

Improving Select Applications of Long-Read DNA Sequencing

by

Timothy J. Dunn

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2024

Doctoral Committee:

Professor Satish Narayanasamy, Chair
Associate Professor Reetuparna Das
Associate Professor Robert Dickson
Associate Professor Jenna Wiens

Timothy J. Dunn
timdunn@umich.edu
ORCID iD: 0000-0003-3429-4329

© Timothy J. Dunn 2024

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Satish Narayansamy for his support and guidance throughout the course of my PhD. His expertise and advice was instrumental in shaping the course of my research, and greatly enhanced my experience at the University of Michigan. Thank you for allowing my research to branch in unexpected directions as my passion for bioinformatics research grew.

I would also like to thank my committee members Reetu Das, Jenna Wiens, and Bob Dickson for agreeing to serve on my thesis committee and for their support along the way. I am additionally thankful to Satish, Reetu, and David Blaauw for establishing a precision-health focused research group during the early stages of my PhD, which provided me with mentors and a sense of community even as the COVID-19 pandemic began disrupting research on campus soon afterwards.

I was fortunate to participate in two research internships during my PhD: developing cloud-based bioinformatics pipelines at Microsoft Research and training large language models at Mayo Clinic. Firstly, I would like to thank Erdal Cosgun at Microsoft Research for making my internship and time at Microsoft's campus in Redmond an incredible experience. I would also like to thank my co-mentors Venkat Malladi and Olesya Melnichenko for their advice and guidance. At Mayo Clinic, I would like to thank Ming Huang for directing my project and Fred Fan for taking over leadership of the project in the final few weeks.

I would be remiss to not mention my undergraduate professors at Clarkson University who inspired me to pursue computer science research. First and foremost, Sean and Natasha Banerjee for overseeing my growth from introductory C++ classes to working on numerous research projects, presenting my first paper at a conference in Vancouver, and completing my undergraduate Honors thesis. I would also like to thank Christino Tamon and Alexis Maciel for their excellent lectures

and their enthusiasm for teaching, computer science, and algorithms, which I found inspiring.

I would also like to thank several teachers from my early education whom I believe particularly impacted me through their passion for teaching and education. Neil Chodorow, from Cobbles Elementary School, for leading Project Challenge. Laura Bobin, from Bay Trail Middle School, for convincing a group of my friends and I to join MathCounts by offering free candy. And from Penfield High School: Jennifer Vibber, Razifa Becevic, and Shayne Watterson for their incredible teaching.

I would like to thank all the amazing graduate students and postdocs I spent time working with on North Campus in BBB 4844. Firstly, those who were in the later stages of the program when I joined and served as mentors during the first few years of my PhD: Jack Wadden, Arun Subramaniyan, Kush Goliya, and Charlie Eckert. Thank you all for your advice and guidance. Next, those who made working from the office a great experience: Emily Bao, Yichen Gu, Xiao Wu, Alireza Khadem, Jonah Rosenblum, Joy Dong, Yufeng Gu, Subarno Banerjee, Steven Schaefer, Noah Kaplan, Hari Sadasivan, and Sumanth Umesh. Sorry for working from home so often for the last two years!

In addition to the many collaborators that have already been mentioned above, I would like to thank Justin Zook and Nate Olson from the National Institute of Standards and Technology, Matthew Holt from PacBio, and Adam English from the Baylor College of Medicine for their contributions to the discussion on methodologies for benchmarking whole genome variant calls. Their inputs helped shape my work `vcfdist` into the tool it is today.

I would additionally like to thank the organizations that have sponsored my research: the National Science Foundation under Graduate Research Fellowship Grant No. 1841052 and precision health Grant No. 2030454, and the Dan and Betty Kahn Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are my own and do not necessarily reflect the view of the National Science Foundation.

Thank you to my many friends at the University of Michigan who made the past several years in Ann Arbor great: Lucas, Gage, Jillian, Cameron, Charlie, Steven, Thomas, Lindsey, Nathan,

Sarah, Marcus, Mikayla, Alex, Abby, Phillip, Nate, Emma, Will, Alyse, and too many more to list. A huge thank you to my awesome roommate Noah Meurs for keeping me sane throughout the COVID pandemic, and to my incredible girlfriend Heather Burrum for supporting me throughout graduate school. I couldn't have done this without you!

Last, but certainly not least, thank you to my family. To my grandparents Jim & Claire and Nancy & George, for supporting my education. To my parents Tom & Sue, for everything they have done for me, which is far too extensive to list here. And to my siblings – Katie, Brian, and Kevin – for all their support. I dedicate this thesis to them.

TABLE OF CONTENTS

Dedication	ii
Acknowledgments	iii
List of Figures	x
List of Tables	xii
Abstract	xiii
Chapter	
1 Introduction	1
1.1 An Accelerator for Portable Virus Detection	2
1.2 Realigning Long Reads in Short Tandem Repeat Regions	3
1.3 Accurate Evaluation of Phased Small Variant Calls	4
1.4 Joint Evaluation of Small and Structural Variant Calls	5
1.5 Thesis Overview	6
2 Background	8
2.1 A Brief History of DNA Sequencing	8
2.2 Key Concepts in Genomics	9
2.2.1 The Human Genome	9
2.2.2 Genetic Variation	11
2.3 The Human Whole Genome Sequencing Pipeline	12
2.3.1 Nanopore DNA Sequencing	12
2.3.2 Basecalling	13
2.3.3 Alignment	14
2.3.4 Variant Calling	15
2.3.5 Variant Call Benchmarking	15
3 SquiggleFilter: An Accelerator for Portable Virus Detection	18
3.1 Introduction	18
3.2 Background	22
3.2.1 Need for a Virus Detector	22
3.2.2 State-of-the-art Virus Detectors	23
3.2.3 Portable MinION Sequencer	25

3.3	Motivation	26
3.3.1	Bioinformatics Pipeline	27
3.3.2	Performance Bottlenecks	28
3.4	Design	30
3.4.1	Constructing the Reference Squiggle	30
3.4.2	Normalizing Query Squiggles	31
3.4.3	Subsequence Dynamic Time Warping	32
3.4.4	sDTW for Virus Detection	33
3.4.5	sDTW is an Effective Filter	34
3.4.6	Multi-stage sDTW Filtering	35
3.4.7	sDTW Algorithm Improvements	35
3.4.8	Need for an Accelerator	36
3.5	Accelerator	37
3.5.1	SquiggleFilter Design	37
3.5.2	Processing Element	40
3.5.3	Normalizer	41
3.6	Methodology	41
3.7	Results	43
3.7.1	SquiggleFilter Hardware Synthesis	43
3.7.2	Performance Analysis	44
3.7.3	sDTW Algorithm Accuracy	45
3.7.4	Benefits of Read Until	46
3.7.5	Looking Forwards: Scalability	47
3.8	Related Work	48
3.9	Conclusion	50
3.10	Summary	51
4	nPoRe: Realigning Long Reads in Short Tandem Repeat Regions	52
4.1	Introduction	52
4.1.1	Nanopore Variant Calling	52
4.1.2	Nanopore Read Alignment	53
4.1.3	Nanopore INDEL Accuracy	55
4.1.4	Related Work	57
4.2	Results	60
4.2.1	Overview	60
4.2.2	Accuracy	60
4.2.3	Read Concordance	61
4.2.4	Timing	63
4.2.5	Score Matrices	63
4.3	Discussion	64
4.4	Conclusion	65
4.5	Methods	66
4.5.1	Overview	66
4.5.2	<i>n</i> -Polymer Repeats	66
4.5.3	Penalty Functions	67

4.5.4	Reference Annotation	69
4.5.5	Alignment	70
4.5.6	<i>n</i> -Polymer INDEL Conditions	71
4.5.7	Backtracking	71
4.5.8	Follow-Banding	72
4.5.9	Time and Space Complexity	73
4.5.10	Datasets	75
4.5.11	Pipeline	76
4.5.12	Haplotype Phasing	77
4.5.13	Truth VCF Standardization	77
4.6	Summary	78
5	vcfdist: Accurate Evaluation of Phased Small Variant Calls	79
5.1	Introduction	79
5.2	Results	83
5.2.1	The Affine Gap Design Space	83
5.2.2	A Standard Complex Variant Representation	86
5.2.3	INDEL & SNP Precision and Recall	87
5.2.4	INDELs are Common in Repetitive Regions	88
5.2.5	Local Phasing of Dependent Variants	90
5.2.6	Stabilizing Precision-Recall Curves	91
5.2.7	Partial Credit	92
5.2.8	Distance-Based Performance Metrics	93
5.2.9	Runtime and Efficiency	94
5.3	Discussion	95
5.4	Methods	96
5.4.1	Code and Data Availability	96
5.4.2	Preprocessing	96
5.4.3	Phasing Input Query and Truth Callsets	97
5.4.4	Clustering Dependent Variants	98
5.4.5	A Standard Variant Representation	99
5.4.6	Superclustering Variants	99
5.4.7	Alignment and Local Phasing	99
5.4.8	Partial Credit	100
5.4.9	Distance Calculations	101
5.4.10	New Summary Metrics	101
5.4.11	Global Phasing Calculation	104
5.4.12	Output	104
5.5	Supplementary Information	104
5.6	Summary	114
6	vcfdist2: Joint Evaluation of Small and Structural Variant Calls	115
6.1	Introduction	115
6.2	Background	115
6.3	Results	119

6.3.1	Jointly Benchmarking Small and Structural Variants	119
6.3.2	Improved Variant Comparisons for Phasing Evaluation	120
6.3.3	Accuracy and Runtime	123
6.3.4	Interpretability of Benchmarking Tools	126
6.3.5	Validating Correctness	127
6.4	Discussion	129
6.5	Conclusion	131
6.6	Methods	132
6.6.1	Code and Data Availability	132
6.6.2	Datasets	132
6.6.3	Preprocessing	133
6.6.4	Joint Evaluation of Small and Structural Variants	134
6.6.5	Phasing Analysis	134
6.6.6	Accuracy Comparison	135
6.6.7	Validation of vcfdist	136
6.6.8	Clustering Algorithms	136
6.6.9	Evaluating Structural Variants	138
6.6.10	Runtime Comparison	138
6.7	Supplementary Information	138
6.7.1	Datasets	138
6.7.2	Joint Benchmarking Example	139
6.7.3	Phasing Analysis	141
6.7.4	HLA-DQB1 Complex Variant Example	142
6.7.5	Validation and Detailed Comparisons	143
6.7.6	Clustering	146
6.8	Summary	149
7	Conclusion	150
	Bibliography	154

LIST OF FIGURES

Figures

2.1	Illustration of the working principles of a nanopore DNA sequencer	13
2.2	Example raw nanopore sequencing current measurements	13
2.3	Complex variant normalization example	16
3.1	Portable MinION sequencer in our laboratory	20
3.2	US COVID-19 testing timeline	23
3.3	Portability of sequencing devices	26
3.4	A Read Until pipeline for targeted reference-guided assembly of a virus genome . . .	26
3.5	Runtime of each stage in a viral genome assembly pipeline	27
3.6	Projected increases in sequencing throughput	29
3.7	Generating an expected reference genome signal	31
3.8	Normalization and alignment of query squiggles	31
3.9	The subsequence dynamic time warping algorithm (sDTW)	32
3.10	Epidemic virus genome lengths	33
3.11	sDTW alignment cost distributions for human and lambda phage reads	35
3.12	SquiggleFilter System-on-Chip integration with CPU and GPU	38
3.13	Architectural design of a SquiggleFilter tile	38
3.14	Architectural design of a SquiggleFilter processing element	39
3.15	Architectural design of a SquiggleFilter normalizer	39
3.16	Latency and throughput comparisons of Guppy and SquiggleFilter	44
3.17	SquiggleFilter Read Until accuracy and performance	45
3.18	Accuracy results for modifications to the standard sDTW algorithm	46
3.19	SquiggleFilter robustness against random mutations	46
3.20	Flowcell lifetime comparison for Read Until vs ordinary sequencing	47
3.21	Projected SquiggleFilter Read Until scalability	48
4.1	Pileup comparison of minimap2 and nPoRe	54
4.2	<i>n</i> -polymer copy number confusion matrices	55
4.3	SNP and INDEL germline small variant calling accuracy	56
4.4	Proportion of true and false positive variant calls in <i>n</i> -polymer regions	56
4.5	Comparison of an affine gap penalty and nPoRe's CNV gap penalty	59
4.6	nPoRe accuracy results stratified by region	61
4.7	Aligned read concordance comparison for minimap2 and nPoRe	62
4.8	Example <i>n</i> -polymer penalty score matrices	64
4.9	nPoRe substitution and penalty score matrices	68

4.10	nPoRe realignment algorithm dependencies and penalties	71
4.11	The nPoRe follow banding matrix transformation $A \rightarrow B$	74
4.12	nPoRe VCF standardization	77
5.1	A simple example of vcfeval and vcfdist evaluations	82
5.2	The affine gap alignment penalty design space	85
5.3	vcfeval precision and recall accuracy results	89
5.4	vcfdist precision and recall accuracy results	90
5.5	The high-level design of the vcfdist software	92
5.6	pFDA Truth Challenge V2 accuracy results	93
5.7	A realistic example of vcfeval and vcfdist evaluation with partial credit	105
5.8	Example textual vcfdist output summary	106
5.9	pFDA Truth Challenge V2 accuracy results for winning submissions	107
5.10	pFDA Truth Challenge V2 aligners	108
5.11	Original and normalized affine-gap parameters	109
5.12	Selection of variant representation design point B	110
5.13	An overview of vcfdist's clustering algorithm	111
5.14	An overview of vcfdist's superclustering algorithm	112
5.15	An overview of vcfdist's phasing algorithm	112
5.16	An overview of vcfdist's alignment algorithm for precision and recall	113
6.1	Pipeline overview for joint evaluation of small and structural variant calling	117
6.2	Joint evaluation of small and structural variant calls	120
6.3	An example joint evaluation of small and structural variant calls	121
6.4	An example WhatsHap compare false positive flip error call	123
6.5	Comparison of vcfdist whole genome variant calling evaluation with prior works	124
6.6	The affine gap design space for variant representation	139
6.7	An example evaluation of a complex variant	140
6.8	Switch and flip error confusion matrices	142
6.9	Confusion matrices for variant comparison method differences	144
6.10	Examples for each category of variant classification difference	145
6.11	Measuring variant representation dependencies	147
6.12	An example demonstrating different variant clustering methods	148

LIST OF TABLES

Tables

3.1	A comparison of popular commercial virus detectors for SARS-CoV-2	24
3.2	SARS-CoV-2 strain comparison	34
3.3	Architectural specifications of evaluated GPUs	42
3.4	SquiggleFilter ASIC synthesis results	43
4.1	nPoRe pipeline runtime breakdown	63
4.2	Example n -polymer reference sequence annotations	70
4.3	An overview of the Clair3 training and evaluation pipelines	76
6.1	Comparison of phasing results for vcfdist and WhatsHap compare	122
6.2	Whole genome evaluation runtime comparison	125
6.3	Comparison of evaluations of the HLA-DQB1 gene	126
6.4	A manual examination of differences between variant calling evaluation methods . . .	128
6.5	Summary of evaluated whole genome sequencing datasets	138
6.6	Sequences for a subregion of the HLA-DQB1 gene	143
6.7	Summary of vcfdist clustering results	147

ABSTRACT

The cost to sequence a human genome has dropped from an estimated \$300 million to well under \$1,000 over the past two decades. In fact, several companies – which can amortize sequencing cost by running hundreds of samples in parallel – have recently claimed to have reached the \$100 human genome. Following such dramatic cost improvements, whole genome sequencing is just starting to regularly be used for cancer profiling, rare genetic disease detection, agricultural breeding, pathogen detection, microbiome bacterial abundance estimation, evolutionary biology research, personalized medicine development, and much more. As sequencing costs continue to drop, as accuracy improves, and as new applications are discovered, DNA sequencing will become increasingly ubiquitous.

At the same time that sequencing cost is rapidly dropping, new sequencing technologies have emerged that offer greater capabilities than ever before. In particular, nanopore-based long-read sequencing has no theoretical limit on the length of a contiguous DNA sequence, or “read”, that can be measured. In comparison to short-read sequencing technologies that have dominated the sequencing market thus far (with maximum read lengths of 100 base pairs), nanopore devices have sequenced entire bacterial chromosomes in a single strand. The current nanopore read length record stands at over four million bases. Longer read lengths result in fewer problems during read mapping and genome assembly, allowing insight into complex regions of the genome and types of genetic variation that have been historically under-studied. Although nanopore devices were originally limited by their approximately 80% per-base accuracy when first publicly released in 2015, this accuracy has increased to over 99% in recent years with the adoption of deep learning

basecallers. Nanopore-based sequencing devices are also the first to come in a portable handheld form factor and offer real-time analysis of raw data as it is being recorded, further expanding potential use cases.

Despite its incredibly promising future, long read sequencing does not come without its own set of challenges. In this thesis, I explore several different applications of long read DNA sequencing and improve upon current methodologies in this new field. First, we present a hardware-accelerated filter that directly analyzes nanopore sequencer output in real time to filter non-viral reads, enabling cheaper detection of pathogenic viruses. Next, we introduce a novel read alignment algorithm that enables more consistent alignment of long reads in highly repetitive areas of the genome, and demonstrate that this improves recall for tandem repeat variant calling. Then, we analyze the design space for complex variant representation and present a new variant calling benchmarking tool that accurately and stably measures performance regardless of the representation of reported variants. Last, we extend this benchmarking tool to jointly evaluate small and structural variants, and demonstrate that doing so results in improved measured performance and enables more accurate phasing analyses.

CHAPTER 1

Introduction

The advent of long-read sequencing technologies has ushered in a new era of genomics by enabling the resolution of complex genomic regions that were previously intractable with short-read approaches. Technologies such as Oxford Nanopore and Pacific Biosciences have been at the forefront of this revolution, offering unprecedented insights into the structural intricacies of the genome. With their ability to generate reads that span tens of thousands of base pairs, these platforms can traverse repetitive elements, capture full-length RNA transcripts, and uncover large structural variations that are otherwise obscured in short-read datasets.

Parallel to technological innovations in sequencing, there have been remarkable strides in accuracy improvements. Advances in basecalling algorithms fueled by deep learning and enhancements in chemistry and hardware have considerably reduced the error rates associated with long reads. As accuracy continues to improve, long-read sequencing is poised to become a staple in both research and clinical settings.

As impactful as these advances have been, they also beget novel computational challenges. While longer reads offer the potential for more accurate genome assembly and mapping, errors inherent to the technology, along with the complexity of certain genomic regions, mandate specialized bioinformatic tools. Consequently, there is a need to develop innovative approaches tailored for long-read data to enable their full potential. This dissertation addresses three critical areas where bioinformatics can substantially contribute to capitalizing on the advantages of long-read sequencing: real-time portable virus detection, long-read realignment in short tandem repeat

(STR) regions, and VCF benchmarking for phased small and structural variant calls.

1.1 An Accelerator for Portable Virus Detection

The COVID-19 pandemic, driven by the SARS-CoV-2 virus, has underscored the global need for rapid and accurate viral detection methods. The primary diagnostic modality, PCR, requires custom primers tailored to a virus’s genetic material, a process which can be slow and error-prone. During the critical initial months of the COVID-19 outbreak, this approach proved inadequate, leading to delayed mass testing capabilities and, consequently, a catastrophic inability to contain the spread. To circumvent such drawbacks and to prepare for future outbreaks, it is imperative to develop a more efficient testing infrastructure capable of rapidly adapting to novel viral threats.

Polymerase Chain Reaction (PCR) has been the cornerstone of viral diagnostics, including for SARS-CoV-2. However, the inherent limitations of PCR, including the need for virus-specific primer design and the logistical challenges in their distribution, have highlighted the necessity for a more versatile and swift detection system. Additionally, the computing power required to handle the vast amount of non-viral genetic material present in biological samples presents a computational bottleneck. These issues motivate the development of a solution that could reduce the time required for diagnostics and enable the global community to respond quickly to emerging pathogens.

We propose an innovative approach leveraging Oxford Nanopore Technologies’ MinION, a device capable of sequencing DNA/RNA without the need for virus-specific PCR amplification. The key feature of our methodology is the utilization of the MinION’s “Read Until” capability to selectively eject non-target genetic material, thereby concentrating computational resources on potential viral sequences. Crucially, we introduce SquiggleFilter, an algorithm and hardware solution that operates directly on the raw electrical signals (“squiggles”) produced by the nanopore sequencing process, bypassing the computationally intensive basecalling step and substantially reducing the time to detect viral sequences. Our dynamic time warping (DTW) based technique performs

real-time, targeted filtering of the genetic data.

The SquiggleFilter algorithm has been rigorously tested and proven to achieve dramatic increases in throughput and reductions in latency compared to existing GPU-based solutions, leading to rapid and energy-efficient viral detection. Capable of reprogramming in response to new viral sequences, our solution demonstrates significant processing speed-ups for a SARS-CoV-2 specimen. Furthermore, our solution enhances the diagnostic power of sequencing by providing not just a binary output but a full genomic sequence of the detected virus, thereby facilitating comprehensive viral surveillance and research. Our hardware design is scalable and anticipates future increases in Oxford Nanopore Technologies’ sequencing throughput, positioning SquiggleFilter as a crucial tool for pandemic preparedness and responsive healthcare intervention.

1.2 Realigning Long Reads in Short Tandem Repeat Regions

The advent of long-read sequencing technologies, particularly Oxford Nanopore Technologies (ONT), has revolutionized genomic sequencing with their ability to span repetitive sequences and provide real-time results. While SNP variant calling accuracy for nanopore sequencing has surpassed an impressive 99.5%, the identification of small insertions and deletions (INDELs) in low-complexity regions such as homopolymers and tandem repeats remains suboptimal, with INDEL recall lingering below 80%. The standard variant calling methods have continually evolved, emphasizing advances in machine learning and data representation. However, insufficient attention has been given to the impact of fine-grained read alignment paths on variant calling accuracy, which we propose is critical to accurately identifying INDELs in low-complexity sequences.

We show that conventional alignment strategies using static gap penalties inadequately represent the actual mutation and sequencing error profiles observed in nanopore data, particularly within n -polymer (homopolymer and short tandem repeat (STR)) regions. With current methods leading to persistent errors in fine-grained read alignment, there is a crucial need for improved algorithms that accurately model the likelihood of INDEL occurrence in repetitive sequences. By

enhancing read alignment precision, we anticipate a significant impact on the accuracy of downstream variant calling, with implications for the diagnosis of genetic conditions associated with repetitive sequence variation.

nPoRe (*n*-Polymer Realigner) is an alignment extension tool that introduces novel gap penalties refined for *n*-polymers. These penalties are calculated based on empirically determined INDEL likelihoods from the dataset, rather than presupposed estimations. The approach accounts for the overrepresentation of INDELs within repetitive sequences due to sequencing errors or actual genomic copy number variations. Our algorithm efficiently realigns reads against a reference by trustingly following the original alignments indicated in an input BAM file and applying a mechanism termed “follow-banding,” which confers significant savings in computational resources.

Our evaluations demonstrate that haplotype phasing paired with *nPoRe* realignment notably improves INDEL recall from 63.76% to 73.04% without compromising precision, thus addressing a critical gap in nanopore variant calling. Furthermore, read concordance measured by Gini purity shows an improvement, indicating better agreement between reads in the realigned dataset. The release of *nPoRe* provides a robust tool that boosts the accuracy of variant calling in challenging genomic regions, thereby enhancing the diagnostic power of sequencing-based assessments and facilitating more precise genomics research.

1.3 Accurate Evaluation of Phased Small Variant Calls

Accelerated advancements in whole genome sequencing (WGS) and the importance of variant calling for clinical and research purposes necessitate reliable benchmarking tools. Despite progress, current evaluations of variant calling accuracy, particularly in tools like *vcfeval*, often yield inconsistent results driven by biases towards specific variant representations. Such representations can misalign with the true variation, highlighting a pressing need for more robust evaluation techniques that universally recognize equivalent variant sets and support development and selection of the best variant calling pipelines for clinical decisions and research.

The foundational issue with tools like `vcfeval` arises from their sensitivity to the multiple valid representations of complex variants, leading to inaccurate assessments. Moreover, evaluations that do not enforce local phasing of variants yield scores contingent on arbitrary representations rather than the underlying genomic truth. Consequently, variant calling pipelines might appear disproportionately effective or flawed depending on their variant representation, introducing a discrepancy between predicted and actual performance. This limitation is especially critical as sequencing extends into challenging genomic territories with higher variant complexity.

To address these concerns and improve the reliability of small variant call benchmarking, we introduce `vcfdist`, a tool that standardizes variant representation based on well-defined alignment parameters, ensuring local phasing accuracy. `vcfdist` also recognizes the value of near-correct calls through a novel partial credit system and employs distance-based metrics to transcend the dependence on variant representation, providing a clearer picture of the variant calling accuracy.

Comprehensive testing on 64 phased Truth Challenge V2 submissions reveals `vcfdist`'s robustness, achieving remarkable consistency in INDEL evaluation across various representations, with stability improving from $R^2 = 0.97243$ to 0.99996 compared to `vcfeval`. By embracing a standardized variant representation, enforcing local phasing, introducing partial credit, and leveraging alignment distances, `vcfdist` ushers in an era of equitable and precise benchmarking for variant calling, crucial for future explorations and improvements in genomics.

1.4 Joint Evaluation of Small and Structural Variant Calls

Before the advent of DNA sequencing technology, structural variants (SVs) were observed as early as 1959. With the initial sequencing of the human genome in 2001 using short-read technologies, focus shifted to studying single nucleotide polymorphisms (SNPs) and small insertions and deletions (INDELs). It eventually became clear that, although less common than SNPs, SVs and INDELs account for a significant portion of genomic diversity. Only recently has the introduction of accurate long-read sequencing technologies significantly improved the unambiguous identification

of large insertions and deletions.

Despite the ability to call both small and structural variants from the same long-read sequencing data, variant benchmarking practices typically separate the evaluation of small variants (under 50 base pairs) and structural variants (over 50 base pairs). This artificial categorization, which stemmed from the limitations of short-read sequencing, compromises the accuracy of variant calling evaluations. When variants are segmented into these categories before benchmarking, potentially equivalent variant calls across size categories are overlooked, leading to inflated error metrics.

To address the limitations inherent in segregating small and structural variant evaluations, we extend `vcfdist` to offer joint benchmarking capabilities. `vcfdist` is now equipped to evaluate phased SNPs, INDELs, and SVs across the whole genome. This unified approach aims to reduce the false discovery and false negative rates commonly reported when evaluating variant categorizations separately.

The extension of `vcfdist`'s capabilities has demonstrated that joint evaluations significantly increase the accuracy of measured performance metrics. Our analysis shows uniform reductions in error rates for SNPs (-28.9%), INDELs (-19.3%), and SVs (-52.4%) across three datasets, indicating `vcfdist`'s comprehensive and precise benchmarking. Moreover, a detailed comparison with other evaluation tools such as `Truvari` and `vcfeval` confirms `vcfdist`'s superior accuracy and interpretability. The additional correction of common phasing evaluation flaws leads to over a 50% reduction in measured flip errors. In effect, `vcfdist` is poised as a more accurate and versatile tool, revolutionizing the joint benchmarking of variant calls.

1.5 Thesis Overview

The main goal of this thesis is to explore the many possibilities created by the advent of cheaper and portable long read sequencing, and to develop solutions for the unique challenges which new long read sequencing technologies face.

First, in Section 3, we present SquiggleFilter, an accelerator for efficient classification of viral and human DNA [1]. Although ONT’s MinION sequencer is portable, basecalling is too computationally expensive to be performed on an edge device. We present an accelerator-based solution that directly filters reads from the raw current measurements, bypassing basecalling entirely. This allows read filtering and viral genome reconstruction to be performed entirely on a portable edge device.

Next, in Section 4, we present nPoRe, which introduces a novel alignment algorithm to improve the performance of pileup-based variant calling [2]. Since the advent of long read sequencing, improvements in read accuracies and lengths has allowed alignment to and analysis of more repetitive regions than ever before. Previously, such regions were excluded from genome analyses. Mutations known as copy number variants are of particular interest in such regions. nPoRe is the first work to incorporate the possibility of copy number variation into each read’s alignment score. This reduces alignment artifacts and improves alignment concordance across reads, resulting in better downstream variant calling results.

While benchmarking the variant calling accuracy improvements due to nPoRe, we noticed that the results reported by state-of-the-art benchmarking tools were dependent on complex variant representations and discarded phasing information. This phasing information is becoming increasingly relevant, since long reads are better able to phase haplotypes. To improve benchmarking consistency and evaluate phased variants, we developed vcfdist. In Section 5, we show that vcfdist is a significant improvement over prior work [3].

Section 6 presents an extension to vcfdist which enables evaluation of structural variants (SVs) in addition to SNPs and smaller INDELs. We demonstrate that jointly evaluating small and structural variants reduces measured false positive and false negative variant calls for more accurate benchmarking. Lastly, we evaluate phasing for small and structural variants using vcfdist and show that alignment-based phasing analysis results in more accurate phasing evaluations than existing tools.

CHAPTER 2

Background

2.1 A Brief History of DNA Sequencing

In 1953, Watson and Crick famously discovered the double-helix structure of DNA while working from crystallographic data produced by Rosalind Franklin and Maurice Wilkins [4, 5]. The helical structure of DNA, or deoxyribonucleic acid, is composed of two spiralling sugar-phosphate backbones and the four nucleotide bases adenine (A), cytosine (C), guanine (G), and thymine (T) extending inwards to form complementary base pairs (A with T and G with C). The ordering or “sequence” of these four bases is what encodes the genetic instructions for the development, functioning, growth, and reproduction of all living organisms.

It wasn’t until the mid 1970s that two fundamental papers led to the widespread sequencing of DNA [6, 7]. This protocol involved breaking terminally-labeled DNA at every repetition of a base, and then using gel electrophoresis to resolve the order of bases using the generated fragment lengths. This approach, known as 1st generation or “Sanger” sequencing, was slow and labor-intensive and resulted in short reads of around 100 nucleotides.

As the sequencing technology matured and new methods such as PCR (polymerase chain reaction) were developed for working with DNA libraries, this sequencing pipeline was fully automated and scaled up. This became known as 2nd generation sequencing and was characterized by massively parallel automated workflows and read lengths up to 1000 base pairs (bp) [8].

Most recently, 3rd generation sequencing technologies are characterized by real-time single-

molecule sequencing of longer reads. Although there are currently several competing long read sequencing technologies, the two most prominent are Pacific Bioscience’s SMRT (single-molecule real-time) sequencing and Oxford Nanopore Technology’s nanopore sequencing. PacBio SMRT sequencing uses nanostructures called zero-mode waveguides to detect the fluorescence of individual nucleotides, and achieves read lengths of over 10,000bp with high accuracy [8].

ONT sequences DNA by applying a voltage difference across a thin membrane embedded with a single nanopore. As DNA translocates through the nanopore, an ammeter measures the resulting current and can be used to reconstruct the bases in the DNA that passed through the nanopore. This allows reading sequences of DNA with arbitrary length (the record currently stands at over 4,000,000bp [9]). Moreover, they have released a portable handheld nanopore sequencer, the MinION, at a price point of around \$1000 USD. In this thesis, I explore several applications that are well-suited to this exciting new technology, and present improvements for each. In the following Section, I first cover key concepts in genomics that are essential for understanding this work.

2.2 Key Concepts in Genomics

2.2.1 The Human Genome

DNA. DNA, short for deoxyribonucleic acid, is a complex molecule that carries the genetic instructions for the development, functioning, growth, and reproduction of all living organisms. It is the hereditary material that passes genetic information from one generation to the next.

The structure of DNA consists of two long strands that twist around each other to form a double helix. Each strand is composed of a sugar-phosphate backbone, with nucleotide bases extending inward from the backbone. The four nucleotide bases found in DNA are adenine (A), thymine (T), cytosine (C), and guanine (G). The base pairs form complementary pairs, with A always pairing with T, and C always pairing with G. This pairing creates the rungs of the DNA ladder-like structure.

Chromosomes. The human genome is the complete set of genetic material or DNA present in humans. It is composed of approximately 3 billion base pairs and is organized into 23 pairs of chromosomes. The first 22 pairs of chromosomes are called “autosomes” and are labelled in descending order of approximate size. The last pair of chromosomes differs based on an individual’s sex, and are therefore called “sex chromosomes”. Females (XX) have two copies of an X chromosome, and males (XY) have one copy each of an X and Y chromosome.

Genes. The human genome contains around 20,000 to 25,000 protein-coding genes, which provide instructions for building proteins. These genes make up only a small fraction (about 1-2%) of the entire genome. The majority of the human genome consists of non-coding DNA, which does not directly code for proteins. Non-coding DNA includes regulatory regions, repetitive sequences, introns, and intergenic regions. Recent research has shed light on the functional importance of non-coding regions, such as their involvement in gene regulation and disease susceptibility.

Alleles. Alleles are alternative forms of a gene that occupy the same position (or “locus”) on a specific chromosome. Each individual inherits two alleles for each gene, one from each parent. The combination of these two alleles is known as an individual’s “genotype”, which can either be “homozygous” if the two alleles are identical or “heterozygous” if the alleles are different. A “haplotype” refers to specific combinations of alleles or genetic variants that tend to be inherited together because they are located nearby on the same chromosome.

Phasing. Human gametes (egg and sperm cells) are unique in that they contain only a single copy of each chromosome, and are “haploid”. All other human (non-gamete, or “somatic”) cells are “diploid”, containing two copies of each chromosome, one from each parent. Cells containing more than two copies of each chromosome are termed “polyploid”; this is common in plants.

Historically, short read sequencing was often able to determine that a heterozygous (occurring on one of the two chromosomal copies) variant was present, but not whether it occurred on the maternal or paternal haplotype. Such variant calls are “unphased”. If we are unable to ascertain whether multiple nearby variant calls are on the maternal or paternal haplotype, but we do know whether or not they occur on the same haplotype relative to one another, then they are “locally

phased”. Regions with known locally-correct phasings are termed “phase blocks”. If a variant is known to occur on the maternal or paternal haplotype, it is (globally) “phased”. With recent advances in long read sequencing, phase blocks are growing ever larger [10].

2.2.2 Genetic Variation

Genetic variation comes in all shapes and sizes, from single base pair mutations to million base insertions. Below, we highlight some of the most common categories of genetic variation.

SNPs. Single nucleotide polymorphisms are the most common type of genetic variation in the human genome. They occur when a single nucleotide (A, T, C, or G) at a specific position in the DNA sequence is replaced with a different nucleotide. Two or more adjacent SNPs are known as a multi-nucleotide polymorphism (MNP).

INDELs. Insertions and deletions occur when a segment of DNA is either inserted or deleted, causing a shift in the DNA sequence. INDELs can range in size from a single base pair to much larger segments, though most classifications restrict the definition to modifications shorter than 50 base pairs.

Complex variants. Any variants which cannot be represented as a single insertion, deletion, or SNP are called “complex”. For example, if the reference allele is AACG but the alternate allele is TCA, this change cannot be represented by a single INDEL or SNP.

SVs. Insertions and deletions longer than 50 base pairs are typically classified as “structural variants”. Although the average genomic variation between two humans is 0.1% in terms of SNPs, when taking SVs into account, this increases to 1.5% [11]. There several subcategories of SVs such as inversions, duplications, translocations, and tandem repeat expansions that can be defined by the properties of the modified and surrounding sequences.

CNVs. Copy number variants are alterations in the number of copies of a particular repeated DNA segment. These variations can involve the deletion or duplication of a segment of DNA, ranging in size from a few hundred base pairs to large chromosomal regions.

STRs. Also known as “microsatellites”, simple tandem repeats are typically defined as regions consisting of the same 1-6 base pair pattern repeating adjacently multiple times. Tandem repeat expansions are a common mutation which refers to an increase in the number of copies of a repeated sequence, and can cause genetic disorders.

2.3 The Human Whole Genome Sequencing Pipeline

2.3.1 Nanopore DNA Sequencing

As shown below in Figure 2.1, nanopore sequencing occurs by feed a single strand of double stranded DNA through a nanopore. This protein nanopore acts as a narrow channel that allows ions to pass from one side of the membrane to the other. Although an ordinary cell membrane is depicted in Figure 2.1, Oxford Nanopore Technologies (ONT) – the primary company that sells nanopore sequencing devices – uses membranes consisting of a synthetic polymer for enhanced durability.

By applying a small voltage of 170mV across this membrane, ions on one side of the membrane will migrate through the nanopore to the other side. The movement of these ions will be impacted by the single-stranded DNA that partially obstructs the nanopore. By attaching an ammeter to this setup and measuring the resulting electrical current (i.e. rate of ion movement), a raw signal can be generated that is a function of the DNA bases currently partially obstructing the nanopore.

Although not pictured in Figure 2.1, a “motor protein” attached to the nanopore and is responsible for unzipping the double stranded DNA into two single strands. Despite being called a motor protein, in current nanopore sequencing designs this molecule is actually more of a braking protein, reducing the rate of DNA movement to approximately 450 bases per second so that the 4kHz current sampler can obtain a sufficient number of measurements per DNA base. This motor protein basically acts as a ratcheting mechanism forcing the DNA to move exactly one base at a time. Although the average rate of DNA movement is 450 bases per second, the duration between each movement is highly variable and governed by a stochastic process, waiting for an ATP molecule to

collide with the motor protein.

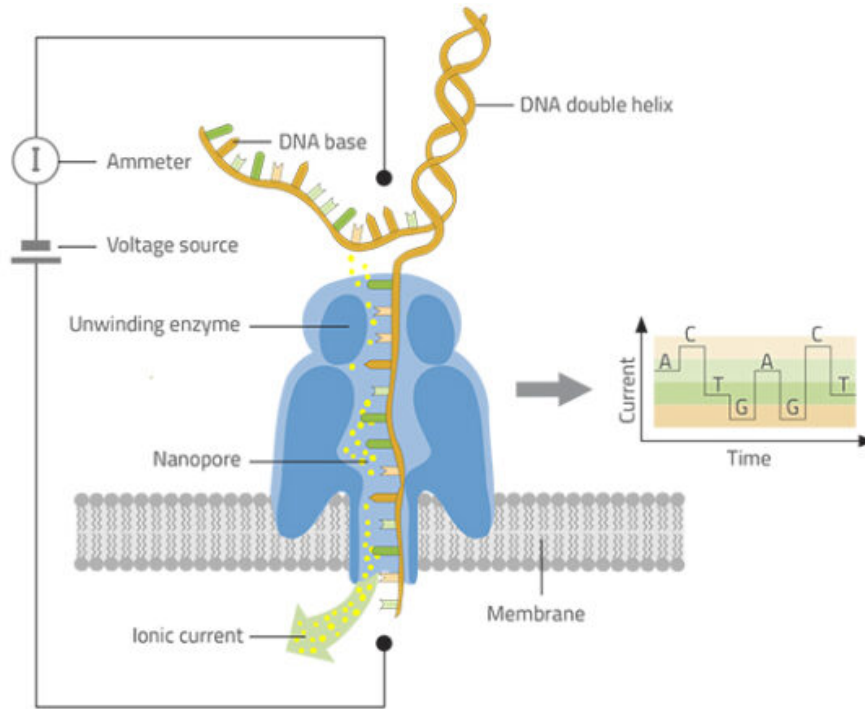


Figure 2.1: Illustration of the working principle of a nanopore DNA sequencer. A strand of DNA is passed through a nanopore, and an ionic current is measured and translated into the DNA sequence (image from [12]).

2.3.2 Basecalling

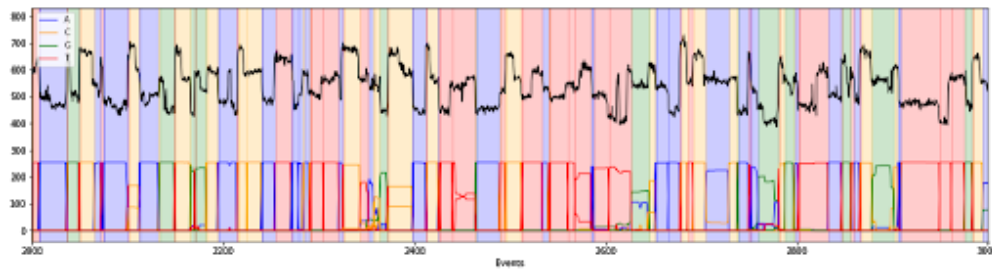


Figure 2.2: Example raw nanopore sequencing current measurements. The top graph (in black) shows the raw nanopore signal trace. The background is segmented and color-coded for each base output by the basecaller at each position (A=green, T=red, C=orange, G=blue). The bottom graph shows the per-base likelihoods output by the neural network basecaller at each position.

The simplistic mapping of one current level to one DNA base in Figure 2.1 is for illustrative

purposes only. Figure 2.2 shows a real example squiggle that has been decoded by a basecaller. The current level measured during nanopore sequencing is affected by multiple bases, since the narrowest physical restriction in a nanopore is several times larger than a single base of DNA. The signal at any given point in time is affected by approximately 5-6 adjacent bases in older nanopores. In recent years, this has increased to 8-10 bases because deep neural networks are able to deconvolute this information and a wider sensing width has been shown to improve basecalling accuracy in repetitive regions such as homopolymers.

Early basecallers used Hidden Markov Models (HMMs) to model the internal state of the sequence of bases currently obstructing the nanopore. Once deep neural networks started performing well at language translation tasks (e.g. French to English), however, these model architectures were adapted to translate raw current measurements into a sequence of bases. ONT's production models first used bidirectional recurrent neural networks (RNNs) using gated recurrent units (GRUs) and long short term memory (LSTMs). Today, their models use a convolutional neural network (CNN) followed by a connectionist temporal classification (CTC) decoder.

2.3.3 Alignment

Once strands of adjacent DNA bases have been basecalled into sequences (such as ACCGTG) known as “reads”, their location within the reference genome must be determined in order to identify variations from the reference sequence. Alignment occurs in two steps: seeding and extension.

2.3.3.1 Seeding

Seeding involves identifying short, exact or near-exact matches (seeds) between the read and the reference genome. These matches serve as starting points for the alignment process. The goal of seeding is to efficiently narrow down the search space, as aligning every read to the entire genome would be computationally expensive. Typically, seeding algorithms use techniques like hashing or indexing to quickly identify potential matches between the read and the reference genome.

2.3.3.2 Extension

Extension is the process of extending the seeds found during the seeding step into longer alignments. Once potential seed matches are identified, the alignment algorithm extends these seeds by aligning the surrounding regions of the read and the reference genome. During extension, algorithms usually use dynamic programming techniques, such as the Smith-Waterman algorithm, to find the optimal alignment between the read and the reference genome. The goal of extension is to accurately align the entire read to the reference genome, taking into account possible gaps, mismatches, and sequencing errors.

2.3.4 Variant Calling

Variant calling is the last stage of secondary analysis in the human whole-genome sequencing pipeline, aimed at identifying differences (variants) between an individual's genome and a reference genome. These variants can include single nucleotide polymorphisms (SNPs), small insertions or deletions (INDELs), or larger structural variations (SVs). Once sequencing reads have been aligned to a reference genome, variant calling involves identifying positions where the individual's sequence differs from the reference, and determining the type and characteristics of these differences. Various statistical methods and algorithms are employed to distinguish true variants from sequencing errors and artifacts. The resulting variant calls provide valuable insights into genetic diversity, population genetics, disease susceptibility, and personalized medicine.

In recent years, variant calling has been dominated by machine learning

2.3.5 Variant Call Benchmarking

Because there is on average only 0.1% difference in the genetic sequence of two individuals [11], it is most efficient to store the results of whole genome sequencing (WGS) in a difference-based format, relative to a human reference genome. VCF format stores each variant on a separate line. For each variant the chromosome, position, reference (REF) allele, and alternate (ALT) allele are

Normalization. Prior works have defined a standard format for reporting individual variants [13, 14]. First, multi-nucleotide polymorphisms (MNPs) and complex variants are decomposed into simple SNPs and INDELs. Next, matching reference and alternate allele bases are trimmed from both sides of all reported variants. Lastly, all INDELs are left-shifted as far as possible. An example of this normalization process is shown in Figure 2.3.

Query ATCGAAAATC

(a)

(b)

(c)

(d)

VCF

POS	REF	ALT
2	A	T
3	G	C
6	AATC	AAATC

POS	REF	ALT
2	A	T
3	G	C
7	A	AA

POS	REF	ALT
2	A	T
3	G	C
4	G	GA

Decomposed

Trimmed

Left shifted

SNP and INDEL variant comparisons. Early works covering SNP and INDEL comparison include bcftools (stats) [15], vcflib (vcfintersect) [16], vcftools (diff) [17], bcbio (variation) [18], SMaSH (calldiff) [19], and vcfeval [14]. In 2015, Real Time Genomics demonstrated in [14] that only vcfeval and vcflib were able to discover all equivalent variant representations in two different VCFs. VarMatch was released in 2016, and is notable for its flexible optimization criteria and genome partitioning algorithm [20]. A few years later, in 2019, the Global Alliance for Genomics and Health (GA4GH) team released a paper [21] suggesting standardized best practices for germline small-variant benchmarking. In this work, they present their own tool, xcmp “happy” [22], but suggest using vcfeval for benchmarking. happy remains widely used to perform stratified analysis of benchmarking results [23].

Structural Variant comparisons. Since structural variants contain large inserted and deleted sequences, most comparison methods rely on heuristic methods to determine approximate equivalence. Truvari compares two SVs using the annotated variant type, reference distance between the SVs, relative size difference, percentage overlap, and alignment edit distance (using edlib [24]). Any two SVs which do not meet the minimum threshold requirements for any of these criteria are considered false positives [25]. SVAnalyzer appears to have followed a similar approach [26]. Recently, Truvari introduced the phab module, which uses mafft [27] multiple sequence alignment to harmonize truth and query SV representations. This ensures that approximate matches will be found if they exist, but requires changing the variant representation and is currently prohibitively slow. hap-eval, released by Sentieon in 2022, and clusters all nearby SVs together before using alignment distance to classify the entire cluster as a true (TP) or false positive (FP) [28]. TT-mars considers an SV to be a TP if applying it to the reference sequence reduces its distance from the truth sequence and the length is approximately correct [29]. Each SV is considered individually (therefore it cannot discover many-to-one or one-to-many SV equivalence), however, and they use four heuristic parameters for filtering TPs.

CHAPTER 3

SquiggleFilter: An Accelerator for Portable Virus Detection

The MinION is a recent-to-market handheld nanopore sequencer. It can be used to determine the whole genome of a target virus in a biological sample. Its Read Until feature allows us to skip sequencing a majority of non-target reads (DNA/RNA fragments), which constitutes more than 99% of all reads in a typical sample. However, it does not have any on-board computing, which significantly limits its portability.

We analyze the performance of a Read Until metagenomic pipeline for detecting target viruses and identifying strain-specific mutations. We find new sources of performance bottlenecks (base-caller in classification of a read) that are not addressed by past genomics accelerators.

We present SquiggleFilter, a novel hardware accelerated dynamic time warping (DTW) based filter that directly analyzes MinION’s raw squiggles and filters everything except target viral reads, thereby avoiding the expensive basecalling step. We show that our 14.3W 13.25mm² accelerator has 274× greater throughput and 3481× lower latency than existing GPU-based solutions while consuming half the power, enabling Read Until for the next generation of nanopore sequencers.

3.1 Introduction

The COVID-19 pandemic caused by the SARS-CoV-2 virus continues on a global scale. Today, diagnostic tests are widely available to detect SARS-CoV-2. Most of these tests involve some

form of Polymerase Chain Reaction (PCR), a common technique for exponentially amplifying DNA/RNA. In order to detect a virus such as SARS-CoV-2, custom “primers” are first designed and manufactured which will only attach to and amplify specific regions of DNA/RNA in the target virus’s genome. After PCR, the virus’s presence or absence can be determined based on whether the amplification was successful or not.

A significant shortcoming of the current approach is that PCR primers are targeted to a specific virus. Custom primer design is a complex, error-prone, and time-consuming process [30] [31]. Even though SARS-CoV-2’s RNA was sequenced in early January 2020, validated SARS-CoV-2 specific PCR primers took several months to develop [31] [32]. Lack of mass testing capability in the early stages of SARS-CoV-2 made it difficult to detect and control its spread, leading to a catastrophic pandemic. While we now have adequate testing capability for SARS-CoV-2, it is not unlikely for another novel virus like SARS-CoV-2 or its variants to emerge in the near future [33], and if it does, we need to be prepared with adequate testing infrastructure in place to detect and control its spread in the early stages.

We envision a programmable virus detector (one that constructs whole viral genomes) that can be deployed worldwide. As soon as an emerging novel virus is discovered and sequenced, the reference genome of the novel virus would be distributed to all the devices, instantly turning them into targeted detectors.

Our solution uses Oxford Nanopore Technologies’ (ONT) MinION Mk1B (henceforth, referred to as the MinION), a new-to-market palm-sized DNA/RNA sequencer. It is fairly low-cost, portable, and can sequence long reads in real time.

We replace targeted PCR with universal PCR [34], which amplifies *all* DNA/RNA. Thus, it avoids the problem of custom PCR primer design and deployment mentioned earlier. However, this introduces a different problem, as up to 99.99% of the DNA/RNA in a typical biological specimen (e.g. saliva) is non-viral [35] (non-target) and most belongs to the host. Amplifying all DNA/RNA preserves this ratio, resulting in the vast majority of sequencing and computing time and cost stemming from processing non-target DNA/RNA.



Figure 3.1: Portable MinION sequencer in our laboratory.

In order to solve this needle-in-a-haystack problem, ONT sequencers have a feature called “Read Until” [36]. As reads (DNA/RNA fragments) are sequenced, they need to be analyzed in real-time. As soon as the computer classifies that the read is non-viral, the sequencer is instructed to eject it, which saves the time and cost of sequencing non-viral reads (99% of all reads). Unfiltered viral reads are used to construct the whole virus genome using reference-guided assembly (alignment and variant calling).

The MinION, however, does not have any on-board computing power to perform such secondary analysis. We analyze the performance of the Read Until bioinformatics pipeline for efficiently sequencing viral pathogens, and realize a portable computing solution that can be integrated with MinION.

We discover new performance bottlenecks that are not addressed by past genomics accelerators [37, 38, 39, 40, 41, 42, 43, 44]. In particular, we find that the Deep Neural Network (DNN) basecaller (software that translates MinION’s electrical squiggles to AGTC bases) dominates the computing time (96%). The aligner and variant caller, which have been the targets of recent accelerator research, constitute a much smaller fraction of compute. We also find that a current edge GPU is inadequate to keep up with the throughput of the MinION. Also, its high latency in classifying a read prevents us from taking advantage of the latency-critical Read Until feature of MinION.

Converting squiggles to bases using a compute-intensive basecaller, and then aligning to check if a read belongs to the target virus is needlessly expensive for classifying it. Instead, we skip the basecaller altogether by directly comparing each read’s squiggles to the precomputed expected signal profile of the target virus’s entire reference genome (the “reference squiggle”). By skipping the compute-intensive basecaller step, we improve efficiency significantly.

We present SquiggleFilter, a hardware/software co-designed filter which identifies non-target reads by directly comparing the real-time measured squiggles to the target virus’s precomputed reference squiggle. A classification decision is made based on the degree of match. We develop a custom subsequence dynamic time warping (sDTW) algorithm [45] to perform this classification. It includes solutions that improve accuracy by adaptively examining longer read prefix lengths when needed. It also includes customizations that result in area efficient hardware.

sDTW-based SquiggleFilter is significantly more efficient than a DNN-based basecaller, and its regular compute-bound characteristic makes it amenable for hardware acceleration. sDTW is a dynamic programming algorithm [46] whose complexity is proportional to the product of the length of the reference (R) and query (Q). Its regular memory access pattern allows us to build a fast and space efficient 1D systolic array accelerator for sDTW with a constant number of processing elements. Fortunately, we find that almost all epidemic viruses have genome references of length 50,000 (R) bases or smaller (see Figure 3.10) [47]. As a result, our accelerator can easily complete the classification in $\sim 2R$ cycles (forward and backward of reference strand), and still meet the strict latency requirement for leveraging Read Until.

Our work makes the following contributions:

- we demonstrate that basecalling is the computational bottleneck in the virus sequencing pipeline. Read alignment and variant calling – targets for prior accelerator work – are not the bottleneck.
- we identify direct squiggle alignment (first proposed in [48]) as a more efficient alternative to basecalling and alignment when enriching low-concentration viral specimens with Read Until.

- we propose multi-stage sDTW and several modifications to vanilla sDTW to realize an accurate and efficient hardware accelerator.
- we co-design a sDTW hardware accelerator to filter non-viral reads, for variable read prefix and almost all infectious viral genome lengths
- we demonstrate that this hardware, unlike current approaches, will enable Read Until to scale with rapidly increasing nanopore sequencing throughput
- we quantify accuracy and efficiency of our classifier using real-world metagenomic datasets, including datasets collected from our wet-lab experiments for Read Until.

Results: We design an edge device with compute capabilities similar to a Jetson Xavier System-on-Chip [49] consisting of SquiggleFilter, an edge GPU, and an 8-core ARM processor. We show that our proposed SquiggleFilter can accurately distinguish target viral DNA/RNA from background human DNA/RNA. We evaluate accuracy using non-contagious lambda phage virus data sequenced in our own lab. In terms of efficiency, we show that our SquiggleFilter accelerator has 274× higher throughput than the conventional software pipeline (using a basecaller) on an edge GPU while only consuming an area of 13.25mm² and power of 14.31W. SquiggleFilter’s throughput is 233.65M samples/s, which far exceeds the maximum throughput of 2.05M samples/s on a MinION [50], and is adequate to handle up to a 114× increase in MinION’s throughput in the future. The latency for classifying any read is 0.043ms, which is insignificant to Read Until decision’s critical path.

3.2 Background

3.2.1 Need for a Virus Detector

While SARS-CoV-2 was discovered – and its RNA genome sequenced – by early January 2020, it was not until several months later that mass testing was available worldwide. Figure 3.2 shows the steady increase in daily COVID-19 tests performed within the United States [51]. A widely

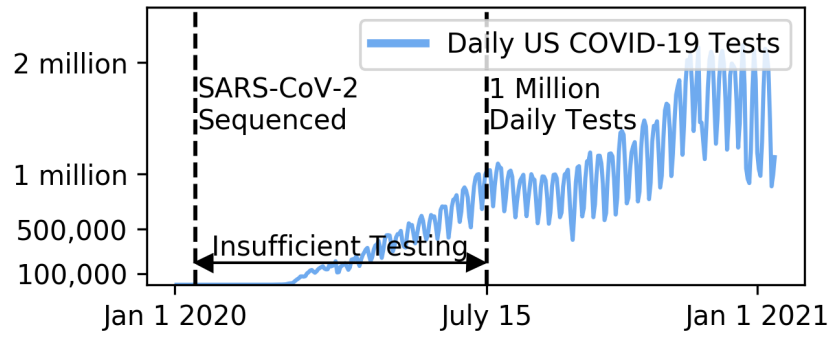


Figure 3.2: US COVID-19 testing timeline [51]

established global testing infrastructure would have helped control the spread of the virus early on, possibly saving hundreds of thousands of lives.

Given the increasing frequency of viral outbreaks, experts are concerned that it is only a matter of time before a new virus threatens the globe [33]. Thus, we need a virus testing technology that can be widely deployed *ahead-of-time*, and reprogrammed to detect and identify mutations in novel viruses as soon as they emerge.

In this work, we focus on controlling the spread of novel infectious viruses in their early stages, as soon as they are discovered and sequenced. Our goal is to enable a universal rapid test that can determine the whole genome of a target virus using reference-guided assembly. Targeting a specific virus enables us to make significant optimizations that help us reduce time and cost of sequencing and compute.

3.2.2 State-of-the-art Virus Detectors

Table 3.1 lists commonly used tests and ONT-based sequencing solutions for SARS-CoV-2. None of the methods except direct RNA or DNA sequencing are programmable, and therefore, are not effective in controlling the pandemic in its early stages. Antigen (paper) tests detect specific surface proteins on the virus. They are cheap, portable, and fast. However, they have low sensitivity and can only detect viruses present at high concentrations.

Molecular tests identify specific regions of interest in a virus’s genome and amplify this DNA if present in the specimen. Polymerase Chain Reaction (PCR) is a common technique used for

Tests	Diagnostic Power	Programmable	Time (min)	Cost (\$)
Antigen-based test				
Paper [52]	presence		15	5
Non-sequencing molecular test				
RT-LAMP [53][54]	presence		60	15
RT-PCR [55]	presence		120-240	<10
Sequencing based molecular test (30× coverage)				
ARTIC [56][55]	98 targets		305	100
LamPORE [57]	3 targets		<65	-NA-
RNA: 1% virus	whole genome	✓	240	110
0.1% virus [58]	whole genome	✓	1206	190
DNA: 1% virus	whole genome	✓	320	105
0.1% virus [59]	whole genome	✓	470	120

Table 3.1: A comparison of popular commercial virus detectors for SARS-CoV-2.

amplification. It has high sensitivity [60] but requires thermal cycling, which can be slow and expensive. LAMP (Loop Mediated Isothermal Amplification) is a more recent technology that obviates the need for a thermal cycler, but its primers are more complicated to design than PCR.

If amplification was successful (i.e., target DNA is present), it can be detected using fluorometry or colorimetry. Most clinical tests for SARS-CoV-2 stop here. However, by sequencing the amplified specimen, we can assemble portions of virus’s genome, depending on the number of targets amplified. ARTIC and LamPORE [57] amplify 98 and 3 genes respectively, and then use ONT’s nanopore sequencing.

Current solutions for virus detection use multiplex primer sets specific to a virus. Primer design is a complex, error-prone and time-consuming process [30] [31]. Thus, they are not an effective solution for early pandemic control. The COVID-19 pandemic highlights this problem, where designing and distributing target-specific primers was challenging, especially when supply chains broke amidst the pandemic.

An alternative to developing custom primers is to directly sequence the specimen following amplification with universal primers, which non-selectively amplify all DNA. This amplification step is required to increase the quantity of DNA, which greatly reduces average capture time (the

time required for a DNA strand to enter a nanopore) and therefore sequencing time. The wet-lab protocol followed, Sequence Independent Single Primer Amplification (SISPA) [62, 63], is universal and hence can be used on all RNA viruses. SISPA has four major steps: (1) RNA extraction, (2) complementary DNA generation, (3) PCR amplification, and (4) final sequencing specimen preparation.

A significant hurdle to SISPA-based sequencing is that following amplification, the specimen contains the genetic material of the target virus among a sea of human and bacterial DNA/RNA. The proportion of target virus DNA/RNA can be as low as 0.01% percent [35]. As a result, the time and cost of sequencing and data processing for this approach is significantly greater than that of custom primer-based solutions.

If this cost barrier can be overcome, this approach would enable detection of novel viruses without requiring months to develop and distribute virus-specific primers. Read Until can greatly increase the efficiency of sequencing by filtering out non-target reads using the virus's reference genome. Current Read Until approaches are limited by insufficient throughput, but our hardware accelerated SquiggleFilter ensures the future scalability of Read Until on higher throughput sequencers.

3.2.3 Portable MinION Sequencer

Oxford Nanopore Technology's (ONT) MinION offers multiple benefits that makes it a uniquely attractive solution for mobile and rapid virus detection.

Long reads: MinION sequencers are capable of measuring long strands of DNA, and can theoretically sequence any strand, regardless of length. The current world record stands at over 4 million bases [9].

Cost: The MinION only costs \$1,000, and offers affordable specimen preparation kits (\$100/use) and flow cells (\$125/use assuming 4× re-use). In comparison, it costs \$80,000-\$100,000 to purchase even the most affordable “Next Generation Sequencing” machines.

Real-time: MinION sequencers provide real-time, streaming output from the device. Stream-

ing signal output enables on-the-fly secondary analyses, and the ability to stop sequencing as soon as the desired coverage is reached.

Portability: A key feature that sets the MinION sequencer apart from all other sequencers in terms of wet-lab, sequencing and compute as shown in Figure 3.3. The portable compute, however, remains inefficient for real-time sequencing.

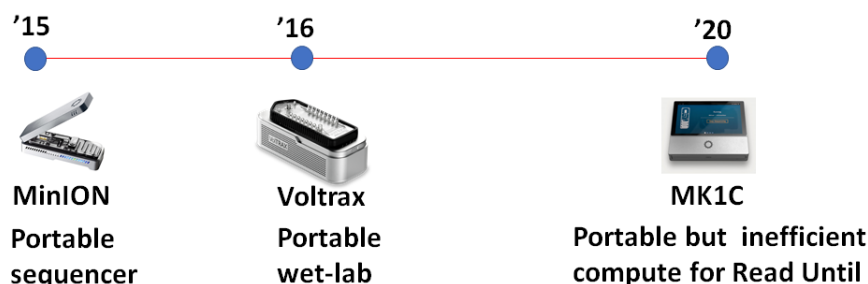


Figure 3.3: Portability of sequencing devices. Sequencing and wet-lab is portable. Compute, though portable, is insufficient for Read Until.

Target enrichment: An especially exciting capability of the MinION sequencer is “Read Until”, which ejects non-target DNA/RNA strands by reversing the electrical potential across the pore. This effectively enables digital enrichment of target DNA/RNA in low-concentration specimens.

However, a slow read classification results in wasted sequencing time. Currently, the MinION has no inbuilt computing power to make Read Until decisions. We additionally find that commodity GPUs are undesirable in terms of both throughput, latency and power.

3.3 Motivation

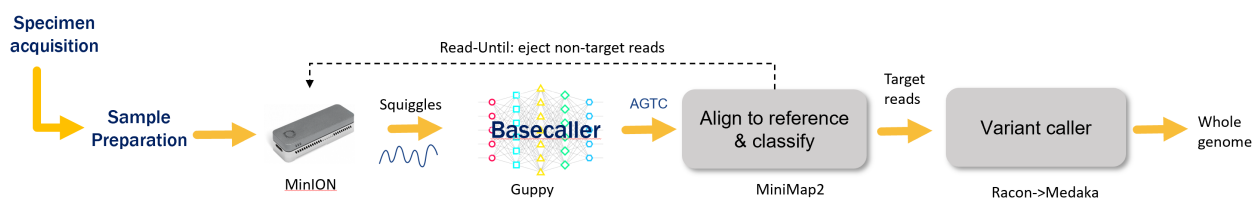


Figure 3.4: A Read Until pipeline for targeted reference-guided assembly of a virus genome.

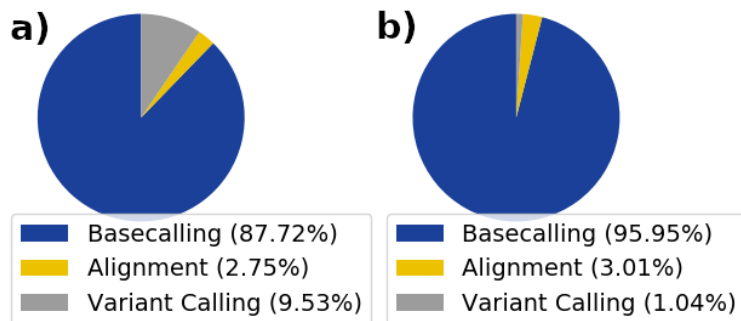


Figure 3.5: Runtime of each stage in a viral genome assembly pipeline. Basecalling is the bottleneck in a Read Until assembly of a SARS-CoV2 genome from specimens with (a) 1%, and (b) 0.1% viral reads.

Our goal is to build a cost and time efficient sequencing pipeline for determining the whole genome of a targeted virus, but without using custom primers for target amplification. We seek to reduce time and cost using the Read Until feature of Oxford Nanopore (ONT)’s palm-sized MinION sequencer.

To this end, we constructed a software pipeline using state-of-the-art bioinformatics tools and analyzed its performance. Our profiling results expose new performance bottlenecks that are different from those targeted in past accelerators for human genome sequencing [37, 38, 39, 40, 41, 42, 43, 44].

3.3.1 Bioinformatics Pipeline

The MinION sequencer measures electrical current signals that represent the bases (A, G, T, C) moving through each pore, recording approximately 10 samples for each base. All the active pores (up to 512 in the MinION) concurrently produce squiggles for the reads flowing through them. These squiggles can be analyzed in real-time as the reads flow through the pores.

Figure 3.4 illustrates the analysis pipeline for the squiggles. A *basecaller* translates squiggles into bases. The latest basecallers (such as ONT’s Guppy [64]) use compute-intensive DNNs, which must be large and deep to attain state-of-the-art accuracy. Guppy processes reads in chunks of 2000 samples, and uses five bidirectional LSTM layers for encoding followed by a custom CTC (Connectionist Temporal Classification) decoder. ONT provides two versions of its basecaller: a

high-accuracy version (Guppy), and another that trades off accuracy for performance (Guppy-lite).

In our Read Until pipeline, squiggles of a read are basecalled in real-time. After a short prefix of a read has been basecalled, it is then processed by an aligner (MiniMap2 [65]) that aligns the read to the target’s reference genome. If a good alignment is found, then the read is classified as a target and passed on to the next stage. Otherwise, a signal is sent to the MinION device, instructing it to eject the non-target read from further sequencing. Thus, the critical computing path for Read Until includes both the basecaller and aligner.

The target reads are collected and analyzed by a variant caller (Racon [66] followed by Medaka [67]). We seek to cover every position in the reference genome by 30 reads (30× coverage). The variant caller analyzes the reads piled up at each reference genome location, and identifies any genomic differences (“variants”) between the sequenced and reference viruses. As the variant caller is not involved in Read Until decisions, it is off the critical path.

3.3.2 Performance Bottlenecks

Figure 3.5 shows the performance bottlenecks of the bioinformatics pipeline (Section 3.3) used to assemble the whole SARS-CoV2 genome, evaluated on the CPU and GPU in Table 3.3. The results are shown for two representative biological specimens, one where the target viral reads constitute 1% of all the reads, and the other 0.1%.

We observe that a large fraction of computing time (96%) goes towards basecalling. This is in spite of using the more efficient, but less accurate, Guppy-lite.

Compute spent towards aligning (MiniMap2) and variant calling (Racon and Medaka) constitutes significantly smaller fraction, especially for specimens with low viral load (0.1%). In contrast, prior work on genomics accelerators targeted aligners and variant callers used for reference-guided assembly of human DNA [37, 38, 39, 40, 41, 42, 43, 44]. There are several reasons for this significant difference, discussed next.

All the reads are aligned to a target viral genome to classify them as target or non-target. This alignment step, however, is significantly less compute intensive compared to aligning to a human

genome, because viral genomes are much shorter ($\approx 30,000$ bases) than human DNA (3 billion bases).

Only a small fraction of target reads (1% to 0.1%) need to be processed for reference-guided assembly of a viral genome. Therefore, the variant caller is invoked only for a small fraction of sequenced reads. Also, given that viral genomes are shorter, we find that the variant caller does not consume much compute resources. Furthermore, the variant caller is not on the critical path for using Read Until, as it is not required for classifying reads.

We find that even a 250W Titan GPU has barely enough basecalling throughput (with low accuracy Guppy-lite) to keep up with a MinION's maximum sequencing throughput. An edge GPU (e.g., Jetson Xavier's) is several times slower than that, and therefore it cannot process all the sequenced reads in real-time to exploit the latency sensitive Read Until feature.

Sequencing throughput, however, continues to grow, as shown in Figure 3.6. Oxford Nanopore Technologies (ONT)'s GridION is only slightly larger, but has $5\times$ the sequencing throughput of a MinION. ONT announced in 2019 that they are working with MinION prototypes that provide $16\times$ sequencing throughput of MinION devices available in the market today. Within the next few years, they plan to release a production flowcell with $100\times$ greater throughput [68].

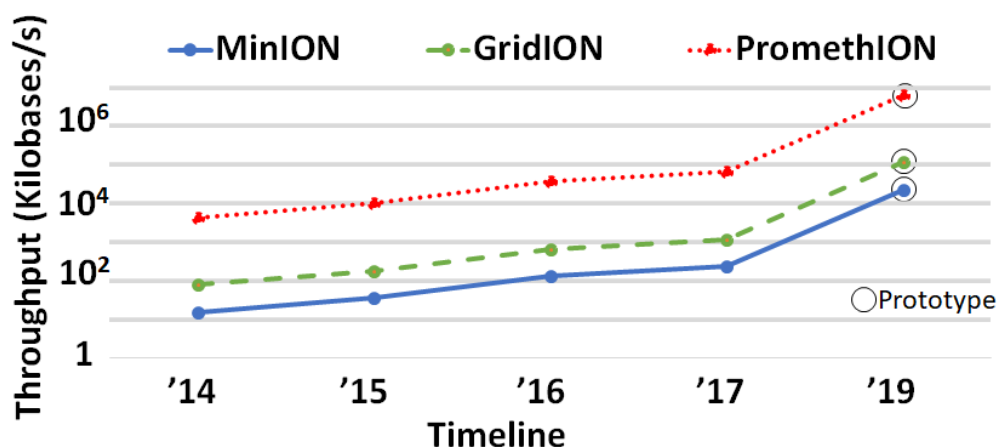


Figure 3.6: Projected increases in sequencing throughput, which is increasing exponentially [69].

Currently, the MinION does not have any on-board compute capability. Our goal is to map all the secondary compute analysis onto an edge system-on-chip so that it can be integrated with

the MinION. We address this growing computing need with our small, low-power accelerated SquiggleFilter, which greatly reduces the basecalling and alignment computation required for non-target reads.

3.4 Design

As discussed in Section 3.3, classifying a read being sequenced by analyzing its short prefix as target or not, in real-time, is the compute bottleneck. Additionally, basecalling for this classification consumes the most compute time.

Instead of using a basecaller (DNNs) and MiniMap2 aligner to classify a read’s prefix, we discuss SquiggleFilter’s algorithm that directly aligns each read’s electrical signals (query) to the target viral genome’s precomputed electrical signal (reference). As a majority of the reads are non-targets, we reduce latency and save much of the work done in basecalling and aligning these non-target reads.

SquiggleFilter aligns the query squiggle with a precomputed reference squiggle of the viral genome using a variant of the dynamic time warping (DTW) algorithm [70]. Recent work has eschewed sDTW due to its $\Theta(NM)$ complexity [71, 72, 73, 74], but we demonstrate that since both queries (read prefixes) and virus genomes are short, it is a practical solution for viral read enrichment. We further demonstrate its effectiveness on real sequencing data for a SARS-CoV-2 specimen.

Finally, we propose multi-stage sDTW filtering to improve efficiency, and discuss several improvements to conventional sDTW that help realize an efficient hardware accelerator.

3.4.1 Constructing the Reference Squiggle

In order to align raw signals to a reference genome, the known sequence of bases must first be converted to an expected current profile [75, 48, 76]. As a strand of DNA passes through a nanopore, the current measured is affected by 5-6 adjacent bases simultaneously. A lookup table is provided

by ONT which contains the expected current (in pA) for every possible combination of six bases (“6-mer”) [77]. This conversion is demonstrated in Figure 3.7, after which the expected signal is normalized using the mean and standard deviation.

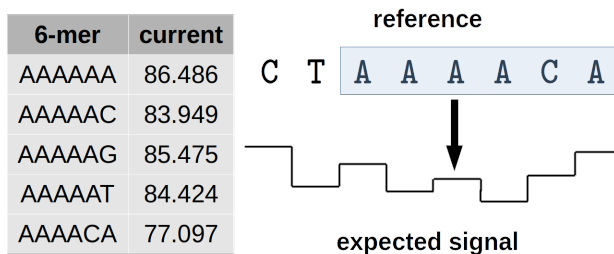


Figure 3.7: Generating an expected reference genome signal.

3.4.2 Normalizing Query Squiggles

Figure 3.8a shows a contrived minimal example of multiple raw nanopore signals corresponding to the same sequence of bases. Due to a variable rate of DNA/RNA translocation through the nanopore, these signals are out-of-sync (transitions between current levels do not occur simultaneously). Using Dynamic Time Warping (discussed next) solves this issue, and signals are aligned to the expected signal profile (shown in red in Figure 3.8b). Slight differences in applied bias voltages at each nanopore cause the measured currents to differ significantly, which is why normalization within each read is additionally helpful (Figure 3.8c).

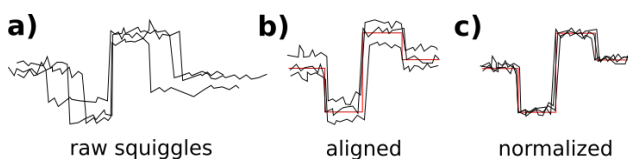


Figure 3.8: Normalization and alignment of query squiggles. **(a)** Three raw current measurements (“squiggles”) for the same sequence of bases. We then show squiggles aligned to the expected signal **(b)** without, and **(c)** with normalization.

3.4.3 Subsequence Dynamic Time Warping

Dynamic Time Warping (DTW) is a dynamic programming algorithm which is commonly used to align out-of-sync signals [45, 78]. Our filter applies subsequence DTW (sDTW), a slight modification of standard DTW which allows the entire query signal to align to any small portion of the reference, rather than forcing end-to-end alignment of both sequences.

The original sDTW algorithm works as follows for subsequence query Q of length N , reference sequence R of length M , and scoring matrix S :

```
def sDTW(Q,R):
    S = zeros(N,M)
    S[0,0] = (Q[0]-R[0])2
    for i in range(1,N):
        S[i,0] = S[i-1,0] + (Q[i]-R[0])2
    for i in range(1,N):
        for j in range(1,M):
            S[i,j] = (Q[i]-R[j])2 + min(
                S[i-1,j-1], S[i,j-1], S[i-1,j])
    return min(S[N,:])
```

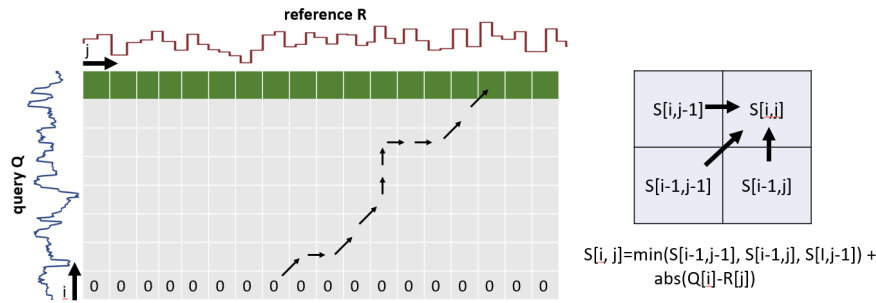


Figure 3.9: The subsequence dynamic time warping algorithm (sDTW).

The above algorithm dynamically computes all possible alignments of the query Q to reference R (keeping only the best ones) while allowing arbitrary many-to-one or one-to-many mappings between the two signal profiles. It is illustrated in Figure 3.9. Matrix S records a running tally of

the net squared differences between the two signals (using the best alignment of $Q[0 : i]$). At the end, $S[N, j]$ (highlighted top row in Figure 3.9) contains the alignment cost of Q to a subsequence of the reference $R[x : j]$, where x is the start of the best alignment ending at j . The minimum value in this row corresponds to the least squared difference in signal between alignments of the signal to the reference, and thus the cost of the optimal alignment.

3.4.4 sDTW for Virus Detection

The majority of viruses which are responsible for human epidemics have relatively small single-stranded RNA genomes [47], as is demonstrated in Figure 3.10. The two notable exceptions are

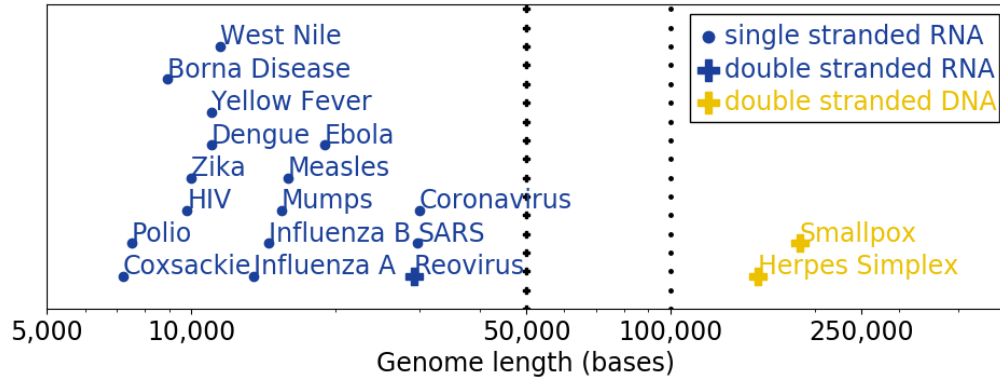


Figure 3.10: Epidemic virus genome lengths.

smallpox and *herpes simplex*, which have larger and more chemically stable double-stranded DNA genomes. Because most viruses have small genomes, we design our filter to operate on viruses with single-stranded genomes of length less than 100,000 bases. Equivalently, the filter works on viruses with double-stranded genomes less than 50,000 bases long. At such short reference genome lengths, it is computationally feasible to compare reads to the entire reference genome for filtering. This would not be a feasible solution for complex organisms such as humans, with genomes approximately 3 billion base pairs long.

3.4.5 sDTW is an Effective Filter

We seek to design a solution that is capable of detecting all strains of a particular viral species. It is therefore important that our filter is tolerant to variants in the sequenced genome relative to the reference genome used by our filter. We found that reference-guided filtering can be accurate regardless of viral strain, since the number of mutations between different strains is low. Table 3.2 presents the number of single base mutations between an assembled virus genome for several known SARS-CoV-2 strains, relative to the original Wuhan reference assembly [79]. No insertions or deletions were observed. Strains were defined using NextStrain’s [80] classification of all sequenced SARS-CoV-2 genomes into groups of shared ancestors, or “clades”, and data was sourced from the GISAID database [81].

Clade	Mut.	GISAID ID	Lab of Origin	Country
19A	23	593737	SE Area Lab Services	Australia
19B	18	614393	Bouake CHU Lab	Ivory Coast
20A	22	644615	Dept. Clinical Microbiology	Belgium
20B	17	602902	NHLS-IALCH	South Africa
20C	17	582807	Public Health Agency	Sweden

Table 3.2: SARS-CoV-2 strain comparison. There are few mutations between SARS-CoV-2 strains, relative to the Wuhan reference genome.

Since there are only a handful of mutations between various SARS-CoV-2 strains, the final sDTW alignment cost will not be significantly impacted. This cost is used to determine whether a given read aligns to the viral reference genome by comparing it to a constant threshold. If the alignment cost exceeds the chosen threshold, then the squiggle did not match well with any subsequence of the reference genome’s expected current profile, and the read can be discarded. Figure 3.11 shows that a static threshold can be used to distinguish between viral and human DNA fragments (discarding reads above the threshold and keeping reads below the threshold) even when only a few thousand signals have been captured. Due to the slight overlap in final alignment costs, some reads will be incorrectly classified when using a static threshold.

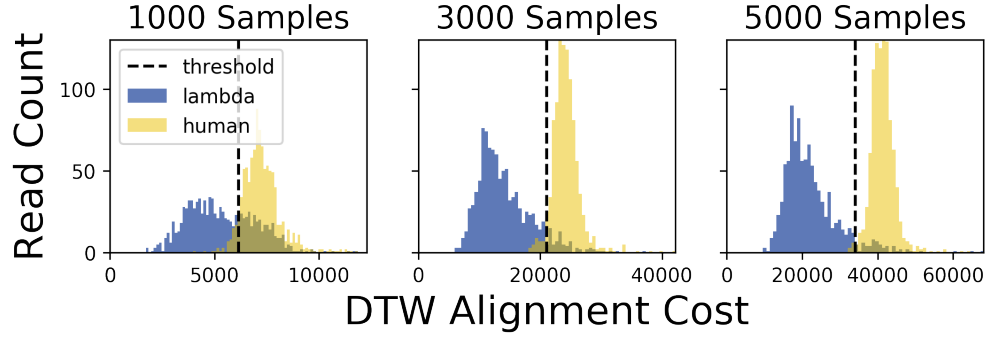


Figure 3.11: sDTW alignment cost distributions for human and lambda phage reads. Reads of 3 prefix lengths (1000, 3000, and 5000 current samples) were aligned to the lambda phage genome.

3.4.6 Multi-stage sDTW Filtering

We observed that as a read’s sequenced prefix length increases, the sDTW alignment cost is more accurately able to distinguish between target and non-target DNA (there is a decrease in overlap between cost distributions in Figure 3.11). However, waiting to make a Read Until decision increases the proportion of non-target DNA sequenced.

Therefore, instead of a single-stage filter that chooses a constant read length and threshold, we can filter in multiple stages. The first stage examines a shorter read length (e.g. 1000 samples), but chooses a less aggressive threshold that may let many non-target reads through. Non-target reads filtered and ejected using Read Until at this stage would be very short. If a read is retained, it is sequenced further. The second stage then examines the longer read prefix (e.g. 5000 samples), and filters using a more aggressive threshold. Intermediate results can be stored to avoid recomputation. In this way, several stages enable the classifier to filter a majority of non-target reads after seeing only a short prefix. Only reads with initial low-confidence are sequenced more before a decision is made. We have designed our hardware accelerator with this (optional) capability.

3.4.7 sDTW Algorithm Improvements

We propose several modifications to sDTW which help improve either our accelerator’s efficiency or accuracy of non-target read filtering.

Absolute Difference: We reduce hardware area and avoid multiplication by using $\text{abs}(Q[i]-R[j])$ as our distance metric instead of $(Q[i]-R[j])^2$.

Integer Normalization: Our solution uses 8-bit fixed point arithmetic during normalization, with no significant impact to classification accuracy (see Figure 3.18).

No Reference Deletions: Since the MinION averages 10 samples per base pair, it is unnecessary during sDTW computation for a single squiggle value to be able to align to multiple bases. We removed the possibility of reference deletions entirely from our dynamic programming computation, so that $S[i, j] = \text{abs}(Q[i]-R[j]) + \min(S[i-1, j-1], S[i-1, j])$.

Match Bonus: This final modification improves filtering accuracy. We found that reads with higher average translocation rates generally have higher alignment costs. To ensure sDTW alignment costs solely represent quality of alignment and are independent of translocation rate, we implemented a “match bonus” that rewards reads for matching additional reference bases, reducing the alignment cost for each matching base by a constant (10) scaled by the number of signals aligned to the previous reference base (thresholded to 10).

3.4.8 Need for an Accelerator

Despite the reduction in computation when compared to basecalling, sDTW alignment is still too slow to run on commodity hardware. sDTW alignment does avoid expensive floating point operations, instead requiring 8-bit integer comparisons and additions/subtractions. sDTW also has a smaller memory footprint (60,000 reference bases) compared to Guppy-lite (284,000 weights) when filtering SARS-CoV-2. Despite memory and operation complexity advantages, however, the number of operations required for sDTW (1,400 million) is greater than that of Guppy-lite (141 million). This is still more efficient than Guppy (2,412 million). In order to meet current and future MinION device requirements for Read Until, it is necessary to design an accelerator.

3.5 Accelerator

We present a System-on-Chip for reference-guided assembly of target viruses, shown in Figure 3.12. Its capabilities are similar to a Nvidia Jetson TX2, except for our SquiggleFilter accelerator. Our SquiggleFilter accelerator classifies and filters non-target reads, which constitute $\geq 99\%$ of all reads in most biological specimens. Thus, a large fraction of computing identified in Section 3.3 is handled by our SquiggleFilter accelerator. Furthermore, our accelerator enables low latency read classification, allowing us to use Read Until to eject non-target reads after sequencing only a short prefix.

Target reads (and any false positives) are processed off of Read Until’s critical path. Only these small fraction of reads need to be basecalled, aligned, and variant called. We find that we can perform these computations on an edge GPU (basecaller) and ARM processor (aligner and variant caller), and still construct the whole viral genome in approximately 10 minutes. Unfiltered non-target reads (false positives due to sDTW algorithm) will fail to align to the viral reference genome after basecalling, and so they will be discarded without affecting the accuracy of conventional reference-guided assembly. The final assembled genome and raw sequencing data is written to a 32GB eMMC 5.1 flash memory, which is sufficient to store one day’s worth of sequencing data.

We now present the 1D systolic array based SquiggleFilter accelerator for our squiggle-level classification algorithm discussed in Section 3.4. It can be programmed to target any novel viral genomes less than 100K bases. It supports variable query length. That is, it can classify read prefixes of different lengths, and thereby supports multi-stage filtering. The size of the systolic arrays and buffers are derived from our analysis of real-world metagenomic data.

3.5.1 SquiggleFilter Design

SquiggleFilter consists of 5 independent tiles (one tile is shown in Figure 3.13). Each can be individually power-gated based on desired filtering throughput. This number was chosen to meet the expected $100\times$ future increase in sequencing throughput. Each read is assigned to an available

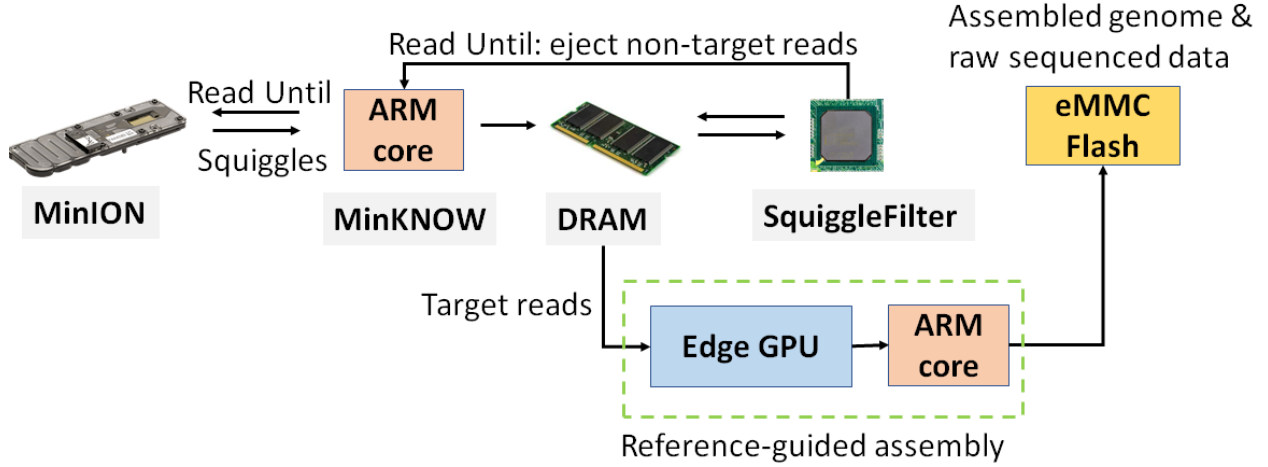


Figure 3.12: SquiggleFilter System-on-Chip integration with CPU and GPU. Our design contains the accelerated hardware filter on an ASIC, integrated with an NVIDIA GPU and an 8-core ARM v8.2 64-bit CPU

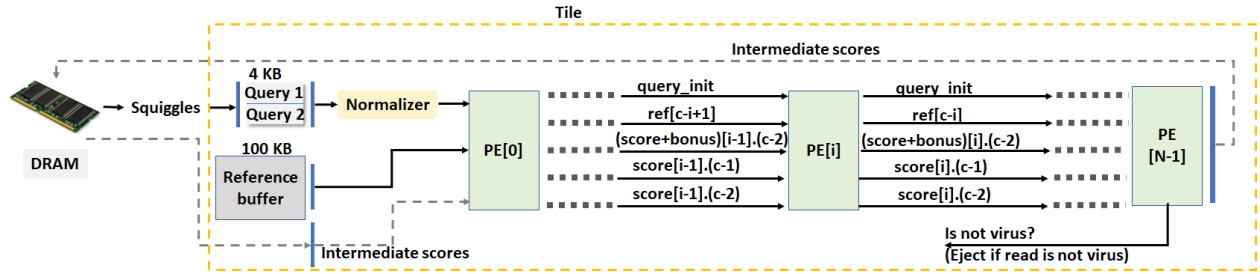


Figure 3.13: Architectural design of a SquiggleFilter Tile. $N=2000$ PEs are connected with streaming inputs and outputs. The last PE determines the classification by comparing its cost to a threshold every cycle. c is the cycle and i is the PE index.

tile for classification. As a read is sequenced, squiggles from a MinION R9.4.1 flow cell are streamed into DRAM in real-time. From there, squiggles are fetched into a tile's query buffers. Two ping-pong query buffers enable simultaneous squiggle loading and normalization. Once the desired length of read prefix has been sequenced, the raw squiggles of a query are normalized and then stored across the processing elements connected in a 1D systolic array.

Each tile also stores a copy of the precomputed reference signal (loaded from flash during an initialization phase) in a reference buffer. The reference samples are then streamed into the systolic array. The entire sDTW matrix is computed in a wavefront parallel manner as described in Section 3.4.7. The final PE determines the final minimum alignment cost, and sends a control

This allows us to reduce access latency and provide sustained throughput to each tile with just one read port. The area cost of duplicating the references is negligible, as reference buffers constitute only 6.98% of total tile area.

Furthermore, our design is independent of reference length and limited only by the reference buffer size provisioned. By loading a new precomputed reference signal onto the on-board flash, SquiggleFilter can easily be reprogrammed to detect a novel virus.

Variable Query Length: As discussed in Section 3.4.6, there exists a trade off between classification accuracy and sequencing length of queries. We find (Section 3.7.4) that read prefix length of 2000 samples yields the most savings using Read Until, when we use a single threshold. Therefore, we use a 1D systolic array of size 2000 PEs.

Our SquiggleFilter design can handle variable read prefix lengths that are multiples of 2000 squiggle samples. To support query lengths longer than 2000 samples and multi-stage filtering, we configure the last PE such that it can optionally write the sDTW costs every cycle to DRAM. This consumes significant memory bandwidth. However, it enables sDTW computation to continue if greater classification accuracy by analyzing a longer prefix is desired. These intermediate costs are then loaded from DRAM and used to initialize the PEs (similar to initial normalized query) prior to computing the costs for a 4000-sample prefix length.

3.5.2 Processing Element

Each PE computes a cell in the sDTW matrix every cycle, using the final algorithm described in Section 3.4.7. At cycle c , each PE (Figure 3.14) checks for the minimum among its previous neighbor's $c - 1$ and $c - 2$ cycle's outputs, modified by a bonus which rewards matching new reference bases. This minimum is then added to the absolute difference of the current query and reference values. Each PE stores the resulting costs and bonuses from its last two cycles for the next PE. Additionally, the last PE contains logic to compare its cost to a predefined threshold which determines whether or not to eject the read. This threshold can be reprogrammed on the SquiggleFilter based on software analysis of the target strain, but we have found it to be relatively

robust across species and sequencing runs. Each PE is $1203\mu\text{m}^2$ and requires 1.92mW when synthesized for a 28nm TSMC chip.

3.5.3 Normalizer

Normalization rescales the raw signals in order to improve classification accuracy when performing sDTW [82], as discussed in Section 3.4.2. The normalizer, shown in Figure 3.15, is a query preprocessor which streams in 10-bit samples from the query buffer for accumulation. After every $n = 2000$ samples, the normalizer updates the mean and Mean Absolute Deviation (MAD), defined as follows:

$$\text{mean} = \bar{x} = \sum_{i=1}^n \frac{x_i}{n} \quad \text{MAD} = \sum_{i=1}^n \frac{|x_i - \bar{x}|}{n}$$

Thereafter, the streamed-in samples are transformed with mean-MAD normalization. The output normalized value is filtered for outliers and then re-scaled to a reduced precision 8-bit integer which is then fed to the tiles for sDTW classification. We find that 8 bits of precision is sufficient for accurate classification (Figure 3.18). For efficiency, we do not convert the ADC sample to floating point, but instead use fixed-point values in the range $[-4, 4]$.

3.6 Methodology

Human DNA datasets containing MinION R9.4 and R9.4.1 flow cells were obtained from the Nanopore Whole-Genome Sequencing Consortium [83] and the ONT Open Datasets [84]. The SARS-CoV-2 dataset contains raw MinION R9.4.1 data available from the Cadde Centre [85]. We sequenced lambda phage DNA in our own laboratory using the ONT Rapid Library Preparation Kit [86] following the Lambda Control protocol with a MinION R9.4.1 flow cell.

We performed basecaller profiling measurements using a Titan XP GPU (server class) and Jetson Xavier GPU (edge class). Their specifications are provided in Table 3.3. We evaluated

both Guppy (`dna_r9.4.1_450bps_hac.cfg`) and Guppy-lite (`dna_r9.4.1_450bps_fast.cfg`) without modification using Guppy version 4.2.2 [64]. MiniMap2 version 2.17-r954-dirty [65] aligned basecalled reads.

First, we measured the basecalling throughput of Guppy and Guppy-lite on a dataset of 33,004 full-length reads. Next, we used the proprietary Python libraries `ont-fast5-api` version 3.1.6 [87] and `ont-pyguppy-client-lib` 4.2.2 [88] to basecall the same reads in chunks of 2000 signals, thereby simulating Read Until on the same dataset. The Python code was instrumented to record latency information, and we tuned the number of reads simultaneously in-flight to optimize performance. This online Read Until processing (due to smaller batch size) resulted in $4.05\times$ lower throughput for Guppy-lite and $2.85\times$ lower throughput for Guppy on the Titan XP. Using these measurements and the relative peak throughputs of the Jetson and Titan, the Read Until performance of the Jetson Xavier was estimated (necessitated by the unavailability of `ont-pyguppy-client` ARM binaries for fine-grained Read Until control on the Jetson).

	Edge GPU	Edge CPU	GPU	CPU
Model	Jetson AGX Xavier	ARMv8.2	Titan XP	2× Intel Xeon E5-2697v3
Cores	512 Volta	8	3840 Pascal	56
Clock	1377MHz	2265MHz	1582MHz	2600MHz

Table 3.3: Architectural specifications of evaluated GPUs.

A memory-efficient multi-threaded implementation of sDTW was written in Python for accuracy analysis, and tested on 1000 reads from each of the datasets mentioned above. In order to determine the relative benefits of Read Until using different classification latencies and accuracies, we developed an analytical model to estimate sequencing runtime. This model accounts for factors such as average read length, desired coverage of the reference genome, average DNA capture time, and the Read Until parameters mentioned previously.

The design was first functionally verified via emulation on Amazon Web Service’s EC2 F1 instance, which uses a 16nm Xilinx UltraScale+ VU9P FPGA. Further, SquiggleFilter was synthesized using the Synopsys Design compiler for 28nm TSMC HPC and the design is clocked at

2.5GHz. 32GB 256-Bit LPDDR4x is connected to the System-on-Chip along with an 8-core ARM v8.2 64-bit CPU.

3.7 Results

3.7.1 SquiggleFilter Hardware Synthesis

ASIC Element	Area (mm ²)	Power (W)
Normalizer	0.014	0.045
Processing Element	0.001	0.002
Tile (1×2000 PEs)	2.423	2.780
Query buffer	0.023	0.009
Reference buffer	0.185	0.028
Complete 1-Tile ASIC	2.65	2.86
Complete 5-Tile ASIC	13.25	14.31

Table 3.4: SquiggleFilter ASIC synthesis results.

Table 3.4 shows SquiggleFilter synthesized to a 13.25mm² ASIC that consumes 14.21W when performing single-stage filtering and clocks at 2.5GHz. It contains 5 fully-independent tiles (which could be individually power-gated to improve energy efficiency). The latency for classifying a 2000-sample read from SARS-CoV-2 is 0.027ms, and for lambda phage is 0.043ms, due to its longer reference genome. This adds insignificant latency to each Read Until decision’s critical path, since it takes around 500ms to sequence a sufficient number of bases to make an accurate decision. The single-tile classification throughputs for SARS-CoV-2 and lambda phage are 74.63M samples/s and 46.73M samples/s respectively, which are both considerably higher than MinION’s current maximum output of 2.05M samples/sec). Additionally, if each tile is configured to perform multi-stage filtering, it will write intermediate results to DRAM, consuming only 10 GB/s main memory bandwidth per tile. Since Jetson Xavier’s main memory supports 137 GB/s, our 5 tile design is feasible.

3.7.2 Performance Analysis

Latency: Figure 3.16a compares GPU-based basecalling latency to our SquiggleFilter accelerator’s latency. Note that we show only basecalling latency as it is the most time consuming step (96% of compute time) of the virus classification pipeline. The measurements demonstrate that it would be impractical to use the high-accuracy Guppy basecaller as its latency is greater than one second, in which time more than 400 bases would have been unnecessarily sequenced for non-target reads. We found that Guppy-lite provides sufficient accuracy for Read Until classification as downstream aligner MiniMap2 is able to account for incorrect basecalls when aligning reads. However, a 149ms basecalling latency for Guppy-lite translates to an additional 60 bases sequenced for each read during classification. Since most non-target reads can be discarded after around 200 bases, this overhead is significant. In comparison, the common-case 0.04ms decision latency of SquiggleFilter ensures that not even a single base pair is unnecessarily sequenced.

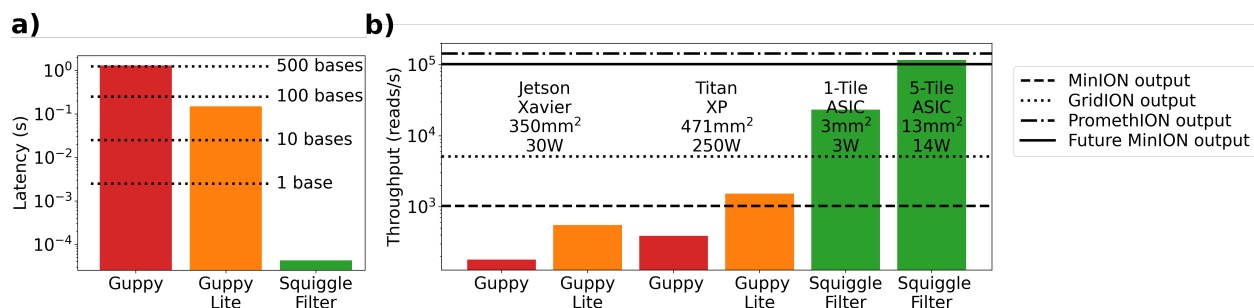


Figure 3.16: Latency and throughput comparisons of Guppy and SquiggleFilter. Results are shown for (a) latency and (b) throughput of Guppy, Guppy-lite and SquiggleFilter during Read Until.

Throughput: Figure 3.16b compares the basecalling throughput of Guppy-lite measured over GPU configurations to SquiggleFilter accelerator’s classification throughput. An edge GPU such as the Jetson does not have sufficient compute power to basecall data from all pores in real-time and keep up with the maximum sequencing throughput of the MinION. We calculated that the Jetson’s throughput would be approximately 95,700 bases per second, which is only 41.5% of the MinION’s maximum output of 230,400 bases per second. In the worst case, Read Until can only be performed using 41.5% of the MinION’s pores when basecalling using Guppy-lite on the Jetson.

The remaining 59.5% of pores are unable to use Read Until, and will sequence full-length human reads. In contrast, SquiggleFilter’s throughput far exceeds MinION’s and GridION’s sequencing throughputs.

3.7.3 sDTW Algorithm Accuracy

Figure 3.17a compares sDTW accuracy to basecalling and alignment on a dataset of 1000 lambda phage and 1000 human reads, with a line plotted for each prefix length. The MiniMap2 alignment quality and sDTW alignment cost thresholds (for determining which reads to sequence and which to reverse) are swept through the range of possible values to show threshold-dependent accuracies. Although the Read Until accuracy obtained by basecalling and aligning slightly outperforms sDTW, this is to be expected since alignment algorithms such as MiniMap2 use numerous scoring heuristics and have matured significantly over the past two decades [65].

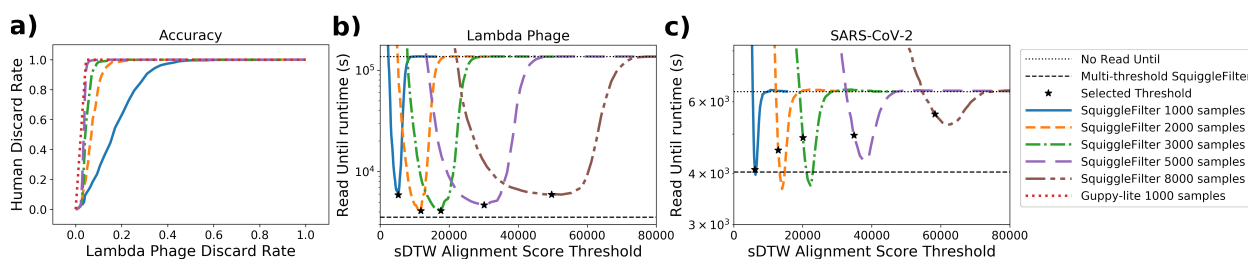


Figure 3.17: SquiggleFilter Read Until accuracy and performance. Results are shown for (a) accuracy on the lambda phage dataset, (b) performance on the lambda phage dataset, and (c) performance on the SARS-CoV-2 dataset.

Figure 3.18 shows the maximal F-score for all of our algorithm modifications and standard sDTW on the same dataset. As expected, accuracy generally increases along with sample prefix length. We found that using both integer normalization and absolute difference for our distance metric reduce filtering accuracy slightly, a compromise which was expected. Eliminating reference deletions results in a slight accuracy improvement. Combining all three of these optimizations results in the lowest accuracy (but most efficient) of all configurations tested. We find that by including our “match bonus”, we can recover lost accuracy and outperform the baseline, with a minor performance penalty. Figure 3.19 furthermore demonstrates that there is not a significant

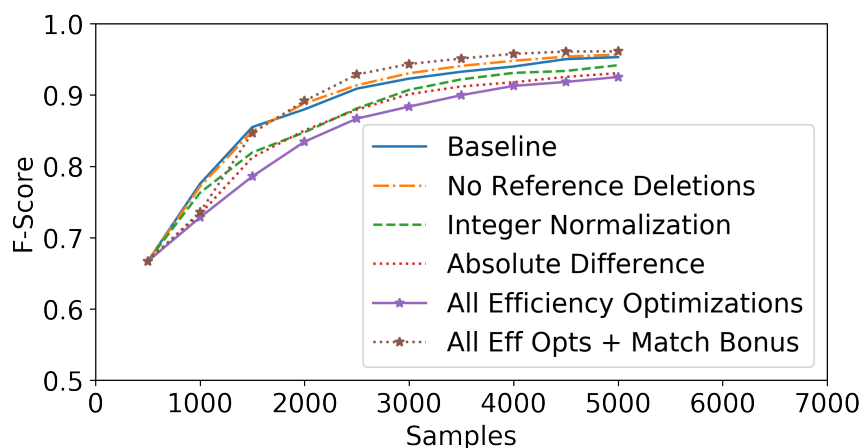


Figure 3.18: Accuracy results for modifications to the standard sDTW algorithm.

loss in filter accuracy until there is more than a 1,000 base difference between the reference genome and viral strain sequenced.

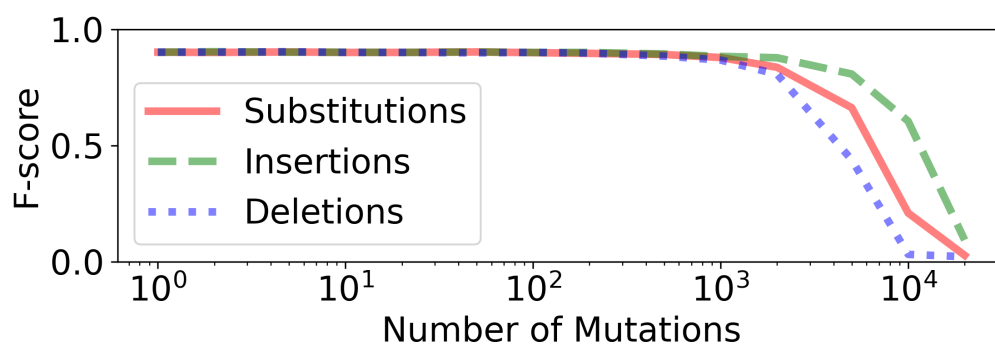


Figure 3.19: SquiggleFilter robustness against random mutations. We show that accuracy is robust against random lambda phage reference mutations.

3.7.4 Benefits of Read Until

Read Until not only saves sequencing time, but also cost. Figure 3.20 shows our wet-lab experiment. After sequencing for a while, washing the flow cell with nuclease and re-multiplexing (rapid alternations of pore voltage bias direction, shown with dotted black line) leads to control and Read Until pores having the same number of active channels. This means that Read Until does not damage the flow cell any more than normal sequencing, but enables more experiments to be run over the lifetime of any flow cell.

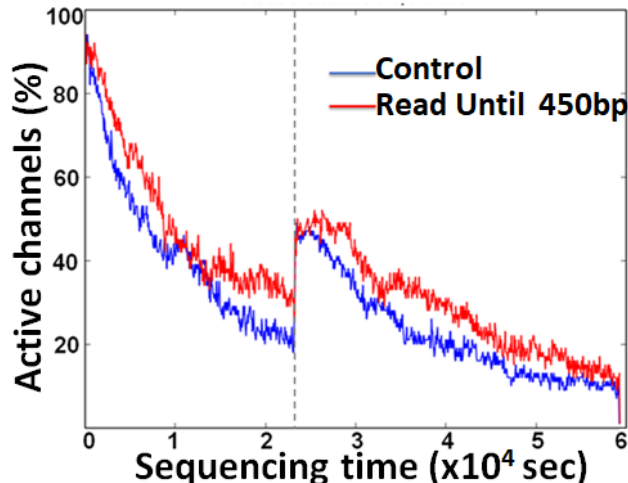


Figure 3.20: Flowcell lifetime comparison for Read Until vs ordinary sequencing. Because flowcell lifetimes are similar, sequencing time savings are directly proportional to flowcell cost savings.

The single-threshold Read Until design space was first explored for our lambda phage dataset. Figure 3.17a shows the accuracy of SquiggleFilter for a variety of Read Until prefix lengths (each line), and for all reasonable sDTW alignment cost thresholds (points on each line). Given this experimentally measured accuracy, the total expected sequencing time to perform Read Until for lambda phage was calculated using our analytical model, and is shown in Figure 3.17b. We found that the best single-threshold configuration for SquiggleFilter outperforms Guppy-lite on this dataset by 12.9% in terms of Read Until runtime. By using multiple thresholds, we can reduce runtime by a further 13.3%.

A similar analysis was then performed for the SARS-CoV-2 dataset, and the results are shown in Figure 3.17c. Optimal sDTW alignment cost thresholds were taken from the Read Until runtime minima from Figure 3.17b, and the corresponding Read Until runtimes using those thresholds are marked for the SARS-CoV-2 dataset.

3.7.5 Looking Forwards: Scalability

Sequencing throughput is expected to increase by 10 – 100× within the next few years, due to new nanopore chemistry enabling a denser configuration with many more channels per flow cell [68]. Figure 3.21 shows that without further improvements to basecalling throughput, current GPUs will

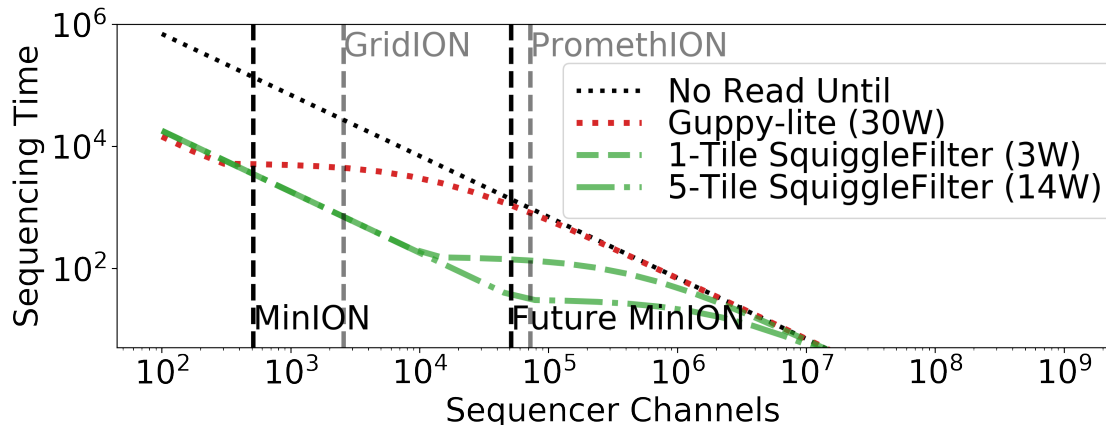


Figure 3.21: Projected SquiggleFilter Read Until scalability.

be unable to keep pace with new sequencing technology. As a result, the time and cost savings gained through Read Until will be largely lost. We can see that Guppy-lite’s slight edge over SquiggleFilter in terms of accuracy has already been lost due to its inability to perform Read Until on 512 pores. In contrast, our SquiggleFilter accelerator can tolerate a 114× increase in sequencing throughput.

3.8 Related Work

The MinION was released in 2014 as the first commercially available nanopore-based DNA/RNA sequencing device [89]. The first Read Until software pipeline was developed two years later, in 2016 [48]. In this seminal work, raw nanopore signal was first segmented into events, and then events were aligned to a lambda phage reference using subsequence Dynamic Time Warping (described in Section 3.4.3). Event segmentation is used to detect the most likely positions in the raw signal where a new base has entered the pore, and could be considered a rudimentary form of basecalling. In fact, it has been used as an essential preprocessing step in several older basecallers [64]. Unfortunately, the throughput measured by this original work on an 8-core ARM processor is 40× lower than the current maximum MinION output.

As basecalling throughput and accuracy has gradually increased over the last few years, the standard approach for Read Until pipelines has been to basecall the signal and use an aligner to

determine if each read aligns to the target genome [72, 73, 90, 71]. This method achieves the highest accuracy, but is not scalable. When pairing a server-class GPU with a handheld MinION device, it is just able to perform Read Until with the required throughput, albeit with significant latency (as shown in Section 3.7.2).

The standard basecallers such as Guppy and Guppy-lite, for which we provide comparisons above, are recurrent neural networks (RNNs) composed of gated recurrent units (GRUs). Performing inference on a network of GRUs requires many sequential operations. Recent basecalling research (in particular, Bonito [91]) has begun to instead favor convolutional neural networks (CNNs), which are inherently more parallelizable than RNNs. As a result, we can expect the next generation of basecallers to provide basecalls with lower latency. By training more lightweight CNN basecallers that just manage to achieve the minimum basecalling accuracy necessary for accurate species classification, it is possible that Read Until solutions requiring basecalling and alignment may become more competitive with sDTW-based solutions.

UNCALLED is a recently developed tool that skips basecalling by doing approximate alignments in 3 steps: event segmentation, FM-index look-ups, and seed clustering [74]. We evaluated UNCALLED and observed that it requires longer prefix lengths than SquiggleFilter for accurate alignment. 23.63% of 2000-sample long chunks from our lambda phage dataset were not alignable. After segmentation, UNCALLED uses an FM-index to filter reads. UNCALLED aligns only ~76% of the lambda reads of 2000 samples on a modern Intel i7-7700 desktop processor, taking 16ms per read. Moreover, ~14% of reads take 353ms per read to be aligned as more samples are required for a decision. ~10% of the reads are left entirely unaligned. On an edge device with an ARM core and lower memory bandwidth, performance would be worse. No existing software-only solution has adequate throughput and low enough latency to effectively perform Read Until on an edge device.

In contrast, our approach shifts to a minimalistic sDTW alignment algorithm, and by designing hardware to accelerate the simple and regular sDTW computation, we can easily meet the desired throughput and latency requirements on an edge device. General purpose DTW accelerators have

already been designed to solve alignment problems in other domains such as audio signal processing [92] and astronomy [82], but nanopore viral DNA/RNA filtering required several application-specific optimizations to meet the desired latency, throughput and accuracy requirements. Our design involves several algorithmic modifications to vanilla sDTW (described in Section 3.4.7), uses an on-chip buffer for efficient repeated alignments to the same reference, replaces all floating-point computation with integer arithmetic for increased efficiency, uses multi-stage filtering for optimal Read Until results, and has been evaluated on a novel virus (SARS-CoV-2).

There has recently been significant work on designing hardware accelerators for genomics applications [37, 38, 39, 40, 41, 42, 43, 44], but these accelerators focus on human genome sequencing. As a result, they efficiently align many (usually short) basecalled reads to a long reference genome with high throughput and accuracy. As noted previously in Section 3.3.2, our problem has very different computational needs. We must selectively filter short noisy raw signals (squiggles) with sufficiently high throughput and low latency to effectively exploit Read Until. We achieve this by replacing the basecaller and aligner with SquiggleFilter.

3.9 Conclusion

In designing a universal virus detector, we identify the basecaller to be a significant bottleneck in filtering non-target reads. This compute problem is only going to get worse, as the throughput of nanopore sequencers is expected to increase by 10-100 $\times$ in the near future. We address this problem using hardware-accelerated SquiggleFilter for filtering non-target reads without basecalling them. We show that our 14.3W 13.25mm² accelerator has 274 $\times$ greater throughput and 3481 $\times$ lower latency than existing approaches while consuming half the power, enabling Read Until for the next generation of nanopore sequencers.

3.10 Summary

This work, titled “SquiggleFilter: An Accelerator for Portable Virus Detection” was published in the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) [1]. The conference was a virtual event hosted in Athens in 2021. SquiggleFilter received an IEEE MICRO 2022 Top Picks Honorable Mention award. Thank you to all my co-authors who contributed to this work: Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, Reetuparna Das, and Satish Narayanasamy.

CHAPTER 4

nPoRe: Realigning Long Reads in Short Tandem Repeat Regions

Despite recent improvements in nanopore basecalling accuracy, germline variant calling of small insertions and deletions (INDELs) remains poor. Although precision and recall for single nucleotide polymorphisms (SNPs) now exceeds 99.5%, INDEL recall remains below 80% for standard R9.4.1 flow cells. We show that read phasing and realignment can recover a large portion of false negative INDELs. In particular, we extend Needleman-Wunsch affine gap alignment by introducing new gap penalties for more accurately aligning repeated n -polymer sequences such as homopolymers ($n = 1$) and tandem repeats ($2 \leq n \leq 6$). At the same precision, haplotype phasing improves INDEL recall from 63.76% to 70.66% and nPoRe realignment improves it further to 73.04%.

4.1 Introduction

4.1.1 Nanopore Variant Calling

As long read technologies have matured and basecalling accuracy has increased to over 99%, their popularity has grown accordingly [93][94]. Long reads are essential for spanning repetitive regions and unambiguously mapping reads. Last year, the first gapless human genome sequence was constructed by the T2T consortium by combining PacBio HiFi and ONT nanopore long reads [95].

Nanopore sequencing in particular has gained popularity due to its impressive read lengths, low cost, real-time results, and direct calling of base modifications[9, 75].

The two current leading nanopore variant callers are Clair3 (developed by the HKUCS Bioinformatics Algorithm Lab) and PEPPER-Margin-DeepVariant (a collaboration between UCSC and Google Health, hereafter referred to as PEPPER) [96, 97]. Both tools have converged on a similar variant calling pipeline: basecalling, read alignment, pileup-based variant calling (using pileup summary statistics), read phasing, and full-alignment variant calling (using all read information).

Despite posting impressive F1 scores (≥ 0.995) for SNP calling, nanopore variant callers struggle with accurately identifying INDELs in low-complexity regions [97, 98, 96]. Most recent nanopore variant calling advances in this area have come from improvements in machine learning and data representation. For example, the move from prior work Clairvoyante [99] to Clair [98] involved “*an entirely different network architecture and learning tasks*”. Clair3 then split the model into a pileup caller to filter out the noise and a higher-dimensional full-alignment caller to make the more difficult decisions [96]. PEPPER examined sorting reads by haplotype and a new architecture, and DeepVariant explored numerous possible data representations for final calling [97, 100]. Orthogonally, we show that improved INDEL calling performance can be achieved through better read alignment by introducing novel gap penalties for homopolymers and tandem repeats, or “*n-polymers*”.

4.1.2 Nanopore Read Alignment

In order to maximize the accuracy of pileup-based variant calling, reads should be aligned such that actual mutations are always aligned to the same location, despite sequencing errors. We find that simply using traditional affine gap penalties is not ideal because gap penalties G_{open} and G_{extend} are static, regardless of context [102]. For example, although our dataset consisted of only 0.8% INDEL errors, homopolymers of length 10 contained an INDEL error 41.8% of the time. Without lowering the INDEL penalty in the context of repetitive sequences, there is a mismatch between the likelihood and alignment penalty of common sequencing errors. This has an outsized impact

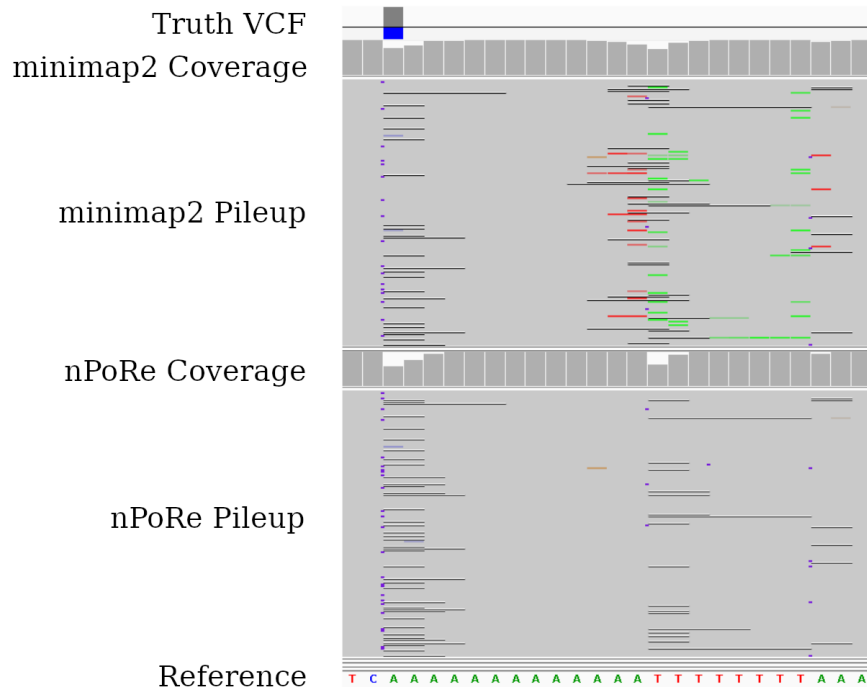


Figure 4.1: Pileup comparison of minimap2 and nPoRe, viewed in IGV [101]. Colored lines represent substitutions, black lines represent deletions, and purple vertical bars indicate insertions. Note that nPoRe alignments contain more INDELs than substitutions, and the starts of these INDELs are more consistently placed.

on fine-grained read alignment, often at the expense of consistently aligning actual mutations.

Figure 4.1 demonstrates a specific example where static INDEL gap costs cause poor alignment concordancy in low-complexity regions. Reads are identical in the two pileups shown; only the alignments differ. In this example, two adjacent homopolymers are basecalled with inconsistent lengths. nPoRe recognizes that these two events were most likely independent, and separates them into two homopolymer length mis-calls/variants. In contrast, minimap2 merges two INDELs whenever possible, or aligns homopolymer length differences as SNPs when one homopolymer is lengthened and the other is shortened, resulting in inconsistent alignment. According to the truth VCF, the first homopolymer of all As had a single deletion. Looking at the third base in the coverage graphs in Figure 4.1, we can see that nPoRe placed a deletion here for a much larger fraction of reads than minimap2.

The likelihood of incorrectly basecalling an INDEL within a homopolymer increases noticeably as homopolymer length increases. Figure 4.2a shows the confusion matrix for actual and

basecalled homopolymer lengths in our dataset. This same trend is visible for tandem repeats of longer length, though to a lesser extent (Figure 4.2b).

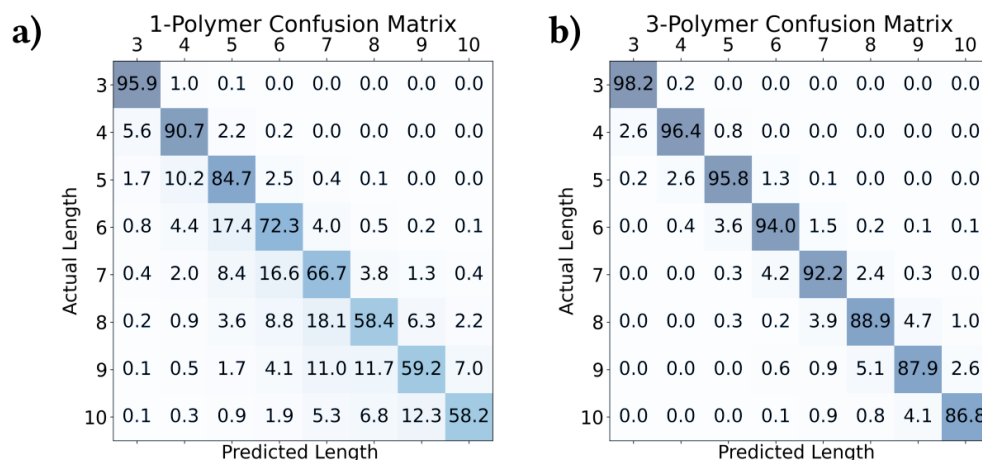


Figure 4.2: n -polymer copy number confusion matrices. Results are shown for (a) 1-polymer and (b) 3-polymer confusion matrices of actual and predicted n -polymer lengths (in percent, by row). “Actual” n -polymer lengths are the corresponding reference n -polymer lengths to which a read is aligned, and “predicted” n -polymer lengths are each read’s basecalled n -polymer length.

4.1.3 Nanopore INDEL Accuracy

Nanopore-based variant callers have historically struggled with INDELs, particularly with recall. Although Clair3 achieves 99.67% precision and 99.60% recall for SNP variant calling, it achieves only 90.86% precision and 64.73% recall for INDELs [96]. PEPPER v4 performs similarly, with 99.61% and 99.62% SNP precision and recall but just over 90% precision and 60% recall for INDELs [97]. The most recent evaluation available shows PEPPER v7 achieving 93% precision and 76% recall for INDELs, at 85× coverage [103].

Our own evaluation confirms these findings, and furthermore attributes the loss of INDEL recall to the first pileup-based variant calling step. Figures 4.3a and 4.3b show SNP and INDEL precision recall curves, respectively, for both Clair3’s pileup and full-alignment models. Note that although the more complex full-alignment model improves precision, it cannot improve recall much because only variant calls and low-confidence reference calls from the previous pileup-based step are considered.

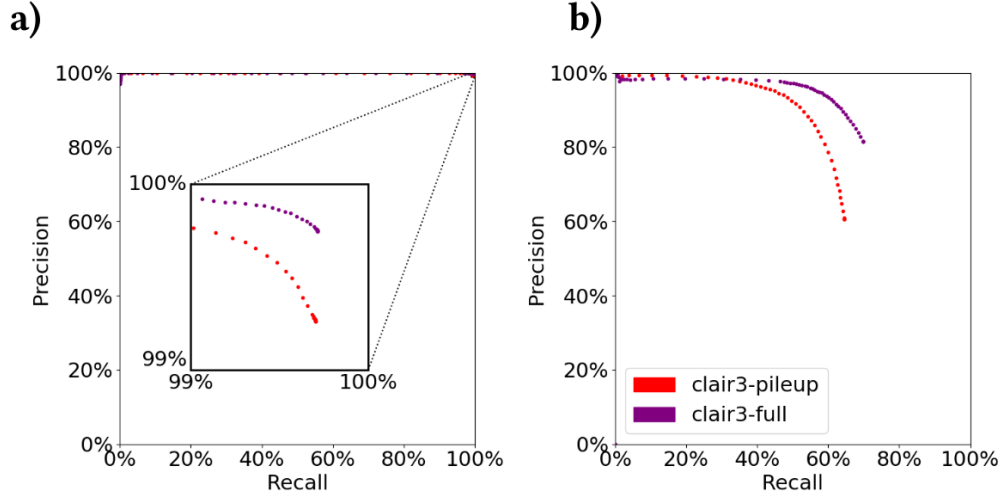


Figure 4.3: SNP and INDEL germline small variant calling accuracy. Results are shown for **(a)** SNP and **(b)** INDEL germline small variant calling accuracy of baseline `clair3-pileup` and `clair3-full` on chr20-22 of GM24385.

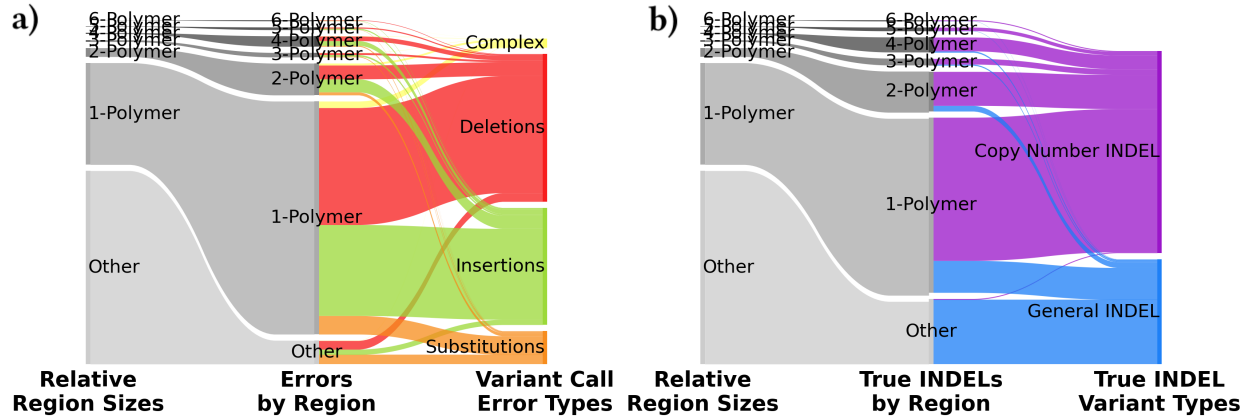


Figure 4.4: Proportion of true and false positive variant calls in n -polymer regions. Sankey diagrams demonstrate the **(a)** proportion of pileup-based variant calling errors, and **(b)** proportion of true INDEL variants contained within n -polymer regions of chr20-22 in GM24385.

Although substitutions comprise a majority (83.75%) of the actual small germline variants in our dataset, INDELs account for 92.36% of the pileup-based false negative and 80.79% of false positive errors. Figure 4.4a shows that of these errors, 92.29% occur within n -polymer regions, despite n -polymer regions covering just 37.07% of evaluated regions. By improving the alignment of reads in these small n -polymer regions, we can have a significant impact on overall variant calling accuracy ($p = 1.51 * 10^{-20}$, one-sided one-proportion z-test).

Ground truth INDEL mutations are over-represented in n -polymer regions as well (79.64%

of all INDELs). This is because Short Tandem Repeat (STR) variation is a common form of mutation due to strand slippage during DNA replication, resulting in one or more copies of a repeated unit being gained or lost. We define copy number INDELs as n -polymers (3+ exact copies of the same repeat unit), with a differing number of copies from the expected reference. For example, AAAA→AAAAA and ATATAT→ATAT meet this definition, but ATAT→ATATAT, AATAATAAAT→AATAAT, and ATATAT→ATATA do not. Despite our relatively strict definition of n -polymer copy number INDELs, however, 65.82% of all INDELs met this classification (Figure 4.4b). nPoRe’s algorithm is directly designed to reduce alignment penalties for n -polymer copy number INDELs and improve alignment in low-complexity regions.

4.1.4 Related Work

Variable gap penalties have been around for a long time. In 1995, Thompson first introduced per-position gap opening and extension penalties [104]. Since then, the sub-field of homologous protein sequence alignment has made extensive use of variable gap penalties (PIMA [105], FUGUE [106], and STRALIGN [107]) due to a high correlation between INDEL likelihood and the existence of protein secondary structures such as α -helices and β -strands. SSALN was the first to use empirically-determined penalty scores (an approach similar to our own) [108], and SALIGN greatly increased the flexibility of the gap penalty function, although with a corresponding increase in computation [109]. MarginAlign similarly used expectation maximization to obtain robust maximum-likelihood estimates for substitution, insertion, and deletion error rates, and then realigned reads for more accurate single-nucleotide variant calling [110]. Unfortunately, none of the numerous extensions these earlier works made to traditional Needleman-Wunsch alignment are directly applicable to the observed problem of long read n -polymer alignment.

Affine gap penalties belong to a larger class of “convex” gap penalties, which also includes piecewise linear and logarithmic gap penalties [111]. These more complex alternatives solve a different problem: reliably grouping several medium-sized gaps into one larger gap. They do this by decreasing the penalty for gap extension with the length of the gap, and are commonly used

for accurate alignment of large structural variants [112]. Such convex gap penalties do not solve the issue of fine-grained read alignment because they are still context-agnostic and at short INDEL lengths are highly similar to an affine gap penalty.

One known strategy to mitigate the effect of homopolymer length basecalling errors is “homopolymer compression”, in which repeated bases in a sequence are collapsed (GAAATCCT→GATCT) [113]. This method is commonly used by graph-based de novo assemblers in the earlier stages of graph construction to improve overlap detection between reads [114, 95, 115]. The recently-developed Verkko assembler goes even further, and compresses n -polymers (ATCATCATC→ATC) [115]. Although n -polymer compression is useful for building a consensus graph, the original reads are generally used to generate the final sequence [114, 115]. Read alignment following n -polymer compression is equivalent to running nPoRe with a null matrix N for n -polymer shortening and lengthening penalties. By defining a non-zero matrix N , this work penalizes copy number changes according to their measured likelihood.

Several existing works focus on the alignment of Short Tandem Repeats (STRs), although most function as INDEL variant callers rather than read realigners [116, 117, 118]. More recently, machine learning based approaches for final variant calling have outperformed these earlier statistical approaches [97, 100, 99, 67]. Several newer works do focus on read realignment, however. ReviS-TER is one such tool for revising mis-aligned/mapped reads through reference reconstruction with local assembly, though this is primarily helpful for improving mapping, not alignment. The Broad Institute has incorporated into their standardized analysis pipeline (Genome Analysis ToolKit, or “GATK”) an IndelRealigner, recognizing that INDELs are frequently mis-called as SNPs at read edges [119]. STR-realigner is most similar to our work. It flags STR regions and aligns them separately, allowing repeated traversal of STRs during alignment [120]. They find that this approach improves the consistency of read alignment in and near repeated regions, improving downstream variant calling. STR-realigner was designed for short reads, however, and its runtime $\Omega(n^2)$ which is perfectly fine for short reads of length 101bp is not unacceptable for long reads which regularly reach lengths upwards of 100kbp.

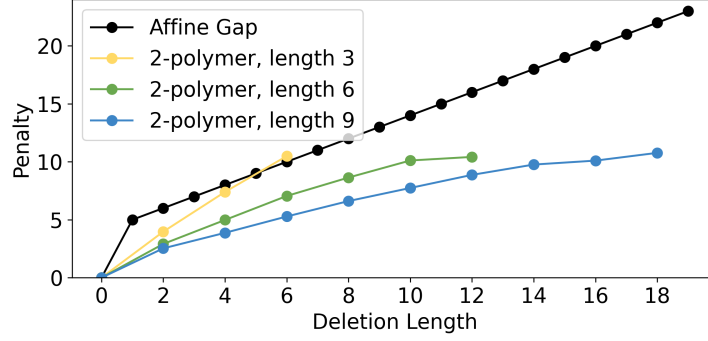


Figure 4.5: Comparison of an affine gap penalty and nPoRe’s CNV gap penalty. The penalty is dependent on the local repeat pattern’s periodicity ($n = 2$) and length ($l = 3, 6, 9$).

Our work introduces a variable gap penalty for n -polymer copy number INDELs, as shown in Figure 4.5. INDELs are more likely to occur in n -polymers, and so we provide a lower context-specific gap penalty, allowing only copy number INDELs. The exact sequence is not considered in this work; all 2-polymers of length 3 are scored the same (e.g. ATATAT and TGTGTG).

This work makes the following contributions:

- We show that context-agnostic affine/convex gap penalties do not accurately reflect the likelihood of nanopore sequencing errors in n -polymer regions
- We extend Needleman-Wunsch affine gap alignment to include context-dependent gap penalties for more accurately aligning n -polymers
- We identify that during germline small variant calling, most INDEL false negative errors occur during the pileup-based variant calling stage
- We introduce “follow-banding” for efficient read realignment
- We develop a VCF standardization method that ensures variants are reported in the same format as our nPoRe realigner
- We show that haplotype phasing and nPoRe realignment significantly improve pileup-based variant calling accuracy

4.2 Results

4.2.1 Overview

This work focuses on improving the accuracy of germline small variant calling (heritable mutations $< 50\text{bp}$ in size). We do so by realigning mapped reads (inputting and outputting in standard BAM format) to improve fine-grained alignment and read concordance by adjusting each read’s CIGAR string. Because we are concerned only with small variants, performing realignment within a $\pm 50\text{bp}$ window of the original mapping/alignment is sufficient. This work is independent of downstream variant caller, and nPoRe can be used in combination with either Clair3 or PEPPER. To evaluate nPoRe, we retrain Clair3 from scratch with minimap2- and nPoRe-realigned reads. We find that when retraining Clair3, it is beneficial to “standardize” the ground truth VCF to report variants in a manner similar to nPoRe-realigned reads (details in Methods). Realigning reads with nPoRe is relatively efficient and results in a significant increase in read concordance, which translates well to an improvement in final variant calling accuracy.

4.2.2 Accuracy

Figure 4.6 reports the performance of all three evaluated Clair3 pipelines, with precision and recall for SNPs and INDELs given separately for each sub-region. Results are reported for both the original and standardized ground-truth VCFs (see Methods). Figure 4.6 shows that *n*-polymer regions are responsible for the majority of INDEL errors, since with these regions excluded, INDEL precision and recall both exceed 95%. Performance in tandem repeat regions alone is relatively good, and homopolymers account for the majority of remaining errors. For a fixed INDEL precision of $2/3$, sorting reads by haplotype (clair3→clair3-hap) improves INDEL recall from 63.76% to 70.66%. Realigning reads with nPoRe (clair3-hap→clair3-npore-hap) further improves INDEL recall to 73.04%.

We chose to perform evaluations using both VCFs because although they contain the exact same information, the “standardized” VCF was more likely to report several INDELs instead of

several SNPs (due to the lower n -polymer shortening/lengthening penalty), and occasionally broke an INDEL up into several smaller INDELs. As a result, the standardized VCF had 18.05% more INDELs (31,104) and 1.45% fewer SNPs (155,163) than the original VCF (25,500 INDELs and 157,454 SNPs). `hap.py`'s `vcfeval` engine assigned partial credit for SNPs less frequently to nPoRe-aligned reads. The standardized VCF resulted in apparent higher INDEL recall for all variant callers, due to the increase in total INDELs.

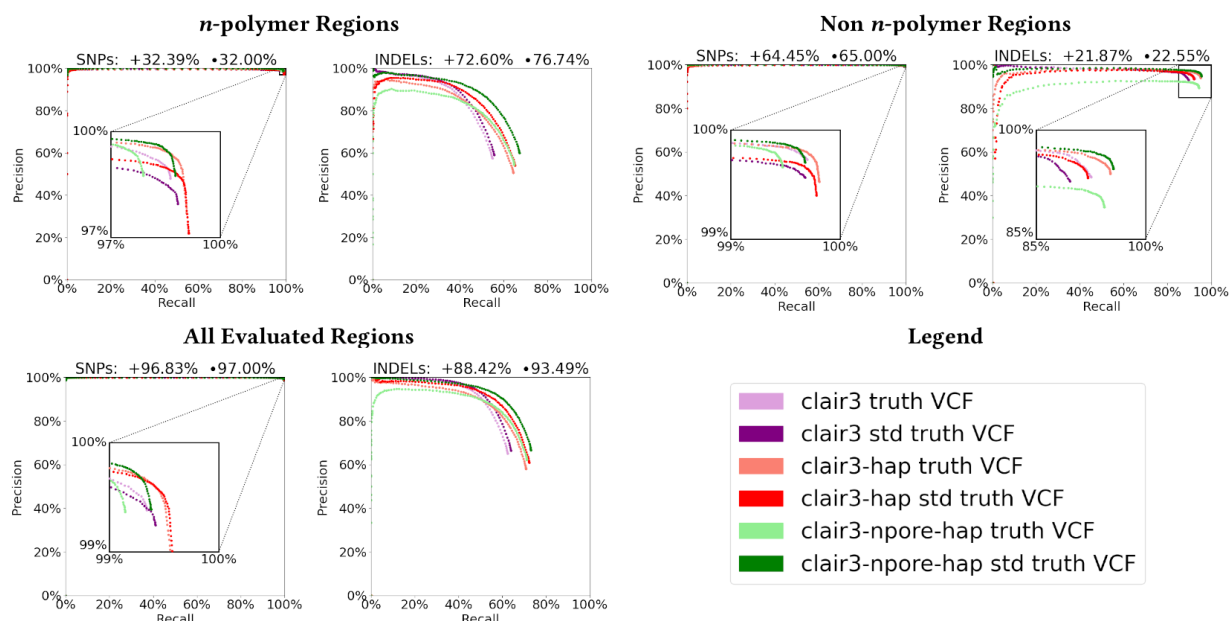


Figure 4.6: nPoRe accuracy results stratified by region. Results are reported for both the original (lighter colors, denoted +) and standardized (darker colors, denoted •) ground-truth VCFs, and titled with the percentage of SNP/INDELs within each region according to both truth VCFs.

4.2.3 Read Concordance

If sequenced reads were to not contain errors, they would all perfectly agree with one another and variant calling would be easy. We would like to maximize the extent to which reads agree with one another, which we term “concordance” and measure per-haplotype and per-position in terms of Gini purity. Gini purity is defined as $GP = \sum_{i=1}^N \mathbb{P}(i)^2$, where N is the number of classes and $\mathbb{P}(i)$ is the probability of class i . Figure 4.7a (lower graph) shows the resulting Gini purity histogram with the classes A, C, G, T, - (deletion) on a logarithmic y-scale. If all reads agree, $GP = 1$. If

50% call T and 50% call C, $GP = 0.5$. In the worst case, where there is an even split between the five classes, $GP = 0.2$. Reference positions with low Gini purity scores are therefore difficult to call, and are a likely source of both false positive and false negative variants. The lower graph in Figure 4.7 compares the Gini purity score distributions in minimap2 and nPoRe-aligned BAMs. It shows a marked $\approx 50\%$ decrease in positions with Gini purity less than 0.5 for the nPoRe-realigned BAM, demonstrating that nPoRe greatly improves alignment concordance across reads in difficult regions.

Read concordance in the phased BAM pileup, evaluated by Gini purity computed per reference position, is shown in Figure 4.7a. Insertion concordancy was evaluated separately, in Figure 4.7b, where the classes are all insertions between base k and $k + 1$ (e.g. $\epsilon, A, AA, AAA, AT, ATT\dots$). We plot insertions separately because the variable number of classes greatly affects the Gini purity score distribution. There appears to be an approximately 10% increase for all imperfect Gini purity scores, which we attribute to nPoRe's increased likelihood of calling INDELs (and as a result, more average classes and greater divergence).

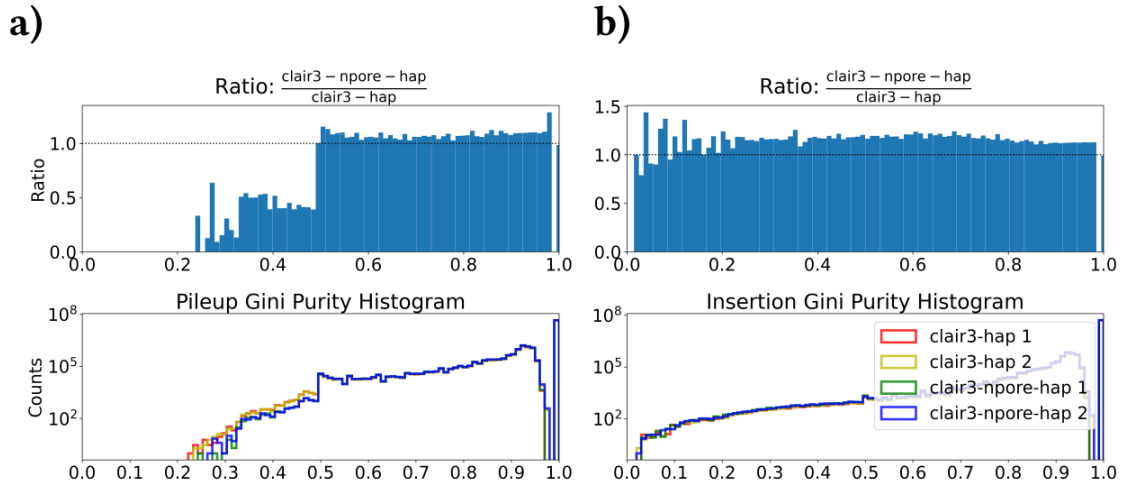


Figure 4.7: Aligned read concordance comparison for minimap2 and nPoRe. Gini purity histograms are shown for (a) pileup columns and (b) insertions.

4.2.4 Timing

We performed our evaluations on a system with 2× Intel Xeon E5 2697v3 2600MHz CPUs and 64GB total RAM. Timing results are shown in Table 4.1. From this evaluation, it is clear that any pipeline stages requiring computation on the full BAM file (marked with *) are considerably more expensive than working with just putative variants, a small fraction of the entire genome. Although our nPoRe realigner accounted for 79.6% of total CPU time, it only accounted for 27.8% of the real runtime, or just under twice as long as it took to index the BAM. nPoRe’s CPU time was 51.6× its real time on our system with 56 total cores, demonstrating that we took full advantage of the available parallelism.

Step	Real Time	CPU Time
*align reads	1,218:01	7,808:24
generate tensors	44:29	301:53
train <code>clair3</code>	38:18	54:22
call variants	86:26	2,392:17
phase variants	347:58	344:05
*phase reads	240:32	228:03
*index BAM	1,138:09	1,750:13
phase truth VCF	364:23	360:03
standardize truth VCF	546:31	3,756:46
*realign reads	2,168:15	111,792:55
*generate haplotype tensors	1,363:23	3,162:48
train model	41:40	66:29
call variants	200:08	8,343:01

Table 4.1: nPoRe pipeline runtime breakdown. Timing results are shown for each stage in the `clair3-npore-hap` pipeline (Table 4.3). Asterisks (*) denote steps for which the full BAM is required, rather than just the VCF.

4.2.5 Score Matrices

Figure 4.8 shows the calculated score matrices for 1- and 3-polymers, corresponding to the confusion matrices in Figure 4.2. In general, n -polymer INDELs are penalized less than the general-case affine gap INDEL penalty. Additionally, insertions are more common than deletions, and INDELs

are more common in n -polymers of shorter repeat unit length (n).

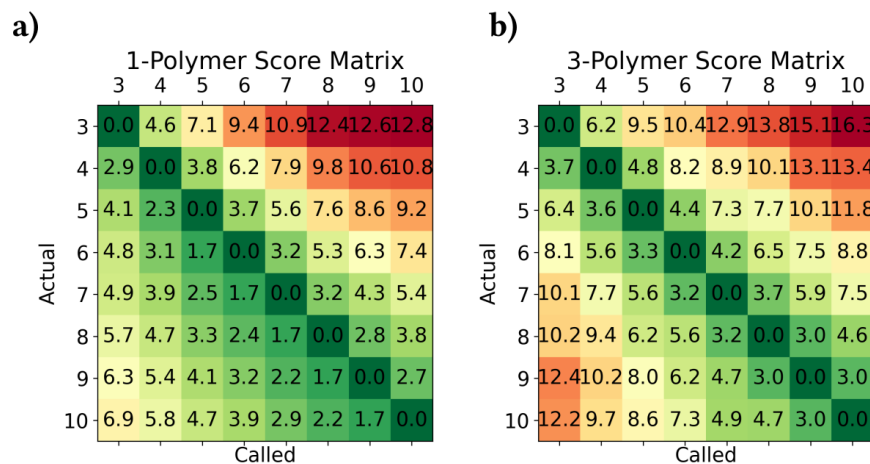


Figure 4.8: Example n -polymer penalty score matrices. Examples are shown for (a) 1-polymer and (b) 3-polymer score matrices. Scores are equivalent to the measured negative log probability of each error.

4.3 Discussion

The current nPoRe algorithm implementation was designed to demonstrate that there is a significant difference in INDEL rates between repetitive and non-repetitive sequences, due to the common occurrence of n -polymer copy number INDELs and sequencing errors. In order to do so, we decided upon a strict definition of n -polymers that requires at least three repetitions of the exact same repeat unit. We found that this strict definition includes around 65% of all INDELs in our dataset. Despite this, there are many repetitive regions in which sequencing errors are common but do not meet our strict definition of an n -polymer. For example, the sequence AAATAAAATAAATAAAT is not an n -polymer because the second repetition of AAAT has an additional A. A more lenient definition of n -polymers would result in a broader application of reduced INDEL gap penalties for repetitive regions and may improve alignment results further. Additional leniency, however, would come at the cost of increased computation.

We find alignment speed to be the greatest practical limitation of our nPoRe aligner, despite writing our alignment kernel in Cython and taking full advantage of the available parallelism. Ge-

nomics datasets are inherently large, and a hyper-optimized implementation with SIMD intrinsics and reduced data width may be necessary for large-scale applications. We have already explored reducing memory usage by shifting to a difference-based n -polymer cost matrix and only storing $2 * n_{\max} + 1$ matrix rows in memory, resulting in about 10-20% performance gains. Replacing the n -polymer cost matrix with a best-fit surface or function would likely improve efficiency further by reducing irregular memory accesses. We consider the main contribution of this work to be identifying fine-grained alignment as a significant source of small variant calling INDEL errors and developing an algorithmic solution. Speed can be improved through further engineering efforts.

The astute reader may notice that our n -polymer copy number INDEL penalties were calculated based on the measured negative log likelihood of occurrence in the original BAM, which as we’ve pointed out, has issues with fine-grained alignment. Even if this were to affect our estimate of n -polymer copy number INDEL likelihoods by $2\times$, however, the effect on INDEL penalty is only $\log 2 \approx 0.69$. Our algorithm has already reduced the cost of a 3-base deletion within a 3-polymer from $G_{\text{open}} + 2 * G_{\text{extend}} = 7$ to the range $[3.0, 3.7]$, depending on the 3-polymer length. If necessary, a second iteration of INDEL likelihood estimation using the nPoRe-realigned BAM could be used to further improve score estimation.

4.4 Conclusion

We identify the main source of nanopore germline small variant calling errors to be copy number INDEL false negatives in n -polymer regions, and show that context-agnostic affine gap penalties do not accurately reflect the likelihood of nanopore sequencing errors. To improve nanopore pileup-based variant calling accuracy, we explore correcting fine-grained read alignment. This work extends Needleman-Wunsch affine gap alignment to include repeat-aware gap penalties for n -polymers. In doing so, we also develop “follow-banding” for efficient long read realignment and a method for standardizing ground-truth VCFs. We demonstrate that read realignment improves

read concordance and variant calling accuracy, and release nPoRe¹ as an open source tool.

Despite being located in low-complexity regions, calling the length of tandem repeats is clinically relevant. There is an entire class of neuropathological disorders associated with copy number variation known as “Tandem Repeat Disorders”, or TRDs. Huntington’s Disease is one such disorder caused by 40 or more repeats of the CAG 3-polymer at the end of the gene *HTT*, instead of a normal 10-30 copies. Other TRDs include Fragile X Syndrome, Kennedy’s Disease, myotonic dystrophy, and several spinocerebellar ataxias [121]. Since nPoRe significantly improves read alignment and variant calling in tandem repeat regions ($p = 1.51 * 10^{-20}$, one-sided one-proportion z-test), it could aid in accurately diagnosing such disorders.

4.5 Methods

4.5.1 Overview

Because we have designed a read **realignment** algorithm, we trust the initial mapping of each read. Each read and its corresponding section of the reference genome are realigned, and a new traceback (alignment path) is computed. In other words, our solution simply adjusts the CIGAR string of each read within the input BAM file to better model the most likely mutations and sequencing errors in an effort to achieve greater concordancy between reads.

Our realignment algorithm is an extension of the Needleman-Wunsch algorithm for global alignment [122]. In addition to including known improvements such as an affine gap penalty and custom substitution penalty matrix [123], our algorithm allows the shortening and lengthening of homopolymers and tandem repeats (i.e. ACACAC→ACACACAC).

4.5.2 *n*-Polymer Repeats

The literature often categorizes sequences consisting of one repeated base as “homopolymers”, and repeated sequences of at least two bases as “tandem repeats” or “copolymers” [97, 21]. Short

¹<https://github.com/TimD1/nPoRe>

tandem repeats (STRs) are often defined as repeated units 2-6 bases in length, and are also known as “microsatellites” or “simple sequence repeats” (SSRs) [124]. Rather than treating these classifications separately for nPoRe, **we define an n -polymer to consist of at least 3 exact repeats of the same repeated sequence, where the repeat unit is of length 1 to 6 bases ($1 \leq n \leq 6, l \geq 3$).** For example, homopolymers such as AAAAAA ($n = 1$) and tandem repeats such as ACACACAC ($n = 2$) and TTGTTGTTG ($n = 3$) are n -polymers. Shorter or irregular repeated sequences such as AATTAATT and ACAACAAACAC are not.

An upper threshold of $n_{\max} = 6$ was selected because there is a marked decrease in the frequency of n -polymers for $n > 6$. Tandem repeats are usually defined with the same upper bound on repeat unit length n for the same reason. Figure 4.4 shows that 6-polymers are already uncommon. Furthermore, nanopore R9.4.1 sequencers fail to accurately call the length of n -polymers because the pore’s effective sensing width is 5-6 bases; i.e. the measured signal depends upon 5-6 adjacent bases simultaneously [77]. An n -polymer for $n \leq 6$ is usually observed as a nearly-constant signal due to exact repetition, from which it is difficult to determine repeat length. For n -polymers where $n > 6$, this is less of a problem, and fewer errors are observed. For a similar reason, a minimum n -polymer length of $l = 3$ was decided upon to classify a repeated sequence as an n -polymer. If $l = 2$, there is never a series of n bases bordered on both sides by another copy of the same n bases. The two copies of n bases are each adjacent to a non-repeating region, and as a result the measured signal is non-constant and few basecalling errors occur.

4.5.3 Penalty Functions

For each read, differences from the reference genome can be attributed to either sequencing errors or actual mutations. Regardless of origin (error or mutation), our goal is to align these reads to the reference in a manner that accurately captures the change that occurred. Existing aligners fail to do this by defining substitution and gap penalties based on estimated rather than measured rates of occurrence, and the algorithms do not account for common sequencing error modes such as tandem repeat length errors in nanopore sequencers. In contrast, we calculate penalties based on

frequency measurements from the input BAM file. We define the penalty score for each difference (whether error or mutation) to be the negative log likelihood of that event occurring. As a result, finding the minimum-penalty alignment path is equivalent to finding the most likely set of errors and mutations that have occurred (assuming independence).

Substitution Penalty Matrix. Figure 4.9 shows the calculation of substitution penalty matrix P from confusion matrix C_P , using Equation 4.1. $\epsilon = 0.01$ was included for numerical stability in the case that certain events were never observed. If we consider bases $x = \text{“ACGT”}$, then $P[i, j]$ is the negative log probability that base $x[i]$ was observed as base $x[j]$, either through a mutation or sequencing error:

$$P[i, j] \approx -\log \mathbb{P}(x[i] \rightarrow x[j]) \approx -\log \frac{C_P[i, j] + \epsilon}{\text{sum}(C_P[i, :]) + \epsilon} \quad (4.1)$$

		Basecall			
		A	C	G	T
Reference	A	2210	4.75	17.99	5.12
	C	4.88	2058	4.02	17.43
	G	17.31	4.03	2064	4.91
	T	5.17	18.23	4.86	2215

		Basecall			
		A	C	G	T
Reference	A	0.01	6.16	4.82	6.08
	C	6.06	0.01	6.25	4.78
	G	4.79	6.25	0.01	6.05
	T	6.07	4.81	6.13	0.01

Figure 4.9: nPoRe substitution and penalty score matrices. We show the **(a)** substitution confusion matrix C_P (counts in millions), and **(b)** the resulting penalty matrix P .

Affine Gap Penalties. Confusion matrices for insertions (C_I) and deletions (C_D) were first generated by measuring the occurrence of small INDELs in the input BAM. Both matrices are 1D, since the expected INDEL length is always zero. Then, penalties were calculated by determining the negative log probability of each INDEL length i occurring: $-\log \frac{C_I[i] + \epsilon}{\text{sum}(C_I) + \epsilon}$. From these penalties, a best-fit gap opening penalty G_{open} of 5 and gap extension penalty G_{extend} of 1 was selected for both insertions and deletions [123].

Tandem Repeat Penalty Matrix. First, confusion matrix C_N of shape $6 \times 100 \times 100$ was generated by comparing expected and observed n -polymer lengths l (up to 100). For each n , or repeat unit size 1 – 6, a penalty matrix was calculated using the following equation, where i is the expected repeat length and j is the measured repeat length.

$$N[n, i, j] \approx -\log \mathbb{P}(n, i, j) \approx -\log \frac{C_N[n, i, j] + \epsilon}{\text{sum}(C_N[n, i, :]) + \epsilon}$$

To improve penalty regularity, particularly for longer n -polymers where few examples were observed, the following two properties were enforced for each possible combination of $k > 0, n, i$ within the bounds of N :

- shorter INDELs are more likely:

$$N[n, i, i \pm k] > N[n, i, i \pm (k + 1)]$$

- longer n -polymers are more likely to contain an INDEL of a given size:

$$N[n, i + 1, (i + 1) \pm k] > N[n, i, i \pm k]$$

4.5.4 Reference Annotation

Reference annotations are used to track eligible n -polymers during alignment. For each possible n -polymer repeat unit length from $n = 1$ to $n_{\max}$, each reference position is annotated with l , the length or number of consecutive repeat units, and idx , the 0-based index of the current repeat unit ($0 \leq idx < l$). Table 4.2 shows example annotations for a short reference sequence for $n = 1$ and $n = 2$. Recall that in order for a sub-sequence to be considered an n -polymer, the pattern must repeat exactly at least three times. Annotations may overlap, and non-zero annotations are only placed at the start of every n -polymer repeat unit.

Reference: A T A T A T A T T T T T A A A G C G C G C	
n=1	<i>l</i> : 0 0 0 0 0 0 0 5 5 5 5 3 3 3 0 0 0 0 0 <i>idx</i> : 0 0 0 0 0 0 0 0 1 2 3 4 0 1 2 0 0 0 0 0
n=2	<i>l</i> : 4 3 4 3 4 3 4 0 0 0 0 0 0 0 3 0 3 0 3 0 <i>idx</i> : 0 0 1 1 2 2 3 0 0 0 0 0 0 0 0 0 1 0 2 0

Table 4.2: Example n -polymer reference sequence annotations.

4.5.5 Alignment

Before aligning read r to reference R , the reference is annotated with n -polymer information as discussed previously. Then, the five matrices D, I, M, S , and L are computed in lockstep one cell at a time, in that order. These matrices of size $|r| \times |R|$ represent the states Deleting, Inserting, Matching, Shortening n -polymers, and Lengthening n -polymers, respectively. For each cell, each matrix stores a tuple $(val, pred, run)$ containing the accumulated penalty *value*, in addition to the predecessor matrix and consecutive movements (*run*) within that matrix for backtracking purposes.

Figure 4.10 demonstrates the cell dependency patterns and penalties in greater detail. For example, when computing cell i, j in S , the reference annotations l and idx , are first retrieved for each n for $R[j]$. All dependencies (marked with $\bullet$ for S) in Figure 4.10 are considered, and the minimum value of these dependencies' cell values plus the associated penalties is calculated and stored in the result cell (marked with $\bullet$ for S). In other words, when looking at $S[i, j]$, for each n , we do:

$$S[i, j + n] = \min(M[i, j] + N[n, l, l - 1],$$

$$S[i, j - run] + N[n, l, l - 1 - run/n])$$

These two movements correspond to starting to shorten a tandem repeat (state $M \rightarrow S$), and continuing to shorten a tandem repeat (state $S \rightarrow S$). All movements into matrices S and L such as these are only allowed conditionally based on reference annotations (described in the following section). Note that if matrices $S\bullet$ and $L\blacktriangle$ are omitted (as well as all $\bullet$ and $\blacktriangle$ dependencies), this algorithm is equivalent to Needleman-Wunsch alignment with an affine gap penalty [122].

4.5.6 *n*-Polymer INDEL Conditions

Unlike matrices D , I , and M , the results for matrices S and L are stored several cells ahead of the current cell, and cell dependencies are only allowed conditionally based on the reference annotations. This ensures that matrices S and L only allow INDELs which change the copy number of tandem repeats and homopolymers. Here are the three conditions c_1, c_2, c_3 used by our algorithm, and referenced in Figure 4.10:

$$\begin{aligned}
 c_1 &= l > 0 && \text{start of repeat unit} \\
 c_2 &= l > 0 \text{ and } idx == 0 && \text{start of } n\text{-polymer} \\
 c_3 &= R[j + 1 : j + 1 + n] == \\
 &\quad r[i + 1 : i + 1 + n] && \text{next } n \text{ bases of } r \text{ match } R
 \end{aligned}$$

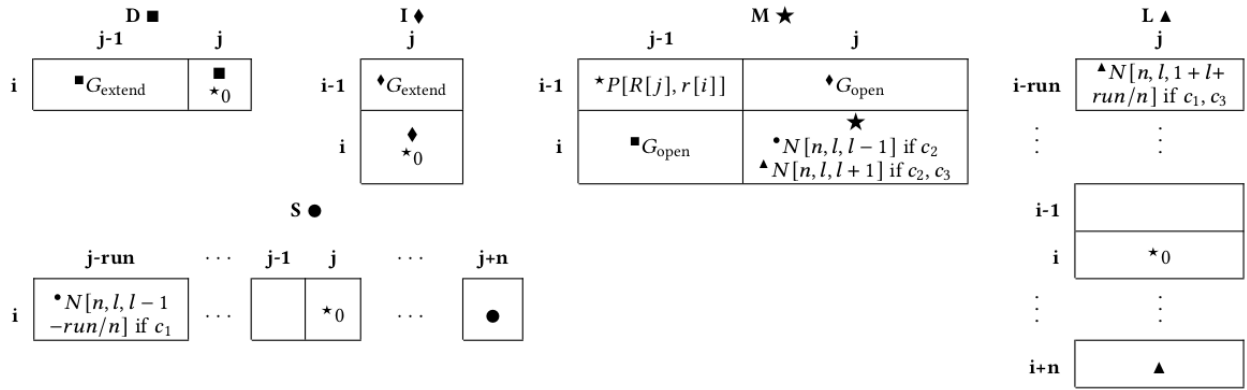


Figure 4.10: nPoRe realignment algorithm dependencies and penalties. Values and dependencies are shown for computing cell i, j in the five matrices D ■, I ◆, M ★, S ●, L ▲, which represent the states Deleting, Inserting, Matching, Shortening, and Lengthening, respectively. Computation for cell i, j occurs in that order, and large symbols (e.g. ■) denote where the result is stored. Superscript symbols (e.g. ■) represent cell dependencies, and a penalty score accompanies it. Each result is the minimum value of all dependency cells plus their accompanying penalty scores. n -polymer shortening and lengthening is only allowed if certain conditions are met (c_1, c_2, c_3), described in the text.

4.5.7 Backtracking

Traceback occurs entirely within matrix M , and relies on *predecessor* and *run* length information computed during the forward pass. The selected optimal alignment path is computed by Algorithm 1 and reported in the output BAM file in the form of a CIGAR string. CIGAR strings are

composed of the symbols M, I, and D. Reference Matches, Insertions, and Deletions, correspond to diagonal, vertical, and horizontal movements in the alignment matrix, respectively. An example alignment is denoted by • in Figure 4.11a. After computation, the computed CIGAR string is collapsed (MMMDMIMM→3M1D1M1I2M).

Algorithm 1 The nPoRe backtracking algorithm.

```

i, j ←  $|r| - 1, |R| - 1$ 
cigar ← ""
while i > 0 or j > 0 do
  (val, pred, run) ← M[i, j]
  if pred == M then
    if r[i] == R[j] then
      cigar += "="
      i, j ← i - 1, j - 1
    else
      cigar += "X"
      i, j ← i - 1, j - 1
    end if
  else if pred == I or pred == L then
    cigar += "I" * run
    i ← i - run
  else if pred == D or pred == S then
    cigar += "D" * run
    j ← j - run
  end if
end while
return cigar[::-1]

```

4.5.8 Follow-Banding

As mentioned previously, the primary goal of our read realignment algorithm is to more accurately model the mutations and sequencing errors in fine-grained alignment. Therefore, we can skip read mapping and trust the read start position reported by the previous aligner in the input SAM/BAM file's POSition field. Read *r* will be aligned to a small subsection of the reference $R[\text{POS} : \text{POS} + m]$, where *m* is the length of the reference sequence corresponding to *r*. Barring any large INDELs, $m \approx |r|$, and we transform our alignment problem from $O(|R||r|)$ to $O(|r|^2)$.

Moreover, we can use the SAM/BAM file’s existing CIGAR string to simplify our alignment problem even further. Our optimal alignment will likely follow a path close to that of the original alignment. Figure 4.11a demonstrates how we can compute the alignment matrix (new optimal path denoted by $\bullet$) in a narrow band $b = 1$ that follows the original alignment (red cells). Dark gray and black cells are not computed. Essentially, we use the CIGAR string to precompute the movement directions for adaptive banded alignment as proposed by Suzuki and Kasahara [125], instead of using a heuristic comparing penalty scores on the band’s edge.

Firstly, all M CIGAR operations are converted to ID, an insertion followed by a deletion. This change is shown in Figure 4.11, supplementing the original red alignment path with light gray cells. Next, computation proceeds one anti-diagonal row of width $2b + 1$ at a time, centered on the alignment path. The computation of anti-diagonal rows shifts either right or downward at each step, governed by the previous CIGAR operation, D or I. These anti-diagonal rows can be stored efficiently in matrix format, as demonstrated in Figure 4.11b. Transforming the banded $|r| \times |r|$ matrix A to a $(2b + 1) \times 2|r|$ matrix B saves significant space because nanopore sequencing read lengths $|r|$ can be up to several million bases [9], while realignment works well with a band width of $b = 30$.

Offset arrays INSS and DELs can be precomputed using the CIGAR (Figure 4.11). Given a cell in matrix B with indices i, j , its position in matrix A can be computed using the following formula:

$$\text{row} = \text{INSS}[i] + b - j \quad \text{col} = \text{DELS}[i] - b + j$$

4.5.9 Time and Space Complexity

Reference annotations. The worst-case time complexity for computing the reference annotations is $O(|R|n_{\max}^2 l_{\max})$, where $|R|$ is the length of the reference R , and $n_{\max}$ is the maximum n -polymer considered, and $l_{\max}$ is the maximum n -polymer length. Since our n -polymer score matrix N is of size $(6, 100, 100)$, $n_{\max} = 6$, and $l_{\max} = 100$. Thus, the time complexity is effectively $O(|R|)$.

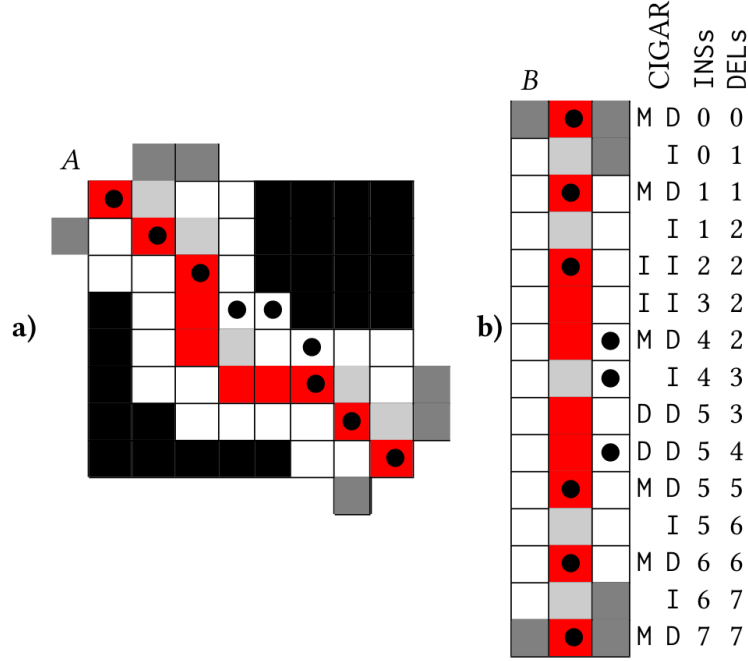


Figure 4.11: The nPoRe follow banding matrix transformation $A \rightarrow B$. Red cells mark the original alignment path. Light gray cells mark supplements to the original alignment path, with diagonal Matches split into a horizontal Deletion plus vertical Insertion. ● marks the new alignment path. Black and dark gray cells are not computed due to being out of band and out of bounds, respectively. Rows in B correspond to banded diagonals of A , centered on the supplemented alignment path.

Furthermore, these annotations must only be computed once, and cost can be amortized over all the reads that are aligned to the reference. We found the time required for reference annotations to be insignificant compared to alignment. These annotations require $O(|R|n_{\max})$ space.

Read alignment. Once the reference annotations and score matrices have been computed, the nPoRe algorithm requires $O(|R||r|)$ time for each read r . The only additional overhead nPoRe incurs over Needleman-Wunsch with affine gaps is computing five Dynamic Programming (DP) matrices instead of three, as well as computing the new cell dependencies. All new penalties are conditional $O(1)$ lookups. As discussed earlier, follow-banding further reduces both the time and space complexity of alignment from $O(|R||r|)$ to $O(b|r|)$, where b is the band width. In total, the cost of aligning all reads is $O(\sum_{i=1}^m b \times |r_i|)$, where m is the number of reads, or equivalently $O(db|R|)$, where d is the average depth of coverage.

All our code is open source and readily available at: <https://github.com/TimD1/nPoRe>.

4.5.10 Datasets

Reference. We used the GrCh38 reference from the Genome-In-A-Bottle (GIAB) consortium [21].

Reads. We obtained our FASTQ files from *ONT Open Datasets*’ May 2021 re-basecalling of HG002 PromethION R9.4.1 data using Guppy 5.0.6. Specifically, we used flow cell PAG07162, prepared using the Short Read Eliminator (SRE) protocol [126]. Depth of coverage was approximately 60×. For training, we used chr1-chr19, and for testing we used chr20-chr22.

Stratification Regions. Stratification BED regions were calculated for $n = 1 \dots n_{\max}$ using the definition of n -polymers provided previously. Regions were extended by a single base on each side (slop=1) to include variants occurring at the edges of n -polymer regions. These BEDS were then merged and complemented as necessary to create stratification BEDS for all n -polymer regions and non n -polymer regions.

Availability of data and materials. The FASTQ dataset supporting the conclusions of this article is available in the [ONT Open Datasets](#) repository. For chromosomes $N = 1$ to 22, we downloaded: s3://ont-open-data/gm24385_2020.11/analysis/r9.4.1/20201026_1644_2-E5-H5_PAG07162_d7f262d5/guppy_v4.0.11_r9.4.1_hac_prom/align_unfiltered/chrN/guppy_v5.0.6_r9.4.1_sup_prom/basecalls.fastq.gz. Specifically, we used flow cell PAG07162 from the May 2021 re-basecalling of HG002 PromethION R9.4.1 data using Guppy 5.0.6; more details regarding this data can be found [here](#).

We used the GrCh38 reference FASTA from the Genome-In-A-Bottle (GIAB) consortium [21], also made available through ONT Open Datasets: s3://ont-open-data/gm24385_2020.09/config/ref/GCA_000001405.15_GRCh38_no_alt_analysis_set.fasta.

The ground truth VCF and BED files were provided by GIAB at the following link: https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/AshkenazimTrio/HG002_NA24385_son/NISTv4.1/GRCh38/.

The source code for nPoRe is available at <https://github.com/timd1/npore>, and archived through Zenodo: <https://zenodo.org/record/6260902>. nPoRe is written in Python/Cython and is available cross-platform through a public Docker container (<https://hub.docker.com/r/timd1/npore>) under the

4.5.11 Pipeline

Step	clair3	clair3-hap	clair3-npore-hap	Program
align reads	✓	✓	✓	minimap2
generate tensors	✓	✓	✓	clair3/CTP.py
train clair3	✓	✓	✓	clair3/Train.py
call variants	✓	✓	✓	clair3/run_clair3.sh
phase variants		✓	✓	whatshap phase
phase reads		✓	✓	whatshap haplotag
phase truth VCF		✓	✓	whatshap phase
standardize truth VCF			✓	nPoRe/standardize_vcf.py
realign reads			✓	nPoRe/realign.py
generate haplotype tensors		✓	✓	clair3-hap/CTPHaps.py
train model		✓	✓	clair3-hap/Train.py
call variants		✓	✓	clair3-hap/run_clair3.sh
evaluate variants	✓	✓	✓	hap.py

Table 4.3: An overview of the Clair3 training and evaluation pipelines.

The full training and evaluation pipelines for all three Clair3 configurations tested are shown in Table 4.3. We used minimap2 version 2.17-r954-dirty [65], clair3 version v0.1-r9 [96], whatshap version 1.0 [127], and hap.py version v0.3.14 [21]. All three variant callers were trained from scratch using our 60× HG002 dataset on minimap2-aligned reads for chr1–chr19, and tested on chr20–chr22. We first extended the retrained Clair3 baseline (clair3), phasing the input reads by haplotype and training a phased pileup candidate caller (clair3-hap). This was done because it significantly improves read concordancy, and leaving reads unphased when calling difficult single-haplotype variants might overshadow concordancy improvements gained by nPoRe’s alignment algorithm. This second baseline enables us to clearly delineate the gains from haplotype phasing and our nPoRe alignment algorithm. The final configuration was

`clair3-npore-hap`, in which we performed ordinary variant calling with `clair3`, phased reads by haplotype, and then realigned them with `nPoRe` prior to variant calling.

4.5.12 Haplotype Phasing

In order to add haplotype phasing information to `Clair3`, a single iteration of the ordinary pileup-based pipeline was first run. Proposed variants were then phased using `whatshap phase`, and reads were tagged by haplotype using `whatshap haplotag`. We then sorted reads by haplotype into three separate BAM files. When generating the input pileup tensor for training `clair3-hap` and `clair3-npore-hap`, for each position a pileup tensor was generated for both unphased reads, reads from the first haplotype, and reads from the second haplotype. These three pileup tensors were then concatenated to create a new input tensor for `Clair3`.

4.5.13 Truth VCF Standardization

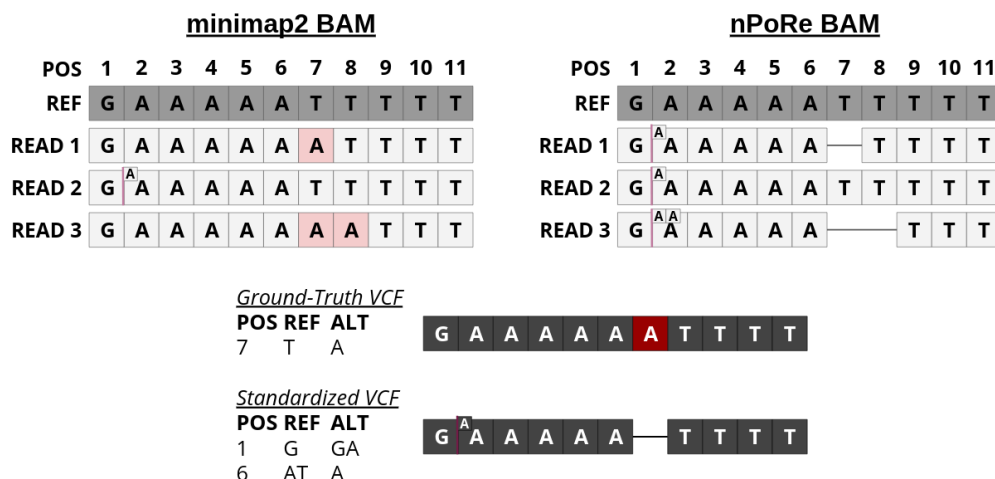


Figure 4.12: nPoRe VCF standardization: the ground-truth VCF is modified to report variants in a manner similar to nPoRe-realigned reads. The resulting sequence is unchanged.

Figure 4.12 shows a simplified example of a typical minimap2 input BAM in comparison to the nPoRe-realigned output BAM. nPoRe is comparatively more likely to call *n*-polymer INDELs than SNPs, due to the reduced INDEL penalty. We found that `clair3-npore-hap` variant calling

performance suffers if we train Clair3 with the original ground-truth VCF, since the realigned reads tend to report variants using an INDEL-heavy representation. To mitigate this, we altered the ground-truth VCF so that it reports variants using the same representation our aligner tends towards. An example of this “standardized” VCF is shown in Figure 4.12.

To achieve this, we copied our reference FASTA to create two haplotype FASTAs, and applied the phased ground-truth variants to each haplotype FASTA, storing the new CIGAR. Using a mapping position of 0, the generated haplotype references, and associated CIGARs, we considered these ground-truth haplotype references to be reads and aligned them to the original reference using nPoRe. Any substitutions, insertions, or deletions in the resulting alignment were then parsed into a new standardized ground-truth VCF file. This process ensures that the new “standardized” truth VCF contains the same exact ground-truth sequence as the original VCF when applied to the reference FASTA, but reports variants in a manner consistent with nPoRe.

4.6 Summary

This work, titled “nPoRe: n-Polymer Realigner for improved pileup variant calling” was published in BMC Bioinformatics Volume 24, 2023 [2]. Thank you to all my co-authors who contributed to this work: David Blaauw, Reetuparna Das, and Satish Narayanasamy.

CHAPTER 5

vcfdist: Accurate Evaluation of Phased Small Variant Calls

Accurately benchmarking small variant calling accuracy is critical for the continued improvement of human whole genome sequencing. We show that current variant calling evaluations are biased towards certain variant representations and may misrepresent the relative performance of different variant calling pipelines. We propose solutions, first exploring the affine gap parameter design space for complex variant representation and suggesting a standard. Next, we present our tool “vcfdist” and demonstrate the importance of enforcing local phasing for evaluation accuracy. We then introduce the notion of partial credit for mostly-correct calls and present an algorithm for clustering dependent variants. Lastly, we motivate using alignment distance metrics to supplement precision-recall curves for understanding variant calling performance. We evaluate the performance of 64 phased “Truth Challenge V2” submissions and show that vcfdist improves measured (SNP, INDEL) performance consistency across variant representations from $R^2 = (0.14542, 0.97243)$ for baseline vcfeval to $(0.99999, 0.99996)$ for vcfdist.

5.1 Introduction

The first draft reference of the human genome was assembled with great difficulty in 2000 at an estimated cost of \$300 million [128]. Following this massive effort, it became standard practice for whole genome sequencing (WGS) analyses to align sequencing reads to this reference to

determine an individual’s genome [129]. Because there is only around 0.1% difference in the genomic sequence of two individuals when excluding structural variants (SVs) [130, 131], the final sequence is stored as a set of variations from the reference genome and reported in variant call format (VCF). These small germline variants (under 50 base pairs) are then classified as either single-nucleotide polymorphisms (SNPs) or insertions/deletions (INDELs). Any variants which cannot be represented as a single SNP or INDEL are called “complex”.

The past decade has been a time of rapid advancement in DNA sequencing chemistries, machine learning, and bioinformatics algorithms [64, 132]. In such an environment, the ability to accurately compare the performance of variant calling pipelines is crucial. Firstly, accurate comparisons are necessary for identifying the current best-performing variant calling pipeline; this choice will impact real clinical decisions [133]. Secondly, researchers must be able to correctly identify promising avenues of further research. Thirdly, the curation of high-quality databases of germline mutations is only made possible with accurate comparison techniques. These databases are later used for linking genotypes with clinically-relevant phenotypes [131, 134].

The current standard benchmarking tool for variant calling is vcfEval, introduced by Real Time Genomics (RTG) in 2015 [14] and later backed by the Global Alliance for Genomics and Health (GA4GH) Benchmarking Team [21]. It is now widely accepted as the standard tool for benchmarking small variant calls [21, 135, 100, 136]. vcfEval uses a clever pruned search algorithm to determine the largest possible matching subsets of query and truth variants. Regardless of the query and truth variant representations, vcfEval is able to determine equivalency as long as there is a complete exact match. A noteworthy extension is VarMatch, which uses a similar algorithm but implements several different optimization criteria and presents an algorithm to partition the inputs into smaller subproblems without a loss of accuracy [20].

Comparing a set of variant calls (a “query” callset) to a known “truth” callset is a deceptively difficult problem, arising from the fact that there are many possible representations for the same set of variants. Prior works have invested significant effort in defining one accepted representation for a given variant. This process of variant normalization requires that variants are decomposed,

trimmed, and left-shifted [20, 13]. Although normalization is sufficient to ensure there is one possible representation for a single variant, there is not a unique representation for multiple adjacent (“complex”) variants. This makes evaluating variant calling accuracy difficult [136].

Despite its impressive advancement over previous state of the art algorithms for variant comparison, vcfeval suffers from several key limitations which we attempt to address. Each limitation increases the extent to which vcfeval’s results depend upon the initial representation of variants in the query and truth VCFs. Firstly, vcfeval has no notion of partial credit and will not report any true positives if no subsets of query variants match exactly with truth variants. Figure 5.1 demonstrates a simplified example of where different variant representations of the same sequence lead to significantly different measured accuracies. Refer to Figure 5.7 for a realistic example. Secondly, vcfeval outputs precision-recall curves for SNP and INDEL variants separately. Although precision and recall are concise metrics for understanding approximate variant calling accuracy, they are dependent on both the reference FASTA and query VCF representation. For complex variants, especially in low-complexity (repetitive) regions, the same query sequence can be represented in numerous equally valid ways with a differing total count of SNPs and INDELs (see Figures 5.1a and 5.2b). Thirdly, vcfeval was designed to handle unphased query variants, and allows arbitrary local phasing of adjacent heterozygous variants. Allowing different phasings leads to different sets of possible query sequences depending on the original variant representation. While vcfeval does contain an experimental option for global phasing (--Xobey-phase), enforcing local phase is not currently supported [14].

In combination, these characteristics bias evaluations towards a particular variant representation that uses many SNPs rather than a few INDELs to represent genomic variation. This bias becomes most evident in high-quality variant callsets where most remaining variant calling errors occur in low-complexity or high-variance regions. As read sequencing technologies have recently improved in terms of both average read length and quality, we are now able to call variants in more difficult regions than ever before [137]. INDELs are orders of magnitude more likely to occur in low-complexity repetitive regions [2], and so it is important that variant calling evaluation is not bi-

(a)

Ref. ACCCTTTTGTG			Query ACCTTTG			Truth ACCCTTTG		
Query VCF Representation 1			Query VCF Representation 2			Truth VCF		
POS	REF	ALT	POS	REF	ALT	POS	REF	ALT
3	CCTTT	C	1	AC	A	4	CTTT	C
			4	CTTT	C			

(b)

vcfeval Summary Statistics								
	TP	FP	FN	PP	Prec.	Recall	F1	F1 Q-score
Query Repr. 1	0	1	1	0	0.00	0.00	0.00	0.00
Query Repr. 2	1	1	0	0	0.50	1.00	0.67	4.77

(c)

vcfdist Summary Statistics								
	TP	FP	FN	PP	Prec.	Recall	F1	F1 Q-score
Query Repr. 1	0	0	0	1	0.67	0.67	0.67	4.77
Query Repr. 2	1	1	0	0	0.50	1.00	0.67	4.77

(d)

vcfdist Distance Summary					
	ED	DE	DE Q-score	ED Q-score	ALN Q-Score
Reference	3	1			
Query Repr. 1	1	1	4.77	0.00	3.01
Query Repr. 2	1	1	4.77	0.00	3.01

Figure 5.1: A simple example of vcfeval and vcfdist evaluations, demonstrating vcfeval’s dependence on variant representation and the usefulness of partial credit. **(a)** Reference, query, and truth sequences, as well as the query and truth VCFs. **(b)** vcfeval and **(c)** vcfdist count of true positive, false positive, false negative, and partial positive variants, as well as the calculated precision, recall, and F1 scores. Note that although both query VCF variant calls result in the exact same query sequence, the summary statistics differ. Partial credit alleviates this problem for vcfdist. **(d)** Distance-based summary statistics reported by vcfdist: edit distance, distinct edits, and alignment distance, which are independent of variant representation. An explanation of these summary statistics can be found in the *Online Methods* section.

ased against newer tools which are more likely to identify copy number variations. Lastly, vcfeval suffers from the inability to evaluate large clusters with many nearby variants due to its exponential complexity [20, 14]. Because these benchmarking issues occur during VCF evaluation, they are broadly applicable to all sequencing technologies.

In this work we present `vcfdist`, an alignment-based small variant calling evaluator that standardizes query and truth VCF variants to a consistent representation, requires local phasing of both input VCFs, and gives partial credit to variant calls which are mostly (but not exactly) correct. We show that the SNP and INDEL precision-recall curves reported by `vcfdist` are stable across variant representations. Furthermore, we introduce alignment distance based metrics for evaluation which are entirely independent of variant representation, and only measure the distance between the final diploid truth and query sequences. We then introduce a novel variant clustering algorithm which reduces downstream computation while also discovering long-range variant dependencies.

We evaluate all 64 submissions from the precisionFDA’s 2020 variant calling “Truth Challenge V2” using both `vcfeval` and `vcfdist` on two high-quality ground truth datasets: the NIST v4.2.1 WGS benchmark, and the “Challenging Medically Relevant Genes” (CMRG) dataset. We find that `vcfdist` improves measured SNP and INDEL performance consistency across variant representations from `vcfeval`’s $R^2 = 0.14542$ and 0.97243 to $R^2 = 0.99999$ and 0.99996 , respectively. We note that `vcfeval`’s SNP R^2 value is only so poor (0.14542) because one of our selected representations (design point A in Figure 5.2) never uses SNPs to represent variants (see Figure 5.6a). Excluding design point A, `vcfeval`’s SNP R^2 value is 0.96888 ; however, we believe this design point is important to include in our analyses for reasons discussed later.

5.2 Results

5.2.1 The Affine Gap Design Space

As demonstrated in Figure 5.1, the main issue with a difference-based format such as VCF is that often there are multiple reasonable sets of variant calls that can be used to represent the same final sequence relative to a reference FASTA. Since DNA sequencing only measures the final sequence, there is no way of knowing which set of variants physically occurred. We can only select a representation which contains the “most likely” set of variants, based on the relative likelihoods of various mutations occurring.

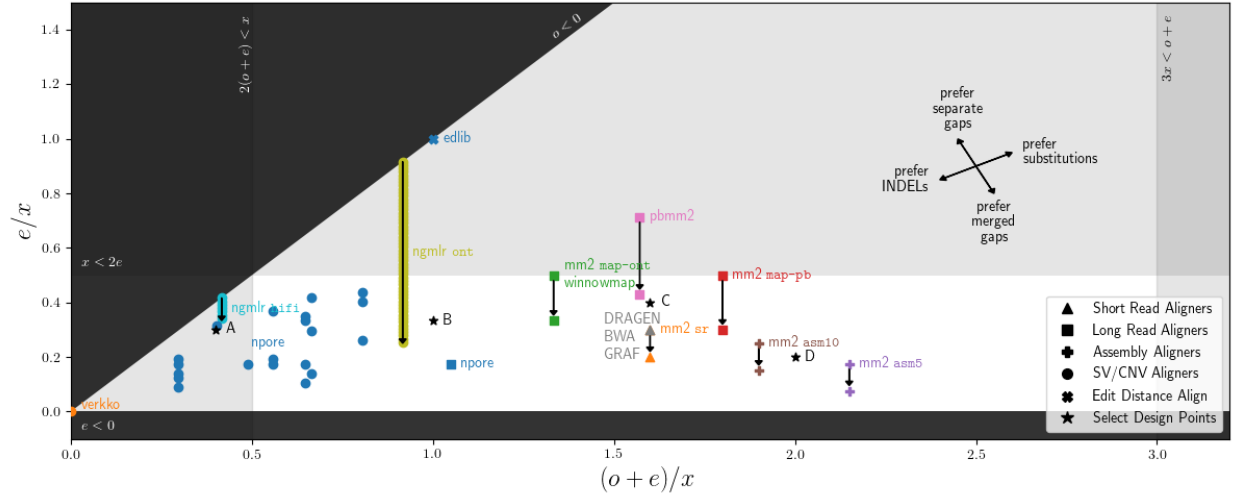
This problem of variant representation can be viewed as a query-to-reference pairwise global alignment problem, and the path of the alignment with the minimum penalty score can be used to derive an edit path, representing the most likely set of variants. This approach was first explored in [138] and termed “Best Alignment Normalization”. Here, we present a more thorough exploration of the design space for normalized variant representation. During alignment, allowed operations include matching, substituting, inserting and deleting bases with corresponding penalties m , x , $g(n)$, and $g(n)$. Under the gap-affine model originally proposed in [102], $g(n) = o + ne$, where n is the length of the gap, o is a gap-opening penalty, and e is a gap-extension penalty. The relative value of substitution, insertion, and deletion penalties is critical, as it determines which variant representation is selected in the VCF.

Recent efforts such as [139] and [140] have demonstrated how to transform gap cost models into other equivalent representations. In Figure 5.2a we normalize penalties such that $m = 0$ and then explore the design space for a general-purpose aligner with an affine-gap cost model. Black shaded areas represent the invalid areas where $o < 0$ or $e < 0$; opening and extending gaps should be penalized, not preferred over matching bases. Gray shaded areas represent parameter selections that poorly reflect the approximate likelihood of certain events in the general case. For example, when $2(o + e) < x$, a substitution will instead always be represented as a single-base insertion and a single-base deletion. Note that although these grayed parameter ranges are not ideal for a general-purpose aligner, configurations such as $2(o + e) < x$ may make sense in repetitive regions of the genome where copy number variants are likely.

Figure 5.2a plots the default parameter configurations for the most commonly used aligners in the 2020 pFDA variant calling challenge. This plot also includes the default parameters for a wider range of tools, such as those used for structural variant (SV) detection (verkko, NGMLR) [115, 112], copy number variant (CNV) detection (nPoRe) [2], edit distance calculation (edlib) [24], and assembly alignment (minimap2’s `asm5` and `asm10` configurations) [65]. The original and normalized affine-gap parameters for each tool configuration are included in Figure 5.11.

The various configurations for minimap2 are depicted with two data points and a transition,

(a)



(b)

Reference AGGCGACA

Query ATACCGAGCTTA

Point A

 $m, x, o, e = 0, 10, 1, 3$

Alignment

```

AGG---CGA-C--A
A--TACCGAGCTTA

```

VCF

POS	REF	ALT
1	AGG	A
3	G	GTAC
6	A	AG
7	C	CTT

Point B

 $m, x, o, e = 0, 3, 2, 1$

```

A-GGCGA-C--A
ATACCGAGCTTA

```

POS	REF	ALT
1	A	AT
2	G	A
3	G	C
6	A	AG
7	C	CTT

Point C

 $m, x, o, e = 0, 5, 6, 2$

```

A-GGCGA---CA
ATACCGAGCTTA

```

POS	REF	ALT
1	A	AT
2	G	A
3	G	C
6	A	AGCT
7	C	T

Point D

 $m, x, o, e = 0, 5, 9, 1$

```

AGGCGAC-----A
A---TACCGAGCTTA

```

POS	REF	ALT
1	AGGC	A
5	G	T
7	C	CCGAGCTT

Figure 5.2: The affine gap alignment penalty design space. (a) The design space for an affine-gap aligner with match, mismatch, gap opening, and gap extension penalties m , x , o , and e . This plot includes the aligners used in pFDA's Truth Challenge V2, as well as assembly, edit distance, and CNV/SV aligners for comparison. All parameters have been normalized so that $m = 0$ (see Figure 5.11 for details). (b) Aligning the same query and reference sequence using different gap parameters (design points A, B, C, and D) results in different sets of reported variants.

since minimap2 uses double affine gap alignment [65]. Under double affine gap alignment, there are two separate gap-opening (o_1, o_2) and gap-extension (e_1, e_2) penalties, and the minimum of $g_1(n) = o_1 + ne_1$ and $g_2(n) = o_2 + ne_2$ is preferred. Typically, $o_2 > o_1$ and $e_2 < e_1$. As a result, the penalties will be equal at $n = \frac{o_2 - o_1}{e_1 - e_2}$; prior to this point, $g_1 < g_2$ and afterwards $g_2 < g_1$. Thus g_1 determines the penalty for a short gap and g_2 determines the penalty for a long gap.

Any tools using homopolymer (HP) compression or simple tandem repeat (STR) compression such as verkko [115] are plotted at (0,0) in Figure 5.2a, since compressed gaps are not penalized in repetitive regions during alignment. Repetitive sequence compression is common practice for long-read de novo assemblers [115, 141, 114], since incorrect estimation of homopolymer run length is the dominant error mode of nanopore sequencing [89]. nPoRe penalizes copy number variation in these repetitive regions only slightly, depending on the copy number and repeat unit length [2]. This results in many different points on the left side of Figure 5.2a. NGMLR, an aligner for structural variant detection, also does not heavily penalize INDELs [112]. In contrast to minimap2, NGMLR uses a smoother convex gap cost model, decreasing the penalty for each additional gap extension as the length of the gap grows. As a result, NGMLR’s gap model is represented in Figure 5.2a as a smooth gap extension penalty decrease.

In Figure 5.2a, design point *A* was selected as the approximate centroid of the CNV/SV aligners. Likewise, point *C* was selected as the approximate centroid of common short and long read alignment parameter configurations, and point *D* was selected for assembly aligners. Lastly, point *B* was selected because this design point will simultaneously minimize the edit distance (ED) and number of distinct edits (DE) in the variant representation, weighting DE twice as heavily as ED. This requires meeting several additional constraints, a plot of which can be seen in Figure 5.12. It is also the approximate midpoint of design points *A* and *C*. All selected points were additionally chosen such that their substitution and gap penalties can be represented by small positive integer constants (see Figure 5.2b).

5.2.2 A Standard Complex Variant Representation

As Figure 5.2b shows, for complex variants there are many reasonable representations which depend upon the selected affine gap alignment parameters. By varying alignment penalties m , x , o , and e , Figure 5.2b shows that aligning the same query and reference sequence will result in a different VCF representation at each of the four selected design points *A*, *B*, *C*, and *D*. If we define a standard set of affine gap parameters, however, then there is usually just one possible representation

for complex variants such as this.

We propose using design point C , where $(m, x, o, e) = (0, 5, 6, 2)$, as the standard set of affine gap parameters for reporting variants. When normalized to $m = 0$ and $x = 1$, $C = (0, 1, 1.2, 0.4)$. Design point C was selected because it is the approximate centroid of widely-accepted parameters used to align reads of three different popular sequencing technologies (Illumina short reads, PacBio HiFi reads, and Oxford Nanopore long reads). For Illumina short reads, DRAGEN, BWA, GRAF, and minimap2 sr use the same normalized alignment parameters $(0, 1, 1.3, 0.3)$ (Figure 5.11). For ONT long reads, minimap2 map-ont and winnowmap use $(0, 1, 0.833, 0.5)$. For PacBio HiFi reads, minimap2 map-pb uses $(0, 1, 1.3, 0.5)$ and pbmm2 uses $(0, 1, 0.857, 0.714)$. Of the 53 submission methodologies documented from the pFDA Truth Challenge V2, 52 aligned reads using one of these above-mentioned gap parameter configurations, all of which are relatively similar. The other aligner used was NovoAlign, whose relative alignment penalties depend upon base quality [142].

5.2.3 INDEL & SNP Precision and Recall

Variant calling accuracy is currently evaluated by measuring separately the SNP and INDEL precision-recall curves, and then reporting the precision and recall where the F1 score is maximized [14, 21, 135, 100]. This metric is useful because it gives an intuitive overview of what percentage of variants were called correctly and incorrectly for SNPs and INDELs separately. Figures 5.3a and 5.3b, however, show that precision-recall curves can change significantly depending on variant representation, even when the evaluated query and truth sequences are the exact same. This is partially because the total number of SNP and INDEL variants reported in a VCF will vary depending on the selected variant representation.

This issue is highlighted in Figure 5.3b by design point A , which represents all SNPs as a 1-base insertion and 1-base deletion (because $2(o + e) < x$; see Figure 5.2a). Since VCFs represented using point A report no SNPs, any false positives will be categorized as INDEL FPs. True positive variants will be SNP TPs when the truth VCF contains a SNP, and INDEL TPs when the truth VCF contains an INDEL. This results in query VCFs at point A having perfect SNP precision, but

a lower INDEL precision than other representations. Similar problems exist for other variant representations, although to a lesser extent. The stabilized precision-recall curves following vcfdist's variant standardization can be seen in Figures 5.3c and 5.4d.

5.2.4 INDELs are Common in Repetitive Regions

Demonstrating that standardizing query VCFs using design point A changes the resulting evaluation may initially appear to be a contrived argument, since point A is not similar to any aligners used in the pFDA Truth Challenge V2. Figure 5.3a shows little difference in the performance of VCFs represented using common aligner parameters such as BWA, minimap2 map-ont, minimap2 map-pb, and pbmm2 [65, 143, 144]. This is because all of these general-purpose read aligners were designed for aligning reads in non-repetitive regions, to which most bioinformatics analyses have been restricted until recent years [145, 146]. As such, their parameters are clustered in a small region of the affine gap penalty design space in Figure 5.2a. We selected design point A precisely because its low gap penalties model how alignment occurs in repetitive regions by dedicated copy number variant (CNV) aligners or any aligners performing homopolymer (HP) or simple tandem repeat (STR) compression.

As the field shifts towards true whole genome evaluation, we expect variant callsets to frequently include merged results from general-purpose aligners/callers in non-repetitive regions and CNV/SV aligners/callers in more repetitive regions. In low-complexity regions, CNVs and other small gaps are orders of magnitude more likely than elsewhere [2], necessitating the inclusion of variants whose representation falls of the left-hand side of Figure 5.2a near design point A. As can be seen from Figure 5.2a, several existing SV and CNV callers already occupy this space. When these repetitive INDEL callsets are merged with callsets from non-repetitive regions, we must ensure that our evaluation of variant calling accuracy remains unbiased.

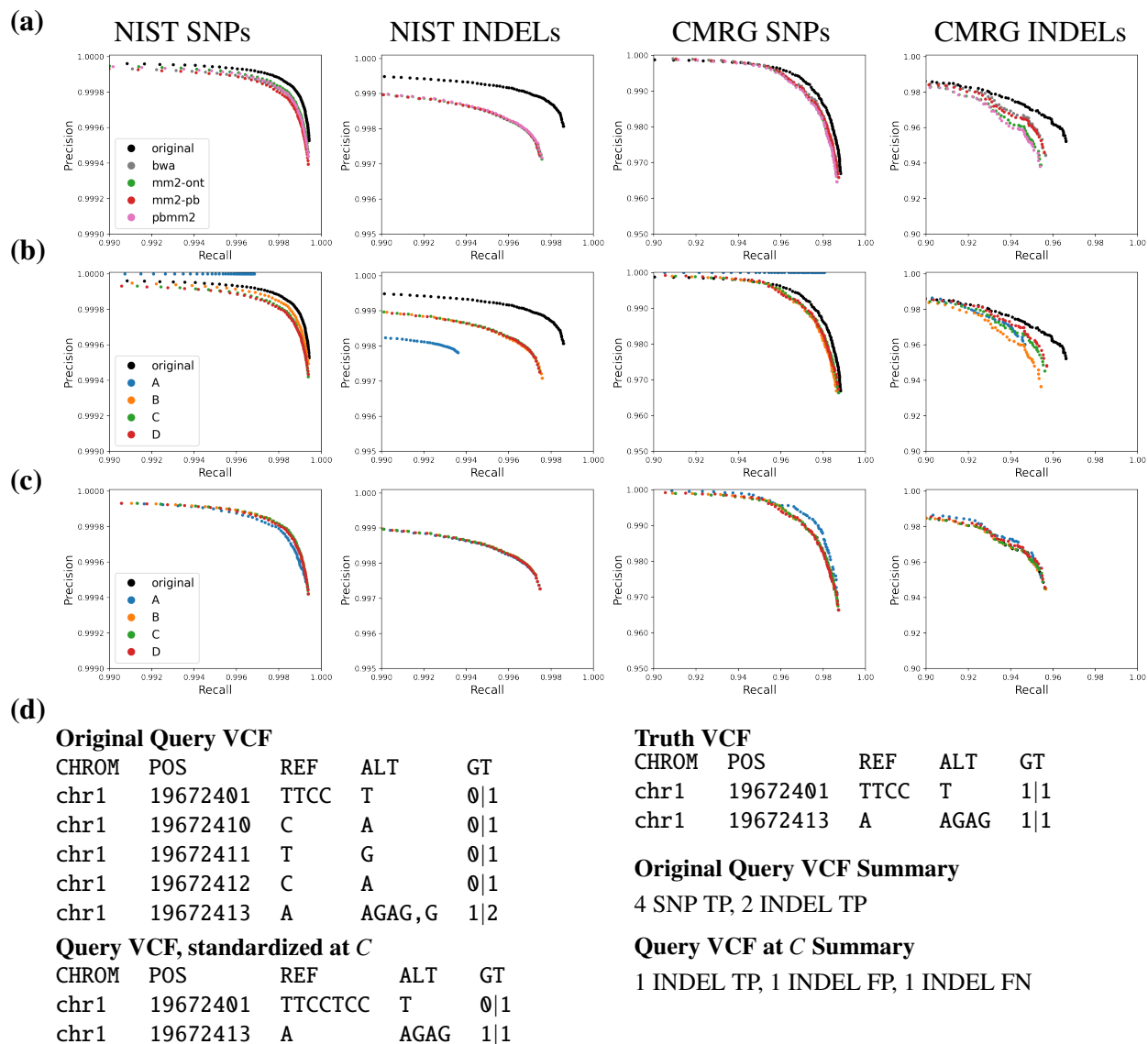


Figure 5.3: vcfeval precision and recall accuracy results, shown for the Truth Challenge V2 submission K4GT3 on the NIST and CMRG datasets. (a) Evaluating the original query VCF and after changing the query variant representation using the alignment parameters of common aligners (see Figure 5.2). (b) Evaluating the original query VCF and after changing the query variant representation to design points A, B, C, and D (see Figure 5.2). (c) Standardizing the five representations from (b) using vcfdist prior to evaluating with vcfeval dramatically improves consistency. (d) A real example demonstrating why the original K4GT3 query VCF appears to significantly outperform other representations in (a) and (b). Because vcfeval discards query phasing information and allows any possible local phasing, the original fractured variant representation is considered entirely correct whereas the more succinct standardized representation at C is not.

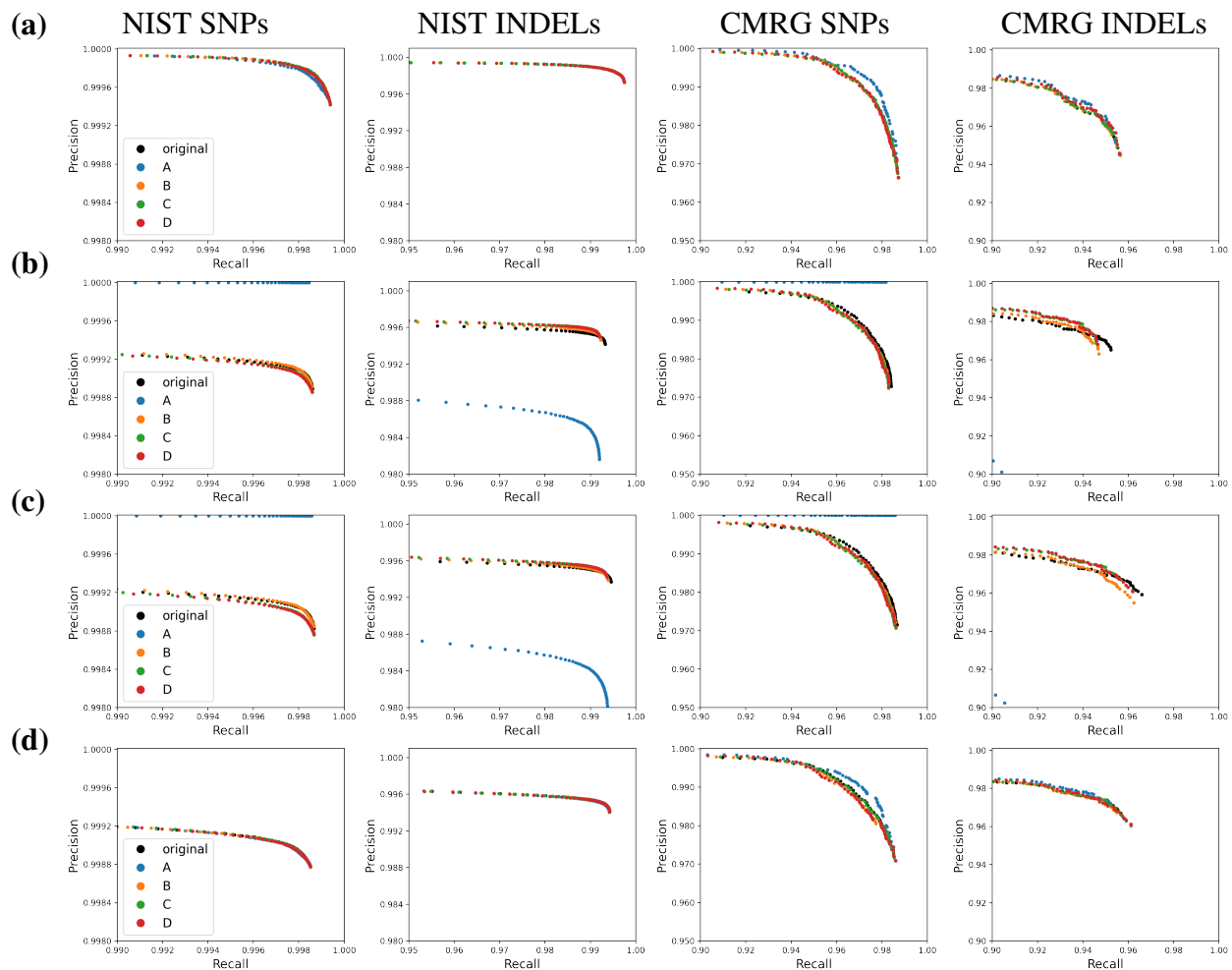


Figure 5.4: vcfdist precision and recall accuracy results, shown for **(a)** standardized vcfeval and **(b,c,d)** vcfdist for Truth Challenge V2 submission K4GT3 on the NIST and CMRG datasets. For all plots, the original query VCF is evaluated before and after changing its variant representation to design points A, B, C, and D (see Figure 5.2) **(a)** This plot is identical to Figure 5.3c, but using axes consistent with the remainder of this figure. **(b)** Evaluation with vcfdist, turning the options for standardization and partial credit off; this can be directly compared to vcfeval’s performance in Figure 5.3b. Note that the original representation no longer outperforms other representations, since local phasing is enforced. **(c)** Evaluation with vcfdist, allowing partial credit but not standardization, resulting in minor CMRG recall improvements. **(d)** Evaluation with vcfdist, allowing partial credit and standardizing variant representation. This results in the most consistent results, and is the recommended usage.

5.2.5 Local Phasing of Dependent Variants

Incorrect local phasing of heterozygous variants leads to an entirely different sequence of bases for both haplotypes, which may impact clinical decisions [147]. Despite the relative ease of local phas-

ing given new long read sequencing technologies, vcfeval does not enforce that evaluated VCFs contain phasing information. In fact, vcfeval discards this information when available, considering any possible local phasing to be correct [14]. To illustrate why this is a problem, the original query VCF representation shown in Figure 5.3d unnecessarily fragments a single heterozygous variant into five heterozygous variants. For these 5 variants, there are now $2^5 = 32$ possible local phasings, each resulting in a different pair of sequences. Perhaps unsurprisingly, one of these 32 phasings (and not the phasing initially reported) results in a match with the ground truth, and all variants are considered true positives. This serves to heavily bias evaluations towards fragmented query variant representations.

In contrast, vcfdist requires locally phased truth and query input VCFs, and requires local phasing to be correct in order to consider variants true positives. This results in a lower precision-recall curve in Figure 5.4d than Figure 5.4a, which we believe more accurately represents the true performance. It also makes evaluation of the original fragmented representation (black) much more consistent with the other representations (Figure 5.4b vs Figure 5.3b). Importantly, vcfdist does not require correct global phasing of variants, and allows for arbitrary switch errors to occur between clusters of dependent variants. Figure 5.5 shows an overview of our tool, vcfdist.

5.2.6 Stabilizing Precision-Recall Curves

The precision-recall curve stabilization gained by using a standard variant representation (point C) is clearly demonstrated in Figures 5.3 and 5.4. In fact, standardizing variant representation is sufficient for prior work vcfeval to obtain consistent results across variant representations (Figure 5.3c). Researchers could use vcfdist as a pre-processing step for variant standardization, and then perform evaluations as usual through vcfeval. It is important to note, however, that consistency does not imply accuracy. The precision-recall curve for vcfdist in Figure 5.4d is not the same curve as for vcfeval in Figure 5.4a. The other improvements presented in regards to clustering, phasing, and partial credit have a large impact on the final results.

As can be seen in Figures 5.6a and 5.6b, the final F1 scores reported on both the NIST

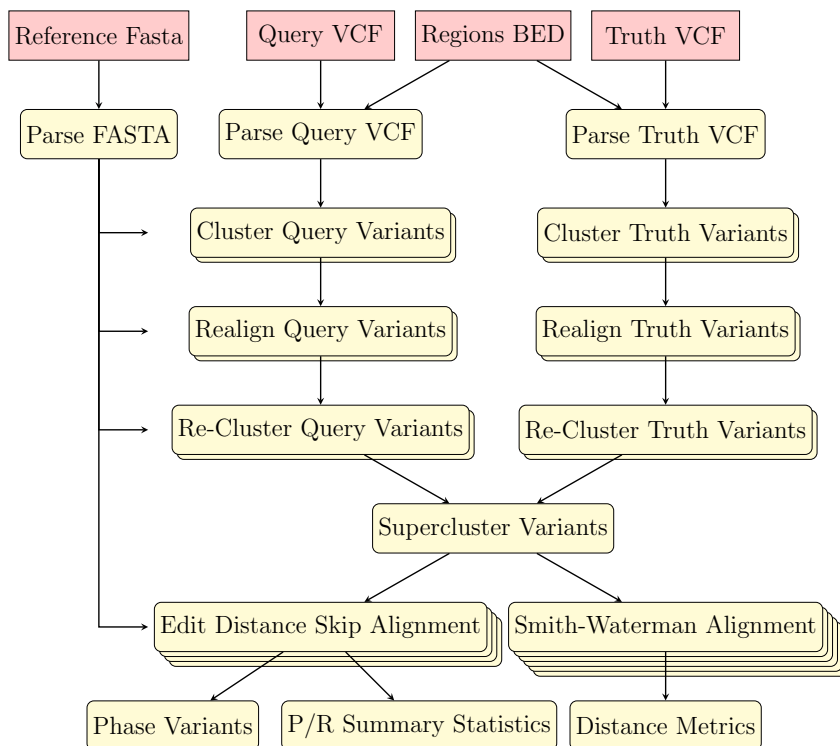


Figure 5.5: The high-level design of the vcfdist software; more details regarding each step can be found in the *Online Methods* section.

and CMRG datasets are much more consistent for vcfdist than vcfeval as variant representation changes. For each graph, the coefficient of determination (R^2) is greater and the AMRC is lower for vcfdist in comparison to vcfeval. We define AMRC (“Average Maximum Rank Change”) in the *Online Methods* section. It is an average measure of how much a particular submission’s rank (in performance relative to the other 63 submissions) would change when using the best versus the worst possible variant representation for that particular submission. Ideally, values of $R^2 = 1$ and $\text{AMRC} = 0$ would indicate that our evaluation is entirely independent of variant representation.

5.2.7 Partial Credit

As demonstrated in Figure 5.1, variant calls which are nearly but not exactly correct are penalized under current evaluation methods. Figure 5.4c shows a slight improvement in variant recall over Figure 5.4b for the CMRG dataset after vcfdist assigns partial credit to mostly-correct variants.

5.2.8 Distance-Based Performance Metrics

As mentioned earlier, precision-recall curves are dependent upon query and truth VCF variant representations, as well as the reference FASTA. In contrast, distance-based performance metrics such as the edit distance, number of distinct edits, and alignment score are determined solely by the query and truth sequences and are unchanged by different variant representations or references. Figure 5.6c demonstrates this. The very minor remaining differences in distance-based metrics are caused by false-positive variant dependencies during clustering. Since slightly different clusters are created, depending on the original variant representations, enforcing consistent local phasing within a cluster may lead to different results.

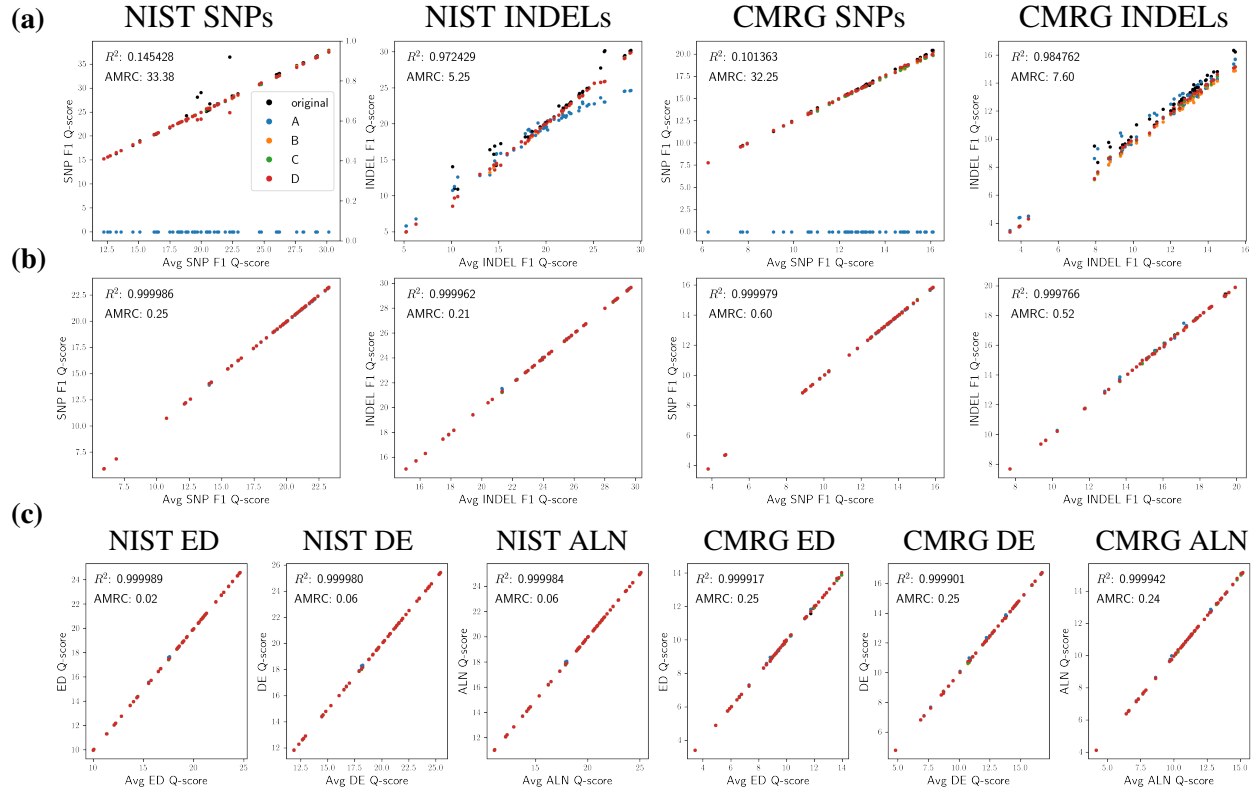


Figure 5.6: pFDA Truth Challenge V2 accuracy results, for (a) vcfeval and (b) vcfdist on both the NIST and CMRG datasets. (c) The same information, using new sequence distance-based summary metrics (see *Online Methods* for a full explanation of each). On each graph, average F1 Q-score is plotted against the F1 Q-score for the original representation and at points A, B, C, and D (see Figure 5.2). Clearly, vcfdist results in (b) and (c) are much more stable in regards to variant representation than vcfeval in (a).

Although a similar level of evaluation consistency can be gained by using a standard variant

representation with vcfdist (Figure 5.6b), distance-based summary metrics are useful to gain a better understanding of overall performance. Figure 5.9b shows that of the three multi-technology pFDA submissions which tied for the winning performance on the NIST dataset, Sentieon’s solution performed best on SNPs, but Roche’s solution performed best on INDELs. Using precision-recall curves for SNPs and INDELs separately, it is hard to get a clear picture of which submission has the best overall performance. This is because SNPs are much more common than INDELs, and the two variant types do not always have similar accuracy characteristics (performance on INDELs is usually worse). Figure 5.9c clearly demonstrates that Roche’s solution performs better overall on the NIST dataset, as it has lower edit distance, distinct edits, and alignment distance. Interestingly, Sentieon’s solution achieves better performance on the CMRG dataset.

5.2.9 Runtime and Efficiency

We evaluated the runtime of submission K4GT3 on the CMRG and NIST datasets for each of the five representations in Figure 5.4 (original, *A*, *B*, *C*, *D*). On the relatively small CMRG dataset, the baseline vcfeval’s single-threaded runtime was 153.6s. In comparison, vcfdist’s average single-threaded runtime was 441.2s and its maximum memory usage was 4.99G. If, however, vcfdist skips VCF realignment and uses the simpler gap-based clustering method, its average runtime decreases to 25.9s and RAM usage is only 3.27GB. For smaller datasets, loading the reference FASTA accounts for the vast majority of memory usage (≈ 3 GB).

On the much larger NIST dataset, vcfdist’s maximum RAM usage only increases to 5.86GB. Its single-threaded runtime (2648.2s) remains approximately 2 $\times$ slower than vcfeval’s (1489.6s), unless realignment and Smith-Waterman clustering are disabled (663.8s). The baseline, vcfeval, additionally supports multi-threading with contig-level granularity. We plan to add multi-threading support to vcfdist with supercluster granularity (allowing core usage to scale beyond 23 for human genomes) once development of other features has ceased. Since other steps in the whole genome sequencing pipeline are the computational bottleneck (basecalling, alignment, and variant calling) [100, 2], this has not yet been a priority.

5.3 Discussion

The first major step towards affordable whole genome sequencing was the development of second-generation sequencing methods, which enabled massively parallel sequencing of short reads of 100-1000 bases [148]. With such short reads, resolving phasing and calling variants in repetitive genomic regions was incredibly difficult. It was in this context that vcfeval was developed. With the introduction of third-generation sequencing methods in recent years, however, the scope of genomic regions under active analysis has expanded. The 2019 NIST v4.2.1 benchmarking dataset used includes 17% more SNVs, 176% more INDELs and 12% larger benchmarking regions than the 2016 NIST v3.3.2 dataset [149]. Additionally, now that datasets often have average read lengths of 10-100Kb, some recent genome assemblies contain phase block N50s of more than 20Mb [150]. As a result, local phasing information is highly accurate.

As the state of the field and technology shifts, so too must the standards for evaluation. The accuracy of third-generation sequencing has improved tremendously in the past few years [64, 132], and most small variant calling errors are now limited to low-complexity regions and complex variants. This work aims to ensure that evaluation is consistent in low-complexity regions and for complex variants where many representations are possible. We do so by proposing a standard variant representation, enforcing consistent local phasing, improving variant clustering, attributing partial credit, and motivating the use of alignment distance metrics for evaluation.

As genome sequencing becomes cheaper and more accurate, and as reads lengthen, a gradual shift is beginning from reference-based variant calling to de-novo diploid assembly [115, 150]. Even if the most common format for storing a human genome progresses from a haploid reference FASTA and an accompanying VCF to a diploid FASTA, there will always be the need to compare human genomes to one another. Whether this comparison is stored using a VCF or graph-based format, the same challenge of multiple possible alignments remains. As a result, we expect this work to remain relevant for the foreseeable future.

5.4 Methods

5.4.1 Code and Data Availability

All code for vcfdist and the benchmarking pipelines developed are available in a public Github repository: <https://github.com/TimD1/vcfdist>. Publicly available software used to phase the input callsets includes minimap2 v2.24-r1122 [65], samtools v1.16.1 [151], and WhatsHap v1.7 [127]. To evaluate the phased callsets, vcfdist v1.0.0, hap.py v0.3.15 [22], and rtg-tools vcfeval v3.12.1 [14] were used.

All datasets used were previously released by other researchers and are publicly available online, using human genome HG002 and reference genome assembly GRCh38. The unphased precisionFDA Truth Challenge V2 VCF submissions and corresponding read FASTAs used to phase the VCFs are hosted on the NIST public data repository at <https://data.nist.gov/od/id/mds2-2336> [136]. The phased ground truth VCF and BED files for the HG002 whole-genome and challenging medically-relevant genes datasets are available in the NISTv4.2.1 and CMRG_v1.00 directories of the Genome In A Bottle Consortium’s FTP release folder, respectively: <https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/Ashkenazi> mTrio/HG002_NA24385_son [137, 145]. The GRCh38 reference FASTA is likewise available at <https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/references/>. For the NIST v4.2.1 dataset, we used the phased truth VCF HG002_GRCh38_1.22.v4.2.1_benchmark_hifiasm_v11_phasetransfer.vcf.gz instead of the unphased HG002_GRCh38_1.22.v4.2.1_benchmark.vcf.gz VCF. More details on the exact files and versions used are available online at <https://github.com/TimD1/vcfdist/tree/master/data>.

5.4.2 Preprocessing

Running our pipeline required two modifications to the phased NIST ground truth VCF. The CMRG ground truth VCF was used unchanged. Firstly, there were erroneously two additional blank FORMAT field values in the Major Histocompatibility Complex (MHC) region on chr6 that

were removed. Secondly, WhatsHap only accepts integer phase set values, whereas the NIST truth VCF contained phase sets named with strings. The phase set (PS) tags in the NIST ground truth VCF were removed using `bcftools annotate` [151]. This does not affect our downstream evaluation because `vcfdist` only assumes variants are phased locally, and not globally.

Several of the pFDA Truth Challenge V2 query VCFs required modification as well. Submissions 0G00R and BARQS reported some reference or ungenotyped variants as haploid, which were filtered. Chromosomes were renamed (i.e. 1 to chr1) for the same two query VCFs. Submissions 0G00R, BARQS, and HB8P3 reported variants on decoy contigs, which were filtered. Additionally, the VCF header for HB8P3 did not store the lengths of each decoy contig. Submission Y8QR8 did not report any contig lengths, which were added manually. Submissions WGQ43 and YGOTK did not contain variant quality scores. Submissions WX8VK and CZA1Y contained an erroneous variant where ALT was the same as REF at position chr6:28,719,765. This was removed.

5.4.3 Phasing Input Query and Truth Callsets

`vcfdist` requires input query and truth VCFs to be locally phased. However, most VCFs submitted to precisionFDA's Truth Challenge V2 were unphased. In order to phase these VCFs prior to evaluation, we downloaded all the original HG002 sequencing read FASTAs provided to participants of the competition. This dataset contained 35× genome coverage of Illumina short paired-end reads, 35× coverage of PacBio HiFi reads, and 50× coverage of Oxford Nanopore reads, sequenced on a PromethION and basecalled with Guppy v3.6.0. All reads were aligned to GRCh38 with `minimap2` using the default parameters corresponding to each sequencing technology (`sr`, `map-pb`, and `map-ont`). Supplementary and secondary alignments were filtered. The aligned reads were then phased using `whatshap haplotag`. Lastly, all variants in the 64 Truth Challenge V2 submission callsets were phased using `whatshap phase` and the phased reads from all three technologies.

Although the CMRG truth VCF was entirely phased, only 99.3% of variants in the NIST truth VCF were phased. For this work, we considered all truth variants to be phased even when they were not (i.e. indiscriminately converting 0/1 to 0|1). Importantly, although this impacts the

measured absolute performance of each query VCF, it does not impact any conclusions this study makes regarding evaluation stability. Due to recent improvements in read lengths and accuracies, most future benchmarking truth VCFs will likely be fully phased.

5.4.4 Clustering Dependent Variants

When multiple variants occur at the same locus, there are often multiple ways that differences between the draft reference FASTA and the resulting sequence could be represented in a VCF. This is still true after performing standard variant normalization techniques. For an example, see Figure 5.2b. In order to fairly evaluate the differences between two VCFs, we must evaluate an entire set of nearby variants (or “cluster”) in such regions simultaneously. VarMatch proposes a clustering algorithm that can be used to identify “separator regions” (long and non-repetitive sections of the reference genome that contain no variants), which can be used to divide variants into independent clusters [20].

Using a similar approach, we note that two clusters of variants can be considered independent of one another for a given set of affine-gap parameters if all optimal alignment paths return to the main diagonal between the two clusters. This “main diagonal” is the reference region between the two clusters which contains no variants. In order to determine whether two adjacent clusters are dependent, we begin extending from the start of the first cluster and note the rightmost reference position to which we can extend while avoiding the main diagonal and having an alignment penalty of lesser or equal value to that of the original representation of variants within the first cluster.

We mirror this process for the second cluster, extending from its end to the leftmost reference position possible while avoiding the main diagonal and limiting the alignment penalty. If the latter position is less than or equal to the first, then these two paths are able to meet off the main diagonal, and these two clusters are dependent. We use an iterative algorithm, initializing each cluster to a single variant. If two adjacent clusters are dependent, we merge them into a single larger cluster. We repeat this process until all clusters have not grown from the previous iteration or the maximum iteration count is reached. An overview of this clustering algorithm, which was also partly inspired

by the uni- and bi-directional Wave Front Alignment (WFA) algorithms [152, 153], can be seen in Figure 5.13.

5.4.5 A Standard Variant Representation

As shown in Figure 5.2a, point C was selected to be the Smith-Waterman design point for a standard variant representation, since it is the approximate centroid of widely-accepted parameters used to align reads of three different popular sequencing technologies (Illumina short reads, PacBio HiFi reads, and Oxford Nanopore long reads). The integer affine-gap parameters corresponding to point C are $(m, x, o, e) = (0, 5, 6, 2)$. Each haplotype was standardized independently, for both the truth and query VCFs. After clustering variants using the previously-described algorithm, within each cluster all variants are applied to the draft reference FASTA and the resulting sequence is realigned using the standard affine-gap parameters for point C . A new set of variant calls is then defined from the resulting alignment string, now in the standard representation.

5.4.6 Superclustering Variants

After clustering and realigning the truth and query variants separately, we need some means of associating query variants with their corresponding truth variants and vice-versa. In order to do so, we group together clusters on both the truth and query haplotypes that reach within 50 base pairs of one another. An example is shown in Figure 5.14. This resulting “supercluster” may be composed of several variant clusters on any of the four haplotypes (two each for the variant and query). All subsequent analyses take place within a supercluster, which are considered fully independent of one another.

5.4.7 Alignment and Local Phasing

Since `vcfdist` assumes variants have been locally phased, we do not consider the possibility of genotype or switch errors within a supercluster. However, because `vcfdist` makes no assumptions

regarding global phasing of the truth and query VCFs, we consider the relative phasing of each supercluster to be unknown and allow phase switches in between superclusters. In order to phase superclusters, we first perform each of the four possible alignments of truth and query haplotypes: Truth1 to Query1, Truth1 to Query2, Truth2 to Query1, and Truth2 to Query2.

During this alignment, we represent each query haplotype as a graph (merged with the reference sequence) to allow skipping query variants without penalty, since there may be false positive variant calls. Note that although this algorithm is similar to ordinary sequence-to-graph alignment, it is not equivalent since reference-to-query transitions (and vice versa) are not allowed within truth variants. An example of this alignment is shown in Figure 5.16. The minimum edit distance is calculated for each of the four alignments, and the phasing which minimizes total edit distance is selected: (X) Truth1 to Query1 and Truth2 to Query2, or (Y) Truth1 to Query2 and Truth2 to Query1. This represents the best possible phasing for the supercluster.

5.4.8 Partial Credit

Once a phasing has been selected, summary statistics (e.g. counting true positive (TP) and false positive (FP) variants) for each supercluster can be calculated using the resulting alignments. Unlike prior work, *vcfdist* attributes partial credit to variant calls which are not entirely correct yet still reduce the overall edit distance. We call these “partial positives” (PP). These partial positives can be used to more accurately represent the true performance of a variant caller, since it is common to report the nearly-correct length of INDEL variants (such as in Figure 5.1). The first step is identifying “sync points”, where the optimal alignment path passes through a reference base on the main diagonal (i.e. a base that is not within either a truth or query variant). Such sync points are marked in red in Figure 5.16d.

Between each sync point, query and truth variants are considered to be true positives if the new alignment’s edit distance is zero, false positives (or false negatives for the truth VCF) where the edit distance is unchanged, and partial positives where the edit distance is reduced. For the purposes of precision and recall calculation, partial positives are converted to a fractional positive

plus a fractional negative using the old and new edit distances. For example, the partial positive query variant call in Figure 5.1 reduces the edit distance from 3 to 1 and is counted as $\frac{2}{3}\text{TP} + \frac{1}{3}\text{FP}$. All statistics are calculated separately for each haplotype, and then summed. As a result, we count a false positive homozygous variant (1|1) as two false positives.

5.4.9 Distance Calculations

When measuring the distance between two sequences, we find it useful to define two metrics: “edit distance” (ED) and “distinct edits” (DE). For example, an insertion of length 5 has an edit distance of 5, but is 1 distinct edit. We would like to minimize both DE and ED simultaneously, placing greater importance on distinct edits than edit distance, since erroneously lengthening a gap is less detrimental than introducing new edits. If we aim to globally minimize $2\text{DE} + \text{ED}$, this minimization problem is equivalent to Smith-Waterman global alignment with the affine-gap parameters $(m, x, o, e) = (0, 3, 2, 1)$. This is because opening a gap (o) increases DE, extending a gap (e) increases ED, and a substitution (x) increases both ED and DE. This affine-gap design point, which is point B in Figure 5.2a, is first used to align the query and truth sequences. Afterwards, the resulting alignment path is used to derive the edit distance and number of distinct edits. Note that although in other contexts “edit distance” colloquially refers to *minimum* edit distance, that is not the case here because design point B attempts to simultaneously minimize DE and may do so at the expense of ED.

5.4.10 New Summary Metrics

For each complex variant or INDEL, we allow partial positive variants to receive partial credit by comparing the reference sequence edit distance (ED_{ref}) and the query sequence edit distance (ED_{query}). This query sequence is derived by applying query VCF variants to the reference sequence.

Partial positives:

$$\begin{aligned} PP_{\text{query}} &= \left(1 - \frac{ED_{\text{query}}}{ED_{\text{ref}}}\right) TP_{\text{query}} + \frac{ED_{\text{query}}}{ED_{\text{ref}}} FP_{\text{query}} \\ PP_{\text{truth}} &= \left(1 - \frac{ED_{\text{query}}}{ED_{\text{ref}}}\right) TP_{\text{truth}} + \frac{ED_{\text{query}}}{ED_{\text{ref}}} FN_{\text{truth}} \end{aligned}$$

In order to evaluate approximate genome variant calling quality, we define the F1 Q-score metric as follows:

$$F1_{\text{Qscore}} = -10 \log_{10}(1 - F1_{\text{score}})$$

We make similar definitions for ED Q-score, DE Q-score, and ALN Q-score. These metrics are Q-score estimates of overall variant calling quality, based on the remaining edit distance, distinct edits, and alignment score of the query to truth sequences, relative to the difference between the reference and truth sequences:

$$\begin{aligned} ED_{\text{Qscore}} &= -10 \log_{10} \left(\frac{ED_{\text{query}}}{ED_{\text{ref}}} \right) & DE_{\text{Qscore}} &= -10 \log_{10} \left(\frac{DE_{\text{query}}}{DE_{\text{ref}}} \right) \\ ALN_{\text{Qscore}} &= -10 \log_{10} \left(\frac{\text{Alignment Score}_{\text{query}}}{\text{Alignment Score}_{\text{ref}}} \right) \end{aligned}$$

Lastly, we define the “Average Maximum Rank Change” (AMRC) to evaluate the stability of a performance evaluation on a dataset in regards Accurate VCF comparison methodologies are critical for determining promising directions for future research in variant calling, for curating clinical mutation databases, and for selecting current state- to variant representation. In this work, we evaluate the Q-score performance of pFDA Truth Challenge V2 submissions $S = \{S^{(1)}, \dots, S^{(i)}, \dots, S^{(n)}\}$. For each submission $S^{(i)}$, we use both the original (O) variant representations and when normalized

to points A , B , C , and D .

$$S^{(i)} = \{S_O^{(i)}, S_A^{(i)}, S_B^{(i)}, S_C^{(i)}, S_D^{(i)}\}$$

First, we note the Q-score performance metrics for each submission $S^{(i)}$:

$$Q^{(i)} = Q_{\text{score}}(S^{(i)}) = \{Q_{\text{score}}(S_O^{(i)}), Q_{\text{score}}(S_A^{(i)}), Q_{\text{score}}(S_B^{(i)}), Q_{\text{score}}(S_C^{(i)}), Q_{\text{score}}(S_D^{(i)})\}$$

Next, we create a sorted sequence (in increasing order) of the median Q-score for each submission, our best guess at each submission's "true" performance.

$$M = \text{sort}(\{m^{(i)} | m^{(i)} = \text{med}(Q^{(i)})\})$$

Then we find the Q-scores for the worst- and best-performing representations of each submission.

$$W = \{w^{(i)} | w^{(i)} = \min(Q^{(i)})\}$$

$$B = \{b^{(i)} | b^{(i)} = \max(Q^{(i)})\}$$

Lastly, we find the average change in rank for submission $S^{(i)}$ when ranking submissions in order of Q-score if we use the best versus the worst possible representation. Note that we must omit the median of the current submission, $\text{med}(Q^{(i)})$, when calculating the rank. We define $\text{pos}(x, Y)$ as the first position that value x could be inserted into sorted sequence Y such that it remains sorted.

$$\text{AMRC} = \sum_{i=1}^n \left(\text{pos}(B^{(i)}, M - \text{med}(Q^{(i)})) - \text{pos}(W^{(i)}, M - \text{med}(Q^{(i)})) \right) / n$$

5.4.11 Global Phasing Calculation

To phase superclusters, we use a simple dynamic programming algorithm which minimizes the total number of switch errors and supercluster phasing errors. A detailed example is shown in Figure 5.15. Firstly, superclusters are first categorized as type *X* or *Y* depending on the best mapping of query to truth haplotypes (*X*: Truth1 to Query1 and Truth2 to Query2, or *Y*: Truth1 to Query2 and Truth2 to Query1). It is also possible for both mappings result in the same minimum edit distance, in which case the supercluster is not categorized. During the forwards pass, switch errors and supercluster phasing errors are minimized, as the global phase state switches back and forth between *X* and *Y*. During the backwards pass, phase blocks are recovered and the total number of switch errors and supercluster phasing errors is reported.

5.4.12 Output

In addition to a concise summary output (shown in Figure 5.8), `vcfdist` outputs multiple verbose TSV files to aid in further analysis of variant calling performance. `vcfdist` also outputs both the original and new standard representations of the query and truth VCFs, as well as a GA4GH-compatible summary VCF that can be further analyzed using `hap.py`'s helper script `qfy.py` [22]. Lastly, several Python analysis scripts are provided which can be used to conveniently display the output of `vcfdist`.

5.5 Supplementary Information

Figures 5.7-5.16 contain supplementary information about `vcfdist`, its outputs, and various design decisions made during the research process.

(a) **Reference** TTCCTTTCTTTCTTCCTTTCTTTCTTTCTT

Query TTCCTTTCTTTCTTTCTT

Truth TTCCTTTCTTTCTT

Query VCF

Alt. Query VCF

Truth VCF

POS	REF	ALT	POS	REF	ALT	POS	REF	ALT
8	CTTTCTTCCTTTCT	C	3	CCTTTCTTTCTTCCTTT	C	3	CCTTTCTTTCTTCCTTT	C
			24	C	CTTC	25	T	C

(b)

vcfeval INDEL Summary Statistics

	TP	FP	FN	PP	Prec.	Recall	F1	F1 Q-score
Query	0	1	1	0	0.00	0.00	0.00	0.00
Alt. Query	1	1	0	0	0.50	1.00	0.67	4.77

(c)

vcfdist INDEL Summary Statistics

	TP	FP	FN	PP	Prec.	Recall	F1	F1 Q-score
Query	0	0	0	1	0.82	0.82	0.82	7.53
Alt. Query	1	1	0	0	0.50	1.00	0.67	4.77

(d)

vcfdist Distance Summary

	ED	DE	DE Q-score	ED Q-score	ALN Q-Score
Reference	17	2			
Query	3	2	7.27	0.00	4.56
Alt. Query	3	2	7.27	0.00	4.56

Figure 5.7: A realistic example of vcfeval and vcfdist evaluation with partial credit. The example is from Truth Challenge V2 winning submission K4GT3 on GRCh38 region chr5:140,941,200-140,941,230. This region is within the protocadherin alpha gene cluster, genes PCDHA1-PCDHA13 in the CMRG dataset. (a) Reference, query, and truth sequences, as well as the query and truth VCFs. An alternate query VCF representation (resulting in the same query sequence) is shown as well. (b) vcfeval and (c) vcfdist count of true positive, false positive, false negative, and partial positive INDEL variants, as well as the calculated precision, recall, and F1 scores. Note that although both the original and alternate query VCF variant calls result in the exact same query sequence, the summary statistics differ. (d) Distance-based summary statistics reported by vcfdist (edit distance and distinct edits), which are independent of variant representation. An explanation of these summary statistics can be found in the *Online Methods* section.

(a)

TYPE	THRESH	TRUTH TP	QUERY TP	TRUTH FN	QUERY FP	PREC	RECALL	F1 SCORE
SNP	Q >= 0	4636790	4640475	5502	32780	0.993	0.999	0.996
SNP	Q >= 0	4636790	4640475	5502	32780	0.993	0.999	0.996
INDEL	Q >= 0	744931	742121	3600	5990	0.991	0.994	0.993
INDEL	Q >= 7	744844	742055	3687	5776	0.992	0.994	0.993

(b)

TYPE	THRESH	EDIT DIST	DISTINCT EDITS	ED QSCORE	DE QSCORE	ALN QSCORE
ALL	Q >= 0	62704	48117	20.36	20.50	20.44
ALL	Q >= 6	62612	48126	20.37	20.50	20.45
ALL	Q >= 61	6819150	5394057	0.00	0.00	0.00
SNP	Q >= 0	37545	37545	20.92	20.92	
SNP	Q >= 0	37545	37545	20.92	20.92	
SNP	Q >= 61	4643522	4643522	0.00	0.00	
INDEL	Q >= 0	25159	10572	19.37	18.51	
INDEL	Q >= 8	24857	10488	19.42	18.54	
INDEL	Q >= 61	2175628	750535	0.00	0.00	

Figure 5.8: Example textual vcfdist output summary, evaluating pFDA Truth Challenge V2 winner K4GT3. **(a)** Precision, recall, and F-score summary statistics **(b)** Distinct edits and edit distance summary statistics. For each type, results are shown for when no variants are filtered (Q >= 0), at the optimal Q-score cutoff, and in (b) when all variants are filtered (Q >= 61).

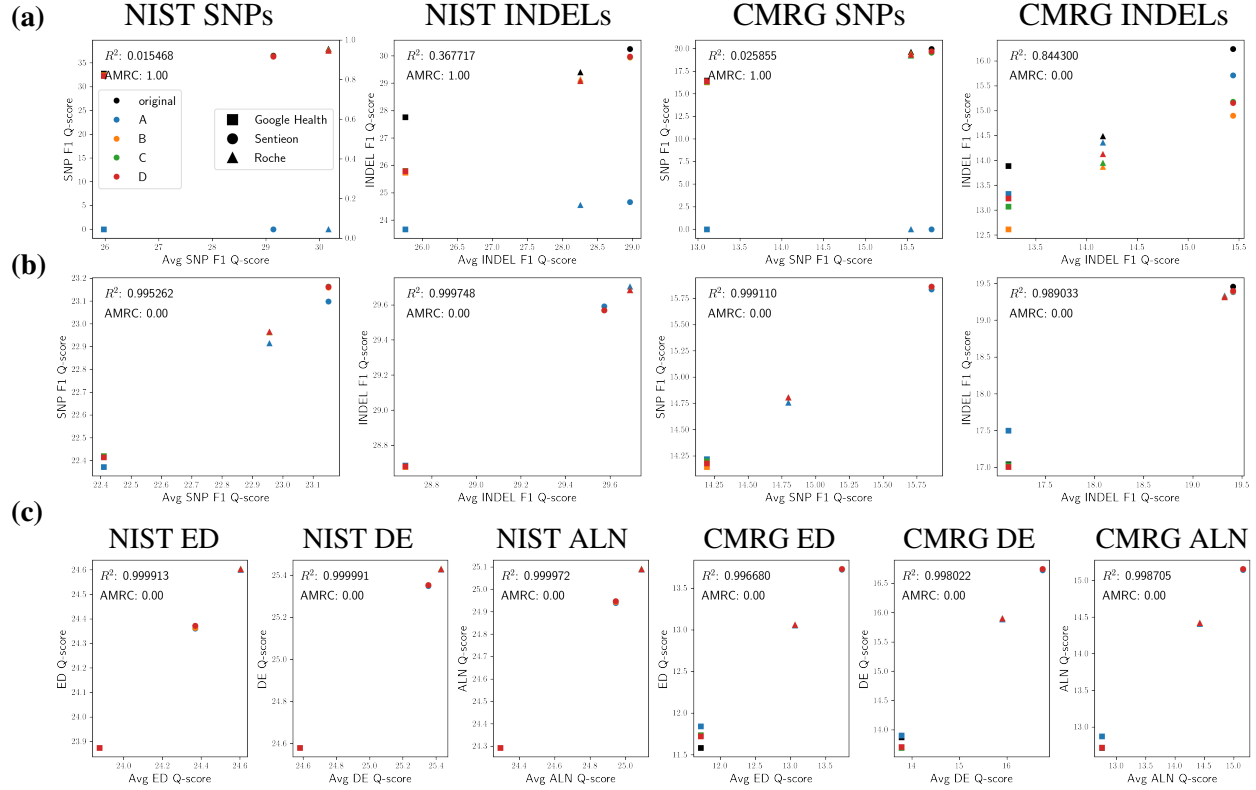


Figure 5.9: pFDA Truth Challenge V2 accuracy results for winning submissions. **(a)** vcfeval and **(b)** vcfdist F1 Q-score plots for the three overall winning Truth Challenge V2 submissions using multiple sequencing technologies on both the NIST and CMRG datasets. **(c)** The same information, using new sequence distance-based summary metrics (see *Online Methods* for a full explanation of each). On each graph, average Q-score is plotted against the Q-score for the original representation and at points A, B, C, and D (see Figure 5.2).

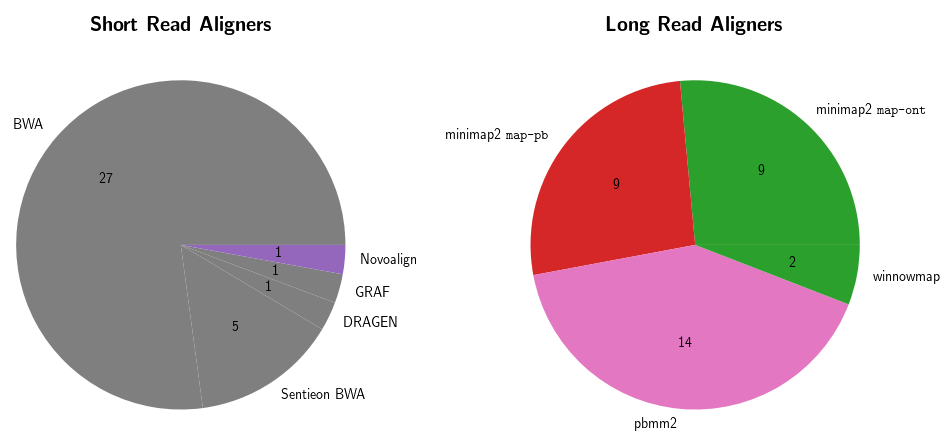


Figure 5.10: pFDA Truth Challenge V2 aligners. 15 of the 20 teams (including all winners) competing submitted their variant calling workflows, so the aligners used for 53 of the 64 total submissions are known [136]. Some submissions combined sequencing data from multiple technologies, and used several aligners.

(a)

Tools	Configs	m	x	o_1	e_1	o_2	e_2
BWA, GRAF, DRAGEN	default	-1	4	6	1		
minimap2	sr	-2	8	12	2	32	1
winnowmap, minimap2	default, map-ont	-2	4	4	2	24	1
minimap2	map-pb	-1	4	6	2	26	1
pbmm2	default, CCS	-2	5	5	4	56	2
minimap2	asm5	-1	19	39	3	81	1
minimap2	asm10	-1	9	16	2	41	1
ngmlr	pacbio	2	-5	-5	-5		-1
ngmlr	ont	3	-3	-1	-1		-0.5

(b)

Tool	Config	m	x	o_1	e_1	o_2	e_2
BWA, GRAF, DRAGEN	default	0	1	1.3	0.3		
minimap2	sr	0	1	1.3	0.3	2.5	0.2
winnowmap, minimap2	default, map-ont	0	1	0.833	0.5	4.167	0.333
minimap2	map-pb	0	1	1.3	0.5	5.3	0.3
pbmm2	default, CCS	0	1	0.857	0.714	8.14	0.429
minimap2	asm5	0	1	1.975	0.175	4.075	0.075
minimap2	asm10	0	1	1.65	0.25	4.15	0.15
ngmlr	pacbio	0	1	0	0.917		0.25
ngmlr	ont	0	1	0	0.417		0.333

Figure 5.11: (a) Original and (b) normalized affine-gap parameters, for tools used in the pFDA Truth Challenge V2. Normalized parameters are plotted in Figure 5.2.

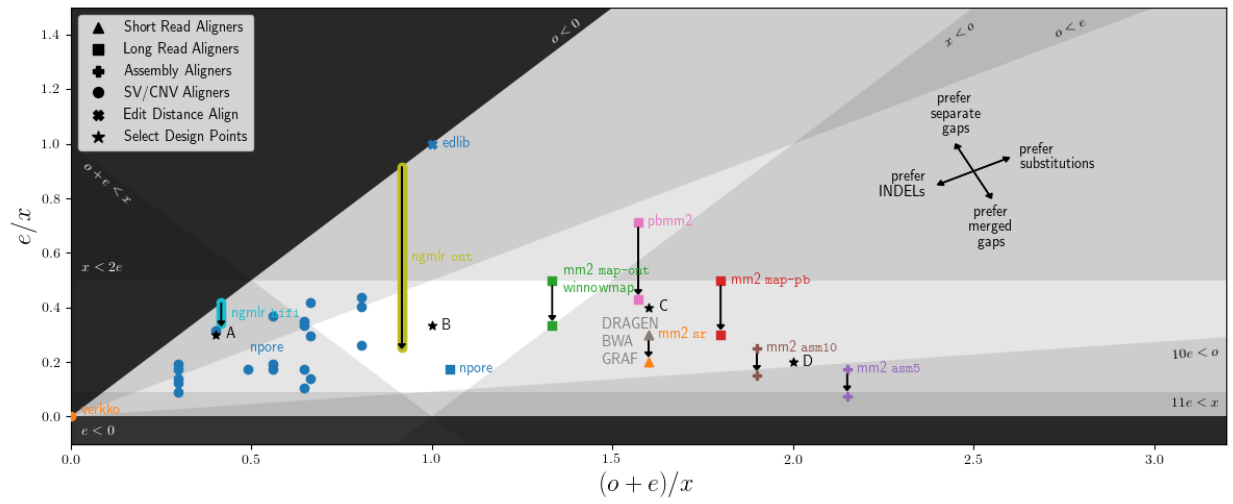


Figure 5.12: Selection of variant representation design point B, which minimizes the edit distance (ED) and distinct edits (DE) in each variant representation, with DE weighted twice as heavily as ED.

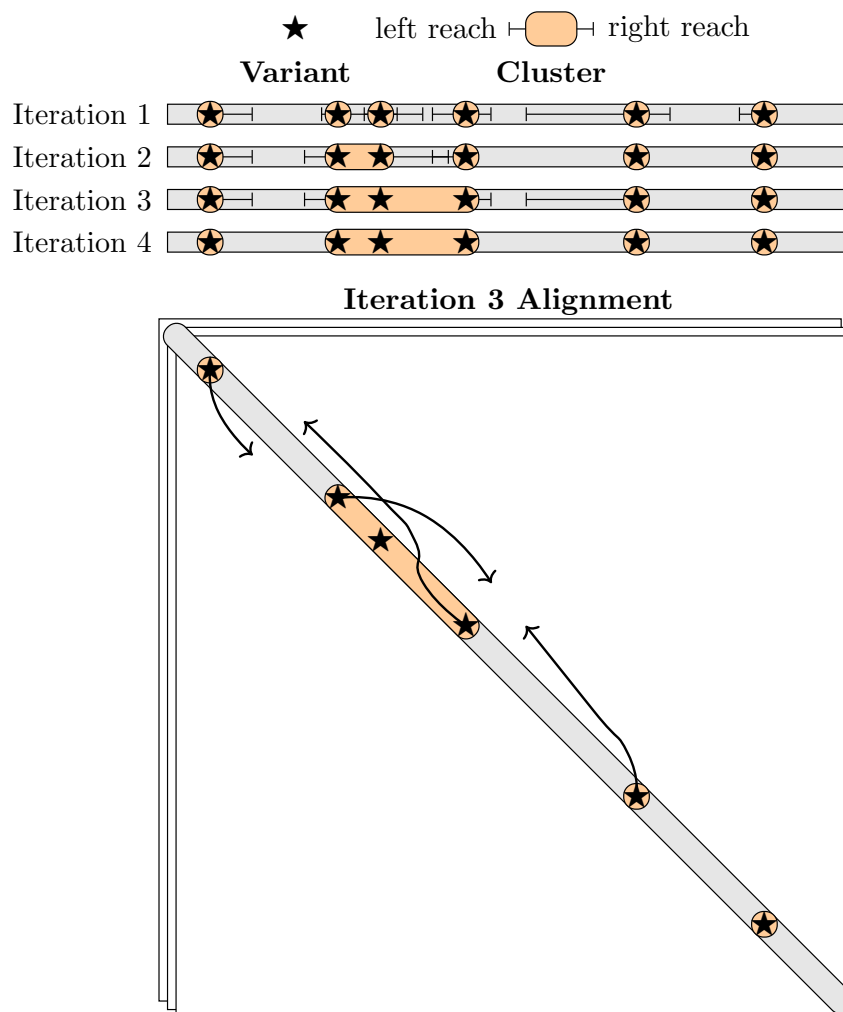


Figure 5.13: An overview of vcfdist’s clustering algorithm. Clusters are initialized to each variant and then iteratively grown by extending leftwards (from each active cluster’s end) and rightwards (from each active cluster’s start) under Smith-Waterman alignment (disallowing reference diagonal matches, the gray shaded region) until a score limit is reached. Clusters are considered “active” if adjacent to a cluster that was merged in a previous iteration. Two clusters are considered dependent if the leftmost extension of the right cluster overlaps with the rightmost extension of the left cluster. Dependent clusters are merged after each iteration until no more clusters are active.

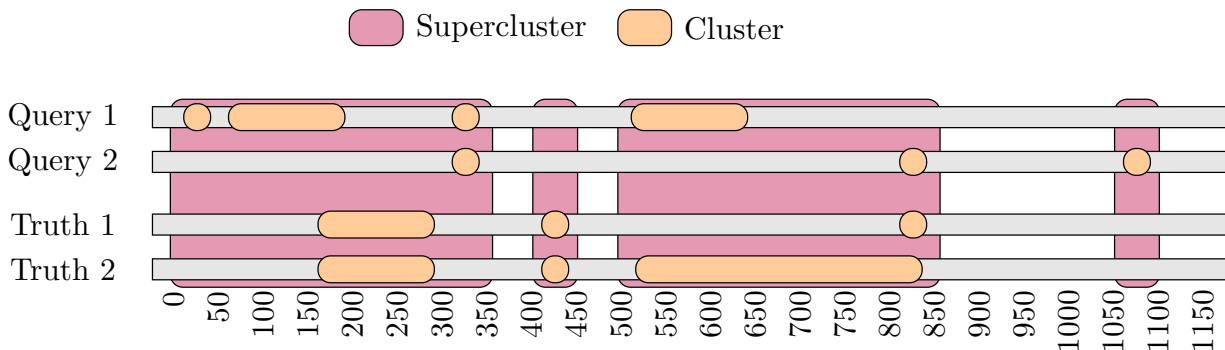


Figure 5.14: An overview of vcfdist's superclustering algorithm. This example demonstrates how variant clusters are grouped into superclusters, which span all haplotypes. Clusters within 50 bases of another cluster (on any haplotype) are merged into a single supercluster.

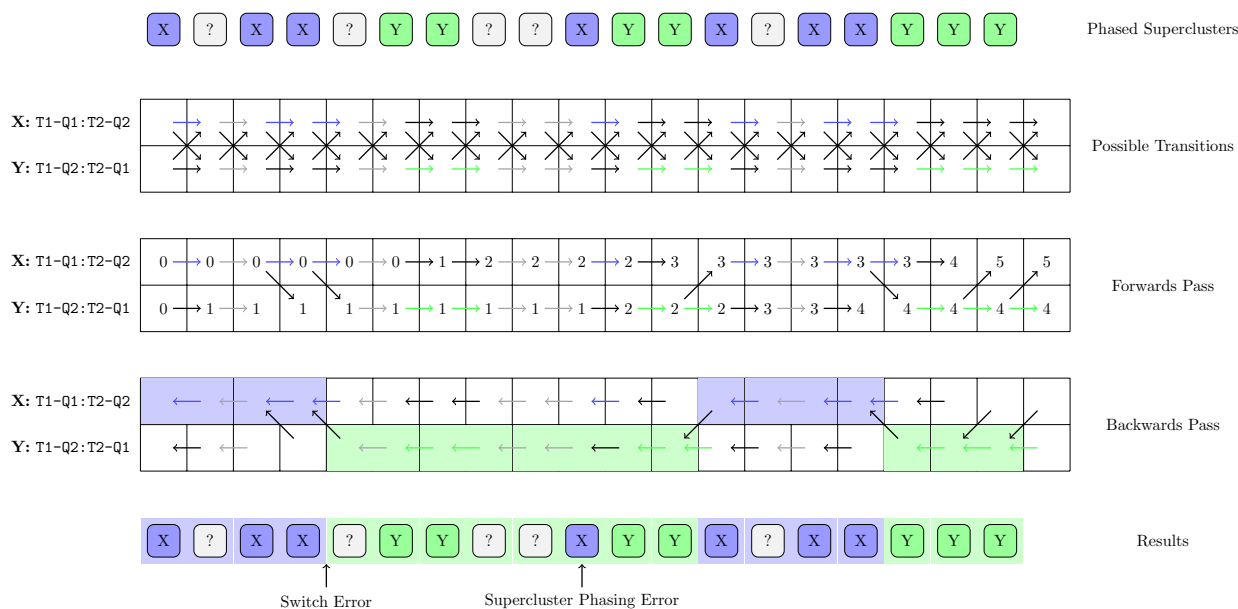


Figure 5.15: An overview of vcfdist's phasing algorithm, which steps through the two possible relative phasings of truth and query haplotypes (X: Truth1 to Query1 and Truth2 to Query2, or Y: Truth1 to Query2 and Truth2 to Query1) for each supercluster. It minimizes the total number of switch errors and supercluster phasing errors using a dynamic programming algorithm. Black arrows show penalized transitions. Note that either phase state X or Y is allowed in an uncategorized supercluster, and that switch errors are always penalized. Phase blocks are highlighted for phase states X (blue) and Y (green).

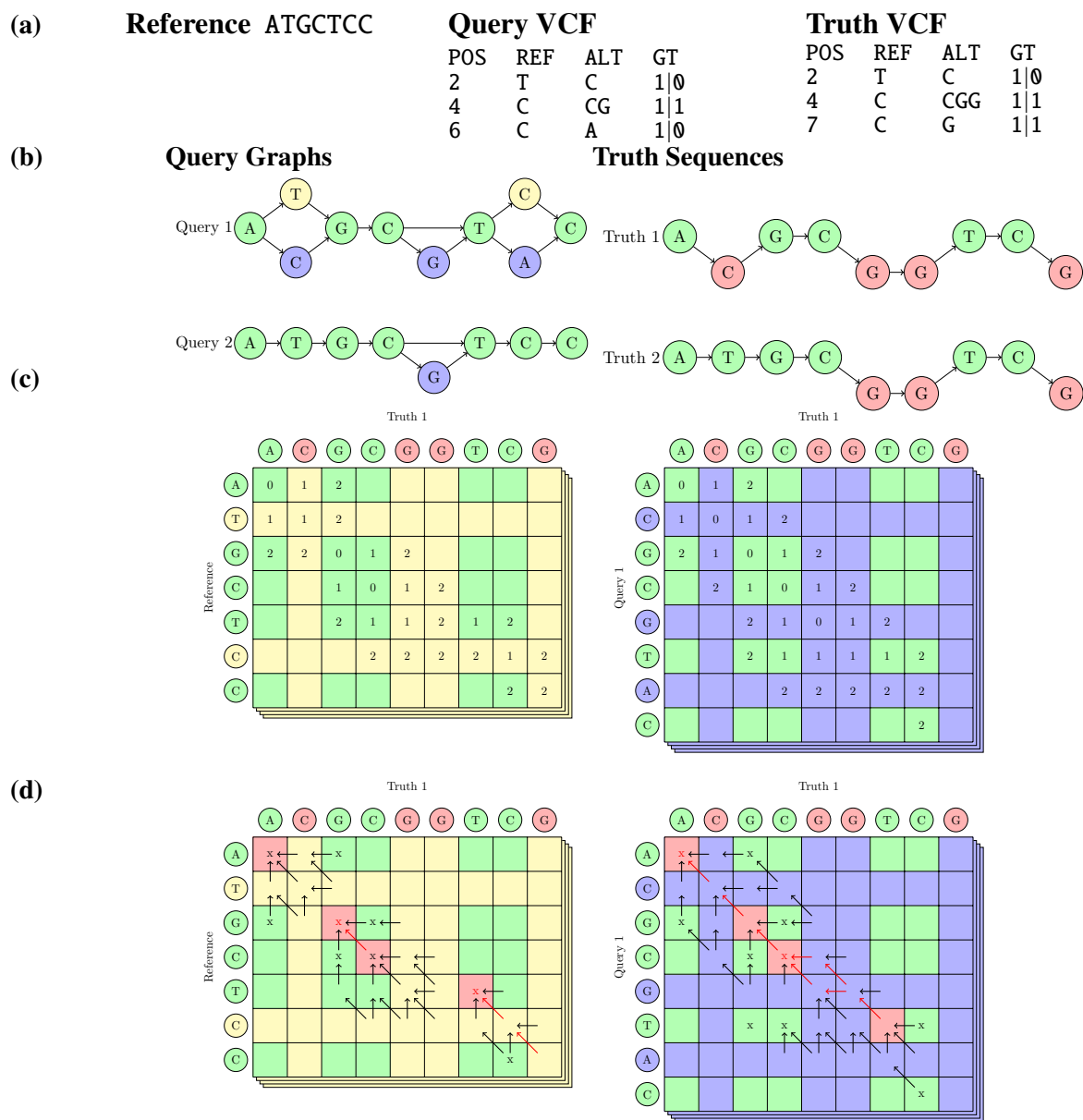


Figure 5.16: An overview of vcfdist’s alignment algorithm for precision and recall, allowing false positive query variants. **(a)** Inputs: reference sequence, phased query VCF, and phased truth VCF **(b)** For truth haplotypes, sequences are generated by applying variants (red) to the reference (green). For query haplotypes, variants (blue) are applied similarly but the reference sequence (green/yellow) is also retained, resulting in a graph. **(c)** For each of the four possible alignments of truth and query haplotypes, the forward-pass Dijkstra minimum edit distance algorithm performs alignment using two matrices. Transitions between the two matrices are allowed when both indices are corresponding reference bases (shown in green). **(d)** During the backwards pass, another Dijkstra search is required to maximize the number of false positive variants, among several possible minimum edit distance paths (red arrows). Corresponding reference bases with a single outgoing path edge are labelled “sync points” (red), and are used for calculating summary statistics.

5.6 Summary

This work, titled “vcfdist: Accurately benchmarking phased small variant calls in human genomes” was published in Nature Communications Volume 14, 2023 [3]. Thank you to my advisor, Satish Narayanasamy, who also contributed to this work.

CHAPTER 6

vcfdist2: Joint Evaluation of Small and Structural Variant Calls

6.1 Introduction

Recent improvements in long-read sequencing accuracy have enabled calling phased small and structural variants from a single analysis pipeline. Despite this, the current standard tools for variant calling evaluation are only designed for either small (vcfeval) or large (Truvari) variants. In this work, we extend vcfdist to be the first variant call benchmarking tool to jointly evaluate phased single-nucleotide polymorphisms (SNPs), small insertions/deletions (INDELs), and structural variants (SVs) for the whole genome. First, we find that a joint evaluation of small and structural variants uniformly reduces measured errors for SNPs (-28.9%), INDELs (-19.3%), and SVs (-52.4%) across three datasets. Next, we correct a common flaw in phasing evaluations, reducing measured flip errors by over 50%. Lastly, we show that vcfdist is more accurate than previously published works and on par with the newest approaches, but with improved result interpretability.

6.2 Background

Prior to the invention of DNA sequencing, structural variants (SVs) larger than 3Mb were observed using a microscope as early as 1959 [154, 155]. Following the initial sequencing of the human genome in 2001 using short read technologies [156], however, the focus of most research

investigations shifted to single nucleotide polymorphisms (SNPs) and small insertions and deletions (INDELs). It quickly became apparent that SNPs are the most common form of genetic variation, accounting for the approximately 0.1% difference in genomic sequence between two individuals [157], or about 3.1 million SNPs. Short-read technologies were well-poised to investigate these differences, due to their short read lengths but high per-base accuracy. It has since been determined that though SVs and INDELs are less common than SNPs, due to their larger size they account for a further 1.4% difference in genome composition between individuals [158], or about 43.2 million bases.

A few years later, in 2009, the first tools to identify structural variants from short-read alignments were developed [159, 160, 161]. Although short-read based structural variant callers remain widely used, they have relatively low recall (10-70%) due to the inherent difficulties of identifying large insertions and deletions from mapped short reads [162]. The accurate detection of structural variants was greatly assisted by the development of new long-read sequencing technologies around 2014, most notably from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) [89, 163]. Although early iterations of each technology had much lower per-base accuracy rates of around 85% [89, 163], longer read lengths led to unambiguous read mappings and more accurate structural variant calls [164]. Since then, accuracy has improved and both PacBio and ONT can sequence reads above 15Kb with 99 to 99.9% accuracy, rivalling the accuracy of short reads [115, 165]. As shown in Figure 6.1, this has led to the recent development of variant calling pipelines built from long-read sequencing data [115].

Once small and structural variants have been called, accurate comparison of variant call files (VCFs) is important for 1) genome-wide association studies (GWAS) [166, 167], 2) precision medicine [168], 3) variant annotation and effect prediction [169, 170], 4) sequencing and variant calling pipeline benchmarking [14, 25], and 5) variant database curation [171, 172]. In short, accurate VCF comparison is necessary for studying the impacts of genetic variants, for understanding the performance of variant calling methods, and for making decisions based on an individual's genetic composition. This information can then be used to identify mutations that cause genetic

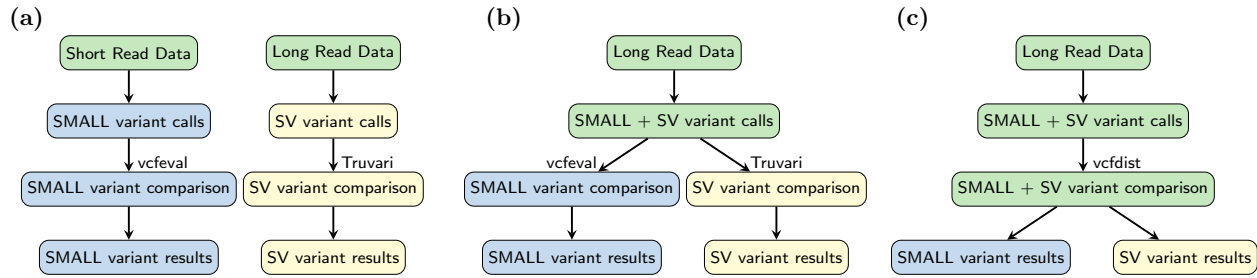


Figure 6.1: Pipeline overview for joint evaluation of small and structural variant calling. **(a)** Until recently, small and structural variants were called separately using different pipelines because they required different DNA sequencing technologies: short reads cannot be unambiguously mapped to call many structural variants accurately, and long reads were not accurate enough for precise small variant calling. **(b)** Due to recent improvements in long-read sequencing accuracy, whole genome sequencing (WGS) pipelines can now identify both small and structural variants (SVs) from the same sequencing data. It is still standard practice to evaluate these variant call categories separately, however, using *vcfeval* [14] for small variants and *Truvari bench* [25] for large INDELs and SVs. **(c)** We propose joint benchmarking of small and structural variant calls in this work, by extending *vcfdist* [3] to evaluate SVs. By comparing query and truth variants across size categories, *vcfdist* is able to detect a greater number of equivalent truth and query variants. This improves benchmarking accuracy, as shown in Figure 6.2.

diseases, to select the best variant calling pipeline for clinical usage, to develop targeted drugs, and to direct future research and funding.

Although small and structural variant calls can now be made from the same analysis pipeline, the current standard practice for VCF benchmarking involves separating small variants (smaller than 50bp) from structural variants (larger than 50bp) prior to benchmarking (see Figure 6.1). This 50bp threshold was selected for historical and technical reasons related to the limitations of short-read sequencing, not because a 50bp threshold is biologically significant in any way [162, 161]. Short-read sequencing’s variant calling performance is poor for INDELs larger than 50bp because the mappability of a 100bp read containing such a large variant is significantly reduced. For this reason, variants below and above this size threshold have been historically evaluated separately. Prematurely categorizing variant calls into small and structural variants prior to benchmarking has a significant impact on measured variant calling performance (see Figure 6.2), since several smaller variants are frequently equivalent to one or several larger variants.

The variant call file (VCF) format was first defined in 2011, and a simple exact variant com-

parison engine was released at the same time as part of vcftools [17]. vcfeval was introduced by Real Time Genomics (RTG) in 2015, and is capable of handling equivalent variant representations [14]. It was designed to evaluate unphased small variant calls, requires exact matches, and evaluates variants up to 1000bp in size. vcfeval has stood the test of time, being the recommended small variant calling evaluator by the Global Alliance for Genomics and Global Health (GA4GH) in 2019 [21, 136]. In 2023, vcfdist was released to evaluate locally phased small variant calls from long-read sequencing pipelines, relaxing vcfeval’s requirement that variants match exactly [3].

Most structural variant calling evaluators similarly allow inexact variant matches. Truvari bench, for example, considers two structural variants equivalent if they are located nearby on the reference, are of similar total size, overlap one another, and have a 70% similar sequence [25]. Although Truvari bench is able to perform whole genome SV comparison, it ignores small INDELs under 50bp by default, and is not currently recommended for evaluating SNPs [25]. Truvari’s refine module extends Truvari bench using an alignment algorithm (WFA [152], MAFFT [27], or POA [173]) to harmonize phased query and truth VCF variant representations for benchmarking INDELs and SVs at least 5bp in size [174].

In this work, we extend vcfdist to be the first tool to jointly evaluate phased SNP, INDEL, and SV calls for whole-genome datasets. Doing so required major internal restructuring and improvements to vcfdist to overcome scalability issues relating to memory and compute requirements. We show that performing a joint analysis of all variant sizes leads to better measured overall accuracy than when evaluating small and structural variants separately, reducing measured false negative and false positive variant calls by 28.9% for SNPs, by 19.3% for INDELs, and by 52.4% for SVs of over 50 bases. We find that vcfdist’s alignment-based analysis obtains more accurate results than vcfeval or Truvari bench and similar accuracy results to Truvari refine, but provides more interpretable results because the representation of evaluated truth and query variants is unchanged. Finally, we jointly evaluate SNP, INDEL, and SV phasing and show that between 42.6% and 92.2% of all phasing flip “errors” that popular phasing analysis tool WhatsHap reports are false positives. Differing variant representations cause variants to appear incorrectly phased, though they are not.

These false positive flip errors then lead to false positive switch errors, which will significantly affect downstream tertiary analyses. We demonstrate that vcfdist is able to avoid these errors in phasing analysis by using alignment-based variant comparison.

6.3 Results

6.3.1 Jointly Benchmarking Small and Structural Variants

In order to understand the impact of jointly benchmarking small and structural variants on measured accuracy, we evaluated three whole genome sequencing (WGS) datasets with vcfdist using several different variant subsets from the truth and query VCFs. More information on these WGS datasets can be found in the Methods section and Table 6.5. Figure 6.2 shows that compared to existing methodologies, which evaluate small variants (in orange) and structural variants (in red) separately, jointly evaluating all variants (in green) leads to lower measured error rates for each variant category.

In Figure 6.2, the hifiasm-dipcall dataset uses alignment parameters which are identical to the draft benchmark Q100-dipcall VCF. As a result, it sees the lowest rates of improvement from a joint evaluation of small and structural variants: a 4.6% reduction in SNP errors, a 1.2% reduction in INDEL errors, and a 24.9% reduction in SV errors. The hifiasm-GIAB-TR VCF uses the same assembly as hifiasm-dipcall with very different alignment parameters, and therefore sees great benefits from a joint evaluation: a 62.5% reduction in SNP errors, a 34.8% reduction in INDEL errors, and a 75.3% reduction in SV errors. The Q100-PAV VCF lies somewhere between these two extremes, with a 19.5% reduction in SNP errors, a 21.7% reduction in INDEL errors, and a 57.0% reduction in SV errors. A visualization of the alignment parameters used for each dataset is included in Figure 6.6.

These performance improvements originate from cases where multiple smaller variants are found to be nearly or exactly equivalent to one or several larger variants. Figure 6.3, below, shows an example where this occurs and a joint evaluation of small and structural variants improves

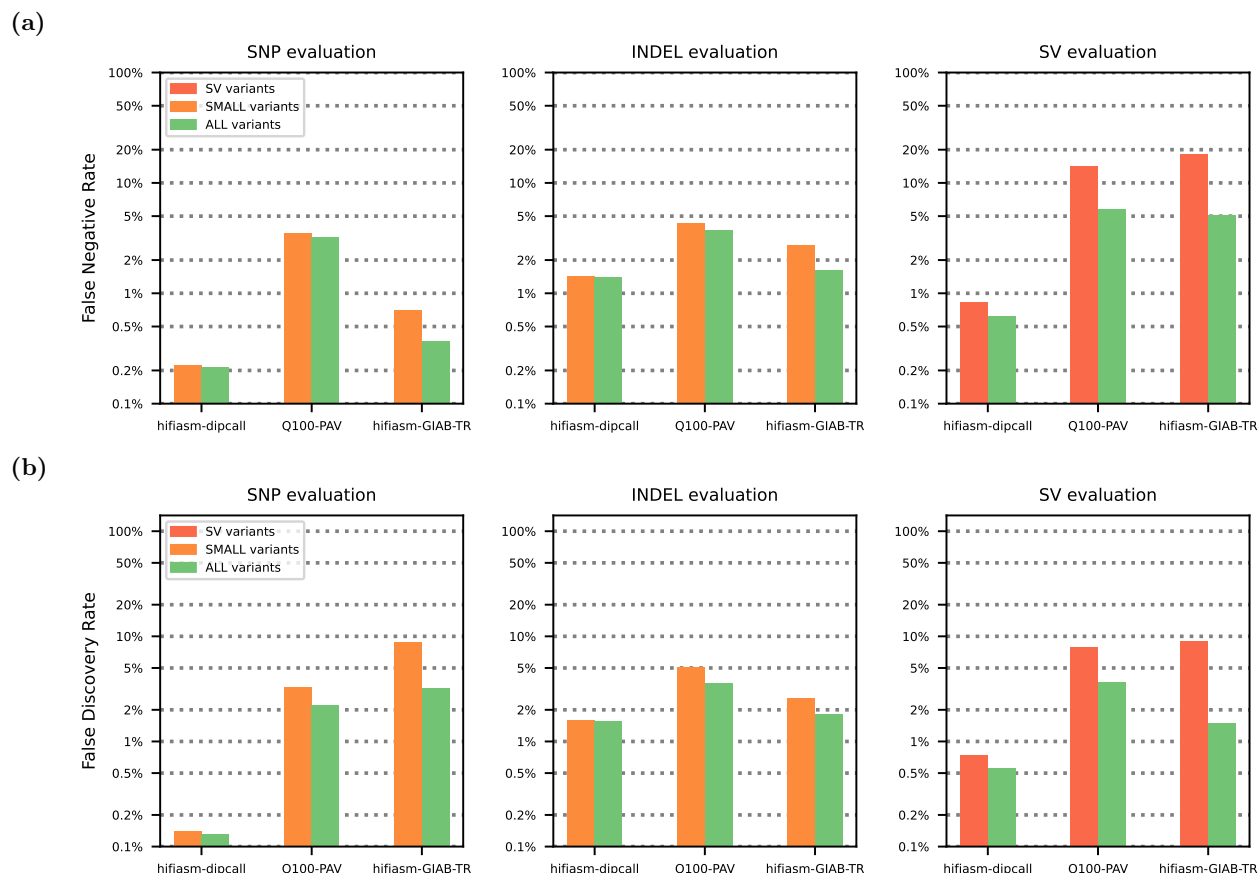


Figure 6.2: Joint evaluation of small and structural variant calls. We evaluated three phased HG002 whole genome sequencing (WGS) variant callsets (described in Table 6.5) on the whole-genome GIAB-Q100 benchmarking BED regions for small and structural variants using vcfdist. We show that compared to existing methodologies, which evaluate small variants (in orange) and structural variants (in red) separately, evaluating all variants at once (in green) leads to higher measured performance for each variant category. **(a)** false negative rate (FNR) and **(b)** false discovery rate (FDR) decrease when all variants are evaluated together, across all datasets. This occurs because correctly determining variant equivalence sometimes requires considering variants from multiple categories. Please note that results are plotted on a logarithmic scale.

measured performance. A similar example containing sequence information in VCF format is provided in Figure 6.7.

6.3.2 Improved Variant Comparisons for Phasing Evaluation

To understand joint phasing evaluation accuracy, we compare vcfdist to WhatsHap, a current standard for phasing evaluation [175, 23]. WhatsHap’s compare module performs one-to-one variant

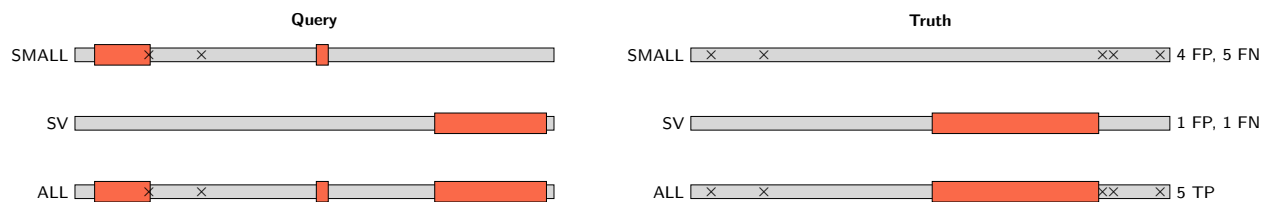


Figure 6.3: An example joint evaluation of small and structural variant calls. This example is taken from the Q100-PAV HG002 variant callset (chr1:3,287,250–3,287,700 on GRCh38), where using `vcfdist` to jointly evaluate small and structural variants improves measured performance. Single nucleotide polymorphisms (SNPs) are marked with black crosses, and deletions are represented as red rectangles. A joint evaluation of all variants discovers that truth and query haplotypes are identical, despite variant representation differences. By prematurely categorizing variants prior to evaluation into small and structural variants, this equivalence cannot be determined and variants would be classified as false positive (FP) and false negative (FN) variant calls instead of true positives (TP).

comparisons between truth and query VCFs to evaluate phasing correctness. For each heterozygous query variant, WhatsHap searches for an identical truth variant and notes whether that truth variant has the same or opposite phasing of the corresponding query variant. Within each phase block, WhatsHap then uses a simple dynamic programming algorithm to minimize the total number of flip errors (in which the phase of a single variant is mismatched) and switch errors (in which the phases of all following variants are mismatched) [175]. Although WhatsHap’s approach seems intuitively correct, it breaks down in repetitive regions of the genome where differing variant representations can result in false positive reported flip errors. Table 6.1 clearly shows that WhatsHap reports far more switch and flip errors than `vcfdist` on the exact same variant calls, particularly for the Q100-PAV and hifiasm-GIAB-TR datasets.

In contrast to WhatsHap, `vcfdist` performs full alignment of all nearby truth and query variants (a “supercluster”), and is able to discover equivalencies in variant representations. As a result, `vcfdist` reports far fewer phasing errors. The Q100-PAV VCF contains the fewest switch and flip errors, likely because it was produced using the same verkko assembly as the draft benchmark Q100-dipcall VCF. For the hifiasm-dipcall and hifiasm-GIAB-TR VCFs, `vcfdist` reports nearly identical switch and flip error rates. We believe this is because they were both produced using the same hifiasm scaffold [176]. In comparison, WhatsHap reports a much higher combined switch

Dataset	Tool	Measured Switch Errors	Measured Flip Errors
hifiasm-dipcall	WhatsHap	610	396
	vcfdist	494	390
Q100-PAV	WhatsHap	324	433
	vcfdist	6	52
hifiasm-GIAB-TR	WhatsHap	1074	1004
	vcfdist	494	396

Table 6.1: Comparison of phasing results for vcfdist and WhatsHap compare. Results are shown relative to the Q100-dipcall truth VCF on the whole-genome GIAB-Q100 BED. WhatsHap consistently reports more switch and flip errors than vcfdist. We demonstrate in Figure 6.8 that most of these supposed phasing errors are actually correctly phased and provide an example below in Figure 6.4.

and flip error rate for the hifiasm-GIAB-TR VCF than for the hifiasm-dipcall VCF. We expect this is because the variant representation used by the hifiasm-GIAB-TR callset differs significantly from that used by the draft benchmark Q100-dipcall VCF, whereas the parameters used by the hifiasm-dipcall VCF are identical (Figure 6.6).

In Figure 6.8, we present an extensive comparison of the switch and flip errors reported by WhatsHap and vcfdist. We find that 42.6% of flip errors reported by WhatsHap proved to be false positives, since the truth and query sequences match exactly when all neighboring variants are considered. An example is shown below in Figure 6.4, where WhatsHap reports a flip error within a complex variant even though both truth and query haplotypes match exactly. A further 49.6% of the flip errors reported by WhatsHap were not classified as flip errors by vcfdist due to insufficient evidence. Since there was no ground truth for these instances, we manually examined 16 random cases from each dataset (48 in total) where the ground truth was unknown. We found that classifying flip errors with WhatsHap resulted in 43 false positives, 4 true negatives, and 1 true positive. In comparison, classifying flip errors with vcfdist resulted in 40 true negatives, 7 false positives, and 1 false negative when compared to a manual examination. As can be seen in Table 6.1, these excess false positive flip error calls by WhatsHap artificially inflate the reported switch error rate as well, which will significantly impact tertiary analyses.

(a)

CONTIG	POS	REF	ALT	FORMAT	TRUTH	QUERY
chr1	32,653,646	T	G	GT:BD:BC	0 1:TP:1.0	0 1:TP:1.0
chr1	32,653,657	TTTG	T	GT:BD:BC	0 1:TP:1.0	
chr1	32,653,658	TTG	T	GT:BD:BC		0 1:TP:1.0
chr1	32,653,658	TTG	T	GT:BD:BC	1 0:TP:1.0	
chr1	32,653,659	TG	T	GT:BD:BC		1 0:TP:1.0
chr1	32,653,665	TG	T	GT:BD:BC		1 1:TP:1.0
chr1	32,653,666	G	T	GT:BD:BC	1 1:TP:1.0	

(b)

	Position	37	46	47	58	59	60	61	66	67
	Reference	GTTTTTTTTT	T	TTTTTTTTTTTT	T	T	G	TTTTTT	G	TTTT
Haplotype 1	Truth	GTTTTTTTTT	T	TTTTTTTTTTTT	T			TTTTTT	T	TTTT
	Query	GTTTTTTTTT	T	TTTTTTTTTTTT	T	T		TTTTTT		TTTT
Haplotype 2	Truth	GTTTTTTTTT	G	TTTTTTTTTTTT				TTTTTT	T	TTTT
	Query	GTTTTTTTTT	G	TTTTTTTTTTTT	T			TTTTTT		TTTT

Figure 6.4: An example WhatsHap compare false positive flip error call. **(a)** The variant call file (VCF) for an example WhatsHap false positive flip error call. Each VCF record shows the variant chromosome (CONTIG) and positions (POS) in addition to the reference (REF) and alternate (ALT) alleles and their genotypes (GT), the benchmarking decision (BD), and benchmarking credit (BC). In isolation, the two base deletion at position 32,653,658 (highlighted in red) appears to be phased differently between the truth and query VCFs (i.e. a flip error). **(b)** The resulting haplotype sequences. When this supposed flip error is considered in the context of the surrounding variants, vcfdist is able to determine that the two sets of truth and query variant calls are equivalent because both truth and query haplotype sequences are exactly the same. As a result, it is clear that no such flip error has occurred and differences between the truth and query VCF are due solely to differing variant representations.

6.3.3 Accuracy and Runtime

Next, we compare vcfdist to previously published works vcfeval [14] and Truvari bench [25], designed for evaluating small and structural variants, respectively. We also benchmark the performance of Truvari’s refine module, a recently developed extension which realigns truth and query variants to one another using MAFFT [27], wavefront alignment (WFA) [152], or partial-order alignment (POA) [173] for more accurate benchmarking. Truvari refine achieves similar accuracy to vcfdist but changes the total counts of truth and query variants, making comparisons across different evaluation tools and pipelines difficult (see Table 6.3 for an example). All current versions of Truvari do not evaluate SNPs, since Truvari was designed for SV evaluation.

At the other end of the spectrum, vcfeval only evaluates variants smaller than 1000 bases. For this reason, we restrict the maximum variant size to 1Kb in Figure 6.5. As variant length increases in Figure 6.5, vcfeval reports an increasingly high error rate compared to vcfdist (90.6%

higher for SNPs, 128% higher for INDELs, and 321% higher for SVs). This is because vcfeval requires truth and query variants to match exactly (which is less likely for larger variants), whereas vcfdist and Truvari do not. A more lenient matching heuristic will lead to strictly fewer false positives and false negatives in Figure 6.5. To avoid falsely inflating vcfdist's performance, we set vcfdist's credit threshold to 70% in order to match Truvari's sequence similarity threshold of 70% as closely as possible. We additionally standardize the method of variant counting across all tools in Figure 6.5 to be consistent with vcfdist because otherwise differences in counting credit for partial allele matches would dominate the results (see Table 6.4).

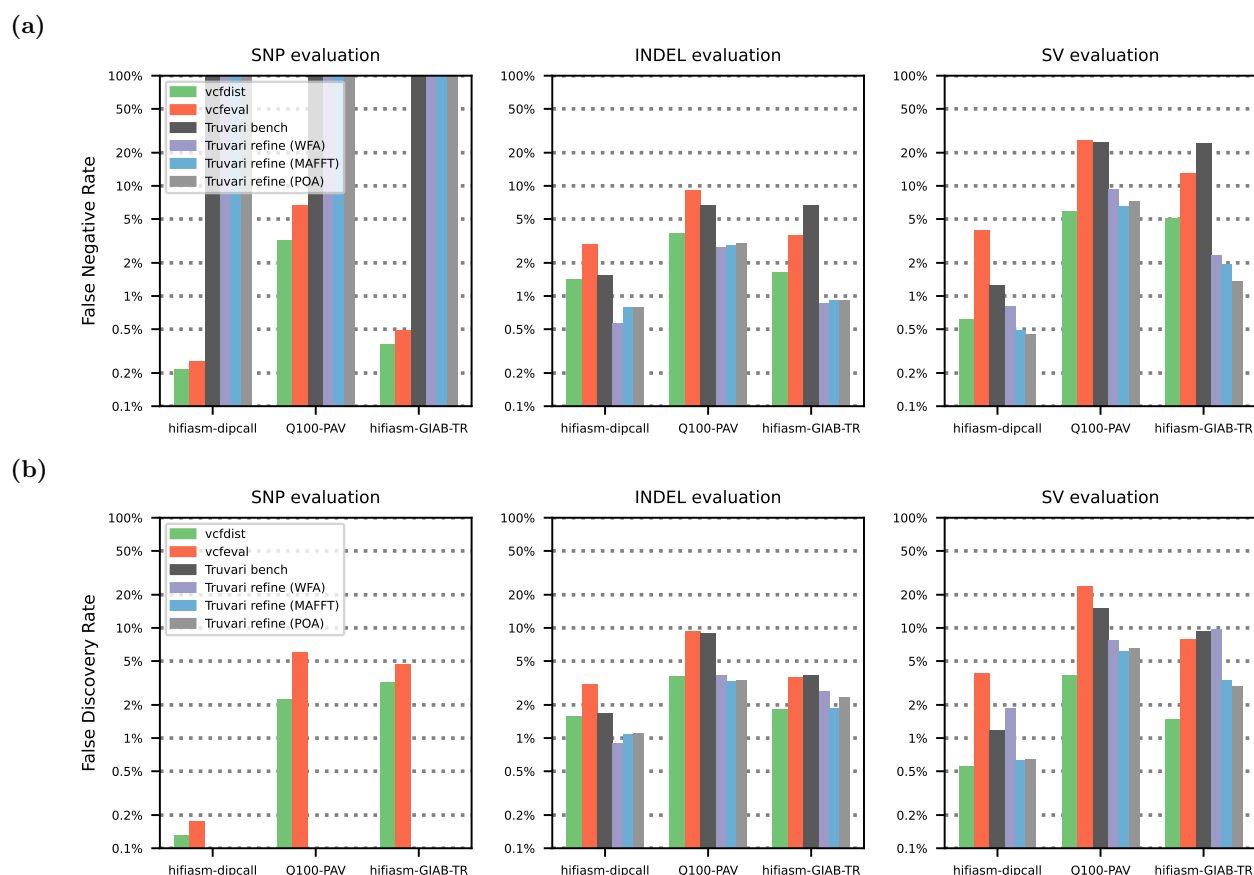


Figure 6.5: Comparison of vcfdist whole genome variant calling evaluation with prior works, which include vcfeval and Truvari bench (and its unpublished refine variants), in terms of measured **(a)** false negative rate (FNR) and **(b)** false discovery rate (FDR) on the GIAB-Q100 BED, which contains benchmarking regions covering 90.3% of the human genome. Note that Truvari does not evaluate SNPs, and that all results are plotted on a logarithmic scale.

Figure 6.5 shows that vcfdist measures lower false negative and discovery rates for all vari-

ant sizes across all three datasets when compared to previously published works Truvari bench and vcfeval. It is able to accomplish this by allowing inexact matches, evaluating groups of variants simultaneously, and allowing variant matches to occur across size categories. In comparison to Truvari refine, vcfdist achieves a similar improvement in benchmarking accuracy but without modifying the variant representations during benchmarking, and evaluating SNPs in addition to INDELs and SVs. Despite vcfdist’s advantages, we find that Truvari refine currently scales better with variant size, since it uses more memory-efficient alignment algorithms. We plan to incorporate wavefront alignment (WFA) [152] into the next release of vcfdist, but for now the maximum recommended variant length is 10Kb. Table 6.2 shows the runtimes of vcfdist, vcfeval, Truvari bench, and Truvari refine on our server; configuration details are provided in the Methods section.

	GIAB-Q100 BED Runtime		
	hifiasm-dipcall	Q100-PAV	hifiasm-GIAB-TR
vcfdist	52:19	1:04:56	59:22
vcfeval	14:40	50:39	46:18
Truvari bench	10:03	10:59	10:46
Truvari refine (MAFFT)	14:10	23:08	22:36
Truvari refine (WFA)	12:26	1:13:53	1:24:03
Truvari refine (POA)	11:57	21:47	24:39

Table 6.2: Whole genome evaluation runtime comparison. Runtime results are shown for vcfdist, vcfeval, Truvari bench, and Truvari refine in *(hh:)mm:ss* format.

Both Truvari and vcfdist perform alignment-based evaluation, which allows detection of variant calls that are mostly but not exactly correct. This is crucial for identifying large structural variants that are less likely to be called perfectly. In contrast, vcfeval finds matching subsets of truth and query variants that result in the exact same haplotype sequence. This computation would be less expensive if not for the fact that vcfeval does not assume the input VCFs are phased. Because there are 2^n possible phasings for n heterozygous variants, vcfeval’s runtime depends more closely on the number and representation of variants than either Truvari or vcfdist. As a result, vcfeval has a wide range of runtimes. It is also important to note that when the number of nearby heterozygous variants is too large, vcfeval fails to complete. This happened for 9,712 variants (0.20%) on the hifiasm-dipcall VCF, for 21,886 variants (0.45%) on the Q100-PAV VCF, and for 136,073 variants (2.50%) on the hifiasm-GIAB-TR VCF.

The runtimes of `vcfdist` and `Truvari`, on the other hand, depend closely on the size of the sequences to be aligned. `Truvari` reduces total runtime by evaluating variants in two stages; only complex regions are passed on to `Truvari`’s `refine` module for more sophisticated evaluation. `vcfdist` segments contigs into independent superclusters using heuristics or a bidirectional wavefront algorithm (biWFA) [153], as shown in Table 6.7.

6.3.4 Interpretability of Benchmarking Tools

The results of variant call benchmarking tools such as `vcfeval`, `Truvari`, and `vcfdist` are frequently used in downstream tertiary analyses. In order to compare the interpretability of these tools, we evaluated the chr6:32,664,600–32,664,899 region of the HLA-DQB1 gene, since it is known to cause difficulties during analysis arising from differing variant representations. The HLA-DQB1 gene is part of a family of genes called the human leukocyte antigen (HLA) complex, and plays an essential role in the human immune system. Deleterious mutations in HLA-DQB1 are highly associated with common autoimmune diseases such as celiac disease [177] and multiple sclerosis (MS) [178].

		SNP Results				INDEL Results				SV Results			
		Truth		Query		Truth		Query		Truth		Query	
		TP	FN	TP	FP	TP	FN	TP	FP	TP	FN	TP	FP
vcfdist	hifiasm-dipcall	32	0	32	0	2	0	2	0	2	0	2	0
	Q100-PAV	32	0	66	0	2	0	3	0	2	0	0	0
	hifiasm-GIAB-TR	32	0	71	0	2	0	1	0	2	0	0	0
vcfeval	hifiasm-dipcall	14	0	14	0	2	0	2	0	2	0	2	0
	Q100-PAV	13	0	13	4	1	0	0	0	2	0	0	0
	hifiasm-GIAB-TR	12	0	12	4	0	0	0	0	2	0	0	0
Truvari bench	hifiasm-dipcall	*	*	*	*	2	0	2	0	2	0	2	0
	Q100-PAV	*	*	*	*	2	0	2	1	0	2	0	0
	hifiasm-GIAB-TR	*	*	*	*	1	1	1	0	0	2	0	0
Truvari refine	hifiasm-dipcall	*	*	*	*	2	0	2	0	2	0	2	0
	Q100-PAV	*	*	*	*	3	0	3	0	0	0	0	0
	hifiasm-GIAB-TR	*	*	*	*	3	0	3	0	0	0	0	0

Table 6.3: Comparison of evaluations of the HLA-DQB1 gene, which is located at chr6:32,664,600–32,664,899. All three query VCFs called both haplotype sequences exactly correct; the resulting sequences are shared in Figure 6.6. As a result, there should be no false negative (FN) or false positive (FP) variant calls counted, and the number of truth VCF true positives (TP) should be consistent across all three datasets. The only tool that correctly ascertains this is `vcfdist`. SNP results for `Truvari` are marked with a * because `Truvari` does not evaluate SNPs.

We found through a manual examination that all three query VCFs called both haplotype sequences in this region exactly correct. A summary of these results is shown in Table 6.3, and the resulting sequences are included in Figure 6.6. Although all sequences were the same, there were significant differences in how this genetic variation was represented. Compared to the first Q100-dipcall haplotype, the Q100-PAV and hifiasm-GIAB-TR VCFs chose to represent a 169-base insertion and a 168-base deletion as a 1-base insertion and 34 SNPs. Relative to the second Q100-dipcall haplotype, the hifiasm-GIAB-TR VCF chose to represent a 1-base insertion and 1-base deletion as 5 SNPs. These differences account for the wide range of query true positive variant counts in Table 6.3.

Note that because both truth sequences were called exactly correct, there should be no false negative (FN) or false positive (FP) variant calls counted, and the number of truth VCF true positives (TP) should be consistent across all three datasets. `vcfdist` correctly counts all variants, as expected. `vcfeval` correctly counts the SVs, but fails to evaluate a large number of the SNP and INDEL variants because there are too many heterozygous variants in close proximity. It excludes these variants from the analysis and proceeds with a warning that the region is too complex to be evaluated. `Truvari bench` does not evaluate the SNPs, and fails to identify several of the INDELs and SVs. This failure occurs because it discards the SNPs prior to evaluation, and therefore does not discover the two cases where numerous SNPs are equivalent to an insertion and deletion. `Truvari refine` also does not evaluate SNPs. It is able to detect that all variant calls are correct, though it does convert both query and truth SVs to an INDEL for the Q100-PAV and hifiasm-GIAB-TR datasets.

6.3.5 Validating Correctness

Lastly, we compare `vcfdist` to existing variant calling evaluation tools in order to verify its correctness. Following variant normalization (described in the Methods section), we organize all variants evaluated by `vcfdist`, `vcfeval`, and `Truvari` in Table 6.4. 57,865 SNPs in this region are excluded from Figure 6.4 because they were not evaluated by `Truvari`; we include these results in

Figure 6.9a. An additional 723 INDELs and SVs occurring at the border of the GIAB-TR BED regions are excluded because they were only analyzed by some of the VCF comparison tools. Variants are then categorized based on the apparent reason for differences in evaluated correctness between the three tools and counted. An example from each of the eight discovered categories is shown in Figure 6.10, and described in further detail below.

Category	Count	Allele Match	Different Thresholds	Complex Variant	Pick Single	Flip Error	Backtracking Tie	Variant Overlap
all agree FP	451							
only vcfeval calls TP	5							5
only Truvari calls TP	256	223	12	12				9
only vcfdist calls TP	23			10	13			
all agree TP	35,376							
only vcfeval calls FP	662	346	307			1	8	
only Truvari calls FP	1						1	
only vcfdist calls FP	52		2	37		6	6	1

Table 6.4: A manual examination of differences between variant calling evaluation methods. We compare INDEL and SV variant calling evaluation by vcfdist, vcfeval, and Truvari, restricting to chr20 of the GIAB-TR tandem repeats BED. SNPs were excluded because they were not evaluated by Truvari. Prior to evaluation, truth and query VCFs were normalized using Truvari phab. This means that the Truvari results reported in this figure are largely equivalent to Truvari refine (MAFFT). Variants that were not evaluated uniformly by the three tools are categorized and counted. Each category is described in greater detail within the text, below, and an example provided in Figure 6.10.

Firstly, all three tools handle allele matches differently, which accounts for the majority of differences in Table 6.4. Truvari will match a query homozygous variant to a truth heterozygous variant and consider both to be true positives. vcfeval will perform the same match but consider the variants to be a false positive and false negative. vcfdist will match the heterozygous variant to one haplotype of the homozygous variant, consider both to be true positives, and then consider the second haplotype of the query homozygous variant to be a false positive. None of these methods is best; rather, each has strengths for certain applications. We caution that users consider these performance differences based on their downstream goals.

The second most common area of disagreement between tools stems from the fact that they have different thresholds for considering variant calls to be true positives. For example, vcfeval requires variants to match exactly, whereas vcfdist requires variants to have a partial credit score

above 0.7 and Truvari requires a sequence similarity above 0.7. For certain edge cases, such as where a length three deletion is called length four, even Truvari and vcfdist may differ.

The next most common differences are intentional implementation differences between the tools. In particular, vcfdist refuses to split a complex variant into multiple variants and consider only a subset of those to be correct. Truvari, by default, only allows a variant to participate in one match (with the `---pick single` parameter), regardless of allele count. vcfeval is the only tool that does not enforce local phasing and allows flip errors of nearby variants to occur.

Lastly, there are rare cases where unintentional implementation differences lead to slightly differing results. Not all backtracking algorithms behave identically, leading to cases in which different algorithms will adjudicate which of a pair of variants is a false positive differently. There are also differences due to directly adjacent or overlapping variants. For example, only vcfeval allows two separate insertions at the same exact location on the same haplotype and Truvari evaluates spanning deletions, whereas vcfeval and vcfdist do not.

6.4 Discussion

In this chapter, we demonstrate that evaluating small and structural variants together is necessary for discovering equivalent sets of truth and query variant calls. Furthermore, we show that intelligent variant comparison, which is able to identify equivalent variant representations, is important for accurate phasing analyses. We then show that vcfdist is now able to scale to whole-genome analysis of phased SNPs, INDELs, and SVs with improved accuracy over prior work. Lastly, we describe and explain the differences between vcfdist and prior work as they relate to our variant calling benchmarking results.

As variant calling performance improves and increasingly complex clusters of variants are evaluated, minor differences in the implementations of evaluation tools such as Truvari, vcfeval, and vcfdist begin to significantly impact the results. Currently, the way each tool handles partially correct variant calls differs greatly. Partial correctness can occur in many ways: a single insertion

is called mostly but not entirely correct, a homozygous variant is called heterozygous (genotype error), a deletion is called with the incorrect length, a heterozygous variant is called on the wrong haplotype (flip error), or only a subset of several variants that comprise a complex variant are called. In Table 6.4, we show that differences in how these cases are handled lead to significant differences in the reported summary metrics (Figure 6.5). Ideally, as a community we would define a standard methodology to handle each of these cases and clearly delineate how to count and categorize these errors. Unfortunately, there are both advantages and disadvantages for each method of categorizing and counting these errors, and the best method to use may depend on the end application. In the absence of a standardized approach, it is important that users of variant calling evaluation tools understand how each tool handles these cases, and the impact that may have on their results.

In this work, we have revisited some of the earlier design decisions we made in `vcfdist` v1. Although we still believe that total alignment distance is a useful supplementary metric to precision and recall curves, we now skip this computation by default, and allow re-enabling it with the `----distance` parameter. We believe that stratifying precision and recall curves by variant size offers many of the same benefits, with a more easily interpretable result. We have also replaced variant calling partial credit with a credit threshold `----credit-threshold [0.7]`. Partial credit is still calculated, but rather than assigning partial false and true positives (which is unnecessarily complicated and non-intuitive), we allow the user to select a partial credit threshold above which variants are considered true positives and below which variants are considered false positives. This more closely aligns with the behavior of other structural variant calling benchmarking tools such as Truvari. Lastly, `vcfdist` no longer realigns truth and query variants to a standard normalized representation by default. We found this behaviour to be undesirable in Truvari refine because it complicates comparisons with other pipelines or datasets. `vcfdist` still retains this capability, however, which can be enabled using `----realign-query` and `----realign-truth`. Although we no longer enable this feature by default, we urge individuals benchmarking variant calling pipelines to be aware of the variant representations used in their truth and query VCFs.

At the moment, `vcfdist` is designed to compare phased variants from a single sample query VCF to a truth VCF. We plan to extend `vcfdist` in the near future to handle unphased variant calls as well, since many genomic datasets do not contain phasing information. Along a similar vein, we would like for `vcfdist` to be able to work with multi-sample population VCFs for use in genome wide association studies (GWAS). We believe joint evaluation of small and structural variants will be incredibly valuable in this context. In order to make this a reality, we will need to continue to improve the efficiency of `vcfdist`. The alignment-based calculation of precision and recall will need to be shifted to a wavefront alignment based implementation [152], and when large or many nearby variants are present, we may need to sacrifice accuracy in order to improve efficiency.

6.5 Conclusion

Recent improvements in long-read sequencing accuracy have enabled calling phased small and structural variants from a single analysis pipeline. Despite this, the current standard tools for variant calling evaluation are only designed for either small (`vcfeval`) or large (`Truvari`) variants. In this work we extend `vcfdist` – previously a small variant calling evaluator – to evaluate structural variants, making it the first benchmarking tool to jointly evaluate phased single-nucleotide polymorphisms (SNPs), small insertions/deletions (INDELs), and structural variants (SVs) for the whole genome.

We find that a joint evaluation reduces measured false negative and false positive variant calls across the board: by 28.9% for SNPs, 19.3% for INDELs, and 52.4% for SVs over 50 bases. Building on `vcfdist`'s alignment-based evaluation, we also jointly analyze phasing accuracy. `vcfdist` identifies that 43% to 92% of all flip errors called by standard phasing evaluation tool `WhatsHap` are false positives due to differences in variant representations. Lastly, we compare the accuracy of `vcfdist` to prior works and demonstrate that it is able to find more true positive variant matches than `vcfeval` and `Truvari bench`. We find that `vcfdist` performs similarly to `Truvari refine`, while also providing more easily interpretable results.

6.6 Methods

6.6.1 Code and Data Availability

All input data have been deposited in a Zenodo repository, publicly available at the following URL: <https://doi.org/10.5281/zenodo.10557082> [179]. The deposited data includes the Q100-dipcall VCF, Q100-PAV VCF, hifiasm-dipcall VCF, hifiasm-GIAB-TR VCF, GIAB-Q100 BED, GIAB-TR BED, and the GRCh38 reference FASTA.

All code for `vcfdist` and the benchmarking pipelines developed are available in a public Github repository (<https://github.com/TimD1/vcfdist>) under a permissive GNU GPLv3 license. The repository has also been deposited in Zenodo, at <https://doi.org/10.5281/zenodo.8368282> [180]. All scripts described below are available in the Github repository <https://github.com/TimD1/vcfdist> [180] in the `analysis-v2/` subdirectory.

6.6.2 Datasets

Q100-dipcall VCF: The v0.9 Q100-dipcall draft benchmark VCF and its associated GIAB-Q100 BED containing small and structural variants were used as the ground truth VCF throughout this chapter [115]. They were created during a collaboration between the Telomere-to-Telomere Consortium (T2T, <https://sites.google.com/ucsc.edu/t2tworkinggroup/home>), the Human Pangenome Reference Consortium (HPRC, <https://humanpangenome.org/>), and the Genome in a Bottle Consortium (GIAB, <https://www.nist.gov/programs-projects/genome-bottle>) in an attempt to establish a diploid whole genome benchmark that is perfectly accurate. The term “Q100” refers to a Phred quality score [181] of 100, or one error per ten billion bases (i.e. zero expected errors per human genome). The v0.9 draft benchmark contains many errors, but improvements are still being made towards this ultimate goal. A combination of data from Oxford Nanopore Technologies (ONT), Pacific Biosciences high-fidelity sequencing (HiFi), Strand-Seq, and Hi-C were used in combination with the trio-based `verkko` assembler [115] and manual review to create a high-quality assembly. Lastly, `dipcall` [182] was used to generate a VCF

of this assembly relative to the GRCh38 reference FASTA.

Q100-PAV VCF: The same verkko assembly [115] was then used by researchers at the National Institute of Standards and Technology (NIST, <https://nist.gov>) to generate a second VCF using the Phased Assembly Variant Caller (PAV) [183]. For this reason, the Q100-PAV variant phasings match very closely with the Q100-dipcall phasings, as can be seen in Table 6.1. Note that by default, PAV merges some non-identical haplotypes, resulting in inexact variant calls.

hifiasm-dipcall VCF: The hifiasm-dipcall VCF was created by the HPRC using a combination of ONT ultra-long (UL), HiFi, Hi-C, and Bionano optical mapping data [176]. First, the trio-based hifiasm assembler [184] was used to create the initial assembly using HiFi and Hi-C data. Bionano optical mapping data was used to verify these scaffolds, and a combination of manual variant curation and polishing with ONT-UL data was used to generate the final assembly [176]. Lastly, dipcall [182] was used to generate a VCF of this assembly relative to the GRCh38 reference FASTA.

hifiasm-GIAB-TR VCF: The v4.20 hifiasm-GIAB-TR VCF and its associated GIAB-TR BED were generated by the Genome In A Bottle Consortium (GIAB) using the same hifiasm assembly, in addition to custom scripts that use minimap2 and paftools [185]. The methodology is described in detail in [174], and was part of an effort to create a high-quality tandem repeat benchmark. Because the same hifiasm assembly was used, the phasing analysis results of hifiasm-GIAB-TR closely match hifiasm-dipcall in Table 6.1, but the variant representation is much different than the other VCFs, as shown in Figure 6.6 (see <https://github.com/ACEnglish/adotto/discussions/4> for details).

6.6.3 Preprocessing

Prior to evaluation, multi-allelic variants were split using bcftools norm v1.17 [15] with the `-m-any` parameter. Where required, the HG002 sample was extracted from the original VCF into a single-sample VCF using bcftools query.

6.6.4 Joint Evaluation of Small and Structural Variants

For each dataset, each subset (small variants (SNPs and INDELs) only, structural variants only, and all variants) of variants was first extracted into a separate VCF using `bcftools view v1.17` [15]. SVs were defined as insertions or deletions greater than or equal to 50 base pairs in size. INDELs are below this size threshold. `vcfdist v2.5.0` [180] was used to compare each VCF to the Q100-dipcall ground truth within the GIAB-Q100 benchmarking regions. Scripts available in the `small_sv_all/` directory of our Github repository [180] were used to calculate and plot the false negative and false discovery rates for each variant categorization within each VCF. The results are shown in Figure 6.2.

6.6.5 Phasing Analysis

The original `vcfdist v1` release contained an experimental phasing analysis algorithm that was untested and unready for production. In this work we extended the original algorithm, described in [3], to perform a proper evaluation of phasing. First, we added support for phase blocks using the input VCFs' `FORMAT:PS` fields. Unlike most other tools, `vcfdist` allows the ground truth VCF to contain phase sets as well. Using the reported phase sets, `vcfdist` now correctly identifies switch and flip errors. In addition to reporting detailed switch and flip error information, `vcfdist` also calculates several useful summary metrics such as phase block NG50 (breaking regions on new phase blocks), switch NGC50 (breaking regions on new phase blocks and switch errors), and switchflip NGC50 (breaking regions on new phase blocks, switch errors, and flip errors). The NG50 metric reports the largest region such that all regions of that size and larger cover at least 50% of the genome. Lastly, we added a phasing threshold so that variant clusters are considered unphased unless one phasing significantly improves the cluster's edit distance from the ground truth versus the other phasing. The default value is set at `---phasing-threshold [0.6]`, or 60% reduction in edit distance, although the results from Figure 6.8c suggest that a higher threshold may be appropriate.

First, because WhatsHap does not allow providing a BED file to mask analysis regions, we use

bcftools filter v1.7 [15] to restrict all three VCFs to the GIAB-Q100 benchmarking BED regions. We then performed phasing analyses using WhatsHap v2.1 [175] and vcfdist v2.5.0 [180]. We then used several scripts, available in the phasing/ directory of our Github repository [180], to compare and plot the resulting flip and switch errors reported by each tool. The results are shown in Figure 6.8 and Table 6.1. In Figure 6.8, a random subset of 16 clusters with unknown phasing from each VCF was selected and manually examined in order to define a ground truth and compare vcfdist’s and WhatsHap’s performances on this subset of cases.

6.6.6 Accuracy Comparison

In order to determine the differences in variant evaluation between vcfdist, vcfeval, Truvari bench, and Truvari refine, we evaluated all three datasets using each tool on the GIAB-Q100 BED. We ran vcfdist v2.5.0 [3] with `-l 1000` to limit the maximum SV length because vcfeval does not consider variants larger than 1000 bases. We ran rtg vcfeval v3.12.1 [14] with the following parameters: `----ref-overlap ----all-records ----vcf-score-field=QUAL`. We ran Truvari bench v4.2.1 with the following command line options: `----no-ref a ----sizemin 1 ----sizefilt 1 ----sizemax 1000 ----pick single ----typeignore ----dup-to-ins`. We then ran Truvari refine v4.2.1 using `----regions candidate.refine.bed` from the previous Truvari bench step with the following parameters: `----use-original-vcfs ----use-region-coords ----recount ----align <method>` where `<method>` was WFA, MAFFT, and POA. We then merged results from Truvari refine and bench by subtracting the `candidate.refine.bed` (used by Truvari refine) from the GIAB-Q100 BED using bcftools subtract v1.17, evaluating the remaining regions with Truvari bench, and then merging the results. Lastly, we used several scripts, available in the vs_prior_work/ directory of our Github repository [180], to compare and plot the false negative and false discovery rates of each tool. These results are shown in Figure 6.5.

The same methodology was used to evaluate the example complex variant in the HLA-DQB1 gene for each tool. Scripts are available in the hla/ directory of our Github repository [180], and

results are shown in Table 6.3.

6.6.7 Validation of vcfdist

In order to validate the correctness of vcfdist’s reported variant calling results, we needed a way to compare how vcfdist classified a single variant to the classifications reported by prior works vcfEval and Truvari refine (MAFFT) for that same variant. We chose to compare against Truvari refine instead of Truvari bench because it is more accurate [174]. Unfortunately, Truvari refine changes the representations of the input truth and query variants, making it difficult to compare benchmarking results across tools. Instead of directly using Truvari refine, we used an equivalent workflow that involves normalizing variant representations using Truvari phab (which internally uses MAFFT) followed by Truvari bench evaluation. This approach enabled us to directly compare the decisions made by vcfdist, vcfEval, and Truvari on the exact same set of variants while also enabling Truvari to be more accurate.

We first normalized chr20 of the GIAB-TR BED from each of our three VCFs with the truth VCF using Truvari v4.2.1 phab (MAFFT). We then converted these VCFs into single sample VCFs of the desired format and split up multi-allelic variants using bcftools reheader, norm, and view v1.18 [15]. Afterwards, we evaluated each VCF using vcfdist v2.5.0 [3], vcfEval v3.12.1 [14], and Truvari bench v4.2.1 [25]. Scripts available in the phab.cm/ directory of our Github repository [180] were used to run and summarize the evaluations. The results are presented in Table 6.4, Figure 6.9, and Figure 6.10.

6.6.8 Clustering Algorithms

A naive variant comparison algorithm would compare each query variant to a single reference variant individually in order to discover matches. While this approach works for the majority of variant calls, there are cases where several query variants are equivalent to one or many truth variants (Figure 6.11a). Several examples of this are shown in Figure 6.12. This is especially true for repetitive regions of the genome, or as the representations of the truth and query VCFs diverge.

In order for a benchmarking tool to recognize these cases of complex equivalency, all the variants involved must be evaluated at once in a group, or “cluster”.

The default clustering algorithm employed by `vcfdist` discovers all cases in which variants could participate in a complex match and groups those variants together into a single cluster. `vcfdist` achieves this by first initializing each cluster to a single variant. Next, the leftmost and rightmost reference positions that can be reached by aligning through each cluster with an alignment cost less than or equal to the cost of the current variant representation are recorded. If the reach of a cluster overlaps with the reach of a neighboring cluster, the two clusters are merged. This occurs until all clusters have stabilized.

In order to handle structural variants, `vcfdist`’s original clustering algorithm (described briefly above and in greater detail in [3]) was required to undergo significant changes to improve efficiency and reduce memory usage. Firstly, the bidirectional Smith-Waterman-Gotoh [102] algorithm used to calculate cluster left and right reaches was converted to a wavefront alignment based [152] equivalent in order to reduce memory usage from $O(n^2)$ to $O(n)$. Next, the alignment cost of each cluster was recalculated (and lowered, when possible) following each cluster merge in order to reduce unnecessary cluster growth in following iterations. The left and right reaches of each cluster were cached across iterations and only recalculated following a merge. A greedy merging strategy was employed to merge multiple clusters at once when possible. Cluster reaches were calculated using iterative reach doubling to avoid unnecessary computation. Lastly, multi-threading support was added for clustering.

`vcfdist` v2.5.0 was run with “gap n ” clustering for $n = (10, 100, 500)$ in addition to the default biWFA clustering using the command line options `----cluster gap 10`. Wall clock runtime was measured using the GNU `time` command on an Intel Xeon E5-2697 v3 CPU, with 56 threads and 64GB RAM. Scripts used to calculate the cluster sizes shown in Table 6.7 from `vcfdist`’s verbose outputs are available in the `clustering/` directory of our Github repository [180].

6.6.9 Evaluating Structural Variants

In order to evaluate larger structural variants using `vcfdist`, we made several changes such as introducing command line parameter `----largest [5000]` to control the size of variants evaluated by `vcfdist`. We also added summary metric reporting for structural variants separately from INDELs and added the `----sv-threshold [50]` flag to control the threshold for this classification. In addition to the numerous clustering efficiency improvements mentioned above, we decreased the memory usage of the precision and recall calculations. Since each cluster can be evaluated independently, we also added multi-threading and work balancing based on cluster size for all intra-cluster computations.

6.6.10 Runtime Comparison

Variants larger than 1000bp were pre-filtered using `bcftools view v1.17` and inversions were filtered using `GNU grep v2.20` because otherwise they significantly impact the runtime of `Truvari refine` (even when excluded from the analysis with `----sizemax 1000`). Wall clock runtime was measured using the `GNU time` command on an Intel Xeon E5-2697 v3 CPU, with 56 threads and 64GB RAM. The scripts with the exact parameters used to run each tool are available in the `vs_prior_work/` directory of our Github repository [180]. The results are shown in Table 6.2.

6.7 Supplementary Information

6.7.1 Datasets

	Q100-dipcall	hifiasm-dipcall	Q100-PAV	hifiasm-GIAB-TR
Consortium(s)	T2T, GIAB, HPRC	HPRC	NIST	GIAB
Assembler	verkko	hifiasm	verkko	hifiasm
Assembly Method	trio-based	trio-based	trio-based	trio-based
Assembly	T2T-HG002-Q100v0.9 polished	HPRC scaffold	T2T-HG002-Q100v0.9 polished	HPRC scaffold
Variant Caller	minimap2, dipcall	minimap2, dipcall	PAV	minimap2, paftools

Table 6.5: Summary of evaluated whole genome sequencing datasets, for each phased WGS dataset used in this chapter.

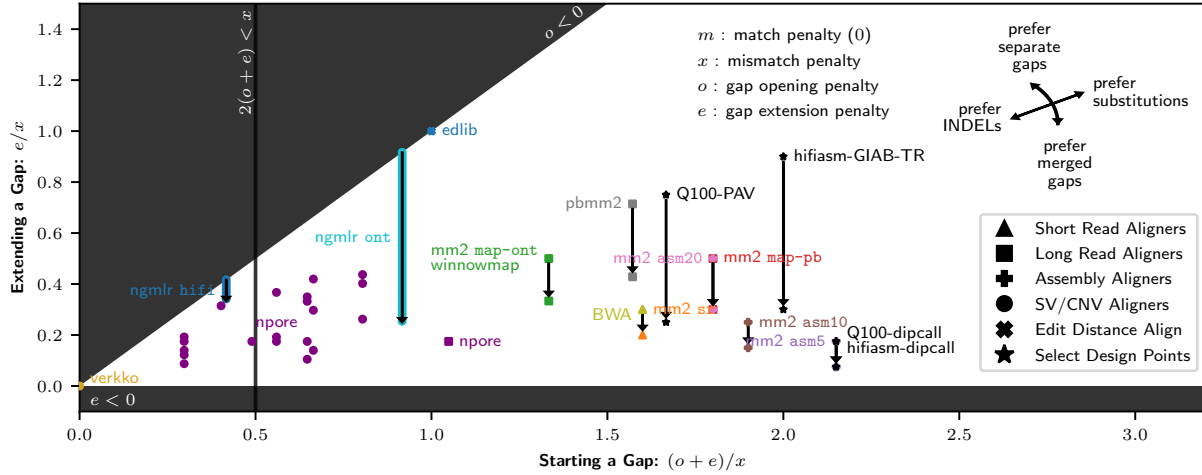


Figure 6.6: The affine gap design space for variant representation, with match, mismatch, gap opening, and gap extension penalties m , x , o , and e . All parameters have been normalized so that $m = 0$, and the penalties for starting $(o + e)$ and extending (e) a gap are plotted relative to substitutions (x) . This plot includes the variant representations used in all four datasets, along with short-read [143], long-read [144, 186], assembly [185], edit distance [24], copy number variant [2], and structural variant [112] aligners for comparison. Each aligner is plotted in a unique color, except for when multiple aligners use identical parameters. For dual affine gap aligners [185, 144, 186], two points are plotted with an arrow indicating the transition to a lower extension penalty e_2 . NGMLR [112] uses a logarithmic gap penalty, and so there is a continuous lowering of e . verkko is plotted at $(0, 0)$ because it uses tandem repeat compression [115]. nPoRe [2] uses different gap penalties for simple tandem repeats (STRs) based on their measured likelihoods, resulting in many plotted points.

6.7.2 Joint Benchmarking Example

SMALL						
CONTIG	POS	REF	ALT	FORMAT	TRUTH	QUERY
chr1	996555	T	C	GT:BD:BC	1 1:FN:0.000	
chr1	996559	G	A	GT:BD:BC		1 1:FP:0.000
chr1	996596	A	AGGACCCCCCGCTGGAGGGG	GT:BD:BC		1 1:FP:0.000
chr1	996674	G	T	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
chr1	996691	G	A	GT:BD:BC	1 1:FN:0.000	
chr1	996707	T	G	GT:BD:BC		1 1:FP:0.000
chr1	996722	G	A	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
chr1	996728	A	AG	GT:BD:BC		1 1:FP:0.000
chr1	996731	A	C	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
SV						
CONTIG	POS	REF	ALT	FORMAT	TRUTH	QUERY
chr1	996282	G	GGGGGCACCCACATCTGGGGCCACAGGATGCAG GGTGGGGAGGGCAGAAAGGCCCCCGCGGGAA	GT:BD:BC	1 1:TP:0.867	
chr1	996348	A	AGGGGCACCCACATCTGGGGCCACAGGATGCAG GGTGGGGAGGGCAGAAAGGCCCCCGCGGGAAAG GGGCACCCACATCTGGGGCCACAGGATGCAGGG TGGGGAGGGCAGAAAGGACCCCCGCTGGAGGGG GCACCTCACGTCTGGGGCCACAGGATGCAGGGTG GGGAGGACAGAAAGGACCCCCGCTGGAG	GT:BD:BC		1 1:TP:0.867
chr1	996728	A	AGGACCCCCGCTGGAGGGGGACCCCCGCTGG AGGGGGCACCCACATCTGGGGCCACAGGATGCA GGGTGGGGAGGGCAGAAAGGACCCCCGCTGGAG GGGGCACCTCACGTCTGGGGCCACAGGAGGCAGG GTGGGGAGGACAGAAAG	GT:BD:BC	1 1:TP:0.867	
ALL						
CONTIG	POS	REF	ALT	FORMAT	TRUTH	QUERY
chr1	996282	G	GGGGGCACCCACATCTGGGGCCACAGGATGCAG GGTGGGGAGGGCAGAAAGGCCCCCGCGGGAA	GT:BD:BC	1 1:TP:1.000	
chr1	996348	A	AGGGGCACCCACATCTGGGGCCACAGGATGCAG GGTGGGGAGGGCAGAAAGGCCCCCGCGGGAAAG GGGCACCCACATCTGGGGCCACAGGATGCAGGG TGGGGAGGGCAGAAAGGACCCCCGCTGGAGGGG GCACCTCACGTCTGGGGCCACAGGATGCAGGGTG GGGAGGACAGAAAGGACCCCCGCTGGAG	GT:BD:BC		1 1:TP:1.000
chr1	996350	A	G	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
chr1	996423	C	T	GT:BD:BC	1 1:TP:1.000	
chr1	996555	T	C	GT:BD:BC	1 1:TP:1.000	
chr1	996559	G	A	GT:BD:BC		1 1:TP:1.000
chr1	996596	A	AGGACCCCCGCTGGAGGGG	GT:BD:BC		1 1:TP:1.000
chr1	996674	G	T	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
chr1	996691	G	A	GT:BD:BC	1 1:TP:1.000	
chr1	996707	T	G	GT:BD:BC		1 1:TP:1.000
chr1	996722	G	A	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000
chr1	996728	A	AG	GT:BD:BC		1 1:TP:1.000
chr1	996728	A	AGGACCCCCGCTGGAGGGGGACCCCCGCTGG AGGGGGCACCCACATCTGGGGCCACAGGATGCA GGGTGGGGAGGGCAGAAAGGACCCCCGCTGGAG GGGGCACCTCACGTCTGGGGCCACAGGAGGCAGG GTGGGGAGGACAGAAAG	GT:BD:BC	1 1:TP:1.000	
chr1	996731	A	C	GT:BD:BC	1 1:TP:1.000	1 1:TP:1.000

Figure 6.7: An example evaluation of a complex variant (from the Q100-PAV dataset) where joint evaluation of small and structural variants changes the benchmarking results from 4 true positives (TP), 4 false positives (FP), and 2 false negatives (FN) to 9 true positives. Each variant call file (VCF) entry reports the variant contig (CONTIG), position (POS), reference allele (REF), alternate allele (ALT), and truth and query information: genotypes (GT), benchmarking decision (BD), and benchmarking credit (BC).

6.7.3 Phasing Analysis

In Figure 6.8b, vcfdist classifies a cluster's flip error status as NONE when the original haplotypes match exactly, and as FLIP when both haplotypes match exactly when the phasing of all variants is flipped. Because the full haplotypes match exactly, the ground truth is known and vcfdist is correct. When neither phasing results in an exact match, the ground truth is labelled UNKNOWN but vcfdist still makes a classification (NONE/FLIP) based on how much closer the edit distance between the truth and query haplotypes is when all variant phasings are flipped. Because the hifiasm-dipcall dataset uses the same variant representation as the Q100-dipcall ground truth, there are few WhatsHap false positive flip errors for this VCF. Once the query VCF variant representation differs from the ground truth, however, the Q100-PAV and hifiasm-GIAB-TR datasets show a large number of WhatsHap false positive flip errors in Figure 6.8b.

Because vcfdist enforces local phasing of variants within a cluster, it may miss single-variant flip errors when the phasing of one of several variants in a cluster is incorrect. In this case, the ground truth would be labelled UNKNOWN because neither phasing would result in the truth and query haplotypes matching exactly. A manual investigation of a random subset of these cases with unknown ground truth in Figure 6.8c shows that in most of these cases, no flip error has occurred. This shows that although vcfdist reports far fewer flip errors in total than WhatsHap, few of these are false negatives. The majority of cases with unknown ground truth are likely also WhatsHap false positives.

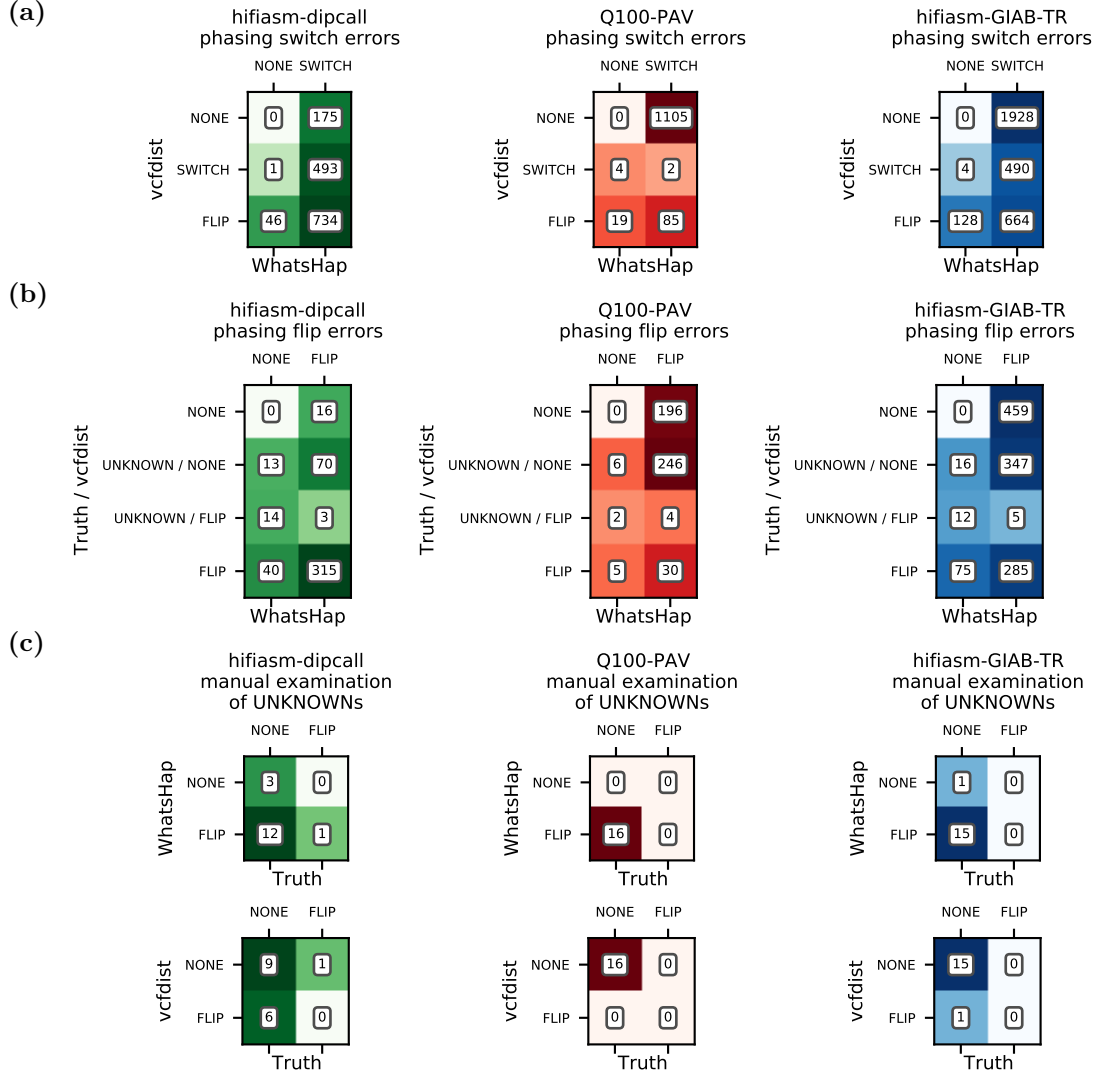


Figure 6.8: Switch and flip error confusion matrices. **(a)** Switch error confusion matrices for vcfdist and WhatsHap for all three whole genome sequencing datasets, evaluated on the GIAB-Q100 BED. Note that WhatsHap considers flips errors to be equivalent to two consecutive switch errors, whereas vcfdist counts each separately. **(b)** Flip error confusion matrices for vcfdist and WhatsHap. Note that cluster ground truths are only labelled as NONE or FLIP when absolutely certain, i.e. if both haplotypes with all variants using either the original or flipped phasing match *exactly*. As a result, many WhatsHap false positive flip calls can be identified. An example is shown in Figure 6.4. **(c)** For UNKNOWN clusters, we manually examined a random subset of 16 clusters from each VCF to determine the ground truth and plot confusion matrices for WhatsHap and vcfdist.

6.7.4 HLA-DQB1 Complex Variant Example

Q100-dipcall haplotype 0

Q100-dipcall hap 0, hifiasm-dipcall hap 0

```
CTGCTGTCCCCCTGCTCTGCCCTAGGTCCCTGCCCCCTCCGATGCACCAGCCCCCAGCAC
CCCCACCGCCTCCTCCTGTCAGCCGGGGTGGAACGAACAAGGCTCAGGTTCCAGAGGCCG
CGCCCCCTTCGCCCCCTCCTGGCGCAGAGACTCTGGGCCCCCTGCCAAGGATGGGCCTCGCA
GACGGGGCAGACGCTCACCTCGCCGCTGCAAGGTCGTGCGGAGCTCCAAGTCCGCGCGC
CTCCTTTCCCTGGGGTGGAATGAACTGGGCTCAGATTTAGAGACCTCGCCCCATCGC
CCCTCCCGGCACAGAACTCGGGGTCTCGGCCAAGGGTGGGCCTCACGGAGGGGCGACGA
CGCTCACCTCCTCTGCAAGATCCCGCGGAACGCCACCTCGTAGTTGTGTCTGCACACC
GTGTCCAACCTCGCCCCGGGTCCCTTCCAGGACTTCCTTCTGGCTGTTCC
```

Q100-PAV hap 0, hifiasm-GIAB-TR hap 1

```
CTGCTGTGTCCCCCTGCTCTGCCCTAGGTCCCTGCCCCCTCCGATGCACCAGCCCCCAGCAC
CCCCACCGCCTCCTCCTGTCAGCCGGGGTGGAACGAACAAGGCTCAGGTTCCAGAGGCCG
CGCCCCCTTCGCCCCCTCCTGGCGCAGAGACTCTGGGCCCCCTGCCAAGGATGGGCCTCGCA
GACGGGGCAGACGCTCACCTCGCCGCTGCAAGGTCGTGCGGAGCTCCAAGTGGTAGTTG
TGTCTGCACACCGTGTCCAACCTCGCCCCGGGTCCCTTCCAGGACTTCCTTCTGGCTGTTCC
C
```

Q100-dipcall haplotype 1

Q100-dipcall hap 1, hifiasm-dipcall hap 1, Q100-PAV hap 0

```
CTGCTTGTCTCCCTGCTCTGCCCTAGGTCCCCGCCCATCTGATGCACCTGCCCCACCAC
TCACGCCGCCAACTCCTGTCCCCTGGGGTGGAATAAACGGGGCTCAGGTTTCAGAGGCCG
CAACCCCATCGCCCCCTCCAGCACAGAGACTAGAGGTCCCGGCCAACGGTGGGCCTCACG
GAGGGGCGACGACGCTCACCTCTCCTCTGCAGGATCCCGCGGTACGCCACCTCGTAGTTG
TGTCTGCACACCCTGTCCACCGACGCCCGGGCCCCCTCCAGGACTTCCTTCTGGCTGTTCC
C
```

hifiasm-GIAB-TR haplotype 0

```
CTGCTTGTCTCCCTGCTCTGCCCTAGGTCCCCGCCCATCTGATGCACCTGCCCCACCACT
CACGCCGCCAACTCCTGTCCCCTGGGGTGGAATAAACGGGGCTCAGGTTTCAGAGGCCGC
AAACCCCATCGCCCCCTCCAGCACAGAGACTAGAGGTCCCGGCCAACGGTGGGCCTCACGG
AGGGGCGACGACGCTCACCTCTCCTCTGCAGGATCCCGCGGTACGCCACCTCGTAGTTGT
GTCTGCACACCCTGTCCACCGACGCCCGGGCCCCCTCCAGGACTTCCTTCTGGCTGTTCC
```

Table 6.6: Sequences for a subregion of the HLA-DQB1 gene. We show full haplotype sequences for the truth (Q100-dipcall) and query (hifiasm-dipcall, Q100-PAV, hifiasm-GIAB-TR) VCFs for a portion of the HLA-DQB1 gene (chr6:32,664,600–32,664,899), with all variants applied to the reference sequence. Substitutions (SNPs) are highlighted in blue, insertions in green, and deletions in red. Please note that the first and second pairs of sequences are identical, despite differing variant representations.

6.7.5 Validation and Detailed Comparisons

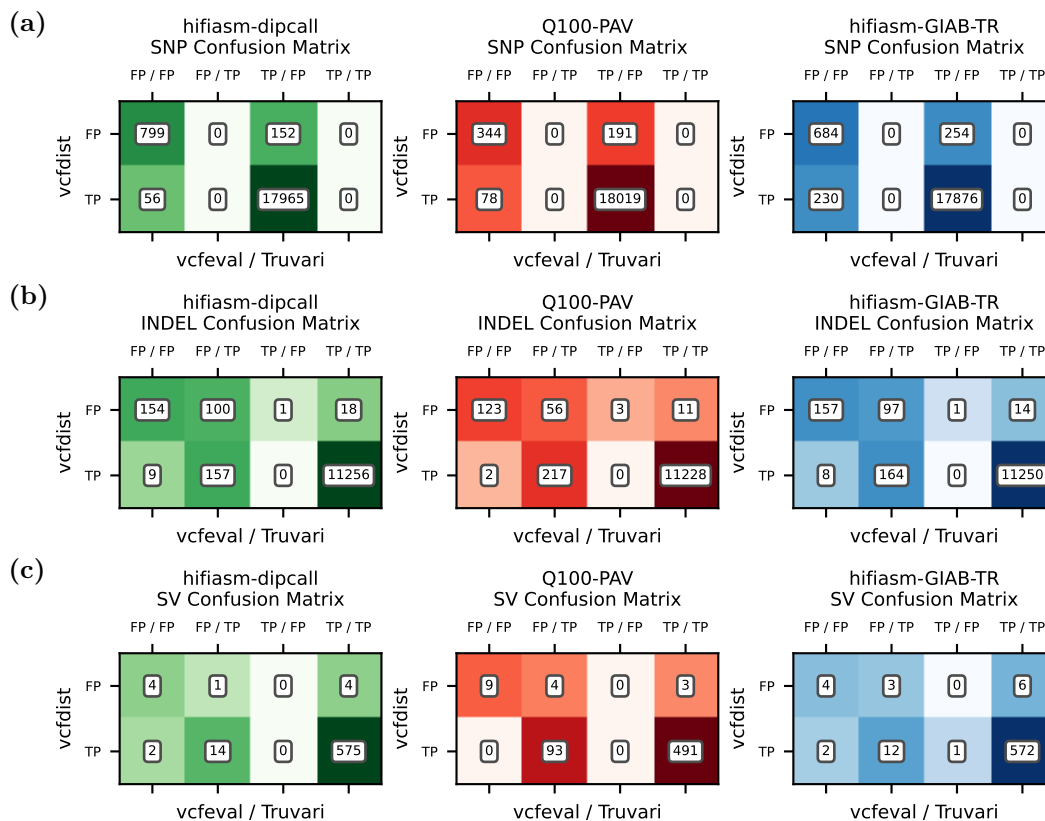


Figure 6.9: Confusion matrices for variant comparison method differences. We show a comparison of (a) SNP, (b) INDEL, and (c) SV variant calling evaluation by vcfdist, vcfeval, and Truvari, restricting to chr20 of the GIAB-TR tandem repeats BED. Prior to evaluation, truth and query VCFs were normalized using Truvari phab. This means that the Truvari results reported in this figure are largely equivalent to Truvari refine (MAFFT).

	POS	REF	ALT	FORMAT	VCFEVAL TRUTH	VCFEVAL QUERY	TRUVARI TRUTH	TRUVARI QUERY	VCFDIST TRUTH	VCFDIST QUERY
Allele Match	40153469	GTATA	G	GT:BD:BK	0 1:FN:am	1 1:FP:am	0 1:TP:am	1 1:TP:am	0 1:TP:gm	0 1:TP:gm
	40153469	GTATA	G	GT:BD:BK						1 0:FP:.
Different Thresholds	39864946	G	GTTT	GT:BD:BC	0 1:FN:0.0		0 1:TP:.75		0 1:FN:.67	
	39864946	G	GTTT	GT:BD:BC		0 1:FP:0.0		0 1:TP:.75		0 1:FP:0.0
Complex Variant	11753707	C	CGTGTGTGTGT	GT:BD:BC	1 0:FN:0.0		1 0:FN:0.0		1 0:FN:.38	
	11753744	G	GTGTGTA	GT:BD:BC	1 0:TP:1.0	1 0:TP:1.0	1 0:TP:1.0	1 0:TP:1.0	1 0:FN:.38	1 0:FP:0.0
Pick Single	18945947	C	CAGAGAGAGAGAG	GT:BD:BC		1 0:FP:0.0		1 0:FP:0.0		1 0:TP:.80
	18945947	C	CAGAGAGAGAG	GT:BD:BC	1 1:FN:0.0	0 1:FP:0.0	1 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0
	18945947	C	CAGAGAGAGAG	GT:BD:BC					1 0:TP:.80	
Flip Error	33052546	G	GTA	GT:BD:BC	1 0:TP:1.0	1 0:FP:0.0	1 0:TP:1.0	1 0:TP:1.0	1 0:TP:1.0	1 0:TP:1.0
	33052546	GTG	G	GT:BD:BC	0 1:TP:1.0		0 1:FN:0.0		0 1:FN:0.0	
	33052548	G	A	GT:BD:BC		0 1:TP:1.0				0 1:FP:0.0
Backtracking Tie	23651946	G	GC	GT:BD:BC	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:FP:0.0
	23651955	C	CC	GT:BD:BC		0 1:FP:0.0		0 1:FP:0.0		0 1:TP:1.0
Variant Overlap	35755334	C	CAT	GT:BD:BC	0 1:FN:0.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0	0 1:TP:1.0
	35755334	C	CATAT	GT:BD:BC		0 1:TP:1.0		0 1:FP:0.0		0 1:FP:0.0
	35755334	C	CATATAT	GT:BD:BC	0 1:TP:1.0		0 1:FN:.67		0 1:FN:.67	

Figure 6.10: Examples for each category of variant classification difference. We show a simple real example for each of the eight categories of variants in Table 6.4 that were evaluated differently between tools. Each variant call file (VCF) entry shows the variant position (POS), reference and alternate alleles (REF and ALT), and supplementary information such as genotype (GT), benchmarking decision (BD), benchmarking category (BK, which is either a genotype (gm) or allele (am) match), and benchmarking credit (BC).

6.7.6 Clustering

For alignment-based variant calling evaluation tools it is necessary to group variants together into many independent clusters that can be evaluated individually, since aligning entire chromosomes is currently computationally infeasible. In order to understand the impact of variant clustering on the measured accuracy and runtime of evaluation tools, we explore and compare several different options for variant clustering. Truvari clusters variants by BED region during evaluation. vcfeval does not cluster variants since it is not alignment-based. Requiring exact variant matches allows it to use a branch-and-bound approach instead. For vcfdist, we implemented two options: a simple yet flexible variant clustering heuristic, and a more sophisticated clustering algorithm that attempts to minimize computation while retaining perfect accuracy.

The simplest option that vcfdist offers is to group all variants less than n bases apart into a single cluster. We call this “gap n ” clustering, as shown in Table 6.7. The average region size, runtime, and measured variant calling performance all depend highly upon n . Using a gap of $n = 10$ results in the fastest runtime, but at the cost of significantly lower measured SNP, INDEL, and SV accuracy. A larger gap of 100 is a reasonable balance between performance and accuracy, and only SV accuracy is noticeably impacted. A much larger gap of 500 almost completely eliminates cases where true positive variants are mistakenly labelled false positives due to equivalent query and truth variants not being clustered together, but at the cost of a much higher runtime.

We develop our own clustering method based on bidirectional wavefront alignment (bi-WFA) [153] that dynamically clusters variants based on whether truth and query variants have the potential to be equivalent. It aligns leftwards and rightwards from each variant cluster, determining the span of reference bases that can be influenced by the current cluster of variants. This algorithm is able to find long-range variant dependencies that may span repetitive regions of the genome longer than 500bp, while also splitting up independent variants that are nearby on the reference. As shown in Table 6.7, this approach achieves even higher accuracy than gap 500 clustering, whilst running more than 10× faster end-to-end.

Clustering approach	None	GIAB-Q100 BED	Gap 10	Gap 100	Gap 500	BiWFA
Total regions	24	300	4,363,640	3,343,352	1,503,147	4,259,395
Total regions size	3,088,269,832	2,789,029,111	18,548,559	69,515,796	550,160,873	24,589,303
Mean region size	128,677,910	9,296,764	4.25	20.8	366	5.77
Max region size	248,956,422	127,461,328	2,522	19,730	65,503	29,663
SNP F1 score	*	*	0.9644	0.9719	0.9722	0.9723
INDEL F1 score	*	*	0.9415	0.9626	0.9631	0.9631
SV F1 score	*	*	0.8244	0.9427	0.9514	0.9516
Runtime	*	*	00:02:57	00:11:53	11:25:06	01:04:56

Table 6.7: Summary of vcfdist clustering results. The efficiency and measured accuracy of evaluating the Q100-PAV VCF with vcfdist on the Q100-dipcall ground truth VCF using different clustering approaches. Accuracy entries marked with * were unable to be computed due to long runtimes caused by large clustered regions. Clustering variants together whenever they are less than 10 bases apart (“Gap 10”) is the most efficient method, although its accuracy is significantly reduced. Bidirectional wavefront alignment (biWFA) based clustering achieves the highest accuracy with reasonable performance.

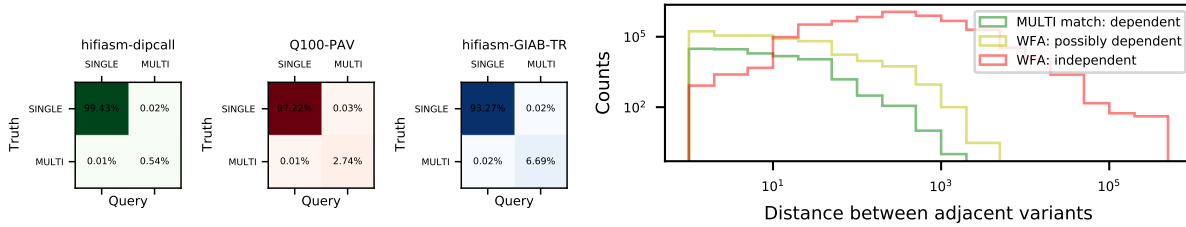


Figure 6.11: Measuring variant representation dependencies. **(a)** As variant representation becomes increasingly different from the Q100-dipcall ground truth (shown in Figure 6.6), the fraction of variants requiring complex comparisons involving multiple truth and query variants increases. **(b)** vcfdist’s biWFA clustering algorithm groups possibly dependent variants (yellow) together, and starts a new cluster when two adjacent variants are independent (red). Of the variants clustered together, only a small fraction are actually dependent (green). However, grouping these variants together is necessary to correctly determine equivalence between sets of truth and query variants.

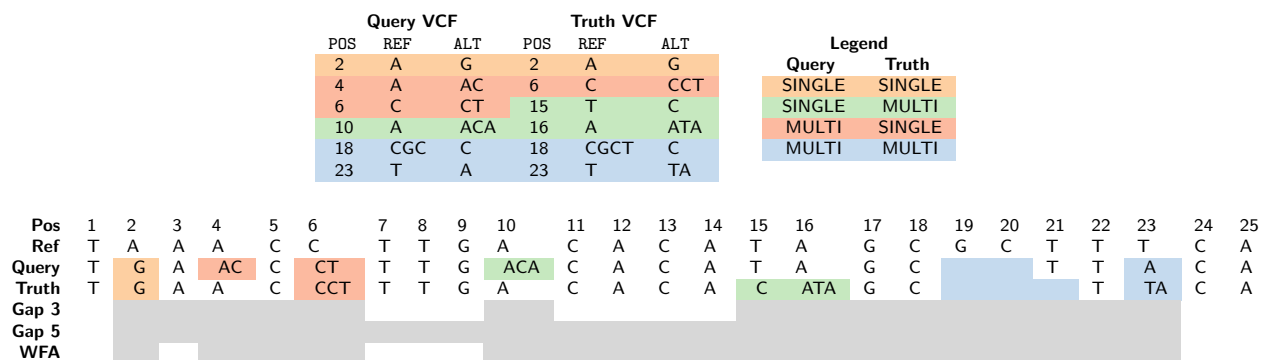


Figure 6.12: An example demonstrating different variant clustering methods. The span of each cluster is shown in gray. Equivalent sets of truth and query variants are depicted in different colors, and one example from each category is included. The four categories describe whether a single or multiple truth or query variants participate in the match. For vcfdist to determine variant equivalence, all dependent variants must be located in the same cluster. Incorrectly separating variants into different clusters results in lower measured accuracy. Unnecessarily grouping variants together results in larger clusters and more expensive evaluation. This tradeoff is shown in Table 5.13.

6.8 Summary

This work, titled “Jointly benchmarking small and structural variant calls with vcfdist”, was submitted to BMC Genome Biology in March 2024. Thank you to all my co-authors who contributed to this work: Justin Zook, Matthew Holt, and Satish Narayanasamy.

T. D. developed vcfdist and performed the computational studies. J. M. Z. suggested and guided the project. T. D. and J. M. H. worked on the phasing analysis. S. N. supervised the work. All authors contributed to the final paper and figures.

CHAPTER 7

Conclusion

Over the past two decades, the cost of sequencing a human genome has dropped by orders of magnitude at the same time that sequencing accuracy has improved, sequencing reads have lengthened, and exciting new sequencing technologies have been developed. In this thesis, I explored and worked on several applications containing challenges that emerged from the introduction of new nanopore-based sequencing technologies.

First, I explored the problem of portable pathogen detection using a handheld MinION nanopore sequencing device. We designed SquiggleFilter, a novel hardware accelerated dynamic time warping (DTW) based filter that directly analyzes a MinION’s raw squiggles and filters everything except target viral reads, thereby avoiding the expensive basecalling step. This was accomplished using the MinION’s built-in “Read Until” feature, which enables ejecting non-target reads in real time as a sample is being sequenced. We show that our 14.3W 13.25mm² accelerator has 274× greater throughput and 3481× lower latency than existing GPU-based solutions while consuming half the power, enabling Read Until for the next generation of nanopore sequencers.

This project really demonstrated to me the unreasonable effectiveness of designing specialized hardware accelerators in improving computational efficiency. As genomics applications become more mainstream, accelerators targeted to specific applications such as basecalling, variant calling, and alignment will likely become more common. This work also explored the tradeoff between accuracy (in terms of Read Until classification) and efficiency (in terms of both latency and throughput). When the accuracy of a solution is more than sufficient to fulfill the needs of an application,

a faster solution with lower accuracy may be preferable. The last insight I gained from this project was that direct computation on raw and messy data that is uninterpretable to humans can yield accurate results without being preprocessed to a more understandable format. In this case, sDTW alignment on raw signal data was performed without basecalling to nucleotide sequences. Since SquiggleFilter was published, several CNN based solutions such as SquiggleNet [187], DeepSelectNet [188], nanoDeep [189], and baseLess [190] have taken this a step further by classifying raw signals using pre-trained neural networks that are effectively black boxes that yield the correct classification. Although transformers have not yet been applied to this problem (likely due to their traditionally larger architectures that result in less efficient decisions), I expect them to outperform CNNs in terms of accuracy on more difficult signal classification tasks (such as when many species are involved, or species have larger genomes).

The next project I worked on was a novel alignment algorithm for improving the consistency of long read alignment in repetitive areas of the human genome; in particular, homopolymers and simple tandem repeats. Although nanopore-based variant calling precision and recall for single nucleotide polymorphisms (SNPs) exceeded 99.5% for R9.4.1 flow cells at the time, INDEL recall lagged far behind [100, 98]. Using nPoRe, I was able to demonstrate that phasing reads by haplotype improves INDEL recall from 63.76% to 70.66% and nPoRe realignment improved it further to 73.04%.

By completing this project, I confirmed my hypothesis that the format in which data is fed to the CNN variant caller is critical for obtaining accurate variant calls. During this study, I trained Clair3 from scratch on the same exact data but with three different pre-processing pipelines, and found that sorting reads by haplotype resulted in significantly better INDEL recall in repetitive regions ($p = 1.25 * 10^{-141}$), as did realigning reads with nPoRe ($p = 1.51 * 10^{-20}$). In future machine learning projects, it is important to consider that resulting accuracy depends not only on the network architecture and quality and volume of training data, but how the training and test data is presented to the network.

I then developed an alignment-based variant comparison tool, vcfdist, to accurately benchmark

small variant calls. Accurately benchmarking small variant calling accuracy is critical for the continued improvement of human whole genome sequencing. I showed that existing variant calling evaluators are biased towards certain variant representations and may misrepresent the relative performance of different variant calling pipelines. This work explored the affine gap parameter design space for complex variant representation and suggested a standard. I evaluated the performance of 64 phased WGS VCFs and showed that `vcfdist` improves measured INDEL performance consistency across variant representations from $R^2 = 0.97243$ to 0.99996. Afterwards, I extended `vcfdist` to jointly evaluate small and structural variants, and made significant contributions to the current state of variant phasing benchmarking. We found that a joint evaluation reduces measured false negative and false positive variant calls across the board: by 28.1% for SNPs, 19.1% for INDELs, and 52.4% for SVs over 50 bases. Lastly, `vcfdist` identified that 43% to 92% of all flip errors called by standard phasing evaluation tool `WhatsHap` are false positives due to differences in variant representations.

From this work, I learned that rigorous evaluation frameworks are essential when developing computation tools, particularly for variant calling. When evaluating the performance of a tool in terms of key performance metrics (such as variant calling precision and recall), it is essential to understand all of the possible limitations of that evaluation, including errors in the ground truth, how the evaluation tools consider partially correct calls, and other nuances in how variants are evaluated. When comparing `vcfdist` to `vcfeval` and `Truvari`, I had to take into account numerous implementation details that seemed minor but majorly affected the measured results. Some examples include the decomposition of complex variants, the inclusion or exclusion of variants overlapping the border of the selected BED regions, how to count variant calls with the correct allele but incorrect genotype, and the threshold for considering a mostly correct variant call a true positive. The most important insight from this work, however, was the extent to which complex variant representation impacts tertiary analyses. In this work, I design a variant comparison tool that is able to reconcile these difference in the context of variant benchmarking, but there are numerous areas of bioinformatics that would likely benefit from more sophisticated variant comparisons, including variant

effect prediction (VEP), genome wide association studies (GWAS), publicly-available databases of known mutations, and clinical interpretations of such variants.

I plan to continue working on improving vcfdist as a benchmarking tool after graduation. In its current state, it is limited to evaluating phased variant calls that are less than 10,000 bases in size. Transitioning the core alignment algorithm from a sequence-to-diploid alignment to a more general sequence-to-graph alignment would enable benchmarking unphased variant calls, allow measuring genotyping errors, and enable more accurate flip error metrics. Shifting to a wavefront alignment implementation of sequence-to-graph alignment would then significantly reduce memory usage, allow faster comparisons, and enable evaluation of much larger variants. I look forward to continuing this work as I start my career in industry.

BIBLIOGRAPHY

- [1] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, “Squigglefilter: An accelerator for portable virus detection,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 535–549, 2021.
- [2] T. Dunn, D. Blaauw, R. Das, and S. Narayanasamy, “npore: n-polymer realigner for improved pileup-based variant calling,” *BMC bioinformatics*, vol. 24, no. 1, pp. 1–21, 2023.
- [3] T. Dunn and S. Narayanasamy, “vcfdist: Accurately benchmarking phased small variant calls in human genomes,” *Nature Communications*, vol. 14, no. 1, p. 8149, 2023.
- [4] J. D. Watson and F. H. Crick, “Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid,” *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.
- [5] D. T. Zallen, “Despite franklin’s work, wilkins earned his nobel,” *Nature*, vol. 425, no. 6953, pp. 15–15, 2003.
- [6] F. Sanger and A. R. Coulson, “A rapid method for determining sequences in dna by primed synthesis with dna polymerase,” *Journal of molecular biology*, vol. 94, no. 3, pp. 441–448, 1975.
- [7] A. M. Maxam and W. Gilbert, “A new method for sequencing dna,” *Proceedings of the National Academy of Sciences*, vol. 74, no. 2, pp. 560–564, 1977.
- [8] J. M. Heather and B. Chain, “The sequence of sequencers: The history of sequencing dna,” *Genomics*, vol. 107, no. 1, pp. 1–8, 2016.
- [9] D. Kilburn, J. Burke, R. Fedak, H. Olsen, M. Jain, K. Miga, S. Mayes, and K. Liu, “High Data Throughput and Low Cost Ultra Long Nanopore Sequencing.”
- [10] D. Porubsky, P. Ebert, P. A. Audano, M. R. Vollger, W. T. Harvey, P. Marijon, J. Ebler, K. M. Munson, M. Sorensen, A. Sulovari, *et al.*, “Fully phased human genome assembly without parental data using single-cell strand sequencing and long reads,” *Nature biotechnology*, vol. 39, no. 3, pp. 302–308, 2021.
- [11] M. Mahmoud, N. Gobet, D. I. Cruz-Dávalos, N. Mounier, C. Dessimoz, and F. J. Sedlazeck, “Structural variant calling: the long and the short of it,” *Genome biology*, vol. 20, no. 1, pp. 1–14, 2019.

- [12] K. Goepfrich and K. Judge, “Decoding dna with a pocket-sized sequencer,” 2018.
- [13] A. Tan, G. R. Abecasis, and H. M. Kang, “Unified representation of genetic variants,” *Bioinformatics*, vol. 31, no. 13, pp. 2202–2204, 2015.
- [14] J. G. Cleary, R. Braithwaite, K. Gaastra, B. S. Hilbush, S. Inglis, S. A. Irvine, A. Jackson, R. Littin, M. Rathod, D. Ware, *et al.*, “Comparing variant call files for performance benchmarking of next-generation sequencing variant calling pipelines,” *BioRxiv*, p. 023754, 2015.
- [15] P. Danecek, J. K. Bonfield, J. Liddle, J. Marshall, V. Ohan, M. O. Pollard, A. Whitwham, T. Keane, S. A. McCarthy, R. M. Davies, and H. Li, “Twelve years of SAMtools and BCFtools,” *GigaScience*, vol. 10, 02 2021. giab008.
- [16] E. Garrison, “Vcfliib: A c++ library for parsing and manipulating vcf files,” *GitHub* <https://github.com/ekg/vcfliib>, 2012.
- [17] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, *et al.*, “The variant call format and vcftools,” *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, 2011.
- [18] B. Chapman, “bcbio.variation: Toolkit to analyze genomic variation data, built on the gatk with clojure,” *GitHub* <https://github.com/chapmanb/bcbio.variation>, 2015.
- [19] A. Talwalkar, J. Liptrap, J. Newcomb, C. Hartl, J. Terhorst, K. Curtis, M. Bresler, Y. S. Song, M. I. Jordan, and D. Patterson, “Smash: a benchmarking toolkit for human genome variant calling,” *Bioinformatics*, vol. 30, no. 19, pp. 2787–2795, 2014.
- [20] C. Sun and P. Medvedev, “Varmatch: robust matching of small variant datasets using flexible scoring schemes,” *Bioinformatics*, vol. 33, no. 9, pp. 1301–1308, 2017.
- [21] P. Krusche, L. Trigg, P. C. Boutros, C. E. Mason, M. Francisco, B. L. Moore, M. Gonzalez-Porta, M. A. Eberle, Z. Tezak, S. Lababidi, *et al.*, “Best practices for benchmarking germline small-variant calls in human genomes,” *Nature biotechnology*, vol. 37, no. 5, pp. 555–560, 2019.
- [22] I. Incorporated, “hap.py,” 2021. Haplotype comparison tools.
- [23] K. Shafin, T. Pesout, P.-C. Chang, M. Nattestad, A. Kolesnikov, S. Goel, G. Baid, M. Kolmogorov, J. M. Eizenga, K. H. Miga, *et al.*, “Haplotype-aware variant calling with pepper-margin-deepvariant enables high accuracy in nanopore long-reads,” *Nature methods*, vol. 18, no. 11, pp. 1322–1332, 2021.
- [24] M. Šošić and M. Šikić, “Edlib: a c/c++ library for fast, exact sequence alignment using edit distance,” *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 2017.
- [25] A. C. English, V. K. Menon, R. A. Gibbs, G. A. Metcalf, and F. J. Sedlazeck, “Truvari: refined structural variant comparison preserves allelic diversity,” *Genome Biology*, vol. 23, no. 1, p. 271, 2022.

- [26] N. Hansen, “Svanalyzer: Tools for the analysis of structural variation in genomes,” 2020.
- [27] K. Katoh and D. M. Standley, “Mafft multiple sequence alignment software version 7: improvements in performance and usability,” *Molecular biology and evolution*, vol. 30, no. 4, pp. 772–780, 2013.
- [28] D. Freed, “hap-eval: A vcf comparison engine for structural variant benchmarking,” 2022.
- [29] J. Yang and M. J. Chaisson, “Tt-mars: structural variants assessment based on haplotype-resolved assemblies,” *Genome Biology*, vol. 23, no. 1, p. 110, 2022.
- [30] N. V. Patel, “Why the CDC Botched Its Coronavirus Testing.” MIT Technology Review.
- [31] M. Park, J. Won, B. Y. Choi, and C. J. Lee, “Optimization of primer sets and detection protocols for sars-cov-2 of coronavirus disease 2019 (covid-19) using pcr and real-time pcr,” *Experimental & molecular medicine*, vol. 52, no. 6, pp. 963–977, 2020.
- [32] J. Quick and N. Loman, “ARTIC V3 Update Notes,” 2020.
- [33] A. J. McMichael, “Environmental and social influences on emerging infectious diseases: past, present and future,” *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 359, no. 1447, pp. 1049–1058, 2004.
- [34] L. Zhang, X. Cui, K. Schmitt, R. Hubert, W. Navidi, and N. Arnheim, “Whole genome amplification from a single cell: implications for genetic analysis,” *Proceedings of the National Academy of Sciences*, vol. 89, no. 13, pp. 5847–5851, 1992.
- [35] A. L. Greninger, S. N. Naccache, S. Federman, G. Yu, P. Mbala, V. Bres, D. Stryke, J. Bouquet, S. Somasekar, J. M. Linnen, *et al.*, “Rapid metagenomic identification of viral pathogens in clinical samples by real-time nanopore sequencing analysis,” *Genome medicine*, vol. 7, no. 1, p. 99, 2015.
- [36] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, “Real-time selective sequencing with rubric: Read until with basecall and reference-informed criteria,” *Scientific Reports*, vol. 9, no. 1, pp. 1–11, 2019.
- [37] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, “Genax: a genome sequencing accelerator,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 69–82, IEEE, 2018.
- [38] T. J. Ham, D. Bruns-Smith, B. Sweeney, Y. Lee, S. H. Seo, U. G. Song, Y. H. Oh, K. Asanovic, J. W. Lee, and L. W. Wills, “Genesis: a hardware acceleration framework for genomic data analysis,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 254–267, IEEE, 2020.
- [39] Y. Turakhia, G. Bejerano, and W. J. Dally, “Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly,” *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 199–213, 2018.

- [40] D. Fujiki, S. Wu, N. Ozog, K. Goliya, D. Blaauw, S. Narayanasamy, and R. Das, “Seedex: A genome sequencing accelerator for optimal alignments in subminimal space,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 937–950, IEEE, 2020.
- [41] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomini, H. Kambalashubramanyam, and P.-E. Gaillardon, “Gencache: Leveraging in-cache operators for efficient sequence alignment,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 334–346, 2019.
- [42] S. K. Khatamifard, Z. Chowdhury, N. Pande, M. Razaviyayn, C. Kim, and U. R. Karpuzcu, “A non-volatile near-memory read mapping accelerator,” *arXiv preprint arXiv:1709.02381*, 2017.
- [43] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand, *et al.*, “Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 951–966, IEEE, 2020.
- [44] L. Wu, D. Bruns-Smith, F. A. Nothaft, Q. Huang, S. Karandikar, J. Le, A. Lin, H. Mao, B. Sweeney, K. Asanović, *et al.*, “Fpga accelerated indel realignment in the cloud,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 277–290, IEEE, 2019.
- [45] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series,” in *KDD workshop*, vol. 10, pp. 359–370, Seattle, WA, USA:, 1994.
- [46] P. Senin, “Dynamic time warping algorithm review,” *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, vol. 855, no. 1-23, p. 40, 2008.
- [47] G. Mahmoudabadi and R. Phillips, “A comprehensive and quantitative exploration of thousands of viral genomes,” *Elife*, vol. 7, p. e31955, 2018.
- [48] M. Loose, S. Malla, and M. Stout, “Real-time selective sequencing using nanopore technology,” *Nature methods*, vol. 13, no. 9, p. 751, 2016.
- [49] NVIDIA, “Jetson agx xavier developer kit.”
- [50] S. Wei, Z. R. Weiss, and Z. Williams, “Rapid multiplex small dna sequencing on the minion nanopore sequencing platform,” *G3: Genes, Genomes, Genetics*, vol. 8, no. 5, pp. 1649–1657, 2018.
- [51] J. Hasell, E. Mathieu, D. Beltekian, B. Macdonald, C. Giattino, E. Ortiz-Ospina, M. Roser, and H. Ritchie, “A cross-country database of covid-19 testing,” *Scientific data*, vol. 7, no. 1, pp. 1–7, 2020.
- [52] Abbott, “Navica App and BinaxNOW COVID-19 Ag Test Card,” 2021. Abbott Point of Care Testing.

- [53] M. A. GOUILH, R. CASSIER, E. MAILLE, C. Schanen, L.-M. ROCQUE, and A. VABRET, “An easy, reliable and rapid sars-cov2 rt-lamp based test for point-of-care and diagnostic lab,” *medRxiv*, 2020.
- [54] NEB, “SARS-CoV-2 Rapid Colorimetric LAMP Assay Kit,” 2021. New England Biolabs.
- [55] LGC, “2019-nCoV CDC-qualified Probe and Primer Kits for SARS-CoV-2,” 2021. LGC Biosearch Technologies.
- [56] ONT, “cDNA PCR Sequencing Kit,” 2021. Oxford Nanopore Technologies.
- [57] P. James, D. Stoddart, E. D. Harrington, J. Beaulaurier, L. Ly, S. Reid, D. J. Turner, and S. Juul, “Lampore: rapid, accurate and highly scalable molecular screening for sars-cov-2 infection, based on nanopore sequencing,” *medRxiv*, 2020.
- [58] ONT, “Direct RNA Sequencing Kit,” 2021. Oxford Nanopore Technologies.
- [59] ONT, “Direct cDNA Sequencing Kit,” 2021. Oxford Nanopore Technologies.
- [60] M. Nagura-Ikeda, K. Imai, S. Tabata, K. Miyoshi, N. Murahara, T. Mizuno, M. Horiuchi, K. Kato, Y. Imoto, M. Iwata, *et al.*, “Clinical evaluation of self-collected saliva by rt-qpcr, direct rt-qpcr, rt-lamp, and a rapid antigen test to diagnose covid-19,” *Journal of Clinical Microbiology*, 2020.
- [61] J. R. Tyson, P. James, D. Stoddart, N. Sparks, A. Wickenhagen, G. Hall, J. H. Choi, H. Lapointe, K. Kamelian, A. D. Smith, *et al.*, “Improvements to the artic multiplex pcr method for sars-cov-2 genome sequencing using nanopore,” *bioRxiv*, 2020.
- [62] G. Moreno and D. O’Connor, “Sequence-Independent, Single-Primer Amplification of RNA viruses V.3,” 2020. University of Wisconsin-Madison.
- [63] ONT, “Metagenomic analysis of SARS-CoV-2 respiratory samples via Sequence-Independent Single Primer Amplification (SISPA) and nanopore sequencing,” 2020. Oxford Nanopore Technologies.
- [64] R. R. Wick, L. M. Judd, and K. E. Holt, “Performance of neural network basecalling tools for oxford nanopore sequencing,” *Genome biology*, vol. 20, no. 1, p. 129, 2019.
- [65] H. Li, “Minimap2: pairwise alignment for nucleotide sequences,” *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
- [66] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, “Fast and accurate de novo genome assembly from long uncorrected reads,” *Genome research*, vol. 27, no. 5, pp. 737–746, 2017.
- [67] C. Wright, “Medaka,” 2020. Medaka - Medaka 1.2.0 documentation.
- [68] C. Brown, “Technology update.” Nanopore Community Meeting, 2019.
- [69] T. Sauvage, W. E. Schmidt, H. S. Yoon, V. J. Paul, and S. Fredericq, “Promising prospects of nanopore sequencing for algal hologenomics and structural variation discovery,” *BMC genomics*, vol. 20, no. 1, pp. 1–17, 2019.

- [70] R. Han, Y. Li, X. Gao, and S. Wang, “An accurate and rapid continuous wavelet dynamic time warping algorithm for end-to-end mapping in ultra-long nanopore sequencing,” *Bioinformatics*, vol. 34, no. 17, pp. i722–i731, 2018.
- [71] R. Ronan, “Read until adaptive sampling.” Oxford Nanopore Technologies.
- [72] A. Payne, N. Holmes, T. Clarke, R. Munro, B. Debebe, and M. W. Loose, “Nanopore adaptive sequencing for mixed samples, whole exome capture and targeted panels,” *BioRxiv*, 2020.
- [73] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, “Real-time selective sequencing with rubric: read until with basecall and reference-informed criteria,” *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.
- [74] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, “Targeted nanopore sequencing by real-time mapping of raw electrical signal with uncalled,” *BioRxiv*, 2020.
- [75] M. Stoiber, J. Quick, R. Egan, J. Eun Lee, S. Celniker, R. K. Neely, N. Loman, L. A. Pennacchio, and J. Brown, “De novo identification of dna modifications enabled by genome-guided nanopore signal processing,” *bioRxiv*, 2017.
- [76] N. J. Loman, J. Quick, and J. T. Simpson, “A complete bacterial genome assembled de novo using only nanopore sequencing data,” *Nature methods*, vol. 12, no. 8, pp. 733–735, 2015.
- [77] O. N. Technologies, “kmer_models,” *GitHub repository*, 2017.
- [78] E. Keogh and S. Kasetty, “On the need for time series data mining benchmarks: a survey and empirical demonstration,” in *Data Mining and knowledge discovery*, vol. 7, pp. 349–371, Springer, 2003.
- [79] P. Zhou, X.-L. Yang, X.-G. Wang, B. Hu, L. Zhang, W. Zhang, H.-R. Si, Y. Zhu, B. Li, C.-L. Huang, *et al.*, “A pneumonia outbreak associated with a new coronavirus of probable bat origin,” *nature*, vol. 579, no. 7798, pp. 270–273, 2020.
- [80] J. Hadfield, C. Megill, S. M. Bell, J. Huddleston, B. Potter, C. Callender, P. Sagulenko, T. Bedford, and R. A. Neher, “Nextstrain: real-time tracking of pathogen evolution,” *Bioinformatics*, vol. 34, no. 23, pp. 4121–4123, 2018.
- [81] Y. Shu and J. McCauley, “Gisaid: Global initiative on sharing all influenza data—from vision to reality,” *Eurosurveillance*, vol. 22, no. 13, p. 30494, 2017.
- [82] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, “Accelerating dynamic time warping subsequence search with gpus and fpgas,” in *2010 IEEE International Conference on Data Mining*, pp. 1001–1006, IEEE, 2010.
- [83] R. E. Workman, A. D. Tang, P. S. Tang, M. Jain, J. R. Tyson, P. C. Zuzarte, T. Gilpatrick, R. Razaghi, J. Quick, N. Sadowski, *et al.*, “Nanopore native rna sequencing of a human poly (a) transcriptome,” *BioRxiv*, p. 459529, 2018.

- [84] O. N. Technologies, “Ont open datasets: Gm24385 dataset release,” 2020.
- [85] CADDE, “Brazil-uk centre for arbovirus discovery, diagnosis, genomics and epidemiology,” 2020.
- [86] ONT, “Rapid Library Preparation Kit (SQK-RAD004),” 2021. Oxford Nanopore Technologies.
- [87] ONT, “ont-fast5-api,” 2020. FAST5 API: a simple interface to HDF5 files of the Oxford Nanopore .fast5 file format.
- [88] ONT, “ont-pyguppy-client-lib,” 2020. PyGuppy: Python bindings for the GuppyClient library.
- [89] ONT, “MinION DNA Sequencer,” 2021. Oxford Nanopore Technologies.
- [90] ONT, “Read until api,” 2021. Oxford Nanopore Technologies.
- [91] C. Seymour, “Bonito: a pytorch basecaller for oxford nanopore reads,” 2019.
- [92] V. Sundaresan, S. Nichani, N. Ranganathan, and R. Sankar, “A vlsi hardware accelerator for dynamic time warping,” in *11th IAPR International Conference on Pattern Recognition. Vol. IV. Conference D: Architectures for Vision and Pattern Recognition*, vol. 1, pp. 27–30, IEEE Computer Society, 1992.
- [93] D. Lang, S. Zhang, P. Ren, F. Liang, Z. Sun, G. Meng, Y. Tan, X. Li, Q. Lai, L. Han, D. Wang, F. Hu, W. Wang, and S. Liu, “Comparison of the two up-to-date sequencing technologies for genome assembly: HiFi reads of Pacific Biosciences Sequel II system and ultralong reads of Oxford Nanopore,” *GigaScience*, vol. 9, 12 2020. gaa123.
- [94] C. Brown, “Technology update.” Nanopore Community Meeting, 2021.
- [95] S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altomonte, L. Uralsky, A. Gershman, S. Aganezov, S. J. Hoyt, M. Diekhans, G. A. Logsdon, M. Alonge, S. E. Antonarakis, M. Borchers, G. G. Bouffard, S. Y. Brooks, G. V. Caldas, H. Cheng, C.-S. Chin, W. Chow, L. G. de Lima, P. C. Dishuck, R. Durbin, T. Dvorkina, I. T. Fiddes, G. Formenti, R. S. Fulton, A. Functammasan, E. Garrison, P. G. Grady, T. A. Graves-Lindsay, I. M. Hall, N. F. Hansen, G. A. Hartley, M. Haukness, K. Howe, M. W. Hunkapiller, C. Jain, M. Jain, E. D. Jarvis, P. Kerpedjiev, M. Kirsche, M. Kolmogorov, J. Korlach, M. Kremitzki, H. Li, V. V. Maduro, T. Marschall, A. M. McCartney, J. McDaniel, D. E. Miller, J. C. Mullikin, E. W. Myers, N. D. Olson, B. Paten, P. Peluso, P. A. Pevzner, D. Porubsky, T. Potapova, E. I. Rogaev, J. A. Rosenfeld, S. L. Salzberg, V. A. Schneider, F. J. Sedlazeck, K. Shafin, C. J. Shew, A. Shumate, Y. Sims, A. F. A. Smit, D. C. Soto, I. Sović, J. M. Storer, A. Streets, B. A. Sullivan, F. Thibaud-Nissen, J. Torrance, J. Wagner, B. P. Walenz, A. Wenger, J. M. D. Wood, C. Xiao, S. M. Yan, A. C. Young, S. Zarate, U. Surti, R. C. McCoy, M. Y. Dennis, I. A. Alexandrov, J. L. Gerton, R. J. O’Neill, W. Timp, J. M. Zook, M. C. Schatz, E. E. Eichler, K. H. Miga, and A. M. Phillippy, “The complete sequence of a human genome,” *bioRxiv*, 2021.

- [96] Z. Zheng, S. Li, J. Su, A. W.-S. Leung, T.-W. Lam, and R. Luo, “Symphonizing pileup and full-alignment for deep learning-based long-read variant calling,” *bioRxiv*, 2021.
- [97] K. Shafin, T. Pesout, P.-C. Chang, M. Nattestad, A. Kolesnikov, S. Goel, G. Baid, M. Kolmogorov, J. M. Eizenga, K. H. Miga, P. Carnevali, M. Jain, A. Carroll, and B. Paten, “Haplotype-aware variant calling with pepper-margin-deepvariant enables high accuracy in nanopore long-reads,” *Nature Methods*, vol. 18, pp. 1322–1332, 2021.
- [98] R. Luo, C.-L. Wong, Y.-S. Wong, C.-I. Tang, C.-M. Liu, C.-M. Leung, and T.-W. Lam, “Exploring the limit of using a deep neural network on pileup data for germline variant calling,” *Nature Machine Intelligence*, vol. 2, no. 4, pp. 220–227, 2020.
- [99] R. Luo, F. J. Sedlazeck, T.-W. Lam, and M. C. Schatz, “Clairvoyante: a multi-task convolutional deep neural network for variant calling in single molecule sequencing,” *bioRxiv*, p. 310458, 2018.
- [100] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, *et al.*, “A universal snp and small-indel variant caller using deep neural networks,” *Nature biotechnology*, vol. 36, no. 10, pp. 983–987, 2018.
- [101] J. T. Robinson, H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov, “Integrative genomics viewer,” *Nature biotechnology*, vol. 29, no. 1, pp. 24–26, 2011.
- [102] O. Gotoh, “An improved algorithm for matching biological sequences,” *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982.
- [103] K. Shafin, “Ont r9.4.1 guppy 5.0.7 sup hg003 whole genome performance evaluation against clair3,” 2021.
- [104] J. D. Thompson, “Introducing variable gap penalties to sequence alignment in linear space,” *Bioinformatics*, vol. 11, no. 2, pp. 181–186, 1995.
- [105] R. F. Smith and T. F. Smmith, “Pattern-induced multi-sequence alignment (pima) algorithm employing secondary structure-dependent gap penalties for use in comparative protein modelling,” *Protein Engineering, Design and Selection*, vol. 5, no. 1, pp. 35–41, 1992.
- [106] J. Shi, T. L. Blundell, and K. Mizuguchi, “Fugue: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties,” *Journal of molecular biology*, vol. 310, no. 1, pp. 243–257, 2001.
- [107] F. Fischel-Ghodsian, G. Mathiowitz, and T. F. Smith, “Alignment of protein sequences using secondary structure: a modified dynamic programming method,” *Protein Engineering, Design and Selection*, vol. 3, no. 7, pp. 577–581, 1990.
- [108] J. Qiu and R. Elber, “Ssaln: An alignment algorithm using structure-dependent substitution matrices and gap penalties learned from structurally aligned protein pairs,” *Proteins: Structure, Function, and Bioinformatics*, vol. 62, no. 4, pp. 881–891, 2006.

- [109] M. Madhusudhan, M. A. Marti-Renom, R. Sanchez, and A. Sali, “Variable gap penalty for protein sequence–structure alignment,” *Protein Engineering Design and Selection*, vol. 19, no. 3, pp. 129–133, 2006.
- [110] M. Jain, I. T. Fiddes, K. H. Miga, H. E. Olsen, B. Paten, and M. Akeson, “Improved data analysis for the minion nanopore sequencer,” *Nature methods*, vol. 12, no. 4, pp. 351–356, 2015.
- [111] N. C. Goonesekere and B. Lee, “Frequency of gaps observed in a structurally aligned protein pair database suggests a simple gap penalty function,” *Nucleic acids research*, vol. 32, no. 9, pp. 2838–2843, 2004.
- [112] F. J. Sedlazeck, P. Rescheneder, M. Smolka, H. Fang, M. Nattestad, A. Von Haeseler, and M. C. Schatz, “Accurate detection of complex structural variations using single-molecule sequencing,” *Nature methods*, vol. 15, no. 6, pp. 461–468, 2018.
- [113] J. R. Miller, A. L. Delcher, S. Koren, E. Venter, B. P. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, and G. Sutton, “Aggressive assembly of pyrosequencing reads with mates,” *Bioinformatics*, vol. 24, no. 24, pp. 2818–2824, 2008.
- [114] S. Nurk, B. P. Walenz, A. Rhie, M. R. Vollger, G. A. Logsdon, R. Grothe, K. H. Miga, E. E. Eichler, A. M. Phillippy, and S. Koren, “Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads,” *Genome research*, vol. 30, no. 9, pp. 1291–1305, 2020.
- [115] M. Rautiainen, S. Nurk, B. P. Walenz, G. A. Logsdon, D. Porubsky, A. Rhie, E. E. Eichler, A. M. Phillippy, and S. Koren, “Verkko: telomere-to-telomere assembly of diploid chromosomes,” *bioRxiv*, 2022.
- [116] G. Highnam, C. Franck, A. Martin, C. Stephens, A. Puthige, and D. Mittelman, “Accurate human microsatellite genotypes from high-throughput resequencing data using informed error profiles,” *Nucleic acids research*, vol. 41, no. 1, pp. e32–e32, 2013.
- [117] M. D. Cao, E. Tasker, K. Willadsen, M. Imelfort, S. Vishwanathan, S. Sureshkumar, S. Balasubramanian, and M. Boden, “Inferring short tandem repeat variation from paired-end short reads,” *Nucleic acids research*, vol. 42, no. 3, pp. e16–e16, 2014.
- [118] A. Ummat and A. Bashir, “Resolving complex tandem repeats with long reads,” *Bioinformatics*, vol. 30, no. 24, pp. 3491–3498, 2014.
- [119] B. Institute, “Indel-based realignment: Improving the original alignments of the reads based on multiple sequence (re-)alignment.” GATK talks, 2016.
- [120] K. Kojima, Y. Kawai, K. Misawa, T. Mimori, and M. Nagasaki, “Str-realigner: a realignment method for short tandem repeat regions,” *BMC genomics*, vol. 17, no. 1, pp. 1–15, 2016.
- [121] C. P. Ryan, “Tandem repeat disorders,” *Evolution, Medicine, and Public Health*, vol. 2019, pp. 17–17, 01 2019.

- [122] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.
- [123] S. F. Altschul and B. W. Erickson, “Optimal sequence alignment using affine gap costs,” *Bulletin of mathematical biology*, vol. 48, no. 5-6, pp. 603–616, 1986.
- [124] H. Fan and J.-Y. Chu, “A brief review of short tandem repeat mutation,” *Genomics, proteomics & bioinformatics*, vol. 5, no. 1, pp. 7–14, 2007.
- [125] H. Suzuki and M. Kasahara, “Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming,” *BioRxiv*, p. 130633, 2017.
- [126] O. N. Technologies, “Rebasecalling of sre and ulk gm24385 dataset,” 2021.
- [127] M. Martin, M. Patterson, S. Garg, S. Fischer, N. Pisanti, G. W. Klau, A. Schöenhuth, and T. Marschall, “Whatshap: fast and accurate read-based phasing,” *BioRxiv*, p. 085050, 2016.
- [128] NHGRI, “The cost of sequencing a human genome.,” 2021. <https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost>.
- [129] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, *et al.*, “The sequence of the human genome,” *science*, vol. 291, no. 5507, pp. 1304–1351, 2001.
- [130] NHGRI, “Genetics vs. genomics fact sheet.,” 2018. <https://www.genome.gov/about-genomics/fact-sheets/Genetics-vs-Genomics>.
- [131] S. T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin, “dbSNP: the NCBI database of genetic variation,” *Nucleic acids research*, vol. 29, no. 1, pp. 308–311, 2001.
- [132] M. Mahmoud, Y. Huang, K. Garimella, P. A. Audano, W. Wan, N. Prasad, R. E. Handsaker, S. Hall, A. Pionzio, M. C. Schatz, *et al.*, “Utility of long-read sequencing for all of us,” *bioRxiv*, pp. 2023–01, 2023.
- [133] D. C. Koboldt, “Best practices for variant calling in clinical sequencing,” *Genome Medicine*, vol. 12, no. 1, pp. 1–13, 2020.
- [134] K. Tomczak, P. Czerwińska, and M. Wiznerowicz, “Review the cancer genome atlas (TCGA): an immeasurable source of knowledge,” *Contemporary Oncology/Współczesna Onkologia*, vol. 2015, no. 1, pp. 68–77, 2015.
- [135] J. Su, Z. Zheng, S. S. Ahmed, T.-W. Lam, and R. Luo, “Clair3-trio: high-performance nanopore long-read variant calling in family trios with trio-to-trio deep neural networks,” *Briefings in Bioinformatics*, vol. 23, no. 5, p. bbac301, 2022.

- [136] N. D. Olson, J. Wagner, J. McDaniel, S. H. Stephens, S. T. Westreich, A. G. Prasanna, E. Johanson, E. Boja, E. J. Maier, O. Serang, *et al.*, “Precisionfda truth challenge v2: Calling variants from short and long reads in difficult-to-map regions,” *Cell Genomics*, vol. 2, no. 5, p. 100129, 2022.
- [137] J. Wagner, N. D. Olson, L. Harris, Z. Khan, J. Farek, M. Mahmoud, A. Stankovic, V. Kovacevic, B. Yoo, N. Miller, *et al.*, “Benchmarking challenging small variants with linked and long reads,” *Cell Genomics*, vol. 2, no. 5, p. 100128, 2022.
- [138] A. Bayat, B. Gaëta, A. Ignjatovic, and S. Parameswaran, “Improved vcf normalization for accurate vcf comparison,” *Bioinformatics*, vol. 33, no. 7, pp. 964–970, 2017.
- [139] J. M. Eizenga and B. Paten, “Improving the time and space complexity of the wfa algorithm and generalizing its scoring,” *bioRxiv*, 2022.
- [140] M. Akulov and R. G. Koerkamp, “Transforming match bonus into cost,” 2022.
- [141] K. Shafin, T. Pesout, R. Lorig-Roach, M. Haukness, H. E. Olsen, C. Bosworth, J. Armstrong, K. Tigyi, N. Maurer, S. Koren, *et al.*, “Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes,” *Nature biotechnology*, vol. 38, no. 9, pp. 1044–1053, 2020.
- [142] N. Technologies, “Novoalign,” 2022. Powerful tool designed for mapping of short reads onto a reference genome from Illumina, Ion Torrent, and 454 NGS platforms.
- [143] H. Li, “Aligning sequence reads, clone sequences and assembly contigs with bwa-mem,” *arXiv preprint arXiv:1303.3997*, 2013.
- [144] A. Toepfer, “pbmm2,” 2022. A minimap2 SMRT wrapper for PacBio data.
- [145] J. Wagner, N. D. Olson, L. Harris, J. McDaniel, H. Cheng, A. Functammasan, Y.-C. Hwang, R. Gupta, A. M. Wenger, W. J. Rowell, *et al.*, “Curated variation benchmarks for challenging medically relevant autosomal genes,” *Nature biotechnology*, vol. 40, no. 5, pp. 672–680, 2022.
- [146] J. Wagner, N. D. Olson, L. Harris, Z. Khan, J. Farek, M. Mahmoud, A. Stankovic, V. Kovacevic, A. M. Wenger, W. J. Rowell, *et al.*, “Benchmarking challenging small variants with linked and long reads,” *BioRxiv*, 2020.
- [147] S. Maestri, M. G. Maturo, E. Cosentino, L. Marcolungo, B. Iadarola, E. Fortunati, M. Rossato, and M. Delledonne, “A long-read sequencing approach for direct haplotype phasing in clinical settings,” *International Journal of Molecular Sciences*, vol. 21, no. 23, p. 9177, 2020.
- [148] B. E. Slatko, A. F. Gardner, and F. M. Ausubel, “Overview of next-generation sequencing technologies,” *Current protocols in molecular biology*, vol. 122, no. 1, p. e59, 2018.
- [149] J. M. Zook, J. McDaniel, N. D. Olson, J. Wagner, H. Parikh, H. Heaton, S. A. Irvine, L. Trigg, R. Truty, C. Y. McLean, *et al.*, “An open resource for accurately benchmarking small variant and reference calls,” *Nature biotechnology*, vol. 37, no. 5, pp. 561–566, 2019.

- [150] E. D. Jarvis, G. Formenti, A. Rhie, A. Guarracino, C. Yang, J. Wood, A. Tracey, F. Thibaud-Nissen, M. R. Vollger, D. Porubsky, *et al.*, “Semi-automated assembly of high-quality diploid human reference genomes,” *Nature*, pp. 1–13, 2022.
- [151] P. Danecek, J. K. Bonfield, J. Liddle, J. Marshall, V. Ohan, M. O. Pollard, A. Whitwham, T. Keane, S. A. McCarthy, R. M. Davies, and H. Li, “Twelve years of SAMtools and BCFtools,” *GigaScience*, vol. 10, 02 2021. giab008.
- [152] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, “Fast gap-affine pairwise alignment using the wavefront algorithm,” *Bioinformatics*, vol. 37, no. 4, pp. 456–463, 2021.
- [153] S. Marco-Sola, J. M. Eizenga, A. Guarracino, B. Paten, E. Garrison, and M. Moreto, “Optimal gap-affine alignment in o (s) space,” *Bioinformatics*, vol. 39, no. 2, p. btad074, 2023.
- [154] P. Jacobs, A. Baikie, J. Strong, *et al.*, “The somatic chromosomes in mongolism,” *The Lancet*, vol. 273, no. 7075, p. 710, 1959.
- [155] L. Feuk, A. R. Carson, and S. W. Scherer, “Structural variation in the human genome,” *Nature Reviews Genetics*, vol. 7, no. 2, pp. 85–97, 2006.
- [156] U. D. J. G. I. H. T. . B. E. . P. P. . R. P. . W. S. . S. T. . D. N. . C. J.-F. . O. A. . L. S. . E. C. . U. E. . F. M. 4, R. G. S. C. S. Y. . F. A. . H. M. . Y. T. . T. A. . I. T. . K. C. . W. H. . T. Y. . T. T. 9, Genoscope, C. U.-. W. J. . H. R. . S. W. . A. F. . B. P. . B. T. . P. E. . R. C. . W. P. 10, I. o. M. B. R. A. . P. M. . N. G. . T. S. . R. A. . Department of Genome Analysis, G. S. C. S. D. R. . D.-S. L. . R. M. . W. K. . L. H. M. . D. J. 11, B. G. I. G. C. Y. H. . Y. J. . W. J. . H. G. . G. J. 15, *et al.*, “Initial sequencing and analysis of the human genome,” *nature*, vol. 409, no. 6822, pp. 860–921, 2001.
- [157] M. Przeworski, R. R. Hudson, and A. Di Rienzo, “Adjusting the focus on human variation,” *Trends in Genetics*, vol. 16, no. 7, pp. 296–302, 2000.
- [158] A. W. Pang, J. R. MacDonald, D. Pinto, J. Wei, M. A. Rafiq, D. F. Conrad, H. Park, M. E. Hurles, C. Lee, J. C. Venter, *et al.*, “Towards a comprehensive structural variation map of an individual human genome,” *Genome biology*, vol. 11, pp. 1–14, 2010.
- [159] K. Chen, J. W. Wallis, M. D. McLellan, D. E. Larson, J. M. Kalicki, C. S. Pohl, S. D. McGrath, M. C. Wendl, Q. Zhang, D. P. Locke, *et al.*, “Breakdancer: an algorithm for high-resolution mapping of genomic structural variation,” *Nature methods*, vol. 6, no. 9, pp. 677–681, 2009.
- [160] A. Abyzov, A. E. Urban, M. Snyder, and M. Gerstein, “Cnvnator: an approach to discover, genotype, and characterize typical and atypical cnvs from family and population genome sequencing,” *Genome research*, vol. 21, no. 6, pp. 974–984, 2011.
- [161] S. S. Ho, A. E. Urban, and R. E. Mills, “Structural variation in the sequencing era,” *Nature Reviews Genetics*, vol. 21, no. 3, pp. 171–189, 2020.

- [162] M. Mahmoud, N. Gobet, D. I. Cruz-Dávalos, N. Mounier, C. Dessimoz, and F. J. Sedlazeck, “Structural variant calling: the long and the short of it,” *Genome biology*, vol. 20, no. 1, pp. 1–14, 2019.
- [163] M. J. Chaisson, J. Huddleston, M. Y. Dennis, P. H. Sudmant, M. Malig, F. Hormozdiari, F. Antonacci, U. Surti, R. Sandstrom, M. Boitano, *et al.*, “Resolving the complexity of the human genome using single-molecule sequencing,” *Nature*, vol. 517, no. 7536, pp. 608–611, 2015.
- [164] P. A. Audano, A. Sulovari, T. A. Graves-Lindsay, S. Cantsilieris, M. Sorensen, A. E. Welch, M. L. Dougherty, B. J. Nelson, A. Shah, S. K. Dutcher, *et al.*, “Characterizing the major structural variant alleles of the human genome,” *Cell*, vol. 176, no. 3, pp. 663–675, 2019.
- [165] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Functammasan, A. Kolesnikov, N. D. Olson, *et al.*, “Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome,” *Nature biotechnology*, vol. 37, no. 10, pp. 1155–1162, 2019.
- [166] E. Uffelmann, Q. Q. Huang, N. S. Munung, J. De Vries, Y. Okada, A. R. Martin, H. C. Martin, T. Lappalainen, and D. Posthuma, “Genome-wide association studies,” *Nature Reviews Methods Primers*, vol. 1, no. 1, p. 59, 2021.
- [167] B. Song, R. Mott, and X. Gan, “Recovery of novel association loci in arabidopsis thaliana and drosophila melanogaster through leveraging indels association and integrated burden test,” *PLoS Genetics*, vol. 14, no. 10, p. e1007699, 2018.
- [168] S. A. Dugger, A. Platt, and D. B. Goldstein, “Drug development in the era of precision medicine,” *Nature reviews Drug discovery*, vol. 17, no. 3, pp. 183–196, 2018.
- [169] W. McLaren, L. Gil, S. E. Hunt, H. S. Riat, G. R. Ritchie, A. Thormann, P. Flicek, and F. Cunningham, “The ensembl variant effect predictor,” *Genome biology*, vol. 17, no. 1, pp. 1–14, 2016.
- [170] K. Wang, M. Li, and H. Hakonarson, “Annovar: functional annotation of genetic variants from high-throughput sequencing data,” *Nucleic acids research*, vol. 38, no. 16, pp. e164–e164, 2010.
- [171] J. B. Holmes, E. Moyer, L. Phan, D. Maglott, and B. Kattman, “Spdi: data model for variants and applications at ncbi,” *Bioinformatics*, vol. 36, no. 6, pp. 1902–1907, 2020.
- [172] M. J. Landrum, J. M. Lee, M. Benson, G. Brown, C. Chao, S. Chitipiralla, B. Gu, J. Hart, D. Hoffman, J. Hoover, *et al.*, “Clinvar: public archive of interpretations of clinically relevant variants,” *Nucleic acids research*, vol. 44, no. D1, pp. D862–D868, 2016.
- [173] C. Lee, C. Grasso, and M. F. Sharlow, “Multiple sequence alignment using partial order graphs,” *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002.

- [174] A. English, E. Dolzhenko, H. Z. Jam, S. Mckenzie, N. D. Olson, W. D. Coster, J. Park, B. Gu, J. Wagner, M. A. Eberle, M. Gymrek, M. J. Chaisson, J. M. Zook, and F. J. Sedlazeck, “Benchmarking of small and large variants across tandem repeats,” *bioRxiv*, 2023.
- [175] M. Martin, M. Patterson, S. Garg, S. Fischer, N. Pisanti, G. W. Klau, A. Schöenhuth, and T. Marschall, “Whatsap: fast and accurate read-based phasing,” *BioRxiv*, p. 085050, 2016.
- [176] E. D. Jarvis, G. Formenti, A. Rhie, A. Guarracino, C. Yang, J. Wood, A. Tracey, F. Thibaud-Nissen, M. R. Vollger, D. Porubsky, *et al.*, “Semi-automated assembly of high-quality diploid human reference genomes,” *Nature*, pp. 1–13, 2022.
- [177] F. Megiorni and A. Pizzuti, “Hla-dqa1 and hla-dqb1 in celiac disease predisposition: practical implications of the hla molecular typing,” *Journal of biomedical science*, vol. 19, pp. 1–5, 2012.
- [178] M. R. Lincoln, S. V. Ramagopalan, M. J. Chao, B. M. Herrera, G. C. DeLuca, S.-M. Orton, D. A. Dymont, A. D. Sadovnick, and G. C. Ebers, “Epistasis among hla-drb1, hla-dqa1, and hla-dqb1 loci determines multiple sclerosis susceptibility,” *Proceedings of the National Academy of Sciences*, vol. 106, no. 18, pp. 7542–7547, 2009.
- [179] T. Dunn, J. M. Zook, J. M. Holt, and S. Narayanasamy, “Phased hg002-grch38 small and structural variant calls [dataset],” 2024.
- [180] T. Dunn, “vcfdist,” 2023. <https://doi.org/10.5281/zenodo.8368282>.
- [181] W. Brockman, P. Alvarez, S. Young, M. Garber, G. Giannoukos, W. L. Lee, C. Russ, E. S. Lander, C. Nusbaum, and D. B. Jaffe, “Quality scores and snp detection in sequencing-by-synthesis systems,” *Genome research*, vol. 18, no. 5, pp. 763–770, 2008.
- [182] H. Li, J. M. Bloom, Y. Farjoun, M. Fleharty, L. Gauthier, B. Neale, and D. MacArthur, “A synthetic-diploid benchmark for accurate variant-calling evaluation,” *Nature methods*, vol. 15, no. 8, pp. 595–597, 2018.
- [183] P. Ebert, P. A. Audano, Q. Zhu, B. Rodriguez-Martin, D. Porubsky, M. J. Bonder, A. Sulovari, J. Ebler, W. Zhou, R. S. Mari, F. Yilmaz, X. Zhao, P. Hsieh, J. Lee, S. Kumar, J. Lin, T. Rausch, Y. Chen, J. Ren, M. Santamarina, W. Höps, H. Ashraf, N. T. Chuang, X. Yang, K. M. Munson, A. P. Lewis, S. Fairley, L. J. Tallon, W. E. Clarke, A. O. Basile, M. Byrskabishop, A. Corvelo, U. S. Evani, T.-Y. Lu, M. J. P. Chaisson, J. Chen, C. Li, H. Brand, A. M. Wenger, M. Ghareghani, W. T. Harvey, B. Raeder, P. Hasenfeld, A. A. Regier, H. J. Abel, I. M. Hall, P. Flicek, O. Stegle, M. B. Gerstein, J. M. C. Tubio, Z. Mu, Y. I. Li, X. Shi, A. R. Hastie, K. Ye, Z. Chong, A. D. Sanders, M. C. Zody, M. E. Talkowski, R. E. Mills, S. E. Devine, C. Lee, J. O. Korbel, T. Marschall, and E. E. Eichler, “Haplotype-resolved diverse human genomes and integrated analysis of structural variation,” *Science*, vol. 372, no. 6537, p. eabf7117, 2021.
- [184] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li, “Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm,” *Nature methods*, vol. 18, no. 2, pp. 170–175, 2021.

- [185] H. Li, “Minimap2: pairwise alignment for nucleotide sequences,” *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
- [186] C. Jain, A. Rhie, H. Zhang, C. Chu, B. P. Walenz, S. Koren, and A. M. Phillippy, “Weighted minimizer sampling improves long read mapping,” *Bioinformatics*, vol. 36, no. Supplement_1, pp. i111–i118, 2020.
- [187] Y. Bao, J. Wadden, J. R. Erb-Downward, P. Ranjan, W. Zhou, T. L. McDonald, R. E. Mills, A. P. Boyle, R. P. Dickson, D. Blaauw, *et al.*, “Squigglenet: real-time, direct classification of nanopore signals,” *Genome biology*, vol. 22, pp. 1–16, 2021.
- [188] A. Senanayake, H. Gamaarachchi, D. Herath, and R. Ragel, “Deepselectnet: deep neural network based selective sequencing for oxford nanopore sequencing,” *BMC bioinformatics*, vol. 24, no. 1, p. 31, 2023.
- [189] Y. Lin, Y. Zhang, H. Sun, H. Jiang, X. Zhao, X. Teng, J. Lin, B. Shu, H. Sun, Y. Liao, *et al.*, “Nanodeep: a deep learning framework for nanopore adaptive sampling on microbial sequencing,” *Briefings in Bioinformatics*, vol. 25, no. 1, p. bbad499, 2024.
- [190] B. Noordijk, R. Nijland, V. J. Carrion, J. M. Raaijmakers, D. De Ridder, and C. De Lannoy, “baseless: lightweight detection of sequences in raw minion data,” *Bioinformatics Advances*, vol. 3, no. 1, p. vbad017, 2023.