our method is closely aligned with [84] with the major difference being the shift from FPGA to GPU. The pruning step is as follows: a critical value C_w determines that a column should be pruned away if the sum of its elements is smaller than C_w. Furthermore, weights that are kept are multiplied by a factor based on the critical value and the sum of the column. This ensures that output magnitudes do not reduce by too much. The results of the columnar pruning exploration in Bonito are presented in Section 4.2.

Neuron selection

The downside of magnitude-based structured pruning is that many of the weights are removed at the same time. While a model's accuracy can be state-of-the-art before pruning, it is often hard to obtain accuracy close to the original after pruning, even with fine-tuning. Wen et al. present intrinsically sparse structures (ISS) for LSTMs [88], one of the first works proposing to learn structured sparsity in RNNs. Later research adapts this method for GRUs [52], which is then improved by introducing the concept of neuron selection to structurally shrink the size of LSTM layers [87]. This approach treats

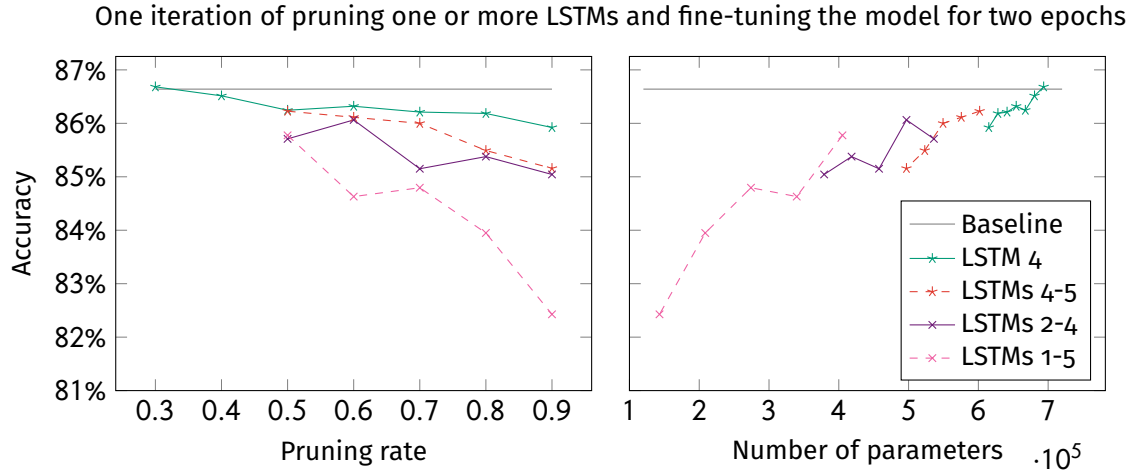


Figure 4.2: Results of pruning one or more LSTM layers in Bonito. The left plot shows the final accuracy by pruning rates, highlighting that pruning more layers by the same rate impacts accuracy. The right plot shows the accuracy compared to the number of nonzero parameters remaining in the model after pruning.

structured sparsity as an attribute that can be turned on or off for each input and hidden neuron in the LSTM layer. This selection mechanism is built into the LSTM as a mask and thus, the sparsity can be learned like any other parameter [87]. Using a penalty hyperparameter the model can be tuned to achieve the desired sparsity/accuracy trade-off. The neuron selection mechanism is adapted and improved on in Chapter 5.

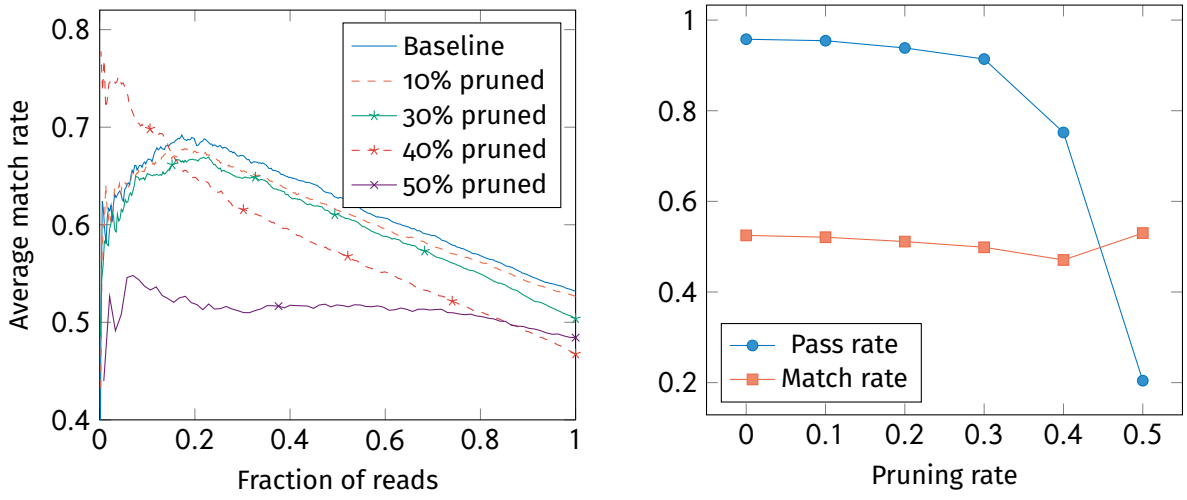
4.2. Results

To get an idea of the obtainable level of sparsity, we follow the L_1 regularization procedure originally described in [34] to prune a pre-trained Bonito-HAC model by zeroing out the weights with the smallest magnitude, until the desired sparsity percentage is reached. Then, the resulting model is fine-tuned for two epochs while keeping the 'pruned' weights set at zero. Additionally, we test the columnar pruning method described earlier.

Unstructured pruning The results of this pruning and fine-tuning are shown in Figure 4.2, with different lines corresponding to pruning different sets of layers. For each configuration, we prune those layers by some percentage, shown in the left plot. Second, we align the results by the number of nonzero parameters left in the model, shown in the right plot. This illustrates a clear, coherent downward trend as the number of parameters is reduced.

While we do not fine-tune the model over the whole dataset for the same number of epochs as the original model has trained, as described in [34], the pattern is clear: pruning is an effective way of decreasing the number of parameters with a minimal decrease in accuracy, i.e. about 1.5% when pruning the middle three layers by 90%. As mentioned before though, unstructured sparsity is not suitable for speedup.

Structured pruning The results for the columnar pruning method are shown in Figure 4.3. Especially the match rates, that remain close to the baseline up to 30% pruning rates, are a positive sign that pruning is a viable optimization method. However, the stark decline of the pass rate, which is how many of the reads can be aligned after basecalling, signals that the model struggles to keep up with difficult reads. Notably, after slowly decreasing over higher pruning rates, the match rate ends up increasing to above baseline levels at a pruning rate of 50%. As noted before, this suggests that difficult reads are skipped after alignment while easier reads are successfully basecalled.



(a) Average match rate of reads sorted by decreasing PhredQ score

(b) Pass rates and match rates showing the knee point and steep dropoff at 0.4 pruning rate

Figure 4.3: Results of the structured columnar pruning method. These results are direct from basecalling after pruning; there is no fine-tuning performed in this case.

5

Learning structured sparsity

5.1. Introduction

The fundamental problem with most pruning methods is the lack of structured sparsity. Intuitively, an element-wise operation on a matrix that is 50% sparse, should take 50% less floating-point operations (FLOPs) compared to a fully dense matrix. Unfortunately, conventional math and tensor libraries cannot take advantage of a low degree of unstructured sparsity, as discussed in Section 4.1.1.

Dorado and other basecallers rely on DNNs for their high accuracy. Over the years, these networks have become larger and more complex, requiring large, powerful and expensive GPUs to keep up with sequencing output. Figure 2.2, presented before in Section 2.2, illustrates that about 43% of the time in the nanopore sequencing pipeline is spent on basecalling, with nearly 90% of this time dedicated to computing the LSTM layers. This hampers field deployment because, while mobile sequencing is very much possible, analyzing the data is not.

The size of many DNNs is prohibitive to deployment on resource-constrained devices due to their high memory requirements and the substantial number of operations needed to process inputs. Furthermore, the energy required to process DNNs is typically well above the limits of mobile devices [34]. Neural networks are often over-parameterized, resulting in significant redundancy in DNNs [24]. Research indicates that it is often possible to prune at least half of the parameters without affecting accuracy, and with an effective retraining strategy, up to 90% of parameters can be removed without significantly impacting accuracy. Recent work [77] explored the possibilities of pruning basecallers specifically, but the authors conclude that this approach does not lead to significant savings in compute without sacrificing accuracy. However, their approach is a single-shot pruning and fine-tuning method, which is known to be suboptimal [36].

This work proposes a pruning approach that is highly effective in reducing the size and computational complexity of DNNs and apply it to basecalling models. Results show that the required number of operations can be reduced by 21× and throughput is increased by 2.4×, while maintaining accuracy within 1% of the baseline model, Bonito-HAC. Pruning neurons decreases the number of calculations needed and the storage size on both disk and in memory, reducing the size and energy requirements of the basecalling pipeline.

In Chapter 4, the limitations of the pruning and fine-tuning pipeline have been discussed. This chapter introduces the concept of neuron selection in Section 5.2 and the addition of a delayed masking scheduler is presented. Alternative pruning and model compression techniques in general are discussed in Section 5.3. These methods did not work well for our application and have been discarded. For the benchmark results as well as a basecalling speed comparison and ablation study, refer to Chapter 6.

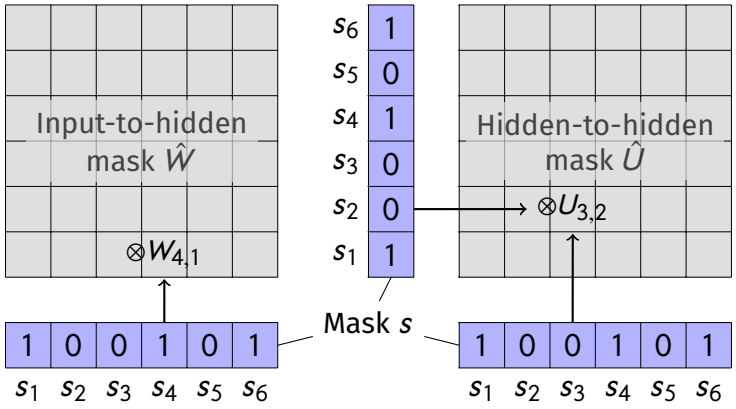


Figure 5.1: Illustration of the masking mechanism

5.2. Neuron selection

The principal optimization method used in this work is structured pruning of LSTMs through neuron selection, proposed originally by Wen et al., which regularizes a neural network by adding a mask over the neurons and penalizing nonzero mask entries [87]. By making this penalty part of the loss function, the network can learn the weights for accurate basecalling as well as which neurons are required to maximize the accuracy, at the same time. In practice, this requires a custom LSTM implementation and thus training is slower than using standard highly optimized LSTM implementations.

After training, the model can be compressed and inference is much faster than the original model, by up to 20× for a simple language processing experiment. Unfortunately, the authors do not provide benchmarks on larger or more difficult tasks. Furthermore, our implementation is, to the best of our knowledge, the first adaptation and implementation of [87].

5.2.1. Method

By introducing a set of binary random variables, which can be interpreted as switches for individual neurons, it is possible to structurally prune the LSTMs through *neuron selection*. Let $W = \{W_i, W_f, W_o, W_c\}$ and $U = \{U_i, U_f, U_o, U_c\}$ be the input-to-hidden and hidden-to-hidden weight matrices, respectively. Furthermore, let mask $s = \{s_i\}$, which controls the presence of the hidden neuron i, where $s_i \in \{0, 1\}$ and $|s|$ is the number of hidden neurons, or the 'hidden size'. While training, the weight matrices are masked by 'turning off' rows and columns when the input or output neurons are masked with s. This gating mechanism is visualized in Figure 5.1. For convenience, the original matrices W and U are re-parameterized to $\hat{W} = W \odot 1s^\top$ and $\hat{U} = U \odot ss^\top$ where 1 is the all-ones vector.

Conceptually, the binary random variable s is sampled from a Bernoulli distribution with $s_i \sim \text{Bern}(\pi_{s_i})$ where π_{s_i} denotes the probability of the random variable s_i taking value 1. However, it is impossible to simply minimize its L_0 norm by gradient optimization, since the Bernoulli distribution is not differentiable. By remodeling to a hard concrete distribution [53], a modification of the concrete distribution [56], we obtain the following procedure to obtain mask variable s:

$$\begin{aligned} u &\sim \text{Uniform}_{[0,1]}, \\ \hat{\tau} &= \sigma((\log u - \log(1-u) + \log \alpha)/\beta), \\ \tau &= \hat{\tau}(\zeta - \gamma) + \gamma, \\ s &= \min(1, \max(0, \tau)), \end{aligned} \tag{5.1}$$

where $\hat{\tau}$ is a sample from the concrete distribution with location and temperature parameters $\log \alpha$ and β, respectively, which is then stretched to the (ζ, γ) interval and clamped in a hard-sigmoid fashion. Note that instead of optimizing α, $\log \alpha$ is directly optimized. This results in a function that is differentiable and can therefore be implemented in and optimized by machine learning frameworks with autograd like PyTorch and Tensorflow. The probability of s being non-zero, which is used in the

loss function, is then computed by the cumulative density function Φ:

$$\Phi_s(s \neq 0|\phi_s) = \sigma(\log\alpha - \beta\log\frac{-\gamma}{\zeta}), \tag{5.2}$$

where $\phi = \{\alpha, \beta, \gamma, \zeta\}$ denotes the parameters of the hard concrete distribution.

5.2.2. Optimizing training time

The original method from [87] imposes an additional mask z over the input neurons of each LSTM layer. However, in our experiments we find that none or only a couple of input neurons get pruned in practice when the penalty is small. This can be attributed to the fact that there are 4096 possible 5-mers that could be in the nanopore[1]. Pruning away the input neurons removes too much state information from the model, which has a large impact on the model accuracy. This effect is observed in the original paper as well, but the authors do not suggest any remedy except for lowering the penalty value for the input neurons. However, this does not lead to better model compression but instead results in keeping all input neurons for our model.

Since the neuron selection mechanism is not implemented in the standard LSTM implementations in PyTorch, we resort to a handwritten implementation in Python, with each tensor multiplication or other operation explicitly written out. This results in highly inefficient training due to overhead, data movement, and lack of parallelism. Therefore, by removing the input neuron mask and its random sampling and preprocessing, training is more efficient. This adjustment decreases the training time by 10–20% while maintaining the effectiveness of the neuron selection mechanism.

Without the extra mask z, the loss function can be simplified a bit. Given that $||x||$ is the input size, the loss and objective functions are formulated as follows:

$$\begin{aligned}\mathcal{L}(W, U, \phi) = \mathbb{E}_{\Phi(s|\phi)}[\mathbb{E}_D(W, U, s)] \\ + \lambda \sum_i (||x|| \cdot \Phi_{s_i}(s_i \neq 0|\phi_{s_i})) \\ + \lambda \sum_{i,j}^{i\neq j} (\Phi_{s_i}(s_i \neq 0|\phi_{s_i})\Phi_{s_j}(s_j \neq 0|\phi_{s_j})) \\ + \lambda \sum_i \Phi_{s_i}(s_i \neq 0|\phi_{s_i})\end{aligned} \tag{5.3}$$

$$(W^*, U^*, \phi^*) = \arg\min_{W,U,\phi} \mathcal{L}(W, U, \Phi) \tag{5.4}$$

5.2.3. Delayed masking

Another new contribution is the addition of a warm-up scheduler to prevent the model from collapsing at the start of training. We find that at high penalty rates, e.g. $> 3 \cdot 10^{-7}$, the autograd optimizer is eagerly pruning the model to reduce the value of the loss function, but this happens so quickly that the model does not have enough time to learn patterns in the data at all. The result is that the model is pruned to a hidden size of (nearly) 0, with the accuracy also sitting at 0%. This effect is a result of the model's performance during training: as can be seen in Figure 5.2 (top), the loss value stays consistently high at 1.4–1.5 until dropping off after 6000 training steps. It is at this state that the model grasps the structure of the nanopore squiggle, as the validation accuracy jumps from 0 to about 70% in subsequent steps.

We propose a penalty scheduler to prevent this issue from happening at larger λ penalty values. By delaying the masking to a later moment, we give the network time to train and learn before more aggressively turning off neurons. For example, using a step function from a modest $0.5 \cdot 10^{-7}$ at the start, to λ's intended value at 10000 training steps (see Figure 5.2) keeps the model at about 90% of its original size for the first part of training. After this time, the full penalty value is used and the

[1]There are always 5 bases in a nanopore at the same time, each base can be one of 4, so there are $4^5 = 4096$ total possible states.

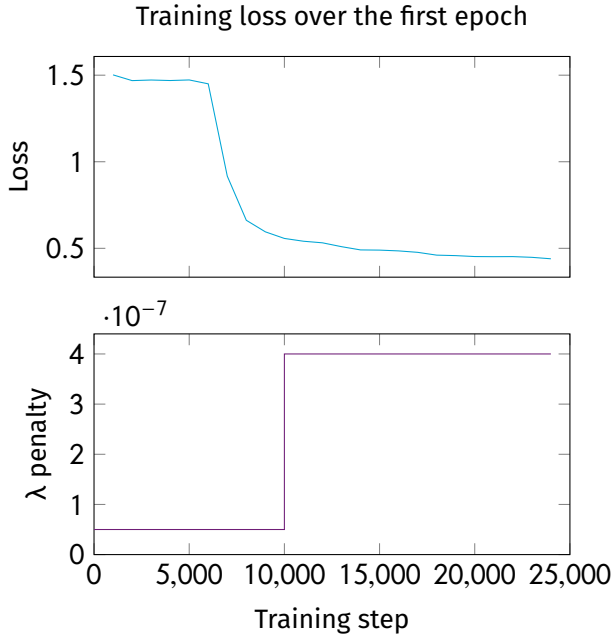


Figure 5.2: Top: training loss during the first epoch. After 6000 steps, the loss value sharply declines in a few thousand steps. When λ is too large, the model size decreases so much within this time, that it is not able to find patterns in the data at all. Bottom: the proposed delayed masking schedule ensures that a baseline accuracy has been established before starting to mask more and more of the model.

model is masked much more aggressively. While we use a step function with a fixed step moment in this paper, more elaborate schemes using linear or sigmoid functions and a dynamic step moment can be tested in the future. The penalty warm-up is a novel contribution that we believe can benefit any application that is learning structured sparsity in a neural network.

5.2.4. Efficient inference

During training, the weight mask is recalculated with every forward step, with each neuron being (de-)activated based on a random sample from its entry in s. As the neurons are 'selected' over the course of the training steps, the probability of a neuron being active is generally not truly zero, to allow correction based on the output accuracy. Consequently, no practical speedup is obtained over the complete starting model. When training is complete, the model can be structurally pruned by removing rows and columns that are all zeros from $\hat{W}$ and $\hat{U}$, as well as the bias vector. Then, these weights can be used for a plain LSTM layer with a reduced hidden size of $||s||_0$.

To motivate and quantify the benefit of smaller LSTM layers, consider that the computational intensity is dominated by the hidden size: the approximate number of FLOPs per batched LSTM step is $N \times D \times (I + D) \times 8$ FLOPs[2] where N is the batch size and I and D are the input and hidden size, respectively. An LSTM with an input and hidden size of 384, like Bonito's, therefore requires 2.4M FLOPs for a single timestep with batch size 1. If the hidden size is reduced to around 120, as shown in Table 6.1 with $\lambda = 6 \cdot 10^{-7}$ for example, the inference step requires just 484k or 0.2× the FLOPs. This reduction of the hidden size D for the LSTM's $O(D^2)$ computational complexity results in a substantially more efficient model.

The layers in the LSTM stack are placed back-to-back or output-to-input, meaning that a reduction of the hidden size in one layer can be directly transferred to the next layer's input size. If the same mask is used, the model outputs are exactly the same, but there is 'free lunch' for speeding up the model and this is actively exploited in our experiments.

[2] $\times 4 \cdot 2$ because the LSTM has 4 gates that each operate on the input, and a multiplication and addition have to be performed for each element.

5.3. Evaluation of alternative pruning methods

In the process of enhancing the efficiency of nanopore basecalling models, we evaluated several alternative pruning methods proposed in the literature. Despite their promise, the methods discussed in this section did not perform as well in our specific application as reported in the original papers. For example, the Selfish-RNN sparse-to-sparse training method proposed in [50] did not perform nearly as well as the standard dense training method. Using the implementation provided by the authors, the final match rate was 10–13% lower than the baseline. This could be due to the model being small from the start, without being trained on the data at full size first.

The C-LSTM method involves compressing LSTM networks using block-circulant matrices, which significantly reduce the number of parameters and computational complexity [85]. The approach uses block-circulant instead of sparse matrices to compress weight matrices and reduces the storage requirement from $O(n^2)$ to $O(n)$ and Fast Fourier Transforms to reduces the computational complexity from $O(n^2)$ to $O(n \log n)$. Although this technique demonstrated impressive results on FPGA implementations, it did not translate well to our GPU-based environment. The block-circulant structure introduced challenges in adapting the model to our specific basecalling tasks, and we were not able to successfully train a model.

A linear surrogate of the LSTM cell using linear recurrence [57] is a promising replacement for the standard LSTM cells in the LSTM layers. As a precursor to linear state-space models like S4 [32], it changes the iterative nature of recurrent models to a parallel scan algorithm. This parallel linear recurrence can be efficiently computed by parallel solvers like GPUs and thus appears to be a good alternative. For our basecalling application, however, experiments show that the accuracy did not get close to the baseline accuracy. Similar to Selfish-RNN, accuracy was about 9–11% below baseline.

Despite the initial promise of these methods, our experiments highlight the importance of domain-specific optimization. The unique challenges posed by nanopore basecalling, such as electrical noise and highly variable sequencing speed, necessitate pruning techniques that can maintain high accuracy while efficiently reducing computational requirements. This suggests that not all sparsification methods are created equal, and that one can not directly apply a single best method to any application. As such, our focus shifted towards structured sparsity and neuron selection, which proved to be most effective for our purposes.

6

Experiments and results

This chapter presents several experiments on the model with masked LSTM layers through neuron selection. Section 6.1 and Section 6.2 present the experimental setup and datasets used, respectively. In Section 6.3, the accuracy of basecalling is compared among the Fast and HAC models, as well as our method over a number of λ penalty values. The corresponding FLOPs savings and throughput increases are presented in Section 6.4. Furthermore, an ablation study is performed in Section 6.5 and the chapter ends with a discussion in Section 6.6.

6.1. Experimental setup

Hyperparameters The model is trained with nearly the same hyperparameter settings as in [65], using the Adam optimizer with an initial learning rate of 1.5e-3, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay = 0. The learning rate is linearly increased for the first 5000 training steps from 0 to the initial optimizer rate as a warm-up, then decreased using a cosine function to a minimum of 1e-5 by the final step. The batch size is doubled from 64 to 128, and the initial learning rate is increased accordingly from 1e-3 to 1.5e-3. Gradients are clipped between -2 and 2 to improve model stability. All standard LSTM parameters are initialized from $\mathcal{N}(0, 0.05)$.

For the neuron selection mechanism, we initially choose $\lambda = 2 \cdot 10^{-7}$ with smoothing parameter set $\phi = \{\beta = 2/3, \gamma = -0.1, \zeta = 1.1\}$. α is a learnable parameter directly trained as $\log \alpha$, initialized from $\mathcal{N}(1.5, 0.1)$ which corresponds to a 90% probability that a neuron is active on average. Since the model contains no dropout layers, the dropout keep ratio does not have to be increased.

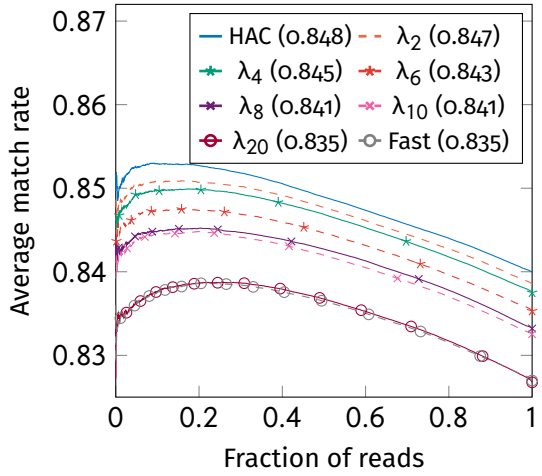
Computational Environment The experiments are run in an HPC environment [22] on nodes with Xeon E5-6448Y 32C CPUs @ 2.10 GHz and an NVIDIA A100 80GB PCIe GPU. In this environment, a full training run of 5 epochs takes 23.5 hours to complete. We use Python 3.9.8 with PyTorch 1.12.1. To deterministically find the throughput numbers in Figure 6.3, we disable the cuDNN backend in PyTorch, as that may (non-deterministically) choose different algorithms based on batch size, model size, free memory, etc. that can drastically impact performance. Other dependencies and versions are listed in the code repository on GitHub[1].

6.2. Datasets

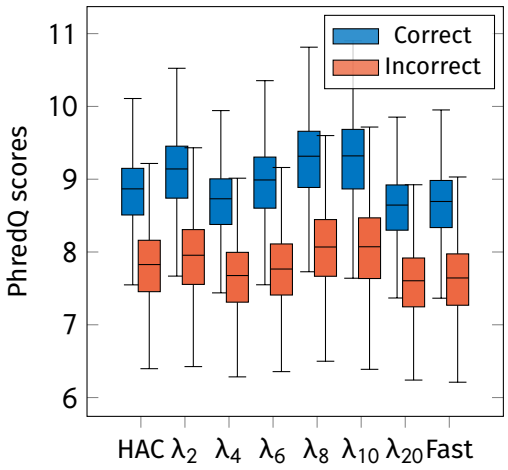
The training, evaluation, and test datasets are the same benchmark datasets from [65], in which the authors propose a standardized benchmark for nanopore basecaller evaluation. We use the cross-species task for training and evaluation of all models. Therefore, our results are not directly comparable to [65], focusing solely on the human task.

The dataset consists of 3 human datasets from [42], 1 Lambda phage dataset sequenced in-house [65], and 60 bacterial datasets encompassing 26 different bacterial species originally published in

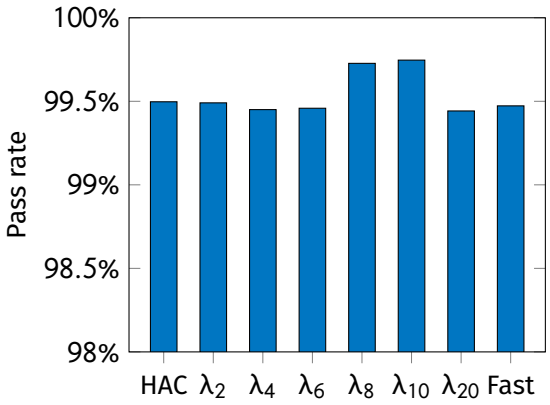
[1]The code is available on `https://github.com/meesfrensel/efficient-basecallers`



(a) Average match rate and AUC of reads sorted by decreasing PhredQ score



(b) Separation of PhredQ scores of correct and incorrectly called reads



(c) Overall pass rates

Figure 6.1: Benchmark results showing Bonito Fast and HAC models, and our work for different λ penalties

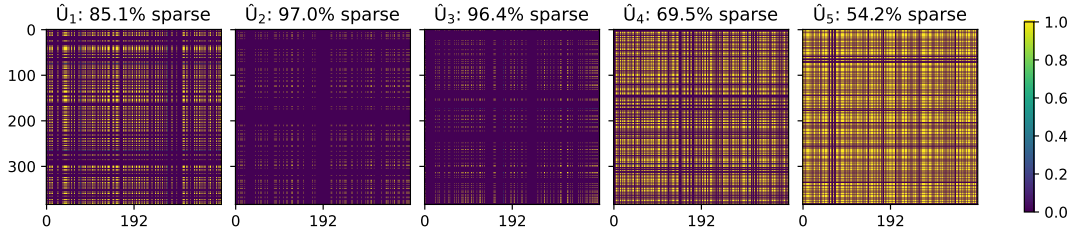


Figure 6.2: Illustration of the masks of the hidden-to-hidden weight matrix $\hat{U}$ for each LSTM layer in the λ_4 model

[89]. All data is sequenced using R9.4 or R9.4.1 pore chemistry. Notably, performance improvements in newer pore chemistries, such as the current R10.1, would similarly enhance our results, reflecting the ongoing advancements in nanopore sequencing technology.

6.3. Basecalling accuracy

For evaluating our work, we reference the benchmark established in [65], which addresses the need for standardized benchmarking in nanopore sequencing, where basecalling accuracy is crucial and often improved through new neural network architectures. Due to varying evaluation metrics and datasets across different publications, it has been challenging to differentiate between data-driven and model-driven improvements. By using this comprehensive benchmark, we ensure fair and consistent comparisons of our basecaller with state-of-the-art models, thereby validating our performance claims.

The primary results are shown in Figure 6.1, highlighting the performance of Bonito fast, HAC, and our method with different pruning rates denoted by λ_x for $\lambda = x \cdot 10^{-7}$. The AUC (Figure 6.1a), with reads sorted by decreasing PhredQ score, shows that all models have a clear correlation between read quality and average match rate. This conclusion is backed up by the high amount of separation in the PhredQ quality scores (in Figure 6.1b), allowing downstream applications to use this as a measure of certainty that a base is correct. Bonito-HAC performs best with an AUC of 0.848 and our pruned model with $\lambda = 2 \cdot 10^{-7}$ following closely behind with an AUC of 0.847. The match rates differ by just 0.13%, while the number of FLOPs required is decreased by 2.2×.

One key observation is the nearly identical results of the λ_{20} and Fast models. The AUC plots in Figure 6.1a overlap to the point that they are indistinguishable apart from the slightly different AUC's of 0.835 and 0.839 for λ_{20} and Fast, respectively.

Pass rates are an important measure against 'cheating': basecalling models could achieve higher match rates by skipping reads that are harder to basecall correctly [65]. However, as can be seen in Figure 6.1c, the models with higher pruning rates do not sacrifice the pass rate for accuracy. This is important but not straightforward, as many experiments with prune/fine-tune methods resulted primarily in very low pass rates.

Overall, the trend is that at higher pruning rates all metrics tend to suffer somewhat equally: we do not observe that pruning affects some metric more than others. This is both positive and negative, as on one hand smaller models could be useful for users who do not care about that one metric, allowing them to save on compute. On the other hand, being able to play with the penalty value to get just the precision you need at the least amount of compute is useful for users who do not have a data center at their disposal.

6.4. Basecalling throughput

Smaller models are theoretically faster at basecalling than large models. As explained in Section 5.2.4, the number of FLOPs for a single inference step is mostly dependent on the hidden size, and the input size has some influence as well. Our method's focus is on reducing the layer's hidden size, which has an $O(n^2)$ effect on the computational complexity. By reducing the hidden size with 1/3, the number of FLOPs decreases by 2.8×, while the accuracy is barely affected. Furthermore, by allowing

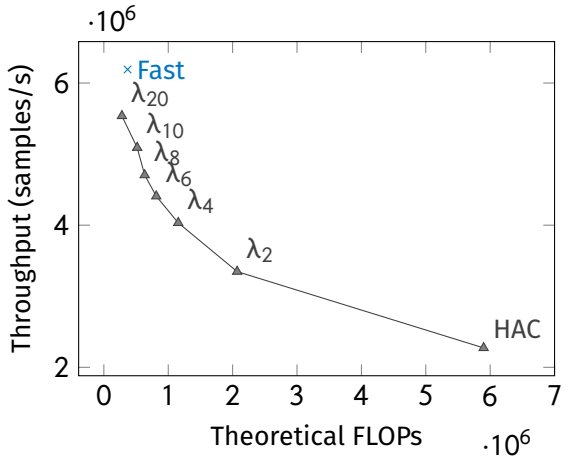


Figure 6.3: Theoretical FLOPs for the LSTM stack compared to the basecalling throughput of the whole neural network

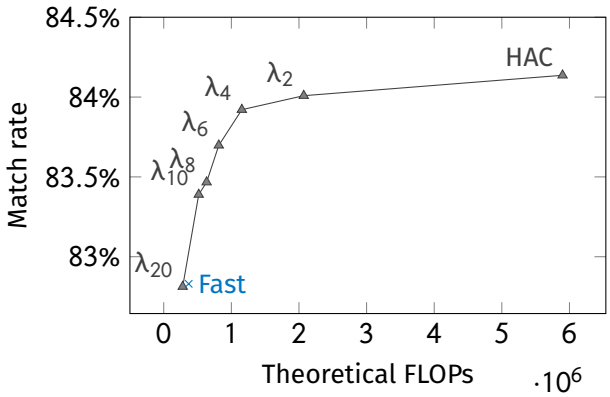


Figure 6.4: Results of the ablation study, with λ_{20} as closest model to Bonito Fast

the match rate to drop by 0.4%, it is possible to reduce the LSTM stack's theoretical FLOPs by 5.1×, and this trend is shown in Figure 6.3 as well.

The possible gains from reducing the hidden size are illustrated with the mask $\hat{U}$, as shown in Figure 6.2. Since we induce structured sparsity, we can 'physically' remove neurons and the corresponding rows/columns from the model. During training, which neurons are important, is automatically decided by backpropagation, and during inference, we benefit by reloading the LSTM cells with smaller hidden sizes. This results in faster inference during basecalling, improving energy usage and throughput.

The resulting throughput speedup achieved with structured sparsity depends on many factors besides the number of FLOPs, which includes batch size, sequence length among others. Figure 6.3 shows the average throughput of the whole neural network (see the architecture in Figure 3.1) for different models, with constant batch size and sequence length. The curve shows the direct relationship between the theoretical gains and real throughput increase: at λ_2, throughput is already 1.5× higher than the baseline. This trend continues with all subsequently smaller models.

Notably, the fastest model λ_{20} is 2.4× faster compared to the HAC baseline and requires 24% less FLOPs compared to the Fast model. However, because of the constantly larger overhead of the convolutional layers and the CTC-CRF decoder, which are not pruned with our method, the λ_{20} model is not able to achieve higher throughput than the Fast model. Instead, the Fast model reaches 2.7× and 1.1× higher throughput than HAC and λ_{20}, respectively.

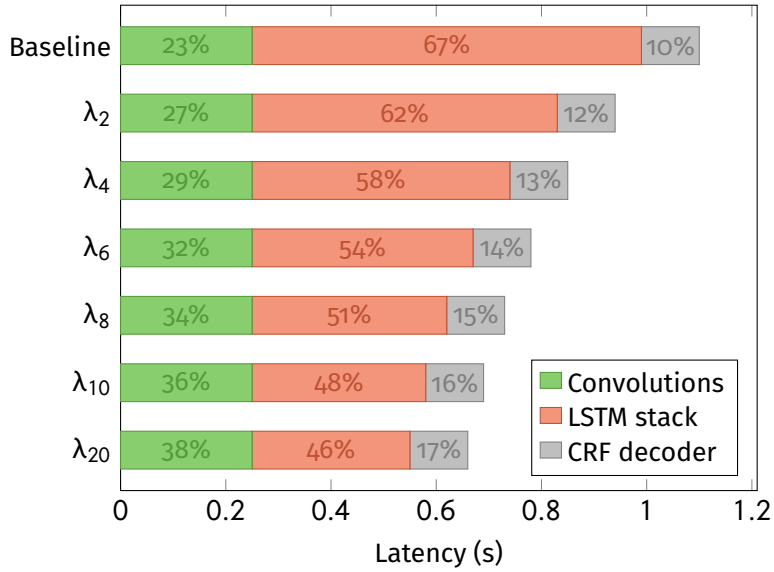


Figure 6.5: Increasing the λ penalty parameter results in smaller and faster models, which is reflected in the Bonito profile timelines shown here. Bonito's suboptimal convolution implementation takes 23% of the time compared to 3.5% for Dorado.

6.5. Ablation study

To thoroughly evaluate the impact of structured pruning on our basecalling model, we conduct an ablation study focusing on varying λ values, which control the neuron penalty during training. Another way to think about this, is how sensitive the model is to varying λ penalty values. This study is essential to understand the trade-offs between model size, throughput, and read accuracy. Our findings are illustrated in Figure 6.4 summarized in Table 6.1.

For $\lambda = 2 \cdot 10^{-7}$, we observed no significant change in basecalling accuracy compared to the baseline model, while achieving a 2.8× reduction in FLOPs. This indicates that a moderate sparsity penalty can effectively prune the network without compromising performance. Increasing the penalty to $4 \cdot 10^{-7}$, the match rate decreases by a further 0.2% while compressing the model to a third of its original size. This level of pruning strikes a good balance between maintaining accuracy and enhancing efficiency.

At higher penalty values, such as $\lambda = 8 \cdot 10^{-7}$, the model experiences a more noticeable drop in accuracy (around 0.7%), but the model FLOPs are drastically reduced, by 9.3× in the LSTM layers. This suggests that while aggressive pruning can significantly decrease computational requirements, it may also impact the model's performance. These results highlight the importance of selecting an appropriate sparsity penalty to balance the trade-offs between model efficiency and basecalling accuracy, which is similar to selecting a basecalling model from Bonito/Dorado's Fast, HAC or SUP models.

Comparing the throughput results to the baseline can help understand the effects of pruning, and outline the limit of speedup. Figure 6.5 illustrates the change in the profile timeline previously shown in Figure 2.2. Bonito has slightly different ratios between the convolutions, LSTM stack, and CRF decoder, with the LSTM stack taking around 67% of the time in the baseline HAC model. This explains the relatively limited speedup overall: although the LSTM stack takes 60% less time, in this benchmark example the overall reduction is just 40%. Because the GPU kernels are synchronized between each model layer to obtain consistent timings, this is much less than the previously reported 2.4× speedup. The important thing to note is the significant reduction in the part that the LSTM stack plays, from 67% to 46%.

Overall, the ablation study underscores the potential of structured pruning to optimize nanopore basecalling models, making high-quality sequencing more accessible for field applications. The results show the great performance benefit of learning which neurons should be active at the same time as learning the connection weights between them. Between Bonito's Fast and HAC models, we consider $\lambda = 6 \cdot 10^{-7}$ the best trade-off between accuracy and speed, measuring just 0.44% below

λ penalty	Match	Hidden sizes	FLOPs	Improvement
Baseline	84.14%	384, 384, 384, 384, 384	$5.90M$	
$2 \cdot 10^{-7}$	84.01%	213, 126, 119, 291, 324	$2.07M$	2.8×
$4 \cdot 10^{-7}$	83.92%	149, 68, 74, 212, 261	$1.16M$	5.1×
$6 \cdot 10^{-7}$	83.70%	111, 48, 55, 179, 229	$0.81M$	7.2×
$8 \cdot 10^{-7}$	83.47%	96, 39, 48, 155, 201	$0.63M$	9.3×
$10 \cdot 10^{-7}$	83.39%	82, 36, 38, 133, 190	$0.52M$	11.4×
$20 \cdot 10^{-7}$	82.81%	55, 22, 21, 88, 145	$0.28M$	21.0×
Fast	82.83%	Input & hidden size: 96	$0.36M$	16.0×

Table 6.1: Ablation study of the penalty parameter of the LSTM layers

HAC's match rate while achieving a $5.1\times$ reduction in FLOP count, ending nicely in between Bonito's models.

6.6. Discussion

We propose learning structured sparsity to make basecalling neural networks substantially more efficient while maintaining high accuracy. This makes nanopore sequencing faster, cheaper, and more accessible by using less compute resources. Moreover, our results show the extent to which the state-of-the-art models are over-parameterized when considering their accuracy. As previous works on efficient basecalling rightly point out, the ever-increasing accuracy, and as a result, size of these models does not outweigh the implications of resulting compute requirements for everyone. Moreover, available power and compute can be a decisive variable in selecting a basecalling model.

Aside from pruning, the other most common approach to model compression, be it for storage or compute savings, is quantization. By representing numbers with fewer bits or even as integers, one can save space and utilize specialized hardware to do these lower precision computations. Mixed precision math is widely used in Dorado and the Rubicon paper performed an in-depth analysis [77] that can be readily integrated with our work, as quantization is orthogonal to pruning. One limitation of our work is the scope: by reducing only the LSTM layers, the LSTM stack's input size is fixed, the convolutional layers cannot be adjusted, limiting the overall reduction in parameters and FLOPs. We therefore recommend further research into whole-model structural pruning, in combination with quantization.

7

Conclusions and recommendations

This thesis investigated bottlenecks in the nanopore DNA sequencing pipeline, specifically in the basecalling step, and identified the five LSTM layers as the primary restriction in fast and efficient basecalling. This finding essential to the subsequent approach. Furthermore, three possible approaches to reduce the impact of basecalling have been explored, which answers the second research question. Quantization, a new model architecture, and pruning are different (and orthogonal) methods to reduce the compute required for basecalling. Previous work has investigated quantization and model architectures at length, so we explored pruning in much more detail.

A new pruning approach is presented to enhance the efficiency of DNNs through the use of structured sparsity in the LSTM layers, and the approach is applied to nanopore DNA basecalling models. By leveraging neuron selection techniques to prune redundant parameters, we can significantly reduce the computational and memory requirements of the basecalling models without compromising accuracy. The approach adds a mask on the LSTM's hidden neurons, that can be turned on or off during training. By making the number of active neurons part of the loss function, the size of each LSTM layer is reduced more and more until it would impact accuracy.

Results show that the second-to-smallest model is 11.4× more efficient and 2.1× faster, with a reduction in match rate of just 0.8% compared to Bonito-HAC, answering the third research question. This advancement is particularly beneficial for field deployments of portable sequencing devices like the MinION, enabling efficient basecalling on resource-constrained devices. Our approach facilitates DNA analysis in various field applications, by allowing users to select an appropriate trade-off between accuracy and basecalling time for their demands and available resources. There is ample room to maneuver this spectrum, with an efficiency increase between $2.8 - 21\times$ and a corresponding accuracy drop of 0.1–1.3% compared to the baseline Bonito-HAC model. Future work will explore further structured optimization techniques and better delayed masking schedulers to further improve the performance and efficiency of basecalling models.

Apart from this, a closer look at the results reveals that the presented pruning method is particularly fit for smaller pruning rates, i.e. up to λ_4. For a $5.1\times$ reduction in FLOPs, match rate drops by just 0.2%. When going beyond this knee point, the match rate does drop off faster than before, although the accuracy is still only reduced by 0.8% at λ_{10}. In general, this suggests that for higher pruning rates, some modifications might be required to retain the maximum possible accuracy. A couple of ideas are an improved penalty scheduler, different or auto-adjusting penalty values per layer, and changing the learning rate based on the number of nonzeros in the mask.

Furthermore, this thesis hasn't considered pruning the other layers in the neural network. For example, the final convolution layer outputs a feature size of 384 into the first LSTM. It is not unthinkable that this feature size can be reduced at larger pruning rates (for higher λ values), shrinking the overall compute requirements even more. The same holds for the output of the last LSTM layer: although it is shrunk to less than half its original size in the smallest models, the tensors have to be 'upscaled'

to fit into the CTC-CRF decoder. This is done by inserting zeros but should be done by shrinking the input layer of the CTC-CRF decoder.

The advent of long-read sequencing has transformed our knowledge of genomics, and with that, our understanding of genetic diversity, evolutionary biology, and the underlying mechanisms of genetic disorders. Today, researchers are using nanopore long-read sequencing to assist surgeons during cancers surgery and to investigate biodiversity all over the world. Increasing the efficiency of basecallers improves accessibility of this technology, and as data and algorithms improve, it is expected that neural networks can be compressed even more. On the other hand, practical long-read sequencing is just a decade old, and it will be interesting to see what biotechnological improvements to pore technology and sequencing hardware can bring over the next decade(s). For now, basecalling is the largest step in the nanopore sequencing pipeline, but it is likely that highly accurate and deterministic sequencing hardware shifts the pipeline's center of gravity towards assembly and polishing over time.

A

Storage requirements

Flow cell output (Gbases)	POD5 size (GBytes)	FASTQ.gz size (GBytes)	Unaligned BAM with modifications (GBytes)	Sum
Flongle: 2.6	18.2	1.69	1.56	21.45 GB
10	70	6.5	6	82.5 GB
15	105	9.75	9	123.75 GB
30	210	19.5	18	247.5 GB
MinION: 48	336	31.2	28.8	396 GB
5x MinION: 240	1200	156	144	1.5 TB

Table A.1: Example file sizes are based on different throughputs from an individual flow cell, with a run saving POD5, FASTQ, and BAM files with a read N50 of 23 kb. Theoretical maximum output of a MinION is 48 Gb when doing 400 b/s for 72 hours, with a Flongle adapter the theoretical maximum output is 2.6 Gb when doing 400 b/s for 16 hours. Data is an extension from the MinION IT requirements.

B

Comparison of systems

As a reference for the FLOPs numbers provided throughout this thesis, this appendix presents a comparison of many systems that are eligible to perform basecalling. Considering that the inference over a single time step of a single sample requires around 0.3 MFLOPs, basecalling the MinION's output of 1.5M samples/s should cost around 0.5 TFLOPS. Due to imperfect implementations and added overhead of the convolutional layers and CTC-CRF decoder, in practice devices with around 2–3 TFLOPS of performance can do live basecalling with the Fast model and about 30–35 TFLOPS is required for HAC basecalling.

Table B.1 lists details of many different platforms that could be suitable for basecalling. Considering that most (somewhat) low-power platforms do not have enough performance to perform live HAC basecalling motivates the need for more efficient neural networks.

System	Form factor	Power	Performance	Fast	HAC
RTX 2080 Ti	Server	350 W	107.6 TFLOPS[1]/ 215.2 TOPS[1]	35 / 71	3 / 7
Laptop with dedicated GPU	All in one	90–200 W	11.6 FLOPS / 100 TOPS[1]	3 / 33	- / 3
Laptop with M.2 Axelera	All in one	38–108 W	100 TOPS	33	3
Laptop with M.2 Hailo[2]	All in one	38–108 W	26 TOPS	8	-
Laptop with M.2 Blaize[3]	All in one	37–107 W	16 TOPS	5	-
Laptop with M.2 Coral[4]	All in one	34–104 W	8 TOPS	2	-
Laptop with USB Coral[5]	USB device	32–102 W	4 TOPS	1	-
Snapdragon 8 Gen 3 (NPU)	Phone/tablet	10–15 W	34 TOPS	11	1
A17 Pro (neural engine)	Smartphone	10–15 W	35 TFLOPS	11	1
Tensor G3 (edge TPU)	Smartphone	10–15 W	8 TOPS	2	-
Jetson AGX Orin	Stand-alone	15–60 W	275 TOPS	91	9
Jetson Orin NX	Stand-alone	10–25 W	100 TOPS	33	3
Jetson Orin Nano	Stand-alone	7–15 W	40 TOPS	13	1
Jetson TX2	Stand-alone	7.5–15 W	1.3 TFLOPS	1[7]	-
Jetson Nano	Stand-alone	5–10 W	0.472 TFLOPS	-	-

TOPS always INT8, FLOPS always FP16.

[1] Tensor-TOPS and -FLOPS

[2] Hailo-8 M.2 AI Acceleration Module, 26 TOPS with unknown power usage

[3] Blaize AI Xplorer X1600E EDSFF PCIe 3 x4, 16 TOPS at 7 W

[4] Coral dual Edge M.2 TPU, 8 TOPS at 4 W

[5] Coral USB Edge TPU, 4 TOPS at 2 W

[6] Based on RTX 4060 Mobile which has a TDP of 115 W

[7] MinION Mk1C uses TX2 internally and is able to basecall live [9]

Table B.1: Comparison of some 'AI' systems, specifically for form factor, peak power usage and peak performance. For real-time basecalling with Fast model, about 3 TFLOPS is enough. High accuracy basecalling in real-time requires 30–35 TFLOPS. The Fast and HAC columns list the theoretical number of sequencing outputs the system can keep up with.

Bibliography

[1] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. 2019. GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data. *BMC Bioinformatics*, 20, 1, (Oct. 2019), 520. DOI: 10.1186/s12859-019-3086-9.

[2] Nauman Ahmed, Vlad-Mihai Sima, Ernst Houtgast, Koen Bertels, and Zaid Al-Ars. 2015. Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. (Nov. 2015), 240–246. DOI: 10.1109/ICCAD.2015.7372576.

[3] Mian Umair Ahsan, Qian Liu, Li Fang, and Kai Wang. 2021. NanoCaller for accurate detection of SNPs and indels in difficult-to-map regions from long-read sequencing by haplotype-aware deep neural networks. *Genome Biology*, 22, 1, (Sept. 2021), 261. DOI: 10.1186/s13059-021-02472-2.

[4] Mohammed Alser et al. 2021. Technology dictates algorithms: recent developments in read alignment. *Genome Biology*, 22, 1, (Aug. 2021), 249. DOI: 10.1186/s13059-021-02443-7.

[5] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology*, 215, 3, (Oct. 1990), 403–410. DOI: 10.1016/S0022-2836(05)80360-2.

[6] Shanika L. Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E. Ritchie, and Quentin Gouil. 2020. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, 21, 1, (Feb. 2020), 30. DOI: 10.1186/s13059-020-1935-5.

[7] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. arXiv:2006.11477 [cs, eess]. (Oct. 2020). Retrieved Jan. 19, 2024 from http://arxiv.org/abs/2006.11477.

[8] Timour Baslan, Sam Kovaka, Fritz J Sedlazeck, Yanming Zhang, Robert Wappel, Sha Tian, Scott W Lowe, Sara Goodwin, and Michael C Schatz. 2021. High resolution copy number inference in cancer using short-molecule nanopore sequencing. *Nucleic Acids Research*, 49, 21, (Dec. 2021), e124. DOI: 10.1093/nar/gkab812.

[9] Miles Benton. 2021. GPU basecalling ONT data. (Nov. 2021). Retrieved Feb. 20, 2024 from https://hackmd.io/@Miles/HJUnkIeOK.

[10] Mark Bicknell. 2023. Dorado — the future of basecalling. Section: Bioinformatics tool. London, (May 2023). Retrieved Apr. 18, 2024 from https://nanoporetech.com/resource-centre/london-calling-2023-dorado-future-basecalling.

[11] Tijmen Blankevoort. 2020. Enabling power-efficient AI through quantization. (Apr. 2020). Retrieved Mar. 7, 2024 from https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/presentation_-_enabling_power-efficient_ai_through_quantization.pdf.

[12] Vladimír Boža, Peter Perešíni, Broňa Brejová, and Tomáš Vinař. 2020. DeepNano-blitz: a fast base caller for MinION nanopore sequencers. *Bioinformatics*, 36, 14, (July 2020), 4191–4192. DOI: 10.1093/bioinformatics/btaa297.

[13] Tom Brown et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*. H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors. Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[14] Junkun Chen, Mingbo Ma, Renjie Zheng, and Liang Huang. 2021. MAM: Masked Acoustic Modeling for End-to-End Speech-to-Text Translation. arXiv:2010.11445 [cs]. (Feb. 2021). Retrieved Jan. 19, 2024 from http://arxiv.org/abs/2010.11445.

[15] Ying Chen et al. 2021. Efficient assembly of nanopore reads via highly accurate and intact error correction. *Nature Communications*, 12, 1, (Jan. 2021), 60. Publisher: Nature Publishing Group. DOI: 10.1038/s41467-020-20236-7.

[16] Michelle Clark and Melissa Severn. 2023. *Wastewater Surveillance for Communicable Diseases: Emerging Health Technologies. CADTH Horizon Scans*. Canadian Agency for Drugs and Technologies in Health, Ottawa (ON). Retrieved Aug. 4, 2024 from http://www.ncbi.nlm.nih.gov/books/NBK601834/.

[17] Guilherme Coppini, Javier Quilez, Chris Seymour, Stefan Dittforth, Doruk Ozturk, and Michael Mueller. 2023. Benchmarking the Oxford Nanopore Technologies basecallers on AWS. (May 2023). Retrieved Jan. 19, 2024 from https://aws.amazon.com/blogs/hpc/benchmarking-the-oxford-nanopore-technologies-basecallers-on-aws/.

[18] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Anna Korhonen, David Traum, and Lluís Màrquez, editors. Association for Computational Linguistics, Florence, Italy, (July 2019), 2978–2988. DOI: 10.18653/v1/P19-1285.

[19] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs]. (June 2022). DOI: 10.48550/arXiv.2205.14135.

[20] Matei David, L J Dursi, Delia Yao, Paul C Boutros, and Jared T Simpson. 2017. Nanocall: an open source basecaller for Oxford Nanopore sequencing data. *Bioinformatics*, 33, 1, (Jan. 2017), 49–55. DOI: 10.1093/bioinformatics/btw569.

[21] Sam Davis. 2024. Transforming Basecalling in Genomic Sequencing. (May 2024). Retrieved July 26, 2024 from https://web.archive.org/web/20240726135925/https://nanoporetech.com/blog/transforming-basecalling-in-genomic-sequencing.

[22] Delft High Performance Computing Centre (DHPC). 2024. DelftBlue Supercomputer (Phase 2). (2024). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[23] Massimo Delledonne. 2017. Species analysis in Biodiversity hotspot: Kabobo, Democratic Republic of Congo. video. (Sept. 2017). Retrieved Apr. 16, 2024 from https://nanoporetech.com/resource-centre/species-analysis-biodiversity-hotspot-kabobo-democratic-republic-congo.

[24] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. 2013. Predicting Parameters in Deep Learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (NIPS'13). Curran Associates Inc., Red Hook, NY, USA, (Dec. 2013), 2148–2156.

[25] Megan B. Diamond et al. 2022. Wastewater surveillance of pathogens can inform public health responses. *Nature Medicine*, 28, 10, (Oct. 2022), 1992–1995. Number: 10 Publisher: Nature Publishing Group. DOI: 10.1038/s41591-022-01940-x.

[26] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018. Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Anna Korhonen and Ivan Titov, editors. Association for Computational Linguistics, Brussels, Belgium, (Oct. 2018), 97–107. DOI: 10.18653/v1/K18-1010.

[27] Mees Frensel, Zaid Al-Ars, and H Peter Hofstee. Learning Structured Sparsity for Efficient Nanopore DNA Basecalling Using Delayed Masking. In review for the ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, (2024).

[28] Nikolaos Gkalelis and Vasileios Mezaris. 2020. Structured Pruning of LSTMs via Eigenanalysis and Geometric Median for Mobile Multimedia and Deep Learning Applications. In *2020 IEEE International Symposium on Multimedia (ISM)*. (Dec. 2020), 122–126. DOI: 10.1109/ISM.2020.00028.

[29] Glen-Oliver F. Gowers, Oliver Vince, John-Henry Charles, Ingeborg Klarenberg, Tom Ellis, and Arwyn Edwards. 2019. Entirely Off-Grid and Solar-Powered DNA Sequencing of Microbial Communities during an Ice Cap Traverse Expedition. *Genes*, 10, 11, (Nov. 2019), 902. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/genes10110902.

[30] Piotr Grzesik and Dariusz Mrozek. 2022. Accelerating Edge Metagenomic Analysis with Serverless-Based Cloud Offloading. In *Computational Science – ICCS 2022* (Lecture Notes in Computer Science). Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors. Springer International Publishing, Cham, 481–492. DOI: 10.1007/978-3-031-08754-7_54.

[31] Piotr Grzesik and Dariusz Mrozek. 2021. Metagenomic Analysis at the Edge with Jetson Xavier NX. In *Computational Science – ICCS 2021* (Lecture Notes in Computer Science). Maciej Paszynski, Dieter Kranzlmüller, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot, editors. Springer International Publishing, Cham, 500–511. DOI: 10.1007/978-3-030-77970-2_38.

[32] Albert Gu, Karan Goel, and Christopher Ré. 2021. Efficiently Modeling Long Sequences with Structured State Spaces. (Oct. 2021). DOI: 10.48550/arXiv.2111.00396.

[33] Jasper Haenen. 2023. *Accelerating DNA basecalling of Nanopore reads on FPGAs*. MA thesis. TU Delft, (Oct. 2023). http://resolver.tudelft.nl/uuid:06115276-884c-4335-a7d8-024dcd59731f.

[34] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (NIPS'15). MIT Press, Cambridge, MA, USA, (Dec. 2015), 1135–1143.

[35] Song Han et al. 2017. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (FPGA '17). Association for Computing Machinery, New York, NY, USA, (Feb. 2017), 75–84. DOI: 10.1145/3020078.3021745.

[36] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22, 1, (Jan. 2021), 241:10882–241:11005.

[37] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2015. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. (July 2015), 221–227. DOI: 10.1109/SAMOS.2015.7363679.

[38] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2018. Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths. *Comput. Biol. Chem.*, 75, C, (Aug. 2018), 54–64. DOI: 10.1016/j.compbiolchem.2018.03.024.

[39] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29, 3451–3460. Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing. DOI: 10.1109/TASLP.2021.3122291.

[40] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. 2022. SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19, 1, (Jan. 2022), 614–623. Conference Name: IEEE/ACM Transactions on Computational Biology and Bioinformatics. DOI: 10.1109/TCBB.2020.3039244.

[41] Miten Jain, Hugh E. Olsen, Benedict Paten, and Mark Akeson. 2016. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17, 1, (Nov. 2016), 239. DOI: 10.1186/s13059-016-1103-0.

[42] Miten Jain et al. 2018. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36, 4, (Apr. 2018), 338–345. Publisher: Nature Publishing Group. DOI: 10.1038/nbt.4060.

[43] Ineke Knot. 2020. How do I use portable genomics in the field? (Aug. 2020). Retrieved Feb. 20, 2024 from https://www.wildlabs.net/event/how-do-i-use-portable-genomics-field.

[44] Hiroki Konishi, Rui Yamaguchi, Kiyoshi Yamaguchi, Yoichi Furukawa, and Seiya Imoto. 2021. Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention. *Bioinformatics*, 37, 9, (June 2021), 1211–1217. DOI: 10.1093/bioinformatics/btaa953.

[45] Sam Kovaka, Shujun Ou, Katharine M. Jenike, and Michael C. Schatz. 2023. Approaching complete genomes, transcriptomes and epi-omes with accurate long-read sequencing. *Nature Methods*, 20, 1, (Jan. 2023), 12–16. Publisher: Nature Publishing Group. DOI: 10.1038/s41592-022-01716-8.

[46] Jin Young Lee et al. 2021. Comparative evaluation of Nanopore polishing tools for microbial genome assembly and polishing strategies for downstream analysis. *Scientific Reports*, 11, 1, (Oct. 2021), 20740. Publisher: Nature Publishing Group. DOI: 10.1038/s41598-021-00178-w.

[47] Heng Li. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34, 18, (Sept. 2018), 3094–3100. DOI: 10.1093/bioinformatics/bty191.

[48] Yuanyuan Li and Trygve O. Tollefsbol. 2011. DNA Methylation Detection: Bisulfite Genomic Sequencing Analysis. In *Epigenetics Protocols*. Trygve O. Tollefsbol, editor. Humana Press, Totowa, NJ, 11–21. DOI: 10.1007/978-1-61779-316-5_2.

[49] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461, (Oct. 2021), 370–403. DOI: 10.1016/j.neucom.2021.07.045.

[50] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. 2021. Selfish Sparse RNN Training. arXiv:2101.09048 [cs]. (June 2021). DOI: 10.48550/arXiv.2101.09048.

[51] Yang Liu et al. 2021. DNA methylation-calling tools for Oxford Nanopore sequencing: a survey and human epigenome-wide evaluation. *Genome Biology*, 22, 1, (Oct. 2021), 295. DOI: 10.1186/s13059-021-02510-z.

[52] Ekaterina Lobacheva, Nadezhda Chirkova, Alexander Markovich, and Dmitry Vetrov. 2020. Structured Sparsification of Gated Recurrent Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34, 04, (Apr. 2020), 4989–4996. Number: 04. DOI: 10.1609/aaai.v34i04.5938.

[53] Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning Sparse Neural Networks through L_0 Regularization. arXiv:1712.01312 [cs, stat]. (June 2018). DOI: 10.48550/arXiv.1712.01312.

[54] Xuan Lv, Zhiguang Chen, Yutong Lu, and Yuedong Yang. 2020. An End-to-end Oxford Nanopore Basecaller Using Convolution-augmented Transformer. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. (Dec. 2020), 337–342. DOI: 10.1109/BIBM49941.2020.9313290.

[55] Morgan MacKenzie and Christos Argyropoulos. 2023. An Introduction to Nanopore Sequencing: Past, Present, and Future Considerations. *Micromachines*, 14, 2, (Feb. 2023), 459. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/mi14020459.

[56] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. arXiv:1611.00712 [cs, stat]. (Mar. 2017). DOI: 10.48550/arXiv.1611.00712.

[57] Eric Martin and Chris Cundy. 2018. Parallelizing Linear Recurrent Neural Nets Over Sequence Length. In https://openreview.net/forum?id=HyUNwulC-.

[58] Rahul Mishra and Hari Gupta. 2023. Transforming Large-Size to Lightweight Deep Neural Networks for IoT Applications. *ACM Computing Surveys*, 55, 11, (Feb. 2023), 233:1–233:35. DOI: 10.1145/3570955.

[59] Lisa D. Moore, Thuc Le, and Guoping Fan. 2013. DNA Methylation and Its Basic Function. *Neuropsychopharmacology*, 38, 1, (Jan. 2013), 23–38. Publisher: Nature Publishing Group. DOI: 10.1038/npp.2012.112.

[60] Niranjan Nagarajan and Mihai Pop. 2013. Sequence assembly demystified. *Nature Reviews Genetics*, 14, 3, (Mar. 2013), 157–167. Publisher: Nature Publishing Group. DOI: 10.1038/nrg3367.

[61] NVIDIA. 2020. NVIDIA A100 GPUs Power the Modern Data Center. (2020). Retrieved July 19, 2024 from https://www.nvidia.com/en-us/data-center/a100/.

[62] Josephine B. Oehler, Helen Wright, Zornitza Stark, Andrew J. Mallett, and Ulf Schmitz. 2023. The application of long-read sequencing in clinical settings. *Human Genomics*, 17, 1, (Aug. 2023), 73. DOI: 10.1186/s40246-023-00522-3.

[63] Oxford Nanopore [@nanopore]. 2023. Our latest platform software (MinKNOW) update supports end-to-end high-throughput projects. The update includes the integration of high-performance basecaller Dorado — to replace Guppy — for real-time, high-accuracy basecalling + methylation. Tweet. (Aug. 2023). Retrieved Apr. 18, 2024 from https://twitter.com/nanopore/status/1689518489529180160.

[64] Oxford Nanopore Technologies. 2024. Nanopore Sequencing Accuracy. (2024). Retrieved Jan. 23, 2024 from https://nanoporetech.com/platform/accuracy.

[65] Marc Pagès-Gallego and Jeroen de Ridder. 2023. Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling. *Genome Biology*, 24, 1, (Apr. 2023), 71. DOI: 10.1186/s13059-023-02903-2.

[66] Peter Perešíni, Vladimír Boža, Broňa Brejová, and Tomáš Vinař. 2021. Nanopore base calling on the edge. *Bioinformatics*, 37, 24, (Dec. 2021), 4661–4667. DOI: 10.1093/bioinformatics/btab528.

[67] Ryan Poplin et al. 2018. A universal SNP and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36, 10, (Nov. 2018), 983–987. Publisher: Nature Publishing Group. DOI: 10.1038/nbt.4235.

[68] Franka J. Rang, Wigard P. Kloosterman, and Jeroen de Ridder. 2018. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome Biology*, 19, 1, (July 2018), 90. DOI: 10.1186/s13059-018-1462-9.

[69] Shanshan Ren, Nauman Ahmed, Koen Bertels, and Zaid Al-Ars. 2019. GPU accelerated sequence alignment with traceback for GATK HaplotypeCaller. *BMC Genomics*, 20, 2, (Apr. 2019), 184. DOI: 10.1186/s12864-019-5468-9.

[70] Shanshan Ren, Koen Bertels, and Zaid Al-Ars. 2018. Efficient Acceleration of the Pair-HMMs Forward Algorithm for GATK HaplotypeCaller on Graphics Processing Units. *Evolutionary Bioinformatics Online*, 14, 1176934318760543. DOI: 10.1177/1176934318760543.

[71] Bita Darvish Rouhani et al. 2023. Microscaling Data Formats for Deep Learning. arXiv:2310.10537 [cs]. (Oct. 2023). DOI: 10.48550/arXiv.2310.10537.

[72] Hiruna Samarakoon, James M Ferguson, Hasindu Gamaarachchi, and Ira W Deveson. 2023. Accelerated nanopore basecalling with SLOW5 data format. *Bioinformatics*, 39, 6, (June 2023), btad352. DOI: 10.1093/bioinformatics/btad352.

[73] Thomas Sauvage, Alexandre Cormier, and Passerini Delphine. 2023. A comparison of Oxford nanopore library strategies for bacterial genomics. *BMC Genomics*, 24, 1, (Oct. 2023), 627. DOI: 10.1186/s12864-023-09729-z.

[74] Nikita Semionov. 2019. *Pruning of Long Short-Term Memory Neural Networks: Passes of Redundant Data Patterns*. MA thesis. Tilburg University. Cognitive science and artificial intelligence, (Dec. 2019). https://arno.uvt.nl/show.cgi?fid=153975.

[75] Mantas Sereika, Rasmus Hansen Kirkegaard, Søren Michael Karst, Thomas Yssing Michaelsen, Emil Aarre Sørensen, Rasmus Dam Wollenberg, and Mads Albertsen. 2022. Oxford Nanopore R10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. *Nature Methods*, 19, 7, (July 2022), 823–826. Publisher: Nature Publishing Group. DOI: 10.1038/s41592-022-01539-7.

[76] Chris Seymour. 2019. Bonito: A PyTorch Basecaller for Oxford Nanopore Reads. (2019). https://github.com/nanoporetech/bonito.

[77] Gagandeep Singh, Mohammed Alser, Kristof Denolf, Can Firtina, Alireza Khodamoradi, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu. 2024. RUBICON: a framework for designing efficient deep learning-based genomic basecallers. *Genome Biology*, 25, 1, (Feb. 2024), 49. DOI: 10.1186/s13059-024-03181-2.

[78] Gopal K. Singh and Mohammad Siahpush. 2014. Widening Rural–Urban Disparities in Life Expectancy, U.S., 1969–2009. *American Journal of Preventive Medicine*, 46, 2, (Feb. 2014), e19–e29. DOI: 10.1016/j.amepre.2013.10.017.

[79] T. F. Smith and M. S. Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147, 1, (Mar. 1981), 195–197. DOI: 10.1016/0022-2836(81)90087-5.

[80] Amrita Srivathsan, Leshon Lee, Kazutaka Katoh, Emily Hartop, Sujatha Narayanan Kutty, Johnathan Wong, Darren Yeo, and Rudolf Meier. 2021. ONTbarcoder and MinION barcodes aid biodiversity discovery and identification by everyone, for everyone. *BMC Biology*, 19, 1, (Sept. 2021), 217. DOI: 10.1186/s12915-021-01141-x.

[81] Starlink. 2024. Starlink Specifications. (2024). Retrieved Aug. 4, 2024 from https://www.starlink.com.

[82] Haotian Teng, Minh Duc Cao, Michael B Hall, Tania Duarte, Sheng Wang, and Lachlan J M Coin. 2018. Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. *GigaScience*, 7, 5, (May 2018), giy037. DOI: 10.1093/gigascience/giy037.

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs]. (June 2017). DOI: 10.48550/arXiv.1706.03762.

[84] Shaorun Wang, Peng Lin, Ruihan Hu, Hao Wang, Jin He, Qijun Huang, and Sheng Chang. 2019. Acceleration of LSTM With Structured Pruning Method on FPGA. *IEEE Access*, 7, 62930–62937. Conference Name: IEEE Access. DOI: 10.1109/ACCESS.2019.2917312.

[85] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (FPGA '18). Association for Computing Machinery, New York, NY, USA, (Feb. 2018), 11–20. DOI: 10.1145/3174243.3174253.

[86] Peter E. Warburton and Robert P. Sebra. 2023. Long-Read DNA Sequencing: Recent Advances and Remaining Challenges. *Annual Review of Genomics and Human Genetics*, 24, Volume 24, 2023, (Aug. 2023), 109–132. Publisher: Annual Reviews. DOI: 10.1146/annurev-genom-101722-103045.

[87] Liangjian Wen, Xuanyang Zhang, Haoli Bai, and Zenglin Xu. 2020. Structured pruning of recurrent neural networks through neuron selection. *Neural Networks*, 123, (Mar. 2020), 134–141. DOI: 10.1016/j.neunet.2019.11.018.

[88] Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. 2018. Learning Intrinsic Sparse Structures within Long Short-Term Memory. arXiv:1709.05027 [cs]. (Feb. 2018). DOI: 10.48550/arXiv.1709.05027.

[89] Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. 2019. Performance of neural network basecalling tools for Oxford Nanopore sequencing. *Genome Biology*, 20, 1, (June 2019), 129. DOI: 10.1186/s13059-019-1727-y.

[90] Tian Xia, Gelin Fu, Chenyang Li, Zhongpei Luo, Lucheng Zhang, Ruiyang Chen, Wenzhe Zhao, Nanning Zheng, and Pengju Ren. 2023. A Comprehensive Performance Model of Sparse Matrix-Vector Multiplication to Guide Kernel Optimization. *IEEE Transactions on Parallel and Distributed Systems*, 34, 2, (Feb. 2023), 519–534. Conference Name: IEEE Transactions on Parallel and Distributed Systems. DOI: 10.1109/TPDS.2022.3225230.

[91] Chunlin Xiao et al. 2022. Personalized genome assembly for accurate cancer somatic mutation discovery using tumor-normal paired reference samples. *Genome Biology*, 23, 1, (Nov. 2022), 237. DOI: 10.1186/s13059-022-02803-x.

[92] Karin Yaniv et al. 2023. Wastewater monitoring of SARS-CoV-2 in on-grid, partially and fully off-grid Bedouin communities in Southern Israel. *Frontiers in Water*, 5. DOI: 10.3389/frwa.2023.1136066.

[93] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. (June 2017), 548–560. DOI: 10.1145/3079856.3080215.

[94] Yao-Zhong Zhang, Arda Akdemir, Georg Tremmel, Seiya Imoto, Satoru Miyano, Tetsuo Shibuya, and Rui Yamaguchi. 2020. Nanopore basecalling from a perspective of instance segmentation. *BMC bioinformatics*, 21, Suppl 3, (Apr. 2020), 136. DOI: 10.1186/s12859-020-3459-0.