박 사 학 위 논 문
Ph.D. Dissertation

인공지능/머신러닝을 이용한 유전체 분석 파이프라인 향상에 관한 연구

Enhancing genome analysis pipeline with AI and ML

2024

정 영 목 (鄭 暎 穆 Jung, Young-mok)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

인공지능/머신러닝을 이용한 유전체 분석 파이프라인 향상에 관한 연구

2024

정 영 목

한 국 과 학 기 술 원

전기및전자공학부

인공지능/머신러닝을 이용한 유전체 분석 파이프라인 향상에 관한 연구

정 영 목

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2023년 12월 7일

Enhancing genome analysis pipeline with AI and ML

Young-mok Jung

Major Advisor: Dongsu Han
Co-Advisor: Young Seok Ju

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering

Daejeon, Korea
December 7, 2023

Approved by

Dongsu Han
Professor of Electrical Engineering

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DEE

정영목. 인공지능/머신러닝을 이용한 유전체 분석 파이프라인 향상에 관한 연구. 전기및전자공학부 . 2024년. 69+v 쪽. 지도교수: 한동수, 주영석. (영문 논문)
Young-mok Jung. Enhancing genome analysis pipeline with AI and ML. School of Electrical Engineering . 2024. 69+v pages. Advisor: Dongsu Han, Young Seok Ju. (Text in English)

초 록

DNA 시퀀싱 기술이 발전함에 따라, 정밀하면서도 비용 효율적인 유전체 분석 파이프라인의 필요성이 점점 더 중요해지고 있다. 본 논문은 인공지능(AI)과 머신러닝(ML)을 유전체 분석 파이프라인의 가장 중요한 두 가지 단계 유전체 정렬과 변이 탐지에 적용하여 향상시키는 방법론을 제시한다. 먼저 머신러닝을 이용하여 가속화된 유전체 정렬 소프트웨어 BWA-MEME을 제시한다. 이 소프트웨어는 learned-index를 사용함으로써 문자열 일치 알고리즘을 향상시켜 유전체 정렬에서 병목 지점인 seeding 단계를 개선한다. 다음으로 딥러닝 기반 변이 탐지 소프트웨어의 레이블이 있는 데이터에 대한 높은 의존도와 다양한 시퀀싱 기술에서 생성되는 서로 다른 오류에 대한 취약성을 개선한다. 제시된 준 지도 훈련 접근 방식은 레이블이 없는 데이터를 활용하여 오류 프로파일을 학습할 뿐만 아니라 도메인 적응 기술을 통합하여 다양한 오류 프로파일에서 발생하는 도메인 불일치를 최소화하였다. 결론적으로, 본 학위 논문은 유전체 분석 파이프라인에 AI와 ML의 새로운 활용 가능성을 제시한다.

핵 심 낱 말 인공지능, 머신러닝, 고성능 컴퓨팅, 유전체, 바이오인포매틱스

Abstract

As DNA sequencing technologies advance, the need for precise yet cost-effective genome analysis pipelines becomes increasingly vital. This dissertation unveils novel methodologies leveraging artificial intelligence (AI) and machine learning (ML) to enhance the two most critical steps in the genome analysis pipeline: read alignment and variant calling. Initially, we present BWA-MEME, an ML-augmented read alignment software. By employing learned indices, this software enhances the exact match search during the seeding phase—addressing a significant bottleneck in short-read alignment. Subsequently, we address challenges inherent to deep learning-based variant callers. These challenges encompass their reliance on vast labeled datasets and their susceptibility to diverse error profiles presented by different sequencing techniques. We devise a semi-supervised training approach that not only utilizes unlabeled data to learn error profiles but also incorporates a domain adaptation technique to minimize discrepancies arising from diverse error profiles. Together, these methods carve out novel pathways in read alignment and variant calling, underscoring the transformative potential of AI and ML within the genome analysis pipeline.

Keywords Artificial intelligence, Machine learning, High-performance computing, Genomics, Bioinformatics

Contents

List of Tables

List of Figures

Acknowledgment

To my family, advisors, and my friends who always support me!

Chapter 1. Introduction

In the realm of modern genetics and genomics, DNA sequencing stands as a monumental breakthrough, ushering a paradigm shift in our approach to disease diagnosis, prognosis, and therapeutic interventions. As techniques like next-generation sequencing (NGS) have matured, the precision and volume of data they generate have expanded exponentially. However, the sheer scale of this data is accompanied by challenges in its analysis and interpretation, particularly in the alignment of billions of short reads to reference DNA sequences and the identification of genomic variants like SNPs, INDELs, and SVs. To truly harness the potential of this vast genomic data, it is imperative to develop genome analysis pipelines that are not only precise but also cost-effective, ensuring that genome analysis remains accessible and feasible for a broader range of applications and research endeavors.

In the intricate landscape of genomics, diverse sequencing methodologies have emerged, each tailored to the specific needs and preferences of different institutions and research labs. Influencing these choices are numerous experimental factors, including the species under study, the source of samples [29, 70], library preparation methods from different vendors [18], the composition of sequences—encompassing read length and coverage—and the sequencer platform. It's not uncommon for individual labs to adhere to specific sequencing methods, influenced by their research focus, funding, or even historical choices. Such specialization inevitably results in varying distributions and characteristics of sequencing data across labs. This data heterogeneity presents a unique opportunity: by recognizing and understanding these nuances, there emerges a tangible pathway to refine genome analysis pipelines, making them more attuned to each specific sequencing method. This approach not only enhances the accuracy of the data interpretation but also elevates the overall efficiency of the genomics workflow.

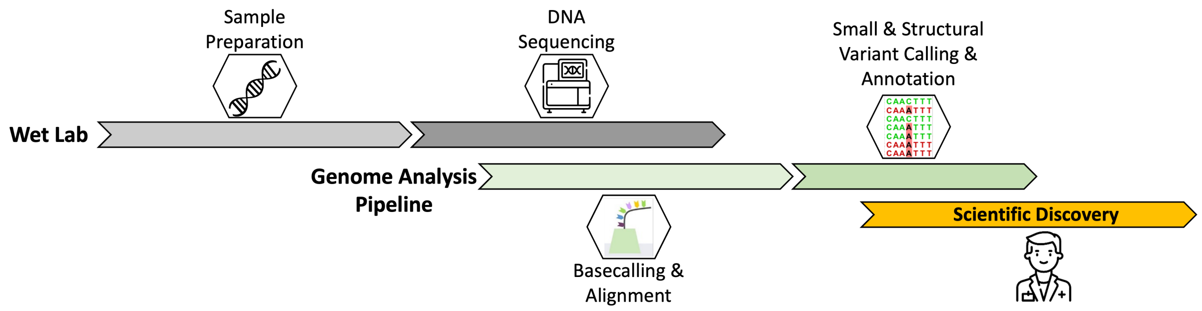


Figure 1.1: Overall workflow of DNA sequencing

In this thesis, we introduce two projects aimed at enhancing the genome analysis pipeline using AI and ML by specializing read alignment and variant calling software for the sequencing method of interest.

In Chapter 2, we dive into the significance and methods of read alignment in genomics, particularly in the realm of next-generation sequencing. With the rapid escalation of sequencing throughput, the alignment of short-reads to a reference DNA sequence has become a task of paramount importance. At the crux of this alignment lies the process of seeding, which involves searching for exact matches of short-read substrings in the reference sequence. Yet, conventional alignment algorithms often falter in performance due to their large number of memory access. BWA-MEME emerges as a solution to this quandary. As the first comprehensive ML-enhannced short-read alignment software, BWA-MEME capitalizes on learned indices to deftly handle the exact match search challenge, streamlining the seeding process. Through rigorous evaluations, BWA-MEME's markedly outperforms BWA-MEM2 in terms of seeding throughput, showcasing a commendable reduction in memory accesses, LLC misses, and the number of instructions while maintaining the consistency of SAM output.

In Chapter 3, the discourse transitions to the domain of deep learning-based variant callers (DVCs). These DVCs have surged to the forefront of small variant detection in DNA sequencing data, thanks to their unparalleled performance. Nevertheless, their dependency on supervised learning, combined with the requirement for expansive labeled datasets, introduces impediments to their generalizability across diverse sequencing techniques, each accompanied by its unique error profile. Even minor deviations within these profiles can significantly dampen DVC robustness, undermining the accuracy of variant calling for specific sequencing methods. This is where RUN-DVC, our proposition, comes into the picture. RUN-DVC introduces two techniques that complement the conventional supervised training approach. By adeptly harnessing semi-supervised learning, RUN-DVC discerns error profiles from unlabeled datasets of distinct sequencing methods. Furthermore, it employs domain adaptation techniques to minimize discrepancies stemming from varied error profiles. Comprehensive evaluations across diverse sequencing methods illuminates the exceptional adaptability and precision of RUN-DVC, highlighting its capacity to pave the way for a broader application of DVCs across an extensive spectrum of sequencing platforms.

Collectively, these approaches embody the central theme and progress toward a genome analysis pipeline with high accuracy, high efficiency, and high reliability. Chapter 4 provides closing thoughts.

Chapter 2. Accelerating read alignment software with machine learning

DNA sequencing has become a critical piece in modern medicine, advancing the practice in disease diagnosis, prognosis, and therapeutic decisions. The state-of-the-art DNA sequencing method is called next-generation sequencing (NGS). Modern NGS hardware generates billions of short reads in a single run. This, in turn, requires the alignment of short reads (i.e., short DNA fragments) to the reference DNA sequence. As large-scale DNA sequencing operations run hundreds of NGS, developing an efficient short read alignment algorithm has become ever more important.

The state-of-the-art alignment process is divided into two phases, seeding and extending, following the seed-and-extend paradigm [46, 50, 42, 17, 48, 44]. The seeding phase searches for exact matches of seeds (substrings of short reads) in the reference DNA sequence, which identifies the possible alignment positions of the short read in the reference DNA sequence. In the extending phase, the seeds from the earlier phase are extended. In the process, the alignment scores of the extended seeds are calculated using the Banded-Smith-Waterman (BSW) algorithm [71]. Many studies [65, 25, 2, 68] have shown that the seeding phase is a major performance bottleneck in the popular alignment software BWA-MEM2 [71] and Bowtie 2 [39]. In particular, finding exact matches of short reads—or substrings of short reads—within the reference DNA sequence, is the main problem that fundamentally constrains the performance of the seeding phase [39, 47, 42]. Thus, in this chapter, we focus on the seeding phase which involves only the exact matching.

For an efficient exact match search, it is necessary to index the reference DNA sequence and perform an in-memory index lookup. Recently, several new index structures have been studied to solve the exact match search problem in an efficient manner. The state-of-the-art index structures fall into two major categories: traditional index-based [71, 65] and machine-learning-based [33, 23]. Examples of the traditional index structures are FM-index [42, 45, 47, 39] and enumerated radix tree (ERT) [65] index. FM-index is a compressed version of the suffix array [20], which progressively extends a substring from a single character and finds its exact match in the reference DNA sequence. We refer to the substring of the short read that is given as input to the exact match search problem as a substring. The typical length of a short read is between 100 to 300. To speed up the process, the ERT index uses an enumerated index table for finding the exact match of the first 15 consecutive bases of the substring. After this, ERT utilizes a radix tree that encodes the suffixes of the reference DNA sequence, which naturally supports multi-character lookups. Despite this, both FM-index and ERT index require $O(N)$ memory accesses, where N is the length of the input substring.

LISA [22, 23] and Sapling [33] use machine-learning-based index structures. LISA proposes a new data-structure called IP-BWT. IP-BWT employs a learned index that supports an exact match lookup in the suffix array. Although it is still a linear time algorithm, using IP-BWT requires fewer memory accesses compared to the original FM-index because it matches 21 bases in a single lookup. However, LISA assumes that seed search only starts at the first base of the short read, whereas the seeding algorithm in BWA-MEM2 requires to start the search at an arbitrary point in the short read. Thus, LISA cannot be used as a replacement for BWA-MEM2. Sapling [33] has shown that employing a learned index in suffix array search can outperform FM-index based algorithms. However, Sapling assumes that the exact match length of the input substring is fixed (i.e., given or known) as in alignment software BLAST [4], BLAT [27], Bowtie2 [39], and Minimap2 [44]. For variable-length seeding, the seed length and the exact match length of the input substring are not fixed. Therefore, Sapling cannot be directly employed in variable-length seeding that is widely used in alignment software, such as BLASR [9], STAR [17], MUMmer4 [54], and BWA-MEM2 [71].

The key limitation of applying aforementioned index structures in variable-length seeding is that it requires memory accesses and instructions proportional to the length of the substring. In reality, variable-length seeding requires exact matching of substrings with lengths ranging from tens to hundreds. Our central tenet is that to obtain high seeding throughput, we must minimize the number of memory accesses required for exact match search of arbitrary length substrings and break the strong dependency between the length of substring and the number of memory accesses.

In this chapter, we present BWA-MEME, the first alignment software that performs exact match search leveraging the learned index. Specifically, BWA-MEME is the first learned-index-based variable-length seeding algorithm that uses a constant number of memory accesses for an exact match search of arbitrary length substrings. We introduce a novel Partially-3-layer RMI (P-RMI) model, and train it to guarantee error bounds for arbitrary length substrings, turning the variable-length exact matching to machine learning model inference.

Building a full-fledged alignment software that leverages learned index in suffix array search, however, involves solving a number of new and non-trivial challenges. First, the learned index has to provide an accurate exact match position of an arbitrary substring in the suffix array. However, it is difficult to guarantee high prediction accuracy in the suffix array which is large and has an imbalanced distribution of suffixes. Furthermore, variable-length suffix or substring must be encoded into the numerical key to use the learned index. Second, using the suffix array search requires a new design for the super-maximal exact match (SMEM) search [41, 42]. SMEM search algorithm must find all seeds covering the given point of the short read while minimizing the number of exact match searches. The seeding algorithm uses a hit threshold to find seeds that have multiple hits in the reference DNA sequence. Thus, the SMEM search algorithm is required to find seeds with the maximum number of hits but is less than or equal to the hit threshold. Finally, minimizing the memory accesses and CPU cache misses introduced by using a learned index is important.

BWA-MEME addresses these challenges by introducing the new learned index structure and algorithms. First, we present the partially-3-layer recursive model index (P-RMI) which adapts well to the imbalanced distribution of suffixes and provides accurate prediction. In this process, we design an algorithm that encodes the input substring or suffixes into a numerical key. The numerical key is given as input to P-RMI where P-RMI provides predicted position in the suffix array and error bound for the prediction. Binary search is performed within the error bound to find the reference position where the substring aligns to. Second, we present an efficient SMEM search algorithm that uses the same or a less number of exact match searches compared to the state-of-the-art SMEM search algorithms. Finally, we reuse the lookup result to exploit the redundancy of the input substrings to the exact match search problem. This further reduces the number of memory accesses and CPU cache misses that occur during the seeding phase.

Our evaluation shows that, 1) BWA-MEME achieve up to 3.45x and 1.42x speedups in seeding throughput and alignment throughput respectively over BWA-MEM2, while ensuring identical output; 2) BWA-MEME drastically reduces the number of instructions executed by 4.60x, memory accesses by 8.77x, last-level-cache (LLC) misses by 2.21x, and data fetched per read by 4.41x compared to seeding algorithm of BWA-MEM2; 3) Due to the fast seeding algorithm, BWA-MEME shows the highest single-end alignment throughput compared to other state-of-the-art alignment algorithms Bowtie2, Whisper2, and Minimap2; and 4) BWA-MEME provides options to balance the trade-off between alignment throughput and the required memory space.

2.1 Background

2.1.1 The Seeding algorithm of BWA-MEM2 and ERT

A maximal-exact match is a substring (of short reads) that cannot be further extended in either direction without a change in the number of hits (exact matches) to the reference DNA sequence. A SMEM is a unique MEM that is not contained in other MEMs. To find all positions of seeds where the short read may potentially align, the seeding algorithm executes a super-maximal exact match (SMEM) search multiple times with various pivot points, minimum seed length thresholds, and hit thresholds. SMEMs found in the SMEM search that are longer than the minimum seed length threshold are selected as seeds. In the following, we denote the SMEM search algorithm of BWA-MEM2 [71] and ERT [65] as SMEM-BWA and SMEM-ERT, respectively.

SMEM search algorithm and extension. We first describe the extension used in SMEM search and explain how extensions are performed to find SMEMs. Let S be a short read, and $S[i, j]$ denote the substring between position i and position j of the short read. Extension from position P_s is finding farthest position P_e where the substring $S[P_s, P_e]$ has a maximum number of hits but is less than or equal to the hit threshold. Therefore, for each extension, the exact match search algorithm is used to find the number of hits for substring $S[P_s, P_e]$. Depending on the direction of extension, it is called forward extension if $P_s < P_e$ and otherwise backward extension. We denote each point where forward and backward extension end as $forward(p)$ and $backward(p)$, respectively.

The goal of a SMEM search algorithm is to find all SMEMs that include the pivot point P_{pivot}. As SMEMs are substrings that cannot be further extended, all SMEMs that include P_{pivot} can be found by; 1) performing a forward extension from P_{pivot} of the short read and; 2) performing a backward extension from every point in $[P_{pivot}, forward(P_{pivot})]$. This method finds all SMEMs that include P_{pivot} but incurs excessive computation.

SMEM search algorithm of BWA-MEM2. Instead of performing backward extension in all points in $[P_{pivot}, forward(P_{pivot})]$, SMEM-BWA performs backward extension only in the point p where substring $S[P_{pivot}, p]$ and substring $S[P_{pivot}, p + 1]$ have different number of hits. This is because for $\forall p \in [P_{pivot}, forward(P_{pivot})]$, if substring $S[P_{pivot}, p]$ and substring $S[P_{pivot}, p + 1]$ have the same number of hits, $backward(p)$ and $backward(p+1)$ are identical. Therefore, during the forward extension from P_{pivot}, all positions where the number of hits changes are marked as left extension points (LEPs).

Figure 2.1 illustrates the design. **(1) Determining LEPs:** SMEM-BWA starts with the forward extension of a single character at the pivot point of the short read. During the forward extension, substrings are extended one character at a time using the FM-index. The LEP bit is set to 1 when the number of hits changes from the preceding substring. In Figure 2.1, the number of hits decreases as the length of forward extended substring increases, and the LEP bit is set to 1 when the number of hits changes. Each dashed line box represents a single extension. **(2) SMEM search:** The backward extension is performed in the positions where the LEP bit is set to 1. After the backward extension is performed in each position, SMEMs are selected from the backward extended substrings. SMEMs are backward-extended substrings that are not contained in other substrings and are longer than the minimum seed length threshold. In Figure 2.1, backward extension is not performed in the substrings with LEP bit set to 0, which is labeled "Not Extended". Substrings that are contained in other longer substrings are labeled "Contained", and substrings that are not contained in the other substrings are selected as SMEM which is labeled "SMEM".

SMEM search algorithm of ERT. One limitation of SMEM-BWA is that it still performs backward extensions on substrings that eventually do not become SMEM. To overcome this limitation, SMEM-ERT performs extensions in a zigzag fashion. Similar to SMEM-BWA, SMEM-ERT performs backward and forward extension in positions where the LEP bit is set to 1. This is intended to avoid finding duplicate SMEMs and reduce redundant extensions.

Figure 2.2 illustrates the two stages of SMEM-ERT: **(1) Obtaining LEP bits:** SMEM-ERT starts with a forward extension at the pivot point of the short read. LEP bits for all characters in the forward extension are obtained from the ERT index. **(2) SMEM searching in zigzag fashion:** Starting at the pivot point, the backward and forward extensions are repeatedly performed until the forward extension reaches the end of the obtained LEP bits. The forward extension starts at the point where the backward extension ends, and the backward extension always starts at the nearest point where the LEP bit is set to 1, as illustrated in Figure 2.2.

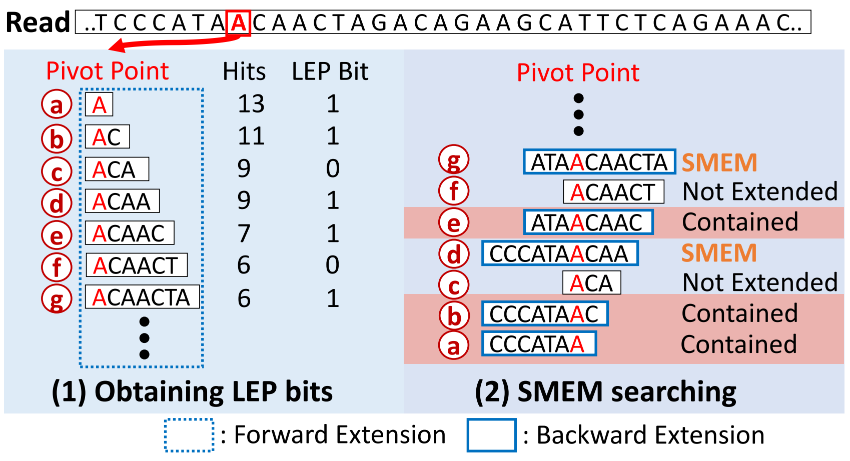
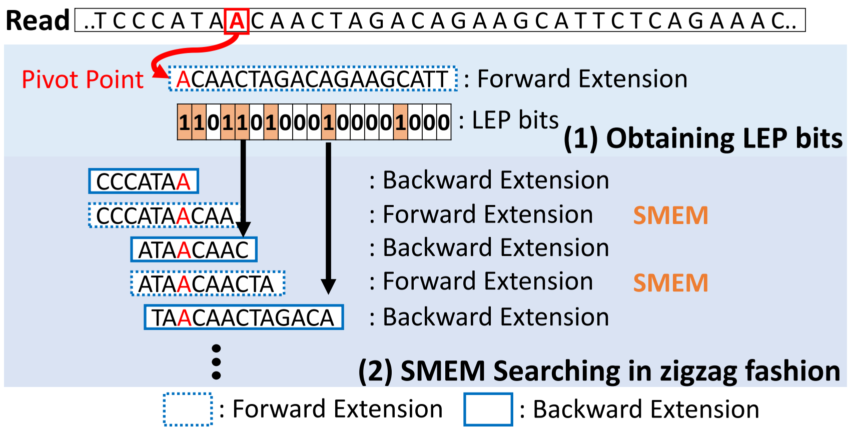


Figure 2.1: **SMEM Search Algorithm of BWA-MEM2 (SMEM-BWA algorithm)**



Read ..T C C C A T A A C A A C T A G A C A G A A G C A T T C T C A G A A A C..
Pivot Point
ACAACTAGACAGAAGCATT : Forward Extension
1101101000100001000 : LEP bits
(1) Obtaining LEP bits
CCCATAA : Backward Extension
CCCATAACAA : Forward Extension SMEM
ATAACAAC : Backward Extension
ATAACAACTA : Forward Extension SMEM
TAACAACTAGACA : Backward Extension
(2) SMEM Searching in zigzag fashion
: Forward Extension
: Backward Extension

Figure 2.2: **SMEM Search Algorithm of ERT (SMEM-ERT algorithm)**

2.1.2 Recursive Model Index Structure

Learned indices use machine learning (ML) models (e.g., linear regression model) to replace traditional index structures or search methods which are used for sorted key-value data (e.g., B-tree, binary search). Recursive model index (RMI) [36] is a commonly used hierarchical structure for learned indices, which uses multiple layers of ML models. The first layer usually has a single model and subsequent layers have a progressively larger number of models. The algorithm of RMI consists of the training and lookup phase. For simplicity, we refer to an ML model as a model.

Training phase. Training is performed layer by layer starting from the first layer. Model in the first layer is trained with whole key-value data to predict the position of keys accurately. Then keys are distributed to the models in the second layer according to the predicted positions of keys. Models in the second layer are trained with the assigned key-value data and keys are again distributed to the models in the third layer. This process continues until the models in the last layer are trained. As prediction may have an error, RMI guarantees a search bound for keys seen in the training phase. The error bound of leaf models is calculated by iterating through keys assigned to each leaf model. We refer to models that are used for the output of RMI as the leaf models. Each leaf model stores the calculated error bound to be used later in the lookup phase.

Lookup phase. The lookup phase of RMI consists of two stages. **Stage 1) Prediction**: Given a key, the first layer model predicts the position of the key, and one of the models in the second layer is selected according to the prediction. The selected model in the second layer subsequently makes a prediction, and one of the models in the third layer is selected. RMI repeats this process recursively until the model in the last layer is selected. The prediction made in the last layer is used as the output of RMI. **Stage 2) Last mile search**: The true position of the key is searched starting from the predicted position of RMI. Binary search can be used if an error bound for prediction is defined in the training phase. Otherwise, linear search or exponential search is used.

2.2 Motivation and Goal

To solve the exact match search problem with minimal time, it is necessary to index the reference DNA sequence and perform an in-memory index lookup. However, it is a challenging task to build an index that fits in limited memory and supports exact match search of substrings (which has a length between 0 to 300) in a long reference DNA sequence (e.g., 3 billion lengths for human reference DNA).

Why use learned index? Applying machine learning based indexing in the short read alignment is attractive considering that reference DNA sequence does not change frequently. Once the models are trained, they can be used without further training unless the new version of the reference DNA sequence is made. In addition, most of the short reads have the perfect match to the reference DNA sequence [12], which means that the test data and the train data are similar. This makes the machine-learning based approach even more effective.

Why apply learned index on suffix array? A suffix array is an array that stores the positions of suffixes sorted in lexicographical order. For a human reference DNA sequence whose size is 6G bases (including its reverse complement), the corresponding suffix array is 31 GB. FM-index is a compressed index created by applying Burrows-Wheeler transform (BWT) [45] on the reference genome. It is one of the popular index structures that trade-offs between lookup speed and main memory consumption [73]. However, it is not uncommon to find recent workstations with tens to hundreds of GB of main memory. Thus, recent aligners [17, 48, 54, 71, 15, 65, 16, 23] leverage larger index structures compared to the one used in FM-index. More recently, LISA presented IP-BWT which integrates learned index to FM-index.

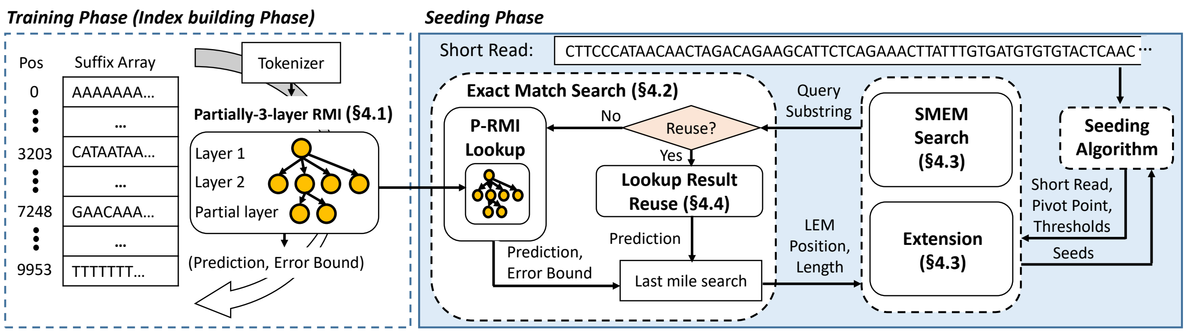


Figure 2.3: BWA-MEME **design overview**. In the training phase (index building phase), P-RMI learns the distribution of suffixes with the 64-bit tokens from the suffix array(described in §2.1.2, §2.3.1). In the seeding phase, the seeding algorithm [42] finds seeds using multiple times of SMEM search (§2.3.3) and extensions(§2.3.3). Note that the extension used in SMEM search should not be confused with the seed extension which is performed after the seeding phase. The SMEM search algorithm of BWA-MEME (SMEM-MEME) performs the backward and forward extensions(§2.3.3) sequentially, starting from the given pivot point. The pivot point is initially set to the first base of the short read and is updated to the position where the forward extension ends. The forward extended substring of the short read becomes the potential SMEM candidate. Subsequently, to find all SMEMs in the short read, an SMEM search is repeatedly performed until the pivot point is set to the last base of the short read. Each extension involves a single longest exact match (LEM) search (Exact-MEME §2.3.2), therefore the Exact-MEME algorithm is invoked multiple times in the SMEM search algorithm. The pseudo code of all algorithms can be found in Supplementary Material.

In particular, LISA processes 21 bases in a single learned index lookup followed by an additional binary search in IP-BWT. For the input substring that is longer than 21, LISA requires multiple learned index and IP-BWT lookups, which result in memory accesses proportional to the length of the substring. Therefore, the number of memory accesses in LISA is lower bounded by the length of the substring, even if the learned index predicts the exact position.

In contrast, directly employing a learned index to suffix array search without using a compressed structure of FM-index, requires a number of memory accesses independent of the length of the input substring. This is because the number of memory accesses required in the suffix array search is bounded by the errors of the leaf models in P-RMI which is a constant value determined at the training phase. The learned index itself requires a single memory access to find the exact match in the best case, which is the minimum achievable value for the exact match search problem. Hence, to achieve the minimal lookup time, we choose to utilize learned index on the suffix array.

Goal of BWA-MEME. Our goal is to build a practical and efficient alignment software that leverages learned index and suffix array search in the seeding phase. The accuracy of the short read alignment is important, therefore the output must be identical with that of BWA-MEM2. This paper considers running alignment software on CPUs only. However, we believe it can be further accelerated by using hardware acceleration.

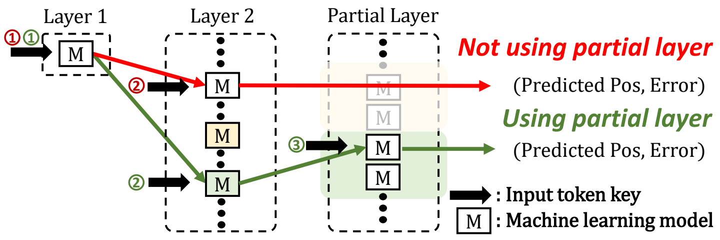


Figure 2.4: Schematic diagram of P-RMI structure and P-RMI lookup. P-RMI constrains the models to form a tree. Hence, every model should have a single parent model pointing to it. The colored long arrow indicates the prediction made by the model and the black thick arrow indicates a token key given as input to the model. The numbers beside the thick arrows are the order of the model inference for each case 1) Not using partial layer and 2) Using partial layer.

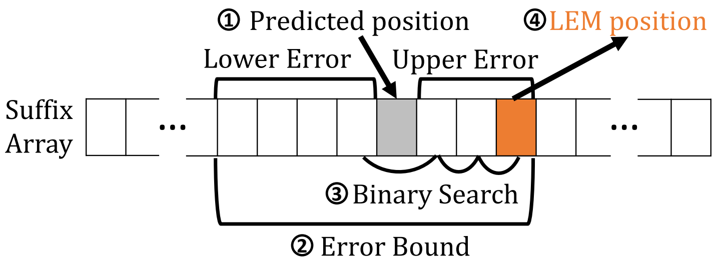


Figure 2.5: How last mile search is performed given the predicted position and the error value. A predicted position Pos_p and an error value are given as input from the P-RMI lookup. The lower bound error $error_l$ and the upper bound error $error_u$ are obtained from the error value. Subsequently, the LEM position is determined by performing a binary search within the error bound (Pos_p-$error_l$,Pos_p+$error_u$).

2.3 Design of BWA-MEME

2.3.1 P-RMI: Partially-3-layer RMI

We present P-RMI that enables an efficient suffix array search by making two enhancements to 2-layer RMI.

Mitigating data imbalance. Due to redundant sequences in the reference DNA, using naïve 2 or 3 layer RMI results in imbalanced distribution of data in the leaf models. The number of keys per leaf model forms a long tail distribution, with a small number of leaf models occupying a significant fraction of keys. A model trained with a small subset of data generally has a higher prediction accuracy. Therefore, mitigating data imbalance is necessary to improve the prediction accuracy of RMI. For the human reference DNA and a 2-layer RMI constructed with 2^{28} leaf models, 85% of the leaf models have data, and the rest are empty. Among the non-empty leaves, 0.22% of them hold 16.5% of key-value data from the suffix array.

To mitigate the imbalance in data distribution with minimum overhead, we introduce partially-3-layer RMI (P-RMI). Instead of fully employing the third layer models, P-RMI adaptively adds an additional layer of models to the second layer models which are suffering from elongated lookup time due to the imbalance in data distribution. Training P-RMI is done layer by layer which is the same as RMI. If the number of assigned keys in the second layer model exceeds the number of key thresholds, an additional layer of models is added to the second layer model. Accordingly, the prediction of the second layer model shifts its role to assign keys to the models in the added layer. The number of models in the

additional layer is calculated according to the number of keys assigned and the target average keys per leaf model. When an additional layer is added, an indicator bit is set and stored with the parameters.
Defining the error bound for last mile search. To minimize the number of memory access in the last mile search, we choose to perform a binary search within the predefined error bound instead of using the exponential search or linear search. However, RMI does not guarantee an error bound for the keys that are not provided in the training phase. The original RMI work provides a solution that can be used in 2-layer RMI when all models within the RMI are monotonic [36, 59, 55]. However, it does not generalize for 3 layer RMI even if the models are monotonic. Thus, we extend the solution to work on our P-RMI.

Our main observation is that the error bound can be guaranteed for the multi-layer RMI, if the leaf models are monotonically increasing functions and the index of the leaf model that a key is assigned to is monotonically increasing with regard to the key. We omit the proof here. Therefore, we build P-RMI with two design constraints. First, P-RMI forces each model to be a monotonically increasing function. Second, the P-RMI constrains the models to form a tree. Using the two constraints, P-RMI calculates an error bound in each leaf model that guarantees to include the true position of arbitrary keys assigned to the model (e.g., keys unseen in the training phase). We provide an explanation in Supplementary Material about how the error bound of P-RMI is guaranteed for arbitrary keys and how the error bound of each leaf model is calculated for P-RMI.
P-RMI configuration. The following factors determine the lookup performance of P-RMI: number of models and types of ML models. Using a large number of models results in a smaller error bound but increases the size of P-RMI. To balance between the lookup performance and the size of P-RMI, BWA-MEME chooses the number of models in the second and additional layer to match the target average keys per model. Also, an additional layer is added to the second layer models where the number of keys exceed the number of keys threshold. We choose 20 as the target average keys per model and 500 as the number of keys threshold. Another important configuration is the type of model as it affects both the prediction accuracy and the size of the index. We found the best performance can be obtained using bit shift operation in the first layer, linear regression models in the second layer, and linear spline models in the additional layer. For the human reference DNA, P-RMI has 2^{28} models in the second layer and 48,047,097 models in the additional layer resulting in a total of 7.6GB.

2.3.2 Exact Match Search with P-RMI

BWA-MEME replaces the exact match search algorithm of BWA-MEM2 with an exact match search algorithm (Exact-MEME) based on suffix array search that uses P-RMI. Specifically, the Exact-MEME algorithm finds the longest exact match (i.e., the longest common prefix) position of the variable-length query sequence in suffix array.
Tokenization of query sequence. Tokenization is the procedure that encodes a variable length suffix or a substring of the short read into a numerical key. The tokenized key should preserve the lexicographical order and be expressive enough to represent the string key. We observe that most suffixes longer than a certain length become unique suffix strings in the reference DNA sequence. Therefore, we choose to use the first K characters of the query sequence and apply 2-bit encoding to the characters. However, the model computation cost substantially increases along with the K size [76]. For example, using a 128-bit (64 bases) key requires a 128-bit machine learning model to be used in RMI. There are no mainstream processors that have hardware support for the floating-point operation with bits larger than 64. This results in a simple 128-bit linear regression model to be 5 times or much slower than the

64-bit linear regression model. Also, we observe that K larger than 32 has marginal gain in prediction accuracy, thus we select 32 for K.

Tokenization of variable-length string is done in two steps. The first 32 characters are obtained from each variable-length string by padding a single or multiple arbitrary characters to the string if the string is shorter than 32 and trimming characters if it is longer. The 32 characters are then encoded into a numeric key using the 2-bit encoding of bases. Note that the tokenized key is not used for last mile search in the suffix array but is used only to predict the position and to obtain the error bound.

P-RMI lookup. P-RMI stores an indicator bit in the second layer models that tells an additional layer of models is added or not. If the indicator bit is not set, the prediction and error value from the second layer is chosen as the output. If the indicator bit is set, the start and end indices of the third layer are obtained from the error value. According to the prediction of the second layer model, the leaf model is selected among the third layer models between the start and end indices.

The error obtained from P-RMI includes the upper and lower bound errors which define the error bound. A binary search is performed within the error bound to find the sorted position of the query sequence in the suffix array. If the sorted position is found and the query sequence exactly matches to the corresponding suffix, the sorted position is selected as the longest exact match position. If the sorted position of the query sequence is between two sequential suffixes, the position of suffix with the longer exact match is selected. For notation simplicity, we refer to the longest exact match found in Exact-MEME as LEM.

Reducing memory accesses in the last mile search. The last mile search stage accounts for a significant portion of memory accesses. Therefore, it is important to reduce the memory accesses. During the last mile search, comparisons are performed several times between suffixes and the input substring. To compare a suffix and the input substring, characters of a suffix should be obtained from the reference DNA sequence which requires two random memory accesses. The first memory access is made to the suffix array which contains the position of the suffix in the reference DNA sequence. Subsequently, another memory access is made to the reference DNA sequence to retrieve the suffix characters. The two memory accesses are random and likely to cause CPU cache miss. To reduce the cache miss, BWA-MEME co-locates position value and the characters of the suffix in a single index data-structure. In particular, the position value and the 32 characters (64-bit) of the suffixes are co-located, allowing any substring shorter than 33 can be compared with a single CPU cache miss. For the human reference DNA sequence, using 32-character suffixes results in 49GB of memory usage.

2.3.3 Making SMEM Search Efficient

This section presents how BWA-MEME find SMEMs using the Exact-MEME algorithm.

Extension with Exact-MEME algorithm. The extension finds a substring that has the maximum number of hits but is less than or equal to the hit threshold. However, the Exact-MEME algorithm finds only the longest exact match (LEM) position [51] without progressively extending a substring.

To remedy this, BWA-MEME designs an extension function that performs a linear search starting from the output LEM position of the Exact-MEME algorithm. The key insight is that exact matching positions are all sequentially positioned in the suffix array and the exact match length of query in suffixes monotonically decreases as the distance from the LEM position increases. The algorithm of the extension function consists of 3 steps. First, the query of extension function is given to the Exact-MEME algorithm which outputs the LEM position of the query in the suffix array. Second, a linear search is performed starting from the LEM position. The linear search defines an exact match range of suffixes where the

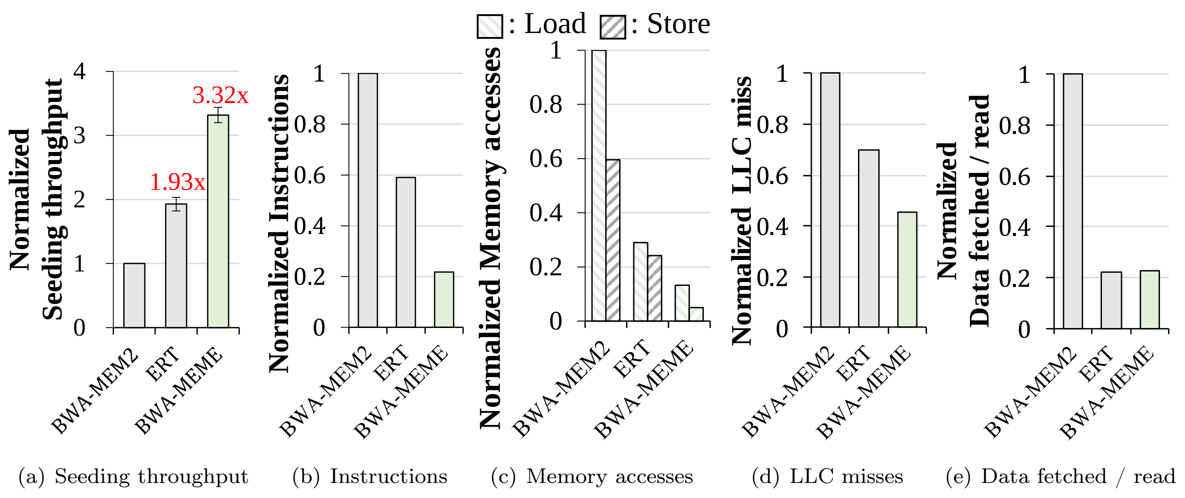


Figure 2.6: **Comparing BWA-MEM2, ERT, and BWA-MEME**

input substring has a partial exact match. The size of the exact match range must be as large as possible but should be less than or equal to the hit threshold. Also, input substring and all suffixes in the exact match range must have an exact match length longer than that of suffixes not in the exact match range. Finally, the start position of the exact match range, the minimum exact match length inside the exact match range, and the size of the exact match range (number of hits) are returned as the output.

Reducing redundant extensions without using LEP bits. Reducing the number of redundant extensions in the SMEM search algorithm is necessary to minimize the number of exact match searches. SMEM-BWA and SMEM-ERT achieve this by tracking the change in the number of hits during the first forward extension. The LEP bits obtained from the forward extension are used to reduce redundant extensions. However, unlike SMEM-BWA or SMEM-ERT, it is infeasible to track the change in the number of hits while extending a substring using the Exact-MEME algorithm. Therefore, we design a new SMEM search algorithm that does not require LEP bits and uses the same or a less number of extensions compared to SMEM-BWA or SMEM-ERT. We observe that repeatedly performing backward and forward extensions starting from the pivot point finds all SMEMs without redundant extensions. The extensions are performed until the extension no longer includes the pivot point. We show the correctness of SMEM output by proving the SMEM output is identical with SMEM-ERT. The proof is provided in Supplementary Material.

2.3.4 P-RMI Lookup Result Reuse

To identify all possible alignment candidates, the seeding algorithm performs multiple SMEM searches on each short read with various pivot points and thresholds. This results in higher sensitivity in alignment [42], but also numerous exact match searches of substrings that have long overlap with each other. However, even if the queries have long overlap with each other, different paths are accessed in P-RMI which incurs random memory accesses, resulting in CPU cache miss. The P-RMI lookup and last-mile search accounts for most of the computation time in the seeding phase, thus reducing their cost brings further speedup.

Substituting P-RMI lookup for ISA lookup. To reduce memory accesses, we use an additional index called inverse suffix array (ISA). At the index building step, BWA-MEME constructs ISA that

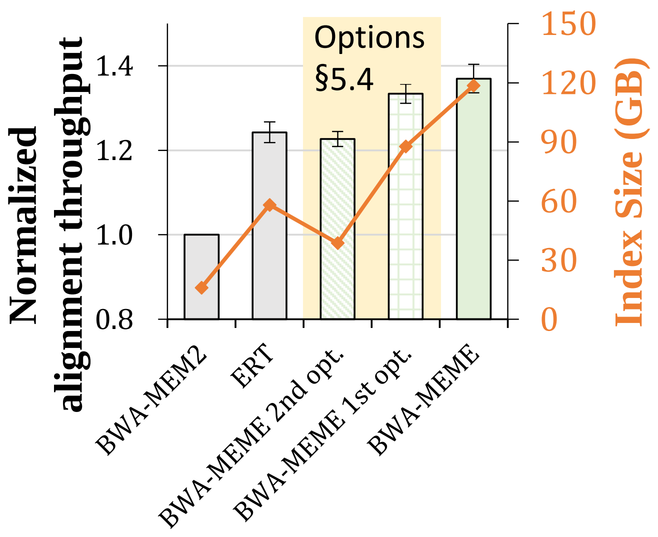


Figure 2.7: **Alignment throughput of BWA-MEM2, ERT, variants of BWA-MEME,and BWA-MEME**

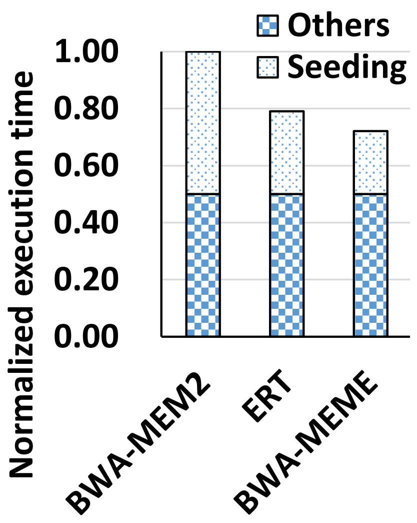
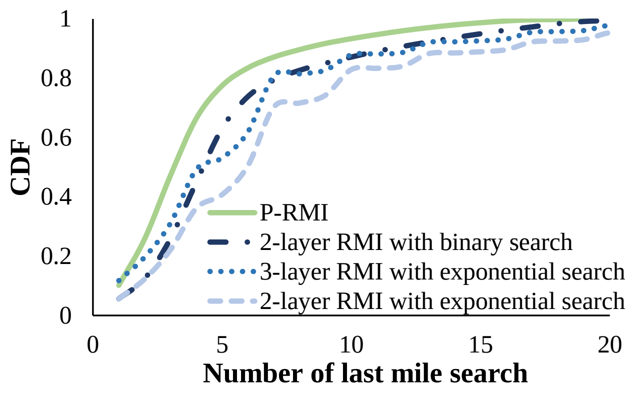


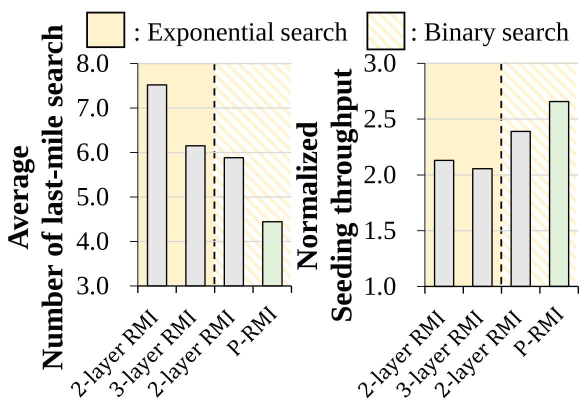
Figure 2.8: Runtime of the seeding phase in alignment. Seeding and seed chaining are included in the time for seeding.

stores the translation from the reference position to the suffix array position (i.e., $ISA[SA[j]] = j \ for \ j \in |SA|$). Assume a query Q_i where the LEM position in the suffix array is known and a new query Q_n are given. BWA-MEME uses the ISA and LEM position of Q_i to predict the LEM position of Q_n if $Q_i[X : X + N]$ overlaps with $Q_n[0 : N]$ where X is the start position and N is the length of the overlap. We refer to the overlapping bases between the identified query and the new query as *overlapping bases*. Let SA and idx be the suffix array and LEM position of Q_i, such that Q_i aligns to $SA[idx]$ position in the reference. Then the overlapping bases must align to $SA[idx] + X$ position in the reference and the LEM position of the overlapping bases in the suffix array is $ISA[SA[idx] + X]$. To find the LEM position of Q_n where $Q_n[0 : N]$ is the overlapping bases, BWA-MEME performs last-mile search starting at $ISA[SA[idx] + X]$ in the suffix array. The benefits of using ISA instead of P-RMI lookup come from better spatial locality and higher prediction accuracy. To predict the LEM position of the Q_n, ISA is accessed within the length of short read distance from $ISA[SA[idx]]$, whereas P-RMI lookup requires completely random access to memory. Also, long overlaps often lead to unique suffixes which results in better prediction than using P-RMI lookup.

When to reuse the lookup result. BWA-MEME decides to reuse P-RMI lookup result in two cases: First, when the short read has a perfect match to the reference DNA sequence, it is guaranteed for the LEM positions of any new queries to align to the perfect exact match position of the short read. The LEM positions of the new queries can be concluded from the LEM position of the perfect exact match without further last-mile search. As a perfect exact match of short read is common in NGS, this reduces a significant amount of memory accesses and CPU cache misses. Second, when the new query partially overlaps with the identified query and the overlapping bases are long enough, the LEM position of the new query is likely to be near the LEM position of the overlapping bases. To find the actual LEM position in the suffix array, we use an exponential search starting from the LEM position of the overlapping bases.



(a) Cumulative distribution of the number of last mile search for P-RMI and the original RMIs



(b) Average number of last mile search and normalized seeding throughput

Figure 2.9: **Comparing P-RMI and original RMIs**

2.4 Results

We evaluate BWA-MEME to answer the following questions:

- Does BWA-MEME have identical SAM output with BWA-MEM2?
- How does it compare with existing algorithms?
- How effective is P-RMI compared to other index structures?
- How robust is BWA-MEME?
- How does BWA-MEME adapt to various memory sizes in servers?

2.4.1 Methodology

Setup. We ran the experiments on Intel Xeon Gold 5220R @ 2.2 GHz with 24 cores, 32KB L1, 24 MB L2, 35.75 MB L3 caches, running Ubuntu 20.04 (Linux kernel 5.4.0). We used the numactl utility to force all memory allocations to a single socket with 225GB of RAM. Unless noted otherwise, 48 threads were used for all experiments with hyper-threading. To analyze the memory access characteristics, we used the Intel Vtunes profiler.

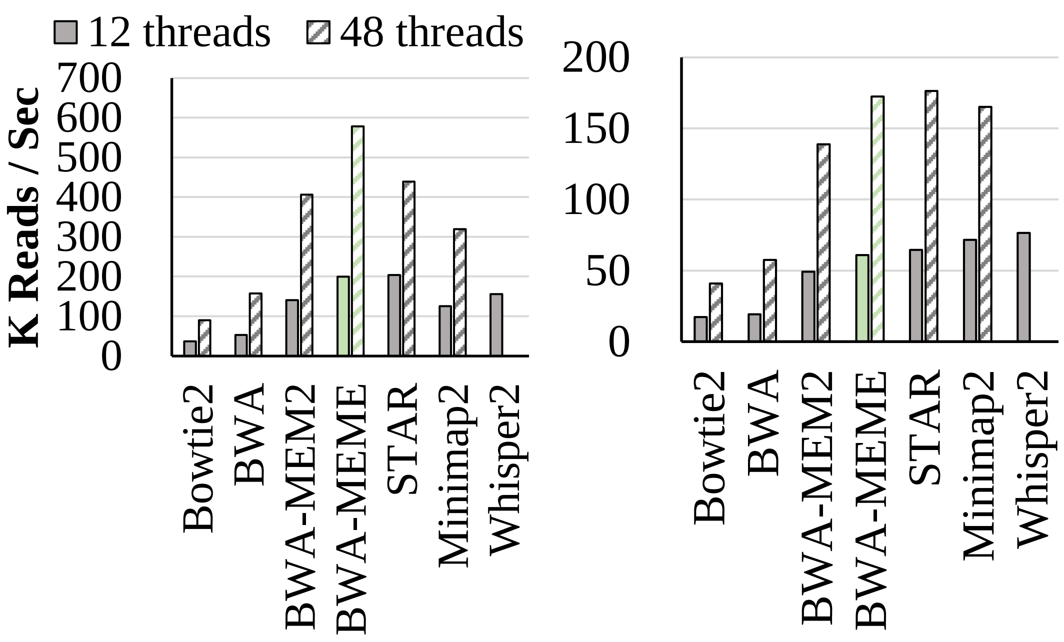
Dataset. We use the reference human genome assembly (human_g1k_v37) and 16 short reads datasets—7 from Illumina Platinum Genomes [19] and 9 from 1,000 Genomes Project Phase 3 [12]. Details are included in Supplementary Material.

Implementation. All algorithms of BWA-MEME are implemented in 3.5k lines of C++ code and integrated into BWA-MEM2 code. BWA-MEM2 is widely used alignment software with various features used by researchers. To guarantee BWA-MEME supports all features supported in BWA-MEM2, we choose to replace the seeding algorithm in BWA-MEM2 with our seeding algorithm. For the correctness of BWA-MEME, we verified BWA-MEME and BWA-MEM2 have identical SAM outputs in all 16 short read datasets. We implement the training process of P-RMI on top of the open-source code based on Rust from the authors of the learned index [36]. The training process outputs the model parameters of P-RMI in binary data which is stored along with the indices of the reference DNA. The model parameters are loaded in the index loading step of BWA-MEME and used for the seeding algorithm.

2.4.2 BWA-MEME Performance Benchmark

Seeding throughput comparison with BWA-MEM2 and ERT. To demonstrate BWA-MEME delivers significant improvement in seeding throughput, we compare BWA-MEME with two state-of-the-art variants of BWA-MEM, BWA-MEM2 and ERT. Note that we cannot compare with LISA and Sapling because they do not provide complete seeding. Figure 2.6 (a) shows the average seeding throughput of BWA-MEM2, ERT, and BWA-MEME for the 16 short read datasets. The seeding throughput of each alignment software in the figure is normalized with respect to the seeding throughput of BWA-MEM2. The error bars represent the standard deviation of the normalized seeding throughput. BWA-MEME achieves average 3.32x speedup over BWA-MEM2 and average 1.72x speedup over ERT. This is because algorithms in BWA-MEME process exact matches in more memory efficient and cache-friendly manner. Due to its efficient design, BWA-MEME completes the job with 4.60x fewer number of instructions, 8.77x fewer memory accesses, 2.21x fewer last-level cache (LLC) misses, and 4.41x less size of data fetched per read, as shown in Figure 2.6 (b), (c), (d), and (e), respectively.
Implications on alignment throughput. Figure 2.7 compares the end-to-end alignment throughput of BWA-MEME and memory requirement. BWA-MEME achieves up to 1.42x and 1.12x speedups over BWA-MEM2 and ERT. Note that the seeding phase accounts for an average of 50% of the runtime in BWA-MEM2. Our seeding algorithm dramatically enhances the seeding throughput. As a result, the seeding phase accounts for 29.9% in BWA-MEME as shown in Figure 2.8.
Comparing BWA-MEME with other alignment algorithms We analyze the alignment throughput, memory usage, and the variant calling results of BWA-MEME and other state-of-the-art alignment software. The competitors include Bowtie2 v2.4.4 [39], BWA 0.7.17 [42], BWA-MEM2 v2.2.1 [71], BWA-MEME, STAR 2.7.9a [17], Minimap2 2.23 [44], and Whisper2 [16]. The STAR aligner is used for RNA-seq, but we added it as a competitor to compare the alignment throughput. Figure 2.10 shows the alignment throughput of each aligner given 12 threads and 48 threads for each single-end alignment and paired-end alignment. The alignment throughput of Whisper2 for 48 threads is not presented in the figure because Whisper2 was not optimized for the case using 48 threads. BWA-MEME showed the highest single-end alignment throughput compared to Bowtie2, Whisper2, and Minimap2. This is due to the efficient seeding algorithm of BWA-MEME. Although the single-end alignment throughput of BWA-MEME and STAR is similar given 12 threads, this is because of the difference in the alignment algorithm. We show in the following section that Exact-MEME is faster than the MMP search algorithm of STAR. In the paired-end alignment, Minimap2 and Whisper2 showed higher alignment throughput compared to BWA-MEME given 12 threads. This is because, BWA-MEME, BWA-MEM2, and BWA use significantly more time in the mate rescue step compared to other alignment software. We also investigated in 150 bp short reads (ERR3239284) and the result can be found in Supplementary Material.

We compared the accuracies of variant calling to compare the mapping quality of each aligner. The results of variant calling with reference to the human genome NA12878 (Ref HG19) can be found in Supplementary Material. We used Strelka2 variant caller [31] and the high confidence set of variants from the Genome In a Bottle (GIAB) [80]. BWA-MEME, Whisper2, and Minimap2 showed similar recall, precision, and F1-score, while Bowtie2 showed noticeably lower recall in SNPs and indels compared to other aligners. Note that BWA-MEME, BWA, and BWA-MEM2 have identical results of variant calling as the alignment results were identical.



(a) single-end alignment (b) paired-end alignment

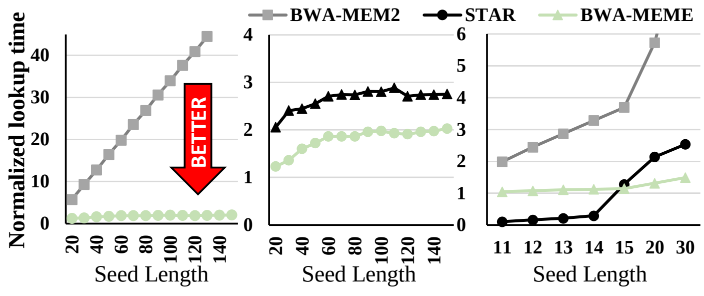
Figure 2.10: Read alignment throughput measured in 101 bp short reads (ERR194147). All aligners were executed with the default option.

	Bowtie2	BWA	BWA-MEM2	BWA-MEME	STAR	Minimap2	Whisper2
12 threads	6.4	13.6	26.4	123	29.6	13.8	23.4
48 threads	6.4	22.3	64.3	153	36	14.3	-

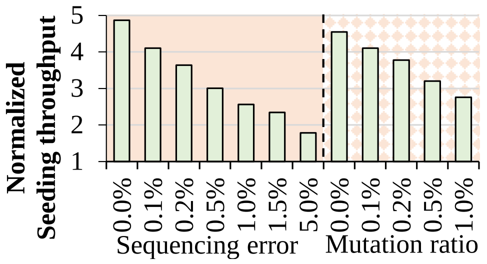
Table 2.1: Maximum memory usage during single-end alignment

2.4.3 Performance Benefit of P-RMI

Comparison with machine-learning based methods. P-RMI contributes to higher seeding throughput because it provides accurate prediction and a small error bound compared to the original RMIs. Figure 2.9 (a) and (b) compare the number of last mile search and seeding throughputs between using P-RMI and original RMIs in BWA-MEME. The seeding throughput is normalized by the seeding throughput of BWA-MEM2. For this evaluation, we use the first 10 million short reads in ERR3239276 from 1,000 Genomes Project Phase 3. To accurately measure the effect of choosing the RMI structure, we exclude the acceleration by lookup result reuse. For 2-layer and 3-layer RMI, we use the linear spline model in the leaf models and the otherwise linear regression model which showed the best performance in the RMI optimizer [55]. In all cases, the number of leaf models is fixed to 2^{28}. For a fair comparison of 3-layer RMI and P-RMI, we use 48,047,097 models in the second layer of 3-layer RMI which is the same number of models used in the additional layer of P-RMI. We apply a binary search when an error bound is provided and an exponential search when an error bound is not provided. As shown in Figure 2.9 (a) and (b), using an exponential search generally requires more last mile search compared to using a binary search. The 3-layer RMI makes accurate predictions compared to the 2-layer RMI, however, it incurs more CPU cache misses while inferencing the models and performing the last mile search. Therefore,



(a) Normalized seed lookup time against seed length



(b) Normalized seeding throughput against sequencing error and mutation ratio

Figure 2.11: **P-RMI analysis**

using the 2-layer RMI with error bound outperforms the 3-layer RMI in seeding throughput. P-RMI combines the two advantages of using a binary search within the error bound and higher prediction accuracy of 3-layer RMI. Hence, P-RMI successfully outperforms the existing RMIs in seeding throughput, at most 29.3%. The comparison of P-RMI and other state-of-the-art learned-index structures [32], including Sapling, can be found in Supplementary Material.

Comparison with traditional-index based methods. We benchmark the seed lookup time using the Exact-MEME algorithm of BWA-MEME, maximal mappable prefix (MMP) search algorithm of STAR [17], and FM-index of BWA-MEM2 given various seed lengths. Note that the MMP is identical to the LEM. We ported the MMP search algorithm of STAR that utilizes the L-mer hash table, as described in Supplementary Material. Figure 2.11 (a) shows the impact of seed length on the seed lookup time of BWA-MEME, STAR, and BWA-MEM2. For this experiment, we used 10 million synthetic short reads [40] for every length. The seed lookup time of BWA-MEM2 significantly increases with regard to the seed length. This is because the number of memory accesses linearly increases as the seed length increases. However, BWA-MEME uses a constant number of memory accesses for all seed lengths. Therefore the seed lookup time does not increase much compared to BWA-MEM2. BWA-MEME is up to 77% faster than the MMP search algorithm of STAR for seeds longer than the L-mer size (15 is used for L, resulting in 2^{30} intervals of suffix array and 7.15 GB in size). For the seeds with length shorter than 15, the MMP search algorithm of STAR is much faster since it finds a seed position with single memory access using the L-mer hash table. The seed lookup time of BWA-MEME and STAR gradually increases along with the seed length between 15 to 60. This is because of the varying search difficulties. For example, there are more duplicate seeds with length 15 than seeds with length 60. We also implemented the SMEM search algorithm (see pseduo algorithm in Supplementary Material) using the MMP search algorithm of STAR and compared the throughput. Figure 5.2 shows that the SMEM search throughput of BWA-MEME is 53.4% higher than that of STAR. More details about the experiment and the results can be found in Supplementary Material.

2.4.4 Robustness of BWA-MEME

Effect of mutation and sequencing error. BWA-MEME outperforms BWA-MEM2 given various mutation ratios in the reference DNA or sequencing error rate of the short reads. Figure 2.11 (b) shows the seeding throughput of BWA-MEME in varying sequencing errors and mutation ratios. We used 10 million of 200 length synthetic short reads [40]. The seeding throughput is normalized by that of BWA-

MEM2. First, the mutation ratio was is fixed to 0.1% to compare the seeding throughput with varying sequencing error rates in the short reads. Next, the sequencing error rate is fixed to 0.1% to compare the seeding throughput with varying mutation ratios of reference. The seeding throughput of BWA-MEME degrades as error increases because the large acceleration comes from processing a long exact matching query with fewer operations. However, BWA-MEME still outperforms BWA-MEM2 in all cases.

Speedup in various reference genomes We now explore how BWA-MEME performs in various reference genomes. Table 2.2 shows the length of the genome, the maximum error, the average error, the seeding throughput speedup of BWA-MEME over BWA-MEM2, and the number of leaf models used in 5 different reference genomes. We used 10 million 101 bp synthetic short reads from each reference genomes to evaluate the speedup. The mutation ratio and the sequencing error rate were fixed to 0.1%. As shown, BWA-MEME is 2.91x-4.10x faster than the BWA-MEM2 for all reference genomes.

Reference genome	Length of genome	Maximum log2 error	Average log2 error	Speedup	Log2 number of models
A	100M	12.6	3.8	3.53	24
B	1679M	22.31	5.08	2.91	28
C	2728M	20.72	5.02	3.22	28
D	3153M	18.6	5.18	3.12	28
E	4915M	16.55	7.79	4.10	28

Table 2.2: P-RMI model analysis in various reference genomes. In each reference genome, the seeding throughput speedup of BWA-MEME over BWA-MEM2 is presented. (A: Caenorhabditis elegans, B: Zebrafish, C: Mouse, D: Human, E: Hordeum Vulgare)

2.4.5 Memory Trade-off Design of BWA-MEME

We demonstrate BWA-MEME is capable of meeting various memory constraints. Figure 2.7 shows the memory requirement and alignment throughput of each variant of BWA-MEME. BWA-MEME uses 118 GB of memory, which consists of P-RMI, suffix array (SA), 64-bit suffixes (32 characters from §2.3.2), and inverse suffix array (ISA). BWA-MEME provides two options that selectively load index data-structures to use less memory space, which comes with a tradeoff in throughput. The first option is to exclude ISA used for lookup result reuse, which brings down the memory requirement to 88 GB. The seeding throughput slightly degrades over the full-mode BWA-MEME, but it still outperforms BWA-MEM2 and ERT, achieving 2.69x and 1.33x speedups in seeding and alignment throughput over BWA-MEM2. The second option excludes both the ISA and the 64-bit suffixes, BWA-MEME uses only the P-RMI and suffix array to process seeding. The memory requirement goes down to 38 GB, and BWA-MEME still achieves 1.71x and 1.23x speedups in seeding and alignment throughput over BWA-MEM2, which is similar to those of ERT.

2.4.6 Additional Results

Other features of BWA-MEME, including the time of index construction, the time of P-RMI training, the index size, the index loading time, the performance of multiple threads, the running time of BWA-MEME in various coverages, were also investigated. Refer to Supplementary Material for these additional results.

Chapter 3. Generalizing deep learning based variant caller with domain adaptation and semi-supervised learning

Detection of genomic variants, such as single-nucleotide polymorphisms (SNPs), short insertion-deletion polymorphisms (referred to as INDELs) and structural variations (SVs), is critical in medical genetics as well as in population genetics and functional genomics. To this end, diverse DNA sequencing platforms have been released in the past two decades [57, 21, 5, 13], and the development of more efficient/accurate variant detection algorithms is one of the topmost interests in the bioinformatics field [43, 80, 82, 6].

The generalizability of variant callers is an important factor [58, 38, 63, 78] as DNA sequencing datasets exhibit diverse background error profiles depending on the sequencing method. These factors include, but not limited to, a few experimental factors, such as the source of samples [29, 70], library preparation methods from multiple vendors [18], the composition of sequences including read length and coverage, and the sequencer platform. Classically, to filter out false positive calls while retaining true variants, many studies or variant callers set up rule-based criteria and finally their own best practice conditions [30, 34]. Afterward, deep learning (DL) based methods, such as DeepVariant [58], were introduced, which outperformed non-DL-based methods in terms of accuracy and generalizability. More recently, research in variant calling has focused on improving the accuracy and efficiency through introducing additional features to the input data [3, 56, 63, 28, 37, 10, 60] or changing the model architecture [52, 53, 60], as well as designing a new variant calling pipeline [63, 78, 35]. However, their reliance on supervised learning still poses a challenge for generalizing to sequencing methods with different error profiles, requiring large amounts of labeled data that demand expert human resources to obtain [79, 81, 82, 6, 74]. Moreover, even a subtle difference in error profiles of the sequencing data that are unseen during training, such as those due to minor changes in the sequencing methods, can challenge the robustness of DL-based variant caller (DVC) and degrade the variant calling accuracy [37, 28].

Here we introduce a new perspective, framing the challenges of the robustness and generalizability in DVC for a sequencing method of interest as domain adaptation and semi-supervised learning problems. We train DVC using labeled datasets and easily obtainable unlabeled datasets from a sequencing method, each considered as a distinct domain. The labeled datasets establish our source domain, while the unlabeled or partially labeled datasets from a sequencing method of interest form our target domain. The generalization of DVC to a sequencing method of interest thus unfolds into solving two problems: (1) if only unlabeled datasets are available from the sequencing method of interest, it can be viewed as an unsupervised domain adaptation (UDA) problem; and (2) if partially labeled datasets are obtainable from the sequencing method of interest, it can be seen as a semi-supervised domain adaptation (SSDA) problem.

In this chapter, we present RUN-DVC, the first semi-supervised training approach for DVC that addresses the above UDA and SSDA problems. In essence, RUN-DVC learns error profiles from unlabeled datasets of a sequencing method of interest using two training modules. First, RUN-DVC employs consistency training, a semi-supervised training technique, making the model generalize well on unlabeled data with unseen error profiles by propagating label information from labeled to unlabeled data. Second, RUN-DVC integrates random logit interpolation, a domain adaptation technique, aiding label propagation by reducing domain discrepancy between source and target domains that arise from varying error profiles.

We evaluate RUN-DVC in comparison with the supervised training approach on generalization scenarios using nine sequencing methods comprising 33 publicly available real-world DNA sequencing datasets [6, 38, 82]. Under UDA scenarios using short-read datasets from Illumina and BGI platforms, RUN-DVC notably increased the variant calling accuracy, enhancing SNP F_1-score and INDEL F_1-score by up to 6.40 %p and 9.36 %p respectively. This demonstrates that RUN-DVC improves the robustness of DVC by learning sequencing error profiles from unlabeled datasets specific to the target domain sequencing method. Moreover, we show the broad applicability of RUN-DVC by applying it to long-read sequencing platforms including Pacific Biosciences (PacBio) and Oxford Nanopore Technology (ONT) sequencing platforms. Finally, we demonstrate that RUN-DVC could match the variant calling accuracy of the supervised training approach using merely half of the labeled datasets in a semi-supervised domain adaptation (SSDA) scenario. This result showcases the potential of RUN-DVC to facilitate a label-efficient generalization of DVC to various sequencing methods, serving as a key advantage in practical deployment.

3.1 Results

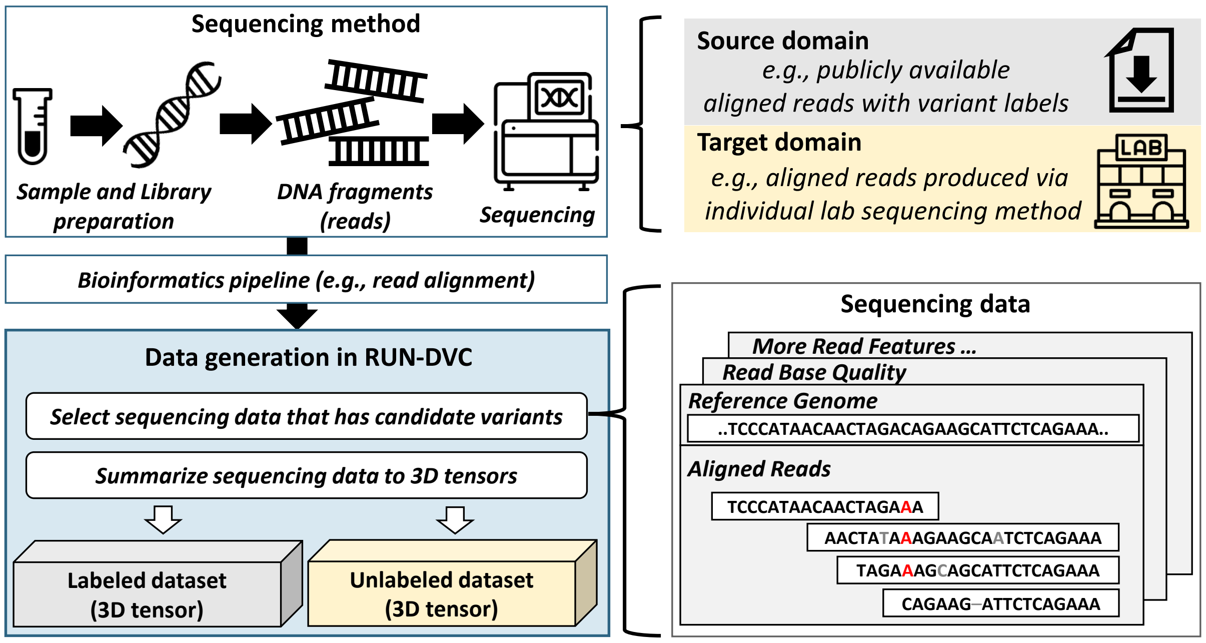
3.1.1 Overview of RUN-DVC.

We developed RUN-DVC, a semi-supervised training approach for DVCs that improves robustness and generalizability to a specific sequencing method of interest by learning error profiles from unlabeled data of the sequencing method. RUN-DVC optimizes the DVC model through a novel loss function that combines unsupervised and supervised losses from two training modules. First, the unsupervised loss is derived from the semi-supervised learning (SSL) module that incorporates consistency training within unlabeled data. This approach uses two differently augmented versions of the same unlabeled data for training, with one serving as a model input and the model prediction on the other as a pseudo-label. By minimizing discrepancies between these, the model propagates labels from labeled data to similar unlabeled data, allowing the model to generalize well from known data to unlabeled data with different error profiles. Second, the supervised loss is derived from the random logit interpolation (RLI) module that aligns embeddings of the source and target domains. The idea is to infer the model twice every iteration with two batches: 1) a batch solely consisting of source domain data and 2) a combined batch of both source and target domain data. Subsequently, the outputs of the source domain data from both batches are interpolated and compared to the ground truth labels. This promotes the model prediction to be accurate, despite fluctuations in batch normalization layer statistics across the source and target domains, thus resulting in a model that better represents both domains.

These training modules complement the supervised training approach without changing the DVC model architecture, positioning RUN-DVC as an alternative training solution for DVCs. Fig. 3.1 illustrates the schematic overview of RUN-DVC.

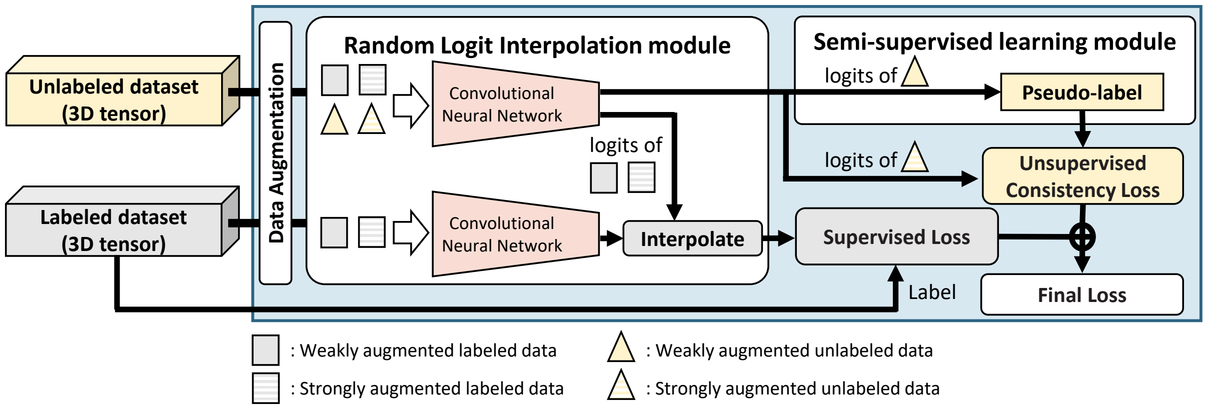
3.1.2 Datasets.

We used 33 publicly available sequencing datasets from GIAB [80, 79, 82, 81] (Genome in a Bottle), the Human Pangenome Reference Consortium [75], and Google [6]. We leveraged the version 4.2.1 GIAB truth variant sets [74] as our ground-truth label for analysis. Sequencing datasets include human samples, NA12878/HG001 from 1000 Genomes [11], and two trio families (HG002-HG003-HG004 and



Sequencing method
Sample and Library preparation
DNA fragments (reads)
Sequencing
Source domain
e.g., publicly available aligned reads with variant labels
Target domain
e.g., aligned reads produced via individual lab sequencing method
LAB
Bioinformatics pipeline (e.g., read alignment)
Data generation in RUN-DVC
Select sequencing data that has candidate variants
Summarize sequencing data to 3D tensors
Labeled dataset (3D tensor)
Unlabeled dataset (3D tensor)
Sequencing data
More Read Features ...
Read Base Quality
Reference Genome
..TCCCATAACAACTAGACAGAAGCATTCTCAGAAA..
Aligned Reads
TCCCATAACAACTAGAAA
AACTATAAAGAAGCAATCTCAGAAA
TAGAAAGCAGCATTCTCAGAAA
CAGAAG–ATTCTCAGAAA

(a) The overview of dataset generation for RUN-DVC. RUN-DVC addresses domain adaptation and semi-supervised learning problems, treating each sequencing method as a distinct domain. The source domain consists of labeled sequencing datasets, for example, publicly accessible sequencing data, while the target domain encompasses unlabeled or partially labeled sequencing datasets from a different sequencing method. A bioinformatics pipeline processes sequencing data through stages including read alignment, sorting, and Indel realignment. Subsequently, sequencing data possessing candidate variants are selected and converted into 3-dimensional tensors. Depending on the availability of variant labels, these datasets are then categorized as either labeled (with variant labels) or unlabeled (without variant labels).



Unlabeled dataset (3D tensor)
Labeled dataset (3D tensor)
Data Augmentation
Random Logit Interpolation module
Convolutional Neural Network
logits of
Convolutional Neural Network
Interpolate
Semi-supervised learning module
logits of
Pseudo-label
logits of
Unsupervised Consistency Loss
Supervised Loss
Label
Final Loss
: Weakly augmented labeled data
: Strongly augmented labeled data
: Weakly augmented unlabeled data
: Strongly augmented unlabeled data

(b) Illustration of the training process of RUN-DVC. RUN-DVC trains the DVC model by optimizing the sum of the supervised and unsupervised loss computed using labeled and unlabeled datasets. During each training iteration, both the random logit interpolation module and the semi-supervised learning module are used to compute the supervised and unsupervised loss, respectively. The supervised loss for labeled datasets is computed by comparing the logits obtained from the random logit interpolation module with the corresponding labels in the dataset. The unsupervised loss is computed by comparing the pseudo-label with the output of the CNN model on the strongly augmented version of the same unlabeled data. We use the class with the maximum value among predictions from the weakly augmented data as the pseudo-label. The details of the process can be found in the Methods section and the pseudocode of the training procedure is presented in the Supplementary Algorithm. 8.

Figure 3.1: Overview of RUN-DVC workflow.

HG005-HG006-HG007) from participants in the Personal Genomes Project [7]. The summary of the sequencing datasets used in experiments is organized in Table 3.1 and its corresponding web links to

Identifier	Sequencing method (number of data)	Sample number
I-Source	Illumina NovaSeq PCR-free 30x data (20 million)	5,6,7
I-A	Illumina NovaSeq PCR-positive 30x data (23 million)	1,2,3,4
I-B	Illumina Hiseq2500x 21-30x data (86 million)	1,2,3,4
I-C	BGISEQ-500 40x data (21 million)	1,2,3,4
I-D	Illumina HiseqX PCR-positive 30x data (20 million)	1,2,3,4
P-Source	PacBio HiFi Sequel II 30x and 50x data (26 million)	1,2,5,5*
P-A	PacBio HiFi Sequel I 30x data (15 million)	2,2*
O-Source	ONT SUP mode Guppy v5.0.14 50x data (34 million)	2,3,4,5
O-A	ONT HAC mode Guppy v5.0.14 50x data (34 million)	2,3,4,5

Table 3.1: Overview of datasets used. Dataset identifier, sequencing method, number of data, and human sample numbers of sequencing datasets are provided. Note that P-A consists of two sequencing datasets from the same sample. The sample number corresponds to the suffix of the sample name (e.g., 1 stands for HG001).

details of sequencing methods are organized in Supplementary Table. 5.14.

Each of the datasets I-A, I-B, I-C, and I-D, exhibits distinct error profiles attributable to the different sequencing methods employed. Specifically, I-A and I-Source were sequenced using the same Illumina NovaSeq sequencer in the same institution, but I-A utilized PCR amplification in the library preparation step. I-D, processed in the same institution as I-Source, employed PCR amplification in the library preparation process and sequenced by an older sequencing platform, Illumina HiseqX. Furthermore, I-B was produced by different institution (10x Genomics) and through an even older Illumina Hiseq2500x platform, while I-C was generated by BGI with the BGISEQ-500 machine. Turning to the PacBio HiFi datasets, P-Source and P-A were obtained from the Sequel II and Sequel I systems, respectively. Finally, O-Source and O-A datasets are ONT sequencing datasets, with base calls made via super accuracy (SUP) and high accuracy (HAC) modes of Guppy 5.0.14 on the ONT PromethION platform, respectively.

3.1.3 Baseline methods.

BaselineBN[8] represents a supervised training strategy employed in existing DVCs supplemented with a minimal domain adaptation technique. This approach trains on labeled datasets from the source domain and also utilizes unlabeled datasets from the target domain to update batch norm statistics, fostering domain alignment[61, 62]. Another approach, referred to as *Full-label*, illustrates the maximum accuracy attainable by the model when trained on fully labeled datasets from the target domain. Data augmentation techniques are utilized during training for both *BaselineBN* and *Full-label*.

3.1.4 Performance on short-read sequencing platforms.

We compare the variant calling performance of RUN-DVC against *BaselineBN* and *Full-label* under four UDA settings using short-read datasets. Specifically, we train RUN-DVC and *BaselineBN* using labeled datasets from I-Source as the source domain and unlabeled datasets from I-A, I-B, I-C, and I-D as the target domains. The *Full-label* is trained using labeled datasets specific to each of I-A, I-B, I-C, and I-D. For the purposes of evaluation, the HG003 sample is excluded from all training datasets. In addition, we provide the accuracy of state-of-the-art methods, Clair3 v1.0.0 [78] and DeepVariant v1.5.0 [58], for validation of the problem and implementation. DeepVariant uses 52 sequencing datasets from various sequencing methods (including I-Source, I-A, and I-D) that account for 815,200,320 training samples, whereas Clair3 uses 12 PCR-free sequencing datasets (comprising I-Source dataset).

Fig. 3.2(a) shows the precision-recall curve and the best F_1 scores achieved by selecting the optimal quality score threshold for each method. The Precision, Recall, and F_1 score of variant calling by each method with default setting can be found in Supplementary Table. 5.15 and Table. 5.16. RUN-DVC demonstrated superior performance over *BaselineBN* across all datasets, utilizing only unlabeled datasets from the target domain. For SNP calling, the performance of *BaselineBN* is only slightly lower than *Full-label* on the I-A, I-C, and I-D datasets. However, there was a significant decrease of 7.91 percentage points (%p) in the SNP F_1 score on the I-B dataset. On the other hand, RUN-DVC performed better than *BaselineBN* by improving the score by 6.40 %p, thus reducing the performance drop compared to *Full-label* to only 1.51 %p. Regarding INDEL calling, *BaselineBN* exhibited substantial performance degradation across all datasets compared to *Full-label*. However, RUN-DVC significantly reduced the performance disparity between *BaselineBN* and *Full-label*, outpacing *BaselineBN* by 2.94 %p, 9.36 %p, 1.69 %p, and 4.21 %p on the I-A, I-B, I-C, and I-D datasets, respectively. These results highlight the ability of RUN-DVC to enhance the robustness of DVC, thereby enhancing variant calling performance in the target domain sequencing method.

State-of-the-art DVCs, such as Clair3 and DeepVariant, displayed discernible performance reductions on certain datasets. Specifically, DeepVariant's performance faltered on I-B and I-C datasets, while Clair3's performance diminished across all datasets. DeepVariant showed no performance degradation in SNP and INDEL calling on the I-A and I-D datasets, which are part of the training dataset of DeepVariant. However, relative to the *Full-label* method, it registered a decrease in the SNP F1 score by 3.68 %p on I-C and a decrease in the INDEL F1 score by 12.90 %p and 2.37 %p on I-B and I-C datasets, respectively. In the case of Clair3, it experienced a 21.99 %p decrease in the SNP F_1 score on I-B and a decline in INDEL F_1 scores by 6.43 %p, 22.16 %p, 2.86 %p, and 6.97 %p on the I-A, I-B, I-C, and I-D datasets, relative to *Full-label*. These observations demonstrate that state-of-the-art DVCs may struggle to maintain robustness when confronted with sequencing methods unseen during training.

3.1.5 Performance on long-read sequencing platforms.

In order to demonstrate the broad applicability of RUN-DVC across various sequencing methods, we compared its performance with that of *BaselineBN* on two UDA settings from each PacBio and ONT dataset. Within these scenarios, P-Source and O-Source were designated as the source domains, while P-A and O-A constituted their respective target domains. Due to the limited availability of publicly accessible PacBio Sequel I datasets generated with the same sequencing method, the HG002 sample's unlabeled datasets were used during the training phase, and the same sample was employed for evaluation. We provide the variant calling accuracy of models provided by Clair3 and DeepVariant (PEPPER) for PacBio Sequel II and ONT Guppy5 SUP mode datasets. Details of training datasets used for Clair3 and DeepVariant can be found in Supplementary Material.

Fig. 3.2(b) shows the results. For both PacBio and ONT datasets, RUN-DVC incorporated haplotype information as an additional feature within the input. Furthermore, for the ONT datasets, RUN-DVC employed an input tensor of a different size, supporting a read depth of up to 89 as opposed to a read depth of 55 utilized for PacBio and short-read datasets (refer to Supplementary Material for more details on input tensor). Nevertheless, RUN-DVC outperformed *BaselineBN* on P-A and O-A datasets even when using the same set of hyperparameters that was used in short-read datasets. These results confirm the versatility of RUN-DVC across diverse sequencing methods ranging from short reads to long reads that use different input sizes and input features.

3.1.6 RUN-DVC effectively learns sequencing error profiles from unlabeled datasets.

To ascertain whether the observed performance improvement stemmed from the learning of error profiles of the target domain sequencing method, we conducted an additional analysis. This involved contrasting the performance of RUN-DVC with both *BaselineBN* and *Full-label* in relation to genomic contexts. RUN-DVC and *BaselineBN* were trained under two UDA settings using labeled datasets from I-Source as the source domain and unlabeled datasets (excluding the HG003 sample) from I-A or I-B as the target domain. We report the variant-calling performance in the genome's difficult-to-map regions (low-mappability and segmental duplications regions) and low-complexity regions (tandem repeats and homopolymer regions) according to the GIAB v2.0 stratification data.

Fig. 3.3(a) shows the variant calling performance on the HG003 sample of the I-A dataset. *BaselineBN* exhibited notable performance degradation in INDEL calling accuracy only in the low-complexity regions. Specifically, in each tandem repeats region and homopolymers region, RUN-DVC achieved F_1-score 0.9374 and 0.8875 which are 1.07 %p and 5.64 %p higher compared to *BaselineBN* that achieved 0.9267 and 0.8311. For whole regions except for low-complexity regions, RUN-DVC (SNP F_1-score: 0.9937, INDEL F_1-score: 0.9931), *BaselineBN* (SNP F_1-score: 0.9938, INDEL F_1-score: 0.9930), *Full-label* (SNP F_1-score: 0.9939, INDEL F_1-score: 0.9932) achieved similar SNP and INDEL F_1-scores. These outcomes align with the known impact of PCR amplifications, which introduce various artifacts on repetitive DNAs [24]. We conjecture that the absence of these artifacts in the PCR-free I-Source dataset is the reason for the *BaselineBN*'s performance degradation in the tandem repeats and homopolymers regions. Notably, RUN-DVC effectively counteracts this effect, improving variant calling accuracy and confirming its ability to learn error profiles, including artifacts and deletions caused by PCR amplification, from unlabeled datasets.

Fig. 3.3(b) shows the variant calling performance on the HG003 sample of the I-B dataset. RUN-DVC significantly outperforms *BaselineBN* in both SNP and INDEL calling accuracy across all regions. Particularly in regions of low mappability, segmental duplications, tandem repeats, and homopolymers, RUN-DVC surpasses *BaselineBN* with SNP F_1-scores improved by 5.335 %p, 6.619 %p, 6.141 %p, and 3.334 %p, and INDEL F_1 scores by 3.134 %p, 3.208 %p, 1.142 %p, and 10.59 %p respectively. In our analysis, the I-B dataset emerged as the most demanding scenario for generalization, exhibiting the most pronounced performance deterioration for all *BaselineBN*, DeepVariant, and Clair3. This is attributed to its uniquely disparate error profiles and the extensive magnitude of errors dispersed throughout the entire genomic regions. Despite this, RUN-DVC excelled, demonstrating its capacity to learn and adapt to significantly different error profiles even in a demanding variant calling case.

Finally, we assess the INDEL calling performance of RUN-DVC for different INDEL sizes as shown in Fig. 3.3(c). RUN-DVC consistently surpasses the performance of *BaselineBN*, underscoring its efficacy for a range of INDEL sizes. In summary, RUN-DVC's strength stems from its ability to learn a multitude of error profiles from unlabeled datasets, showcasing versatility that is not confined to specific regions or types of variants.

3.1.7 Ablation study.

To elucidate the individual contributions of the two training modules of RUN-DVC to the overall performance, we perform an ablation study. In this experiment, we compare *BaselineBN*, "RUN-DVC w/o RLI" that was trained solely via the semi-supervised learning module without the use of random logit interpolation module, and the full RUN-DVC. All methods are trained on two UDA settings using labeled datasets from I-Source and unlabeled datasets (excluding the HG003 sample) from I-A or I-D datasets.

Fig. 3.4(a) shows the results of RUN-DVC, "RUN-DVC w/o RLI", and *BaselineBN* on HG003 sample of I-A and I-D datasets over 4 independent runs. In the case of the I-A dataset, RUN-DVC delivered the highest accuracy (SNP F_1-score: 0.9935, INDEL F_1-score: 0.9387), outperforming "RUN-DVC w/o RLI" (SNP F_1-score: 0.9934, INDEL F_1-score: 0.9243) and *BaselineBN* (SNP F_1-score: 0.9934, INDEL F_1-score: 0.9069). For the I-D dataset, while RUN-DVC demonstrated superior accuracy (SNP F_1-score: 0.9938, INDEL F_1-score: 0.9413) as compared to *BaselineBN* (SNP F_1-score: 0.9927, INDEL F_1-score: 0.8697), "RUN-DVC w/o RLI" exhibited a downturn in SNP calling accuracy (SNP F_1-score: 0.9290, INDEL F_1-score: 0.9181). Furthermore, the variant calling performance on the I-D dataset by "RUN-DVC w/o RLI" displayed remarkable instability.

Fig. 3.4(b) shows the validation loss on the target domain over the course of training iterations for *BaselineBN*, "RUN-DVC w/o RLI", and RUN-DVC. Remarkably, the validation loss of "RUN-DVC w/o RLI" on the I-D dataset exhibited substantial fluctuations and failed to converge to a favorable solution, unlike RUN-DVC. This suggests the crucial role of the RLI module in aligning domains and ensuring stable learning on the target domain.

Overall, these observations underscore the vital role of the SSL and RLI modules in the efficacy of RUN-DVC. It is also pertinent to note that, we proposed multiple data augmentation strategies for DNA sequencing datasets that are also crucial for the overall performance of RUN-DVC. Further details and analysis on these data augmentation strategies can be found in Supplementary Fig. 5.6.

3.1.8 Using RUN-DVC for foreseeing when DVC fails to generalize.

To demonstrate the utility of RUN-DVC as an indicator for generalization failure of DVC in a sequencing method of interest, we examined the variant calling outcomes of RUN-DVC and the supervised trained model. This exploration entails analyzing disparities in quantities, focusing on both biallelic changes in SNPs and INDELs of varying sizes, and comparing these between RUN-DVC and *BaselineBN* under the UDA setting. The results are shown in Supplementary Fig. 5.5.

Our evaluation of the number of biallelic base alterations offers meaningful insights into the potential generalization failure of *BaselineBN* in relation to SNP identification performance. Notably, we did not observe any significant discrepancy in the number of biallelic base alterations in the I-A, I-C, and I-D datasets. However, a noticeable difference surfaced in the I-B dataset, in which a large performance degradation in SNP calling of *BaselineBN* was observed.

As for INDEL calling accuracy, the number of INDEL variants by size serves as a key indicator of potential generalization failures in *BaselineBN*. Across the I-A, I-B, I-C, and I-D datasets, the total number of disparities for each INDEL size between RUN-DVC and *BaselineBN* was 34,528, 222,193, 15,497, and 47,324 respectively, which seems to correlate with the degree of performance degradation.

In addition, we also assess the distribution of genotypes as shown in Supplementary Fig. 5.4(b). Intriguingly, we observe that the variant calling quality scores of *BaselineBN* could either exceed or fall short of those from RUN-DVC and *Full-label*, despite a considerable decrease in variant calling accuracy. Therefore, in light of these findings, we suggest that a comparison of both the quantities of biallelic base changes and the numbers of INDELs of varying sizes, produced by RUN-DVC and the model trained through the supervised method, could serve as effective indicators for instances of generalization failure of DVCs.

3.1.9 RUN-DVC enables label-efficient generalization of DVC to a sequencing method of interest.

To assess the generalizability of RUN-DVC, we conducted a comparison of the validation loss and variant calling accuracy between RUN-DVC and *BaselineBN* under varying quantities of target domain labeled datasets. The results over 2 independent runs are shown in Fig. 3.5 (the numbers can be found in Supplementary Table. 5.20). This evaluation was conducted within two SSDA scenarios, utilizing I-Source as the source domain and I-A and I-B as target domains. The validation loss was determined by evaluating the performance of both methods against identical validation datasets, comprising 5% of the total target domain datasets that were excluded from the training phase. The variant calling accuracy was measured on the HG003 sample that was excluded from the training phase. Importantly, RUN-DVC consistently surpassed the variant calling performance of *BaselineBN* given the same amount of labeled datasets. Additionally, in terms of validation loss, RUN-DVC exhibited almost double the label-efficiency compared to *BaselineBN*. These findings underscore the superior generalizability of RUN-DVC, manifested by its enhanced ability to effectively propagate label information from the labeled datasets to unlabeled datasets.

3.2 Discussion

We present RUN-DVC, the first semi-supervised training approach for DVC that improves the robustness and enables label-efficient generalization to a target sequencing method. RUN-DVC leverages the consistency training and random logit interpolation techniques, allowing it to learn error profiles from unlabeled data of the target sequencing method using the knowledge obtained in labeled data. These training techniques are complementary to the supervised training approach, positioning RUN-DVC as an alternative training solution for existing DVCs.

In the UDA experimental setup, RUN-DVC exceeds the performance of the supervised training approach in the accuracy of variant calling. This signifies the potential of RUN-DVC to enhance the robustness of DVC against specific sequencing methods by learning error profiles in unlabeled datasets. Furthermore, we have established the applicability of RUN-DVC to long-read sequencing platforms such as PacBio and ONT. This demonstrates the extensive adaptability of RUN-DVC to diverse sequencing methods, encompassing both short reads and long reads. Looking forward, we believe that incorporating RUN-DVC into existing DVCs [52, 53, 56, 37, 60, 3, 58, 28, 10, 72] will expand their utilization across a variety of sequencing methods, leveraging the potential of unlabeled data.

One of the prominent applications of RUN-DVC is the generalization of DVC across various species. RUN-DVC operates without making specific assumptions regarding the sequencing method, including the species. This makes it possible to apply RUN-DVC to sequencing data from species other than humans. However, our evaluation was constrained by the limited availability of sequencing datasets using the same sequencing method, and the lack of benchmark datasets across different species, preventing a comprehensive assessment in this regard.

We have demonstrated that RUN-DVC can be used to foresee the generalization failure of DVC by examining numbers and types of variants produced by both RUN-DVC and the supervised trained model. This can be used to determine whether further training of the DVC model is required to achieve a higher variant calling accuracy in the target sequencing method.

Finally, RUN-DVC outperforms the supervised training method by achieving higher variant calling accuracy with fewer labeled datasets, thus illustrating that RUN-DVC offers a more label-efficient training solution. For individual laboratories employing custom sequencing methods and seeking to establish an accurate variant calling pipeline with DVC, RUN-DVC is expected to be particularly valuable, especially when combined with the retraining solution [1].

We acknowledge several limitations of RUN-DVC. First, the RLI module is applicable only for DVCs that use a CNN model with batch normalization layers, a characteristic that is true for the majority of existing DVCs [78, 63, 3, 53, 52, 58, 28]. Nevertheless, shifting to a different model architecture other than CNN might necessitate the development of new methods to replace the RLI module. Second, we have provided evidence that RUN-DVC, operating on a single hyperparameter setting, is effective across a range of sequencing datasets, outperforming the supervised training method. However, we acknowledge that the application of optimal hyperparameters and data augmentations might drive further enhancements in RUN-DVC's performance (see Supplementary Fig. 5.6 for an ablation study on data augmentations). A third limitation lies within the SSDA setting where the target domain's labeled datasets selection is randomized. We postulate that the integration of active learning strategies could potentially augment RUN-DVC's label efficiency. Lastly, RUN-DVC has yet to be validated for somatic mutation calling—an application with different task requirements and fundamental assumptions. Future investigations should address this by extending the use of RUN-DVC to include somatic mutation callers, particularly in cases where the sequencing datasets exhibit a greater degree of diversity.

3.3 Methods

3.3.1 Evaluation metrics

We used Illumina's Haplotype Comparison Tools [38] (hap.py) and GIAB v4.2.1 truth variant data to benchmark variant-calling results. Hap.py generates three metrics Precision, Recall, and F1-score for each of the categories SNP and Indel, respectively. From the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), hap.py computes the three metrics as Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, and F1-score $= \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$.

3.3.2 Input/output and model architecture

Selecting candidate variants. RUN-DVC selects candidate variants for training data or variant calling using the following algorithm. For each position in the reference genome, all the reads that overlap the position are collected. Next, RUN-DVC selects a position as a candidate variant if two conditions are met: (1) the number of aligned reads surpasses the coverage threshold. (2) the percentage of mismatches between the reference genome and the aligned reads is above the allele frequency threshold.

Input tensor. RUN-DVC takes a 3-dimensional tensor as input for the DNN model, which is structured similarly to an image. For instance, in the case of short reads, the input tensor has a shape of 55 rows, 33 columns, and 7 channels. Specifically, a 3-dimensional tensor comprises information about a candidate variant region. Each channel in the input tensor represents a unique feature of the sequencing data including reference bases, variants in reads, strand information, mapping quality, base quality, the proportion of candidate variants, insertion bases, and phasing information. The columns correspond to positions in the reference genome, while the rows represent individual reads aligned to those positions. A detailed explanation of each channel is provided in Supplementary Material, and an example of an input tensor is shown in Supplementary Fig. 5.8.

Output of the DNN model. During inference, the DNN model performs four classification tasks on the input. These tasks involve predicting four variables: G, Z, L_1, and L_2, where G corresponds to 21 possible genotypes and Z represents the zygosity of the variant. The variables L_1 and L_2 indicate the length of the two variants in the diploid organism, with possible values ranging from -16 to 16. Note that, the values of -16 and 16 for L_1 and L_2 represent variants with lengths larger than 16. Specifically, the set of possible values for each variable is as follows:

- $G \in$\{ AA, AC, AG, AT, AIns, ADel, CC, CG, ..., DelDel\}
- $Z \in$ \{Homozygous reference, Heterozygous, Homozygous non-reference\}
- $L_1, L_2 \in \{-16, ..., 16\}$

Constructing labeled training dataset. RUN-DVC constructs a labeled training dataset from the candidate variants using ground truth labels. Specifically, each data point is selected from candidate variants, which consists of pairs of a tensor that summarizes a candidate variant and its corresponding variant label that includes four labels G, Z, L_1, and L_2. N_m true variants and N_r non-variants are selected from candidate variants, in a ratio specified by the target ratio $\gamma = \frac{N_r}{N_m}$. RUN-DVC constructs labeled training dataset using $\gamma = 1$ [78].

Constructing unlabeled training dataset. Different sequencing methods can generate candidate variants with vastly different true variants to non-variants ratios. For instance, the ONT dataset may have 100 times more non-variants than true variants, while the Illumina dataset has a similar number of non-variants and true variants. This results in the ONT dataset having 50 times longer training iterations and sizes compared to the Illumina dataset. However, it has been shown including non-variants beyond a certain number offers marginal improvement in performance [78].

To reduce the number of non-variants in the unlabeled dataset, an RNN model trained on the labeled dataset is used to obtain *confidence scores* for candidate variants that are probabilities of being true variants. The RNN model consists of two bidirectional long short-term memory (Bi-LSTM) layers with 128 and 160 LSTM units, and uses the pileup input proposed by Clair3 (see Supplementary Material for details of pileup input). We observed the RNN model shows high accuracy in filtering non-variants from candidate variants (see Supplementary Table. 5.18 and Table. 5.19 for details). Filtering out non-variants reduces the size of the unlabeled dataset while still including the true variants necessary for training.

The unlabeled training dataset is constructed using all candidate variants when the total number of candidate variants in a single sample is below a threshold value of T. However, if the total number of candidate variants exceeds the threshold, the top T candidate variants are selected based on the *confidence scores*. In our experiments, 80 million is used for the threshold value T. Our approach ensures that the training dataset contains enough true variants for effective training while reducing the size of the dataset and training time by removing non-variants.

Model architecture. RUN-DVC trains a convolutional neural network (CNN) model on 3-dimensional input tensors that each comprises information about a candidate variant region. The CNN model comprises three blocks, each consisting of a basic convolution block followed by a standard residual block. After the blocks, an adaptive max pooling layer is used at the end of the blocks that outputs 512 features. The basic convolution block includes a convolutional block, a batch normalization layer, and a Leaky Relu activation layer, which decreases the width and height of features. The classifier head comprises two dense layers for each task that output 256 and 128 features, respectively. The architecture of the CNN model is presented in Supplementary Fig. 5.7.

3.3.3 Training objectives of RUN-DVC

RUN-DVC solves the unsupervised and semi-supervised domain adaptation problem by leveraging the labeled dataset from the source domain and the unlabeled dataset from the target domain. In the case of the SSDA setting, the labeled samples from the target domain are included in the labeled dataset. RUN-DVC minimizes the loss function $\mathcal{L}$ that consists of two losses: (1) a supervised classification loss $\mathcal{L}_l$ on the labeled sample, (2) an unsupervised classification loss $\mathcal{L}_u$ on the unlabeled sample. The DNN model in RUN-DVC performs four classification tasks, hence, the loss function to be optimized is expressed as:

$$\sum_{t\in\{G,Z,L_1,L_2\}} \mathcal{L}^{(t)} = \sum_{t\in\{G,Z,L_1,L_2\}} (\mathcal{L}_l^{(t)} + \mathcal{L}_u^{(t)}) \tag{3.1}$$

In essence, RUN-DVC optimizes the sum of both supervised and unsupervised loss functions to solve unsupervised and semi-supervised domain adaptation problems. We provide the pseudocode of the training procedure in Supplementary Algorithm. 9.

Weak and strong data augmentations. RUN-DVC adapts the concept of weak and strong augmentation from computer vision. In computer vision, weak augmentation comprises augmentations such as flipping and rotating, whereas strong augmentation includes complex transformations such as changes in color and image structure. For RUN-DVC, weak augmentations consist of subsampling and vertical shifting, while strong augmentations additionally employ feature distortions. During training, the augmentations are applied to batches of data from both labeled and unlabeled datasets at each iteration. This is in contrast to Clair3 and DeepVariant, where augmentations are applied before constructing the training dataset. Applying augmentations during training enables the same data to be augmented in a different way during each epoch of training, which enhances the model's resilience against variability. Further details regarding weak and strong augmentations can be found in the *Data Augmentations* subsection.

Notation. We represent a batch of labeled samples from the source domain by B_l, and a batch of unlabeled examples from the target domain by B_u. Each batch is composed of samples that are weakly and strongly augmented, which we denote respectively by $B_{l,weak}$, $B_{l,strong}$, $B_{u,weak}$, and $B_{u,strong}$.

Supervised classification loss is obtained using random logit interpolation. RUN-DVC addresses the domain adaptation problem by using random logit interpolation [8] (RLI) during training. RLI interpolates the logits (the outputs of the last layer in the DNN model before the activation layer) from both source and target domains to generate more representative batch statistics in batch normalization layers. During each iteration of training, the DNN model is inferred twice to obtain two logits, O_c and O_l. O_c is computed using a batch $\{B_l, B_u\}$ that includes both labeled and unlabeled datasets, whereas O_l is computed using a batch $\{B_l\}$ that only contains the labeled dataset:

$$O_c = f(\{B_l, B_u\}; \theta) \quad O_l = f(\{B_l\}; \theta) \tag{3.2}$$

Importantly, the batch normalization layers in the DNN model are updated during the computation of O_c, whereas they remain fixed during the computation of O_l. This ensures that model parameters and batch normalization layers are adapted to the target domain during training.

The supervised classification loss $\mathcal{L}_l$ is computed using cross entropy loss between ground truth label Y and the final logits O_{RLI}, where the final logits O_{RLI} for B_l labeled samples are obtained by randomly interpolating O_c and O_l, as shown below:

$$O_{RLI} = \sum_{i=0}^{\|B_l\|} \alpha^{(i)} O_c^{(i)} + (1 - \alpha^{(i)}) O_l^{(i)} \mathcal{L}_l = \frac{1}{\|B_l\|} \sum_{i=0}^{\|B_l\|} H(Y_s^{(i)}, O_{RLI}^{(i)}) \tag{3.3}$$

where $\alpha \in \mathbb{R}^{B_l \times k}$ is a vector of random values drawn from uniform distribution $\mathcal{U}^{B_l * k}(0, 1)$, function f denotes the DNN model that outputs k logits for each sample, and $H(Y, P)$ denotes the cross-entropy between label Y and logits P.

Unsupervised classification loss is obtained via semi-supervised learning. The semi-supervised learning module employs the consistency training technique [64] to learn from the unlabeled datasets. At each iteration, the model is trained to minimize the loss between the pseudo-labels generated from the weakly augmented data and the output generated from the strongly augmented data. This improves the model's robustness against input data variability as the outputs of weakly and strongly augmented data are encouraged to be consistent. Also, the label information is gradually propagated from the labeled data to the unlabeled data [77].

The unlabeled samples' logits O_u is obtained from the logits O_c of combined batches as follows:

$$O_u = \sum_{i=\|B_l\|}^{N} O_c^{(i)} \quad \text{where } N = \|B_l\| + \|B_u\| \tag{3.4}$$

O_u contains the logits of both weakly and strongly augmented unlabeled samples, denoted as $O_{u,weak}$ and $O_{u,strong}$, respectively:

$$O_{u,weak} = \sum_{i=0}^{\|B_{u,weak}\|} O_u^{(i)} \quad O_{u,strong} = \sum_{i=\|B_{u,weak}\|}^{\|B_u\|} O_u^{(i)} \tag{3.5}$$

To prevent the model from training with inaccurate pseudo-labels during the early stages of training, a mask is used to select confident pseudo-labels. Specifically, a threshold τ is used to identify high-confidence predictions from weakly augmented data and generate masks as follows:

$$Mask(Softmax(O_{u,weak})) = \mathbb{1}(\max(Softmax(O_{u,weak})) > \tau) \tag{3.6}$$

For each ith sample, pseudo label $P^{(i)}$ is generated by selecting the class with the highest logit from $O_{u,weak}^{(i)}$:

$$P^{(i)} = argmax(O_{u,weak}^{(i)}) \tag{3.7}$$

Finally, the unsupervised loss (i.e., consistency loss) $\mathcal{L}_u$ is computed as follows:

$$\mathcal{L}_u = \frac{\mu}{\|B_u\|} \sum_{i=0}^{\|B_u\|} Mask(Softmax(O_{u,weak}^{(i)})) H(P^{(i)}, O_{u,strong}^{(i)}) \tag{3.8}$$

Here, μ represents the weight assigned to the unsupervised loss and $H(P, O_{u,strong})$ denotes the cross-entropy loss between P and the logits of the strongly augmented samples.

3.3.4 Data augmentations

We observed that existing data augmentations [78, 58] are insufficient for preventing overfitting of the model and achieving high performance in RUN-DVC (see Supplementary Fig. 5.6). Therefore, we propose a set of new augmentations and augmentation policies that are tailored for DNA sequencing datasets.

List of data augmentations. In the following, we provide overviews of the augmentations we proposed. Visualized examples of our augmentations are provided in Supplementary Fig. 5.8.

- **Subsampling:** Sub-sampling augmentation drops a specified portion of reads in the 3-dimensional tensor, with the proportion of reads to be removed specified as an input and the selection of reads to be dropped made randomly using a uniform distribution. This helps the DNN model become more robust against variation in the number of reads aligned in the candidate variant region.

- **Shifting and flipping:** Vertical shifting proved effective in preventing overfitting and improving the overall performance of RUN-DVC, while horizontal shifting did not lead to a performance increase. We chose not to use vertical flipping, as it reverses the order of aligned reads, which the model does not encounter at inference. Similarly, horizontal flipping was deemed unsuitable because it has the potential to alter variant labels [66, 38]. For example, in DNA sequencing, variants are often left-aligned, whereas most DNA sequencing processing tools output left-aligned variants. Thus, horizontal flipping would require changing left-aligned variants to right-aligned before flipping, which could alter the variant label.

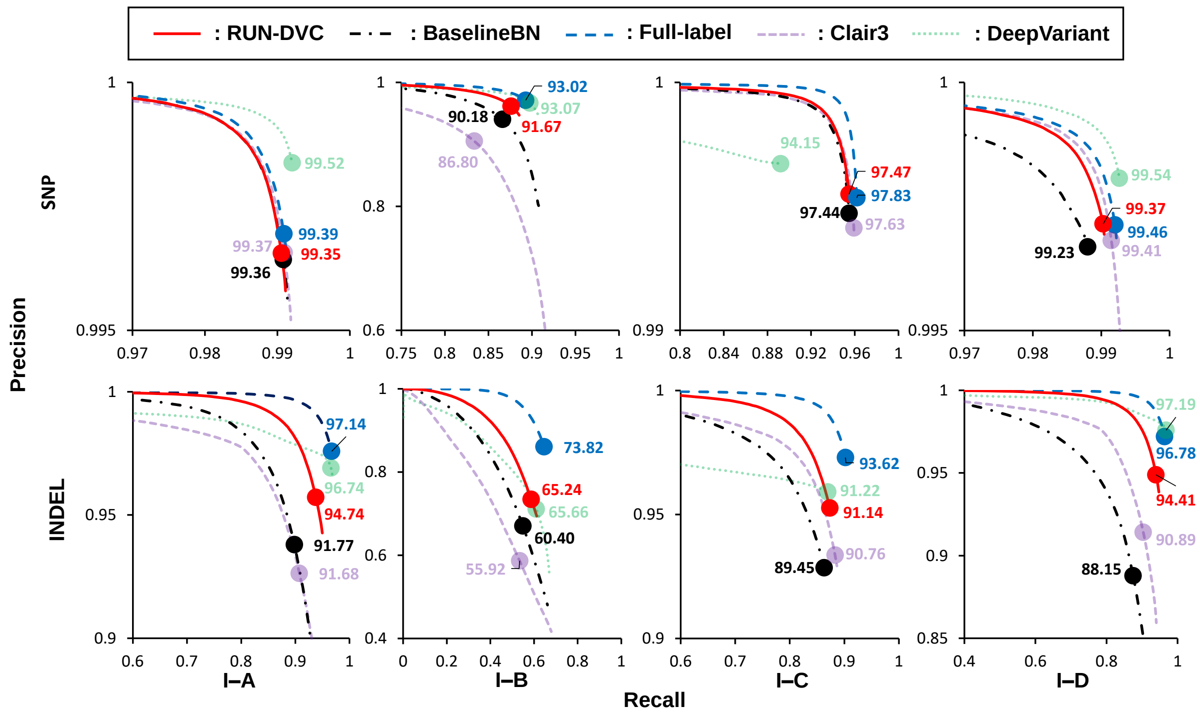
- **Feature distortions:** Distortions were applied to different channels of the tensor, including reference bases, variants in the reads, base quality, mapping quality, and haplotype information. We randomly selected bases not located at the candidate variant's position using a uniform distribution to distort the reference base channel, with the number of reference bases to change specified as input. We also introduced random false variants into the tensor using a uniform distribution to improve the model's robustness to changes in the distribution of false variants. The number of false variants is given as input, and their positions are selected using a uniform distribution. We distorted mapping and base quality by setting mapping quality values to the maximum value or adding noise sampled from a normal distribution to base quality values. For haplotype information, we dropped the information by setting all values in the channel to the value used for the unphased case.

Weak and strong data augmentations. Our study incorporated two distinct data augmentation strategies within RUN-DVC, as delineated in Supplementary Algorithm. 8. The first strategy, termed weak augmentation, implemented subsampling and vertical shifting with a probability of 50%. In contrast, the second strategy, referred to as strong augmentation, made use of the random augmentation (RandAugment) policy [14], allowing for the selection of augmentations from feature distortions alongside subsampling and vertical shifting augmentations. The number of augmentations selected from feature distortions was capped at two, a limit we observed as necessary to achieve higher performance and circumvent overfitting during the training process.

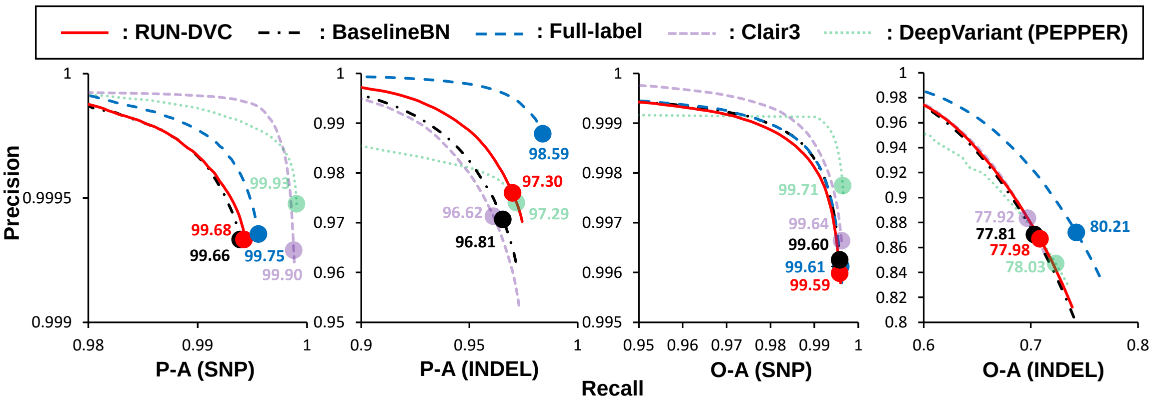
3.3.5 Training and hyperparameters

The stochastic weight averaging [69, 26] technique is used in the training. A running average of model weights is kept during training that is used for validation and model saving. Specifically, the average of model weights is updated every 32 iterations with a decay ratio of 0.99.

In all experiments, the same hyperparameters are used. Radam [49] optimizer is used with initial learning rate $l_r = 3e^{-5}$. Additionally, an exponential learning rate scheduler was employed, reducing the learning rate by a factor of 0.97 per epoch. The number of training epochs is set to 50 epochs where the number of iterations in one epoch is determined by the number of batches in the source dataset. We used a batch size of 1000 for the labeled and unlabeled datasets, 0.9 for the confidence threshold in masking, 0.05 for the weight of unsupervised loss, and $1e^{-5}$ for the weight decay. For all evaluations, we train models from scratch to focus on evaluating the performance of UDA and SSDA themselves. The best model is selected based on the validation loss in the source domain.

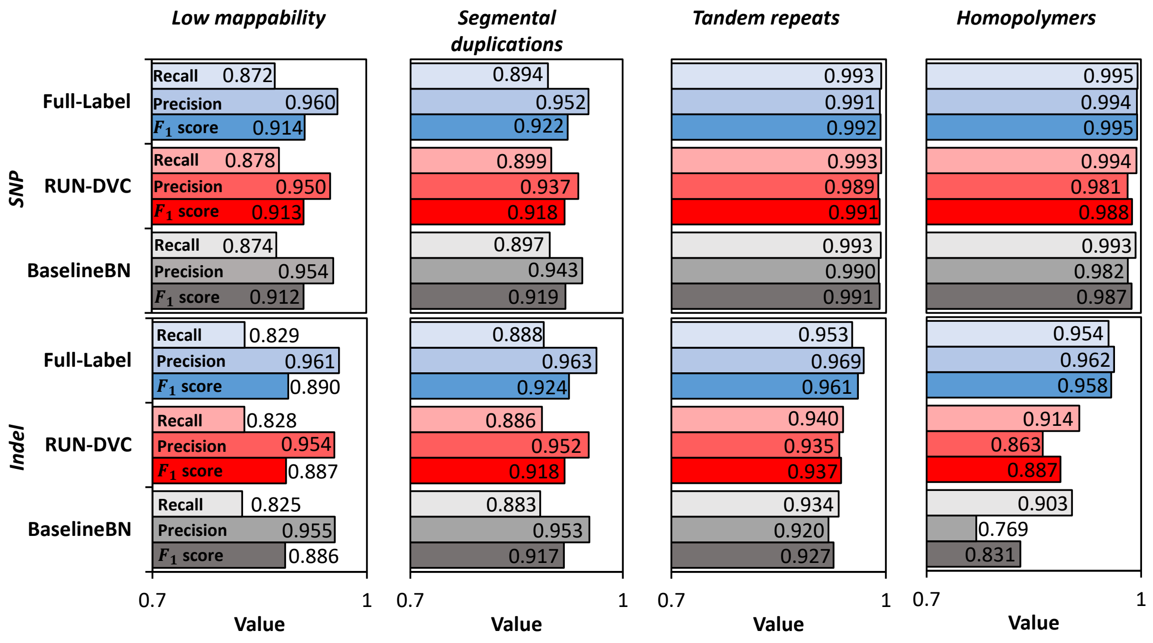


(a) Precision-recall analysis on the HG003 sample of I-A, I-B, I-C, and I-D datasets.

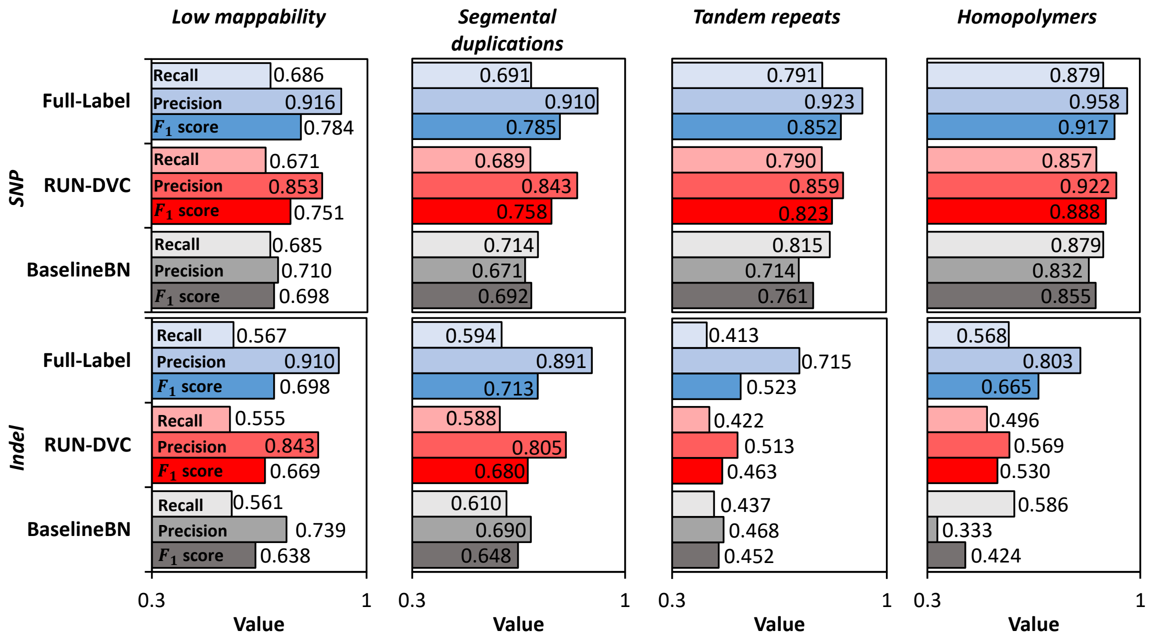


(b) Precision-recall analysis on the HG002 sample of the P-A dataset and the HG003 sample of the O-A dataset.

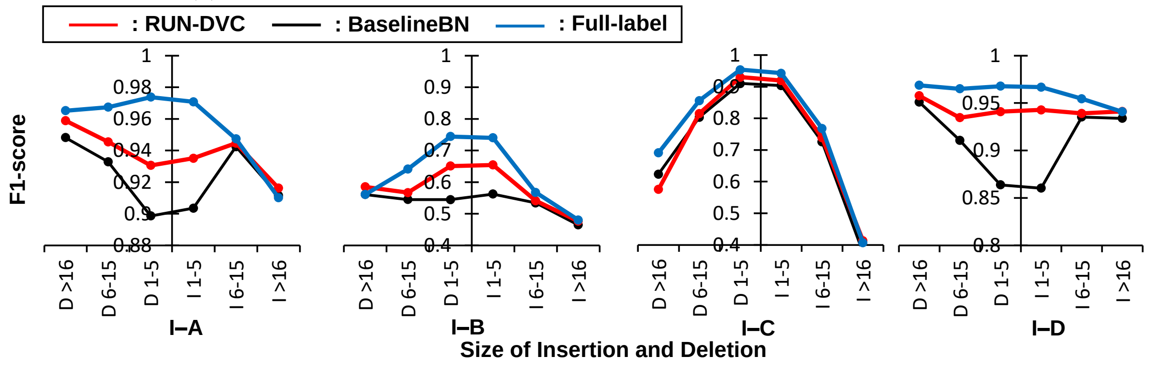
Figure 3.2: Performance of RUN-DVC under UDA setting. The quality scores are used to make precision-recall curves. The highest F_1-score (percentage) achieved by each method is marked with a circle. The precision, recall, and F_1 score of PASS calls are available in supplementary Table. 5.15, Table. 5.16, and Table. 5.17.



(a) Performance analysis by genomic regions on HG003 sample of I-A dataset.

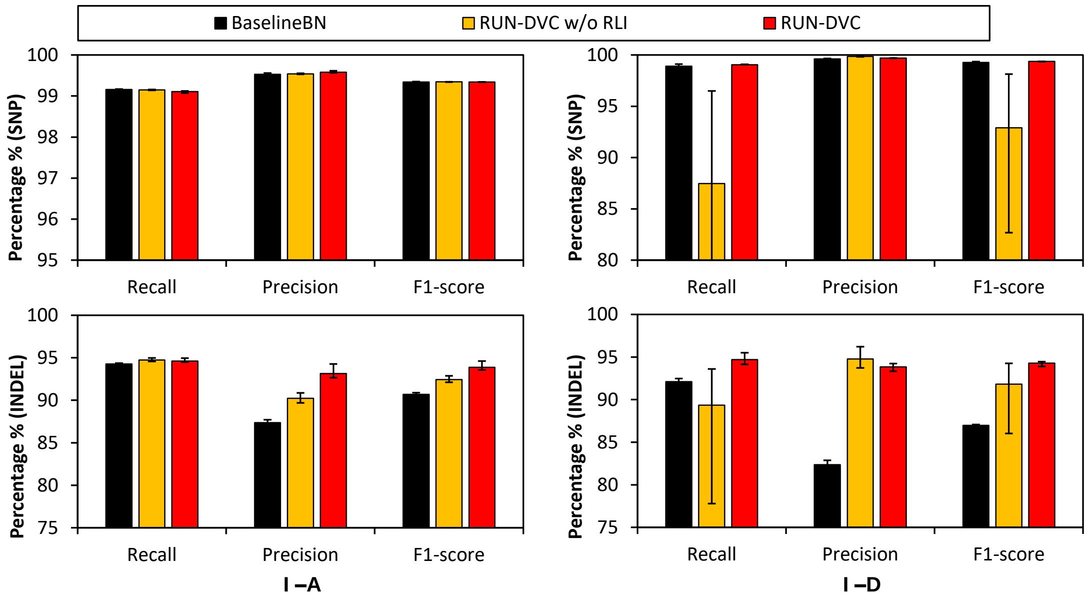


(b) Performance analysis by genomic regions on HG003 sample of I-B dataset.

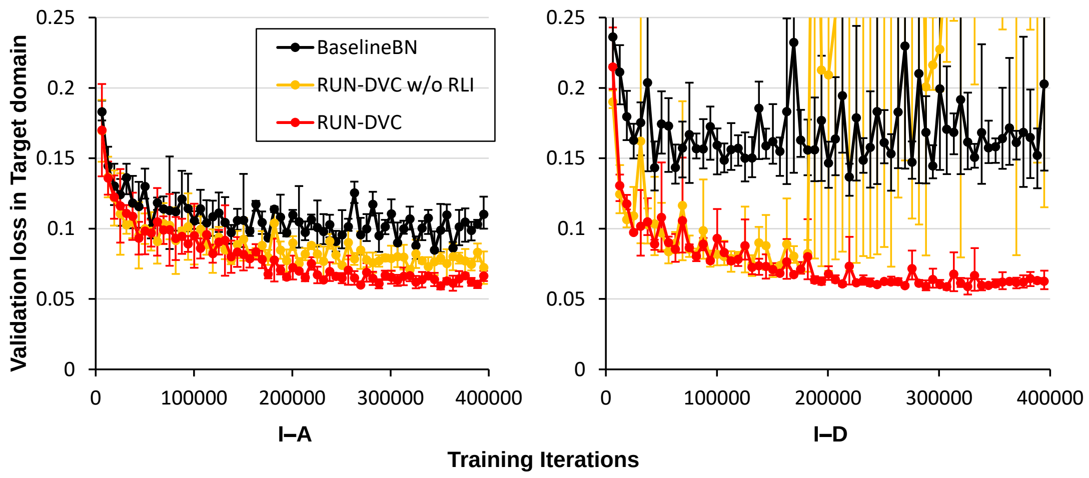


(c) Performance analysis for different INDEL sizes on HG003 sample. (x-axis represents the size of INDELs, I: insertion, D: deletion)

Figure 3.3: Performance analysis of RUN-DVC.



(a) Performance of *BaselineBN*, "RUN-DVC w/o RLI", and RUN-DVC in HG003 sample of I-A and I-D dataset.



(b) Validation loss curve of *BaselineBN*, "RUN-DVC w/o RLI", and RUN-DVC during training.

Figure 3.4: Ablation study. Error bars represent the range between the maximum and minimum values.

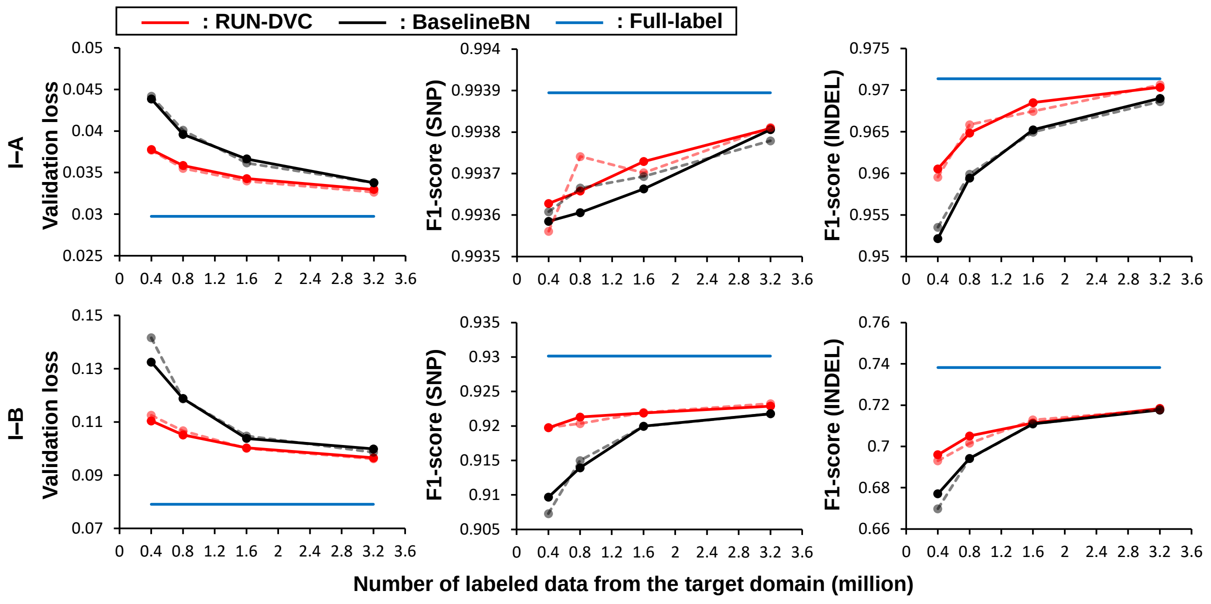


Figure 3.5: Performance of RUN-DVC under various quantities of labeled datasets in SSDA setting. Transparent lines are the results using a different random seed for target domain labeled datasets selection.

Chapter 4. Conclusion

In this thesis, we demonstrated how genome analysis pipeline can be specialized for the sequencing method of interest. To this end, we have proposed two pivotal methodologies: (1) BWA-MEME, an innovative ML-enhanced approach for read alignment, and (2) RUN-DVC, a specialized training framework tailored for deep learning-based variant callers. Central to these methods is their emphasis on precision, efficiency, and nuanced adaptability to the vast spectrum of sequencing techniques.

In Chapter 2, we elucidated the integration of the learned-index into read alignment software. The challenge that still persists is managing the volume of memory accesses during the exact match search, which is deeply interwoven with the performance of the learned-index. As we venture forward, there is an undeniable need to refine the learned-index architecture to be optimized for the reference genome. Additionally, initiatives focused on improving memory access latency and bolstering CPU cache size are expected to yield considerable enhancements in the read alignment throughput.

Chapter 3 presented a method to generalize deep learning-based variant callers to distinct sequencing methods using minimal labeled datasets. As we look toward the future, there exists a compelling potential to extend this methodology to address challenges in somatic mutation calling. Furthermore, embracing techniques such as active learning or applying noisy-label training emerges as promising avenues. These strategies can play a pivotal role, especially when procuring extensively labeled datasets becomes challenging.

In summary, the methods and insights provided in this thesis pave the way for a more accurate, efficient, and generalizable genome analysis pipeline.

Chapter 5. Supplementary Material

5.1 BWA-MEME

5.1.1 How error bound of P-RMI is guaranteed for arbitrary input keys

ML models used in the leaf models of P-RMI are monotonically increasing functions and the position of keys is a piecewise constant function of the key (constant value changes at each key in the training dataset). We show that error bound is guaranteed for arbitrary keys by how error bound is calculated in 3 cases, 1) Two adjacent keys are assigned to identical leaf model, 2) Two adjacent keys are assigned to two adjacent leaf models, 3) Two adjacent keys are assigned to two nonadjacent leaf models.

First, given two adjacent keys k_i and k_{i+1} assigned to a leaf model $model_n$, a prediction error of arbitrary keys between k_i and k_{i+1} is a monotonically increasing function of the key as it is the difference between the monotonically increasing function (output of $model_n$) and the constant function (true position). Therefore, the maximum and the minimum error of $model_n$ in arbitrary keys between k_i and k_{i+1} can be determined by evaluating the errors at k_i and k_{i+1}.

Second, when two adjacent keys k_i and k_{i+1} are each assigned to two different adjacent leaf models $model_n$ and $model_{n+1}$ (by the second constraint of P-RMI k_{i+1} must be assigned to a leaf model with larger index). This implies there exists a key $k_b \in (k_i, k_{i+1})$ where the prediction function changes (prediction function changes to leaf model $model_m$ from $model_n$ at key k_b). Thus, monotonicity holds at each interval $[k_i, k_b]$ and $[k_b, k_{i+1}]$ and the error must be evaluated at k_b to obtain the minimum and the maximum error of the leaf models $model_n$ and $model_m$. Instead of finding the key k_b, P-RMI calculates the minimum and the maximum error of $model_n$ by evaluating the error at keys k_i and k_{i+1}. This is possible because the leaf model is a monotonically increasing function and thus the error evaluated at k_b must be between the errors evaluated at k_i and k_{i+1}.

Third, when two adjacent keys are each assigned to two nonadjacent leaf models $model_n$ and $model_m$. By the second constraint used in P-RMI, all arbitrary keys should be assigned to leaf models between $model_n$ and $model_m$. We evaluate the maximum and the minimum errors of every empty leaf models between $model_n$ and $model_m$ by evaluating the errors at keys k_i and k_{i+1}. Note that, the empty leaf models between $model_n$ and $model_m$ are set to a constant function (constant value is set to position of k_i).

Error bound in leaf models of P-RMI is calculated using the algorithm 1. The maximum and the minimum error can be obtained by evaluating the error for 1) all keys assigned to the leaf model, 2) the smallest key that can be assigned to the leaf model, and 3) the largest key that can be assigned to the leaf model. Instead of finding the exact boundary between two leaf models, we use the last key of the previous model and the first key in the next leaf model. This is because it is easier to compute and gives a smaller minimum error or larger maximum error which still guarantees error bound for arbitrary keys.

Algorithm 1 Calculating upper error and lower error of leaf model N in P-RMI

```
1: Input: (Key, position) dataset K (K[n]=dataset assigned to nth leaf model) and model index n
2: Output: Upper_Error and Lower_Error of model n
3: procedure ErrorRangeCal(K, n)
4:     p_idx ← previous leaf model index that is not empty
5:     n_idx ← next leaf model index that is not empty
6:     Keys ← K[p_idx][-1] + K[n][:] + K[n_idx][0]
7:     for (Token, Position) ∈ Keys do
8:         Pred = rmi_lookup(Token)
9:         Error = Position − Pred
10:        if Upper_Error < Error then
11:            Upper_Error = Error
12:        end if
13:        if Lower_Error > Error then
14:            Lower_Error = Error
15:        end if
16:    end for
17:    Lower_error = abs(Lower_error)                    ▷ absolute value of Lower error
18:    Err = Lower_error « 32 | Upper_error
19:    Return Err
20: end procedure
```

5.1.2 Sapling only supports fixed-length seeding

Sapling only supports fixed-length exact match search and does not support finding the longest prefix of input substring that matches to the reference genome. It can be found in the source code sapling_api.h that Sapling compares the query and the reference for the given query length. Also, the proof-of-concept aligner in Sapling uses fixed-length seeding and seed extension for read alignment. The reason is that error bound of Sapling is only guaranteed for the keys seen in the training phase and is not guaranteed to find the longest exact match position of arbitrary keys. The error bound in Sapling is calculated by the distance between the prediction and the closest K-mer from the predicted position. Some unseen queries with MMP longer than K would not be found within the error bound. Even if the error bound is calculated according to the distance between prediction and the farthest matching K-mer, the error bound is not theoretically guaranteed to find the MMP of arbitrary queries. In particular, Sapling uses a global maximum error bound if the query is not found in the 95 percentile error bound. This may include most of the MMP positions of unseen queries. However, there still exists a probability that the global maximum error bound may not include the MMP position of some queries (i.e. in the leaf model that has the maximum error bound, some keys that are unseen in the training phase can have larger errors than the evaluated maximum error in the training phase).

In addition, the error bound used by Sapling results in poor learned-index lookup time. Figure 5.1 shows that the seeding throughput of P-RMI is more than 3x higher than that of Sapling. Table 5.1 shows the maximum error and the 95 percentile error used for the Sapling model (2-layer RMI with 2^{28} leaf models, we also used 32-mer for tokenization for a fair comparison with P-RMI). This is because using a global maximum error instead of a per-model error generally degrades the overall lookup performance.

To sum up, P-RMI is the first to support MMP with uncompressed suffix array search using learned index and is designed to use a much smaller error bound which results in faster lookup.

5.1.3 MMP search algorithm of STAR aligner

We ported STAR/genomeSAindex.cpp and STAR/ReadAlign_maxMappableLength2strands.cpp to BWA-MEME code to build the L-mer table index and to find the maximal mappable length of the query (The test code of STAR aligner can be found in Github). For a fair comparison, we utilized our hardware-optimized binary search in the MMP search algorithm of STAR. Also, we applied our comparison function which uses a 2-bit representation of the query and a bit-wise operation, while STAR used an 8-bit representation. We implemented the Algorithm 6 and compared the seeding throughput of the MMP search algorithm of STAR and Exact-MEME. Figure 5.2 shows that the seeding throughput of BWA-MEME is 53.4% higher compared to that of STAR. The seeding throughput is normalized with respect to the seeding throughput of BWA-MEM2.

5.1.4 Employing learned index in the suffix array search requires a constant number of memory accesses which is independent with the length of the input substring

To find the exact match position of the input substring, the memory accesses incur during the model inference and the last mile search. The number of memory accesses in RMI model inference depends on the number of layers in the RMI and is a constant number. Subsequently, memory access occurs in the last mile search where a binary search is performed within the error bound. Each comparison during a binary search requires an $O(1)$ memory access. As the error bound is a constant number that is determined at the index building step, the binary search in error bound incurs $O(1)$ memory accesses. Both RMI inference and the last mile search incur $O(1)$ memory accesses. Therefore, the exact match search problem can be solved with $O(1)$ memory accesses when employing the learned index in the suffix array search.

5.1.5 Comparison with Sapling and other learned-index structures

	95-percentile	Maximum
Lower error	12	1851771
Upper error	12	1129710

Table 5.1: Maximum and 95 percentile error of Sapling model in the human reference genome.

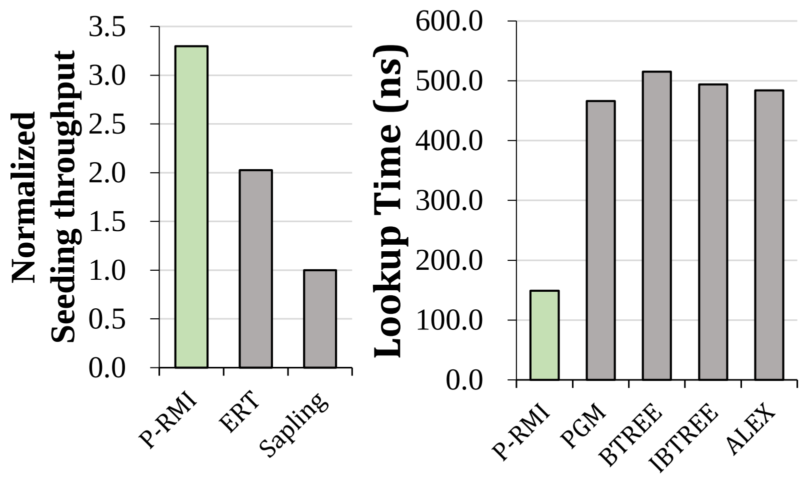


Figure 5.1: **Comparison of P-RMI, Sapling, and other learned-index structures**

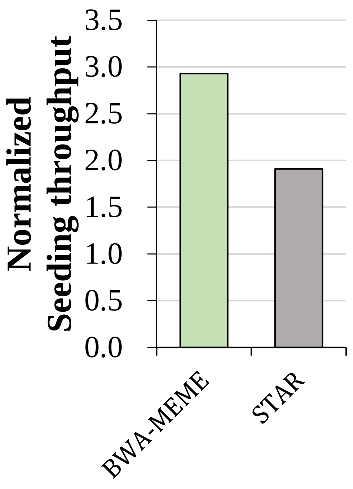


Figure 5.2: **Comparison against MMP search algorithm of STAR**

Figure 5.1 shows the performance of the Sapling model and P-RMI. We trained and integrated the Sapling model (2-layer RMI with piecewise linear function using global maximum error and 95 percentile error) to BWA-MEME and measured the seeding throughput. To support finding the longest exact match (i.e. maximal mappable prefix) of query in the suffix array, we calculated the errors of Sapling models with the method used in the original learned-index paper. While P-RMI is more than 3x faster than Sapling, Sapling and BWA-MEM2 show a nearly identical seeding throughput. This is because using the 95 percentile and the maximum error results in a larger number of last mile searches. Figure 5.1 also shows the lookup time of various state-of-the-art learned-index structures. We used the SOSD benchmark to measure the lookup time in the suffix array dataset. Some learned-index structures do not support duplicate keys and were not tested (Radix spline, TrieSpline, . . .). There were 3 state-of-the-art learned-index structures tested (RMI, PGM, and ALEX). The state-of-the-art learned-index structures show poor performance as shown in Figure 5.1. We believe this is because the suffix array of the reference genome is large, sparse, and imbalanced. State-of-the-art learned-index structures show great performance in the compact and relatively small dataset.

	Index Size (GB)	Index Loading time (SSD 500MB/s)	Index Loading time (Ramdisk)
BWA-MEM2	10	20	6
ERT	58	124	37
BWA-MEME 2nd opt	38.6	90	31
BWA-MEME 1st opt	87.6	174	57
BWA-MEME	118.6	235	73

Table 5.2: Index size (GB) and loading time (seconds) of each alignment software.

	C.elegans	Zebrafish	Mouse	Human	Hordeum Vulgare
Maximum log2 error	12.6	22.3	20.7	18.6	16.6
Average log2 error	3.80	5.08	5.02	5.18	7.79
Seeding time BWA-MEM2	1.54	2.20	2.20	2.26	3.18
Seeding time BWA-MEME	0.44	0.76	0.68	0.73	0.77
Speedup of BWA-MEME	3.53	2.91	3.22	3.12	4.10
Log2 number of models	24	28	28	28	28
Length of genome	100M	1679M	2728M	3153M	4915M

Table 5.3: Analysis of P-RMI and BWA-MEME in the various reference genomes. The seeding time is the normalized CPU ticks.

	C.elegans	Zebrafish	Mouse	Human	Hordeum Vulgare
Index building time	314	5505	9933	11707	16157
P-RMI training time	31	608	759	900	1237
Length of genome	100M	1679M	2728M	3153M	4915M

Table 5.4: The index building time of BWA-MEME. The time is recorded in seconds.

5.1.6 Scalability of BWA-MEME

We evaluated the scalability of BWA-MEME using 4, 8, 16, 24, and 48 threads. We measure the time required for both seeding and alignment using the whole short read dataset ERR194147. The throughput is presented in the table which is normalized with regard to the throughput of BWA-MEM2.

	4 threads	8 threads	16 threads	24 threads	48 threads
ERT	1.67	1.79	1.85	1.83	2.32
BWA-MEME	2.11	2.30	2.35	2.31	2.93

Table 5.5: Scalability of seeding throughput

	4 threads	8 threads	16 threads	24 threads	48 threads
ERT	1.36	1.37	1.40	1.37	1.27
BWA-MEME	1.53	1.55	1.57	1.55	1.42

Table 5.6: Scalability of alignment throughput

Aligner	48 threads		12 threads	
	Running Time (sec)	Memory Usage (GB)	Running Time (sec)	Memory Usage (GB)
BWA	4999	22.3	14951	14.1
BWA-MEM2	1937	64.3	5620	26.4
BWA-MEME	1361	153	3953	123
Bowtie2	8760	6.4	21621	6.4
STAR	1794	36	3872	29.5
Minimap2	2466	14.325	6278	13.8
Whisper2	15866	49.4	5063	23.4

Table 5.7: Running time and memory usage of each aligner in single-end alignment using ERR194147 short read dataset.

Aligner	48 threads		12 threads	
	Running Time (sec)	Memory Usage (GB)	Running Time (sec)	Memory Usage (GB)
BWA	13714	34	40925	14.6
BWA-MEM2	5675	68.1	15983	30.1
BWA-MEME	4563	168	12929	138.1
Bowtie2	19250	4.16	45360	3.5
STAR	4466	35	12180	29.6
Minimap2	4770	14.8	11100	13.8
Whisper2	15696	53	10295	24.3

Table 5.8: Running time and memory usage of each aligner in paired-end alignment using ERR194147 short read dataset.

Aligner	48 threads		12 threads	
	Running Time (sec)	Memory Usage (GB)	Running Time (sec)	Memory Usage (GB)
BWA	3605	24.6	11068	14.1
BWA-MEM2	1544	50.2	4557	29.4
BWA-MEME	1162	156	3398	137
Bowtie2	6224	3.8	45360	3.5
STAR	964	35	2580	29.6
Minimap2	1628	14.9	4394	13.4
Whisper2			3404	16.3

Table 5.9: Running time and memory usage of each aligner in single-end alignment using ERR3239284 short read dataset.

Aligner	48 threads		12 threads	
	Running Time (sec)	Memory Usage (GB)	Running Time (sec)	Memory Usage (GB)
BWA	15912	32.1	51216	21.3
BWA-MEM2	9169	53	28091	33.3
BWA-MEME	8331	191	25686	159
Bowtie2	15674	4.3	30392	3.5
STAR	3015	34.8	8583	29.5
Minimap2	5562	14.6	15798	13.5
Whisper2			7919	22.5

Table 5.10: Running time and memory usage of each aligner in paired-end alignment using ERR3239284 short read dataset.

5.1.7 Variant calling result of BWA-MEME, BWA, BWA-MEM2, Bowtie2, Minimap2, and Whisper2

The results of variant calling are presented in Table 5.11 and Table 5.12. The results of variant calling from BWA-MEME, BWA, and BWA-MEM2 were identical as the alignment results were identical. We investigated the results of variant calling in both 50x coverage average short read and 30x coverage short read (downsampled from 50x coverage short read).

Aligner	Metric	SNP		Indel	
		ALL	PASS	ALL	PASS
BWA-MEME	Recall	0.991766	0.986759	0.975432	0.973303
	Precision	0.984909	0.998163	0.973736	0.986036
	F1-score	0.988326	0.992428	0.974583	0.979628
Minimap2	Recall	0.990698	0.98342	0.97583	0.97353
	Precision	0.989795	0.998982	0.976251	0.987951
	F1-score	0.990246	0.99114	0.976041	0.980688
Bowtie2	Recall	0.971084	0.953774	0.945685	0.935857
	Precision	0.993755	0.999443	0.929759	0.968702
	F1-score	0.982289	0.976075	0.937654	0.951996
Whisper2	Recall	0.991456	0.988885	0.978602	0.976366
	Precision	0.980155	0.996372	0.970302	0.985537
	F1-score	0.985773	0.992615	0.974435	0.98093

Table 5.11: Statistics on the called SNPs and indels of NA12878 sample using 50X coverage WGS. SAM outputs of BWA, BWA-MEM2, and BWA-MEME were identical. The highest values are marked red.

Aligner	Metric	SNP		Indel	
		ALL	PASS	ALL	PASS
BWA-MEME	Recall	0.990629	0.984348	0.955124	0.950498
	Precision	0.971217	0.997766	0.966499	0.980674
	F1-score	0.980827	0.991012	0.960778	0.96535
Minimap2	Recall	0.988935	0.980186	0.954047	0.949197
	Precision	0.97757	0.99868	0.968734	0.982698
	F1-score	0.98322	0.989346	0.961335	0.965657
Bowtie2	Recall	0.969563	0.950361	0.92528	0.913215
	Precision	0.984672	0.999267	0.94071	0.967336
	F1-score	0.977059	0.974201	0.932931	0.939497
Whisper2	Recall	0.98981	0.986241	0.959635	0.954674
	Precision	0.966164	0.995766	0.962116	0.98031
	F1-score	0.977844	0.990981	0.960874	0.967322

Table 5.12: Statistics on the called SNPs and indels of NA12878 sample using 30X coverage WGS. The highest values are marked red.

5.1.8 Comparison of mapping times for various coverages

We show how running time varies with different coverages. For this evaluation, we used the ERR194147 short reads and sub-sampled it with various coverages. The index was loaded from NVMe SSD which provides up to 1 GB/second of disk IO. Also to prevent the index loading from the memory cache, we flushed the memory cache before all experiments.

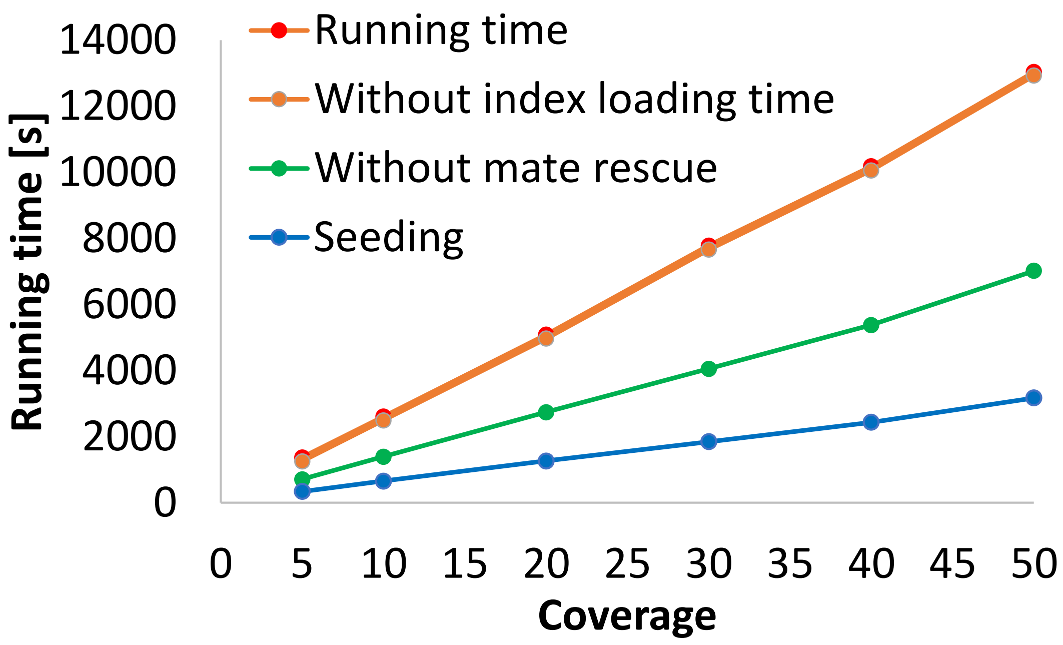


Figure 5.3: **Comparison of mapping times for various coverages. The figure shows the running time of paired-end alignment with BWA-MEME using 12 threads.**

5.1.9 Proof: SMEM search of BWA-MEME has identical SMEM output with SMEM-ERT

Let R be a short read sequence that consists of A, C, G, T. Let $R[i, j]$ denote the substring starting at position i and ending at position j of short read R. The SMEM searching stage of SMEM-ERT starts at the pivot point. The backward and forward extensions are repeatedly performed until the forward extension reaches the end of the obtained LEP bits. The forward extension starts at the point where the backward extension ends, and the backward extension always starts at the nearest point where the LEP bit is set to 1. We prove performing backward extension and forward extension without obtaining LEP bits has identical SMEM output.

Theorem 5.1.1. *Repeatedly performing backward and forward extensions starting from the pivot point finds all SMEMs that are identical with SMEM-ERT.*

Definition 5.1.1 (Extension)**.** Forward extension performed in the point Pos of the short read R is denoted $forward(pos)$. Likewise, backward extension performed in the point Pos of the short read R is denoted $backward(Pos)$. The extension is performed until it can't be further extended and the output of the extension is the position where the extension ends. Hence, $forward(backward(Pos)) \geq Pos$ should always hold.

Definition 5.1.2 (Left extension point bit)**.** The left extension point bits are obtained in the forward extension in the pivot point P of the short read R. LEP bit is obtained for all $[P, forward(P)]$, if number of hits of substrings $R[P, P+n]$ and $R[P, P+n+1]$ are different, LEP bit in $P+n$ is set to 1 or otherwise LEP bit is set to 0.

Performing backward and forward extension at the point where the LEP bit is set to 1 is the same algorithm as SMEM-ERT. Thus we prove performing backward and forward extension in the point where the LEP bit is set to 0 has an identical result with starting from the closest point which has the LEP bit set to 1. The proof of Theorem 5.1.1 directly follows from the next three lemmas

Lemma 5.1.2. *Let pos be a point where the LEP bit is set to 0 and P a pivot point of the short read. For $\forall p_s \in [backward(pos), pos]$, $forward(p_s) > pos$.*

Proof. From extension definition, $forward(p_s)$ should be equal or larger than pos. If $forward(p_s) \equiv pos$, there exists unique hits of substring $R[p_s, pos]$ in reference. For $\forall p \in [p_s, pos]$, there exists hits of substring $R[P, pos]$ exists where $R[P, pos+1]$ does not exact match. It is contradiction to assumption that LEP bit is set to 0 in pos, therefore for $\forall p_s \in [backward(pos), pos]$, $forward(p_s) > pos$. □

Lemma 5.1.3. *Let pos be a position where LEP bit is set to 0 in the SMEM searching stage. For all pos, $backward(pos) \equiv backward(pos + 1)$.*

Proof. We divide the possible cases of $backward(pos)$ and $backward(pos + 1)$ in to three.

- Case 1 $backward(pos) > backward(pos + 1)$: Backward extension from pos cannot be shorter than backward extension from $pos + 1$ which leads to a contradiction with Definition 5.1.1.
- Case 2 $backward(pos) < backward(pos+1])$: $backward(pos)$ can be smaller than $backward(pos+1])$ only if $forward(backward(pos)) \equiv pos$. If $forward(backward(pos)) > pos$, it is contradiction to $backward(pos) < backward(pos + 1])$. Also, from lemma 5.1.2 $forward(backward(pos)) \equiv pos$ is contradiction to the assumption that LEP bit is set to 0 in pos.

- Case 3 $backward(pos) = backward(pos+1])$: As case 1 and case 2 are excluded, for all pos, $backward(pos)$ and $backward(pos+1])$ should be identical.

All cases except case 3 lead to contradiction therefore we have that $backward(pos) \equiv backward(pos+1])$ for any position where LEP bit is set to 0. $\square$

Lemma 5.1.4. *Let pos_1 be the closest position from pos where LEP bit is set to 1 in forward direction. For all $p \in [pos, pos_1], \exists C \in [0, pos)$, s.t. $C = backward(p)$. Thus, performing backward extension and forward extension sequentially from $\forall p \in [pos, pos_1]$ results in $forward(C)$.*

Proof. From Lemma 5.1.3, it is given $backward(pos) \equiv backward(pos+1)$. For $\forall n \in [1, pos_1 - pos)$, LEP bit in position $pos+n$ is 0 and it is proven from Lemma 5.1.3 that $backward(pos+n) = backward(pos+n+1)$. Therefore, for $\forall p \in [pos, pos_1], \exists C \in [0, pos)$, s.t. $C = backward(p)$ and $forward(C) \equiv forward(backward(p)])$

$\square$

Dataset	read length	number of reads	Source
ERR194146	101	813180578	Illumina Platinum Genomes
ERR194147	101	787265109	Illumina Platinum Genomes
ERR194158	101	859371011	Illumina Platinum Genomes
ERR194159	101	707646124	Illumina Platinum Genomes
ERR194160	101	775617169	Illumina Platinum Genomes
ERR194161	101	843454257	Illumina Platinum Genomes
ERR3239276	150	396570406	1000 Genomes Project Phase 3
ERR3239277	150	363937308	1000 Genomes Project Phase 3
ERR3239278	150	342631544	1000 Genomes Project Phase 3
ERR3239279	150	420210145	1000 Genomes Project Phase 3
ERR3239280	150	391766960	1000 Genomes Project Phase 3
ERR3239281	150	365635559	1000 Genomes Project Phase 3
ERR3239282	150	367637337	1000 Genomes Project Phase 3
ERR3239283	150	391766960	1000 Genomes Project Phase 3
ERR3239284	150	374824132	1000 Genomes Project Phase 3

Table 5.13: Short read data used for evaluation.

Algorithm 2 Tokenization of query sequence

```
1: Input: Query sequence Q
2: Output: Tokenized sequence
3: procedure GET_KEY_OF_READ(Q)
4:     Token = 0
5:     for i ← 0 to min(len(Q),32) do
6:         c = Q[i]
7:         if c == Ambiguous base then
8:             Break
9:         end if
10:        Token = Token << 2
11:        Token = Token | 2bit_encode(c)
12:    end for
13:    while i < 32 do
14:        Token = Token << 2
15:    end while
16:    Return Token
17: end procedure
```

5.2 RUN-DVC

5.2.1 Details of inputs for CNN and RNN model.

Input tensor for CNN model. The 3-dimensional tensor comprises multiple channels, with each channel providing information about the candidate variant region. Although much of our tensor design is comparable to that used in Clair3, we made three critical modifications. First, we assign the mapping quality value not just to positions with aligned reads, but also to positions where a deletion is indicated. This modification is intended to circumvent a potential confounding scenario where the mapping information is zero when the deletion spans beyond the window size of the tensor. Second, in the target variant channel, we allocate the allele frequency value solely to the position of the candidate variant. This adjustment was made to enable the horizontal shift data augmentation techniques. Third, we changed the values used for encoding bases and strand information. Our refined three-dimensional tensor comprises eight distinct channels. These channels encode information about reference bases, observed variants in reads, strand information, mapping quality, base quality, the position of candidate variants, the bases involved in insertions, and phasing information. However, in the case of short-read sequencing data, the phasing channel is omitted, resulting in a total of seven channels. The input size for our tensor differs depending on the sequencing platform. For short-read sequencing platforms and the PacBio sequencing platform, we utilized an input tensor with dimensions of 33 columns by 55 rows. However, for the ONT sequencing platform, we employed an input tensor of 33 columns by 89 rows to account for the increased read depth. By default, we assign a value of zero to all positions where either a deletion is indicated or no reads are mapped.

- **Reference bases:** Each position of the aligned read is assigned an integer based on the reference base at that position (A: 75, C: -50, G: 50, T: -75).

Algorithm 3 Lookup of P-RMI

```
1: Input: Tokenized query sequence
2: Output: Predicted position and Error of tokenized query sequence
3: procedure P-RMI_LOOKUP(Token)
4:     model_index ← first_layer_model(Token)
5:     Pred, Err ← second_layer_models(Token, model_index)
6:     if Err >> 63 then
7:         Third_model_start_index ← (Err >> 32) & 0x7fffffff
8:         Third_model_max_index ← Err & 0xffffffff
9:         Pred = min(Pred, Third_model_max_index)
10:        model_index = Third_model_start_index + Pred
11:        Pred, Err = Additional_layer_models(Token, model_index )
12:    end if
13:    Return Pred, Err                    ▷ Err contains Lower_error and Upper_error
14: end procedure
```

Algorithm 4 Exact-MEME algorithm

```
1: Input: Query sequence Q
2: Output: MEM position and length of MEM
3: procedure LEM_SEARCH(Q)
4:     Token ← Tokenization(Q)
5:     Pred,Err ← P-RMI_lookup(Token)
6:     Lower_error ← (Err >> 32) & 0x7fffffff
7:     Upper_error ← Err & 0xffffffff
8:     Search_bound ← { Pred - Lower_error, Pred + Upper_error }
9:     MEM position, Length ← BinarySearch(Q, Search_bound)
10:    Return MEM position, Length
11: end procedure
```

- **Mutations:** Positions differing from the reference base are assigned an integer based on the type of variant present. SNPs are assigned an integer based on the alternative base at the position using the same base-value mapping as in the reference bases channel. INDEL variants are assigned -25 and 25, respectively, at positions corresponding to the starting location of the INDEL on the left end.

- **Strand information:** Each read position is assigned an integer depending on the strand the read is aligned to: 50 for the forward strand and -50 for the reverse strand.

- **Mapping quality:** An integer between 0 and 100 is assigned to all mapped read positions, scaled up from the original Phred mapping score (range 0 to 60), and capped at 60 if it exceeds that value.

- **Base quality:** Each aligned base is assigned an integer between 0 and 100, based on the scaled-up Phred base quality score (original range 0 to 40) and capped at 40 if it exceeds that value.

- **Target variant:** The candidate variant position (default is the column center) is assigned an

Algorithm 5 Extension using Exact-MEME algorithm

```
1:  Input: Query sequence Q and the hit_threshold
2:  Output: start position of hit range, length of MEM, number of hits
3:  procedure COMPARE(Q, pos)
4:      return exact match length of Q and SA[pos]
5:  end procedure
6:  procedure EXTENSION(Q, hit_threshold)
7:      Lem_pos, Lem_len ← Lem_search(Q)
8:      upper_b = Lem_pos + 1; lower_b = Lem_pos - 1;
9:      while upper_b - lower_b - 1 < hit_threshold do
10:         while Lem_len == compare(Q, upper_b) do
11:             upper_b = upper_b + 1
12:         end while
13:         while Lem_len == compare(Q, lower_b) do
14:             lower_b = lower_b - 1
15:         end while
16:         last_lem_len = Lem_len
17:         Lem_len = max(compare(Q, upper_b), compare(Q, lower_b)
18:     end while
19:     return (lower_b+1), last_lem_len, (upper_b-lower_b-1)
20: end procedure
```

integer between 0 and 100, indicating the percentage of reads supporting the specific mismatch pattern. For instance, if 10 out of 20 reads support the reference allele "A", 5 support the alternative "C", 3 support "T", and 2 support an insertion "AT", the integer assigned would be 0, 25, 15, and 10, respectively.

- **Insertion bases:** Inserted bases are encoded following each insertion's starting position, using the same base-value mapping as the reference bases channel.
- **Phasing information:** All positions of each read are assigned an integer value based on phasing: -50 for HP1, 20 for unphased, and 50 for HP2 reads. If reads are phased, they are sorted in the order "unphased, HP1, HP2" across all eight channels.

Input tensor for RNN model. The pileup input tensor consists of 594 integers, representing 33 genome positions with 18 features at each position. These features include counts of read support for the four nucleotides (A, C, G, T), insertions (I_1 and I), deletions (D_1 and D), and R on both the positive strand (+) and negative strand (-). The presence of a '1' superscript indicates that only the indel with the highest read support is counted if there are multiple indels at a given candidate site (i.e., all indels are counted if there is no '1' superscript). The 'R' indicates the following positions of an indel.

5.2.2 Commands

Please see the GitHub for commands used for RUN-DVC. We provide scripts regarding dataset generation, training, and variant calling which heavily make use of GNU Parallel [67].

Algorithm 6 SMEM-MEME algorithm for BWA-MEME

```
Input: short read, pivot point, hit_threshold, min_seed_len
Output: SMEM list
procedure SMEM_SEARCH(read, pivot, hit_threshold, min_seed_len)
    smem_list ← []; read_rc ← read.reverse_complement();
    search_pivot = pivot
    while 1 do
        Q = read_rc[len(read)-search_pivot: -1]                ▷ Backward extension
        _, len, _ ← extension(Q, hit_threshold)
        search_pivot = search_pivot - len + 1
        if search_pivot > pivot then
            break                       ▷ Extension no longer include pivot point
        end if
        Q = read[search_pivot:-1]                               ▷ Forward extension
        pos, len, num ← extension(Q, hit_threshold)
        if num < hit_threshold and len > min_seed_len then
            for Ref_Pos in SA[pos,...,pos + num] do
                smem_list.append( Ref_Pos )
            end for
        end if
        search_pivot = search_pivot + len
    end while
    Return smem_list
end procedure
```

Command used for DeepVariant

```
docker run google/deepvariant:1.5.0 /opt/deepvariant/bin/run_deepvariant -model_type WGS
-ref {Reference} -reads {BAM} -output_vcf {OutputFile} -num_shards {ThreadNum} -regions
{BED}
```

Command used for PEPPER

```
sudo docker run  -u ‘id -u $USER‘:‘id -g $USER‘  -ipc=host  kishwars/pepper_deepvariant:r0.8
run_pepper_margin_deepvariant call_variant  -b "${BAM_FILE}"  -f "/data/Homo_sapiens_assembly38.fast
-o "/output/"  -regions /data/HG003_GRCh38_1_22_v4.2.1_benchmark.bed  -p "${DATASET}"  -t
"${THREADS}  -ont_r9_guppy5_sup
```

Command used for VCF report in DeepVariant

```
docker run google/deepvariant:1.5.0 /opt/deepvariant/bin/vcf_stats_report -input_vcf ${INPUT_VCF}
-outfile_base ${OUTPUT}
```

Command used for Clair3

```
docker run hkubal/clair3:latest /opt/bin/run_clair3.sh -model_path="/opt/models/{MODEL_NAME}"
-ref_fn={Reference} -bam_fn={BAM} -output={OutputFile} -threads={ThreadNum} -bed_fn={BED}
-platform={platform}
```

5.2.3 Links for datasets

Genome stratification file: https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/genome-stratifications/ **GIAB truth variant sets:** https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/ **GIAB sequencing datasets:** https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/ **Google sequencing datasets:** https://console.cloud.google.com/storage/browser/brain-genomics-public **Human Pangenome Reference Consortium datasets:** https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=NHGRI_UCSC_panel/

5.2.4 Details of training datasets of Clair3 and DeepVariant (PEPPER)

The training datasets used for Clair3 and DeepVariant can be found in below links.
DeepVariant for PacBio dataset: https://github.com/google/deepvariant/blob/r1.5/docs/deepvariant-details-training-data.md
Clair3 for PacBio dataset: https://github.com/HKU-BAL/Clair3/blob/main/docs/training_data.md
Clair3 for ONT dataset: https://github.com/HKU-BAL/Clair3/blob/main/docs/guppy5_20220113.md

Algorithm 7 All Seeding algorithm

```
1:  Input: short read
2:  Output: SMEM list
3:  procedure SEEDING(read)
4:      1st_smems, 2nd_smems, 3rd_smems ← [], [], [];
         /*First stage seeding*/
5:      search_pivot = 0
6:      hit_threshold = 1
7:      min_threshold = 19
8:      1st_smems.append( SMEM_SEARCH(read, search_pivot, hit_threshold, min_threshold) )
         /*Second stage re-seeding*/
9:      for smem in 1st_SMEMs do
10:         hit_threshold = smem.hit_threshold + 1
11:         search_pivot ← Middle position of smem
12:         Q = read_rc[len(read)-search_pivot: -1]                ▷ Backward extension
13:         _, len, _ ← extension(Q, hit_threshold)
14:         search_pivot = search_pivot - len + 1
15:         Q = read[search_pivot:-1]                               ▷ Forward extension
16:         pos, len, num ← extension(Q, hit_threshold)
17:         if num < hit_threshold and len > min_seed_len then
18:             for Ref_Pos in SA[pos,...,pos + num] do
19:                 2nd_smems.append( Ref_Pos )
20:             end for
21:         end if
22:     end for/*Last stage additional seeding*/
23:     hit_threshold = 20
24:     search_pivot = 0
25:     while search_pivot < len(read) do
26:         Q = read[search_pivot:-1]                               ▷ Forward extension
27:         pos, len, num ← extension(Q, hit_threshold)
28:         if num < hit_threshold and len > min_seed_len then
29:             for Ref_Pos in SA[pos,...,pos + num] do
30:                 3rd_smems.append( Ref_Pos )
31:             end for
32:         end if
33:         search_pivot = search_pivot + len
34:     end while
35:     smem_list ← 1st_smems + 2nd_smems + 3rd_smems
36:     Return smem_list
37: end procedure
```

Algorithm 8 Data Augmentation Procedure

```
1: data ← Sequencing data in 3D tensor
2: List_Aug ← ["row_drop", "vertical_shift"]
3: List_Feature_Aug ← ["Reference", "Mapping_quality", "Base_quality", "Target_variants", "Phasing"]
4: if weak augmentation then
5:     if random.random() < 0.5 then
6:         data.ApplyAug(Intensity=0.3, List_Aug)
7:     end if
8: else if strong augmentation then
9:     data.ApplyAug(Intensity=0.7, List_Aug)
10:    data.ApplyRandaug(Intensity=0.5, Num_Aug=2, List_Feature_Aug)
11: end if
```

Algorithm 9 RUN-DVC Loss Function

```
1: procedure RUNDVC_Loss($label_s$, $lc$, $lspp$, $source\_total$)
2:     /*
3:     param $label\_s$: labels of the source domain batch
4:     param $lc$: logits obtained from the combined batch (source + target)
5:     param $lspp$: logits obtained from the source batch (only source)
6:     param $source\_total$: size of the weakly and strongly augmented source batch
7:     */
8:     $source\_batch\_size \leftarrow source\_total//2$
9:     $target\_batch\_size \leftarrow (lc.size(0) - source\_total)//2$
10:    $lsp \leftarrow lc[: source\_total]$
11:    if $USE\_RLI$ then
12:        $lambd \leftarrow torch.rand\_like(lsp)$
13:        $final\_logits\_source \leftarrow (lambd * lsp) + (1 - lambd) * lspp$
14:    else
15:        $final\_logits\_source \leftarrow lsp$
16:    end if
17:    $source\_loss \leftarrow CrossEntropyLoss(final\_logits\_source, label_s)$
18:    if $USE\_SSL$ then
19:        $logits\_target\_weak \leftarrow lc[source\_total : -target\_batch\_size]$
20:        $logits\_target\_strong \leftarrow lc[-target\_batch\_size :])$
21:        $mask \leftarrow Softmax(logits\_target\_weak) > Confidence\_Value$
22:        $pseudo\_labels \leftarrow argmax(logits\_target\_weak)$
23:        $target\_loss \leftarrow CrossEntropy(logits\_target\_strong, pseudo\_labels)$
24:        $target\_loss \leftarrow mask \times target\_loss$
25:        Return $source\_loss$, $target\_loss$
26:    else
27:        Return $source\_loss$
28:    end if
29: end procedure
```

Name	Sequencing Platform	Sample and Library preparation method
I-Source	Illumina Novaseq PCR-free	[6]
I-A	Illumina Novaseq PCR-plus	[6]
I-B	Illumina Hiseq2500	https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/10XGenomics/README_10Xgenomes.txt
I-C	BGISEQ-500	https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/stLFR/readme.txt
I-D	Illumina HiseqX PCR-plus	[6]
P-Source	PacBio CCS Sequel II	https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_CCS_15kb_20kb_chemistry2/GRCh38/README.txt https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_SequelII_CCS_11kb/HG002_GRCh38/README.txt
P-A	PacBio CCS Sequel I	https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_CCS_10kb/GRCh38_no_alt_analysis/README.txt https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_CCS_15kb/GRCh38_no_alt_analysis/README.txt
O-Source, O-A	ONT datsets	http://www.bio8.cs.hku.hk/guppy5_data/ https://github.com/HKU-BAL/Clair3/blob/main/docs/guppy5_20220113.md#how-to-use-the-guppy5-model

Table 5.14: Details of sequencing datasets used.

		I-A			I-B		
Method	Variant type	Recall	Precision	F1-score	Recall	Precision	F1-score
BaselineBN	Indel	94.06%	87.21%	90.51%	66.22%	48.06%	55.70%
	SNP	99.14%	99.56%	99.35%	90.76%	80.12%	85.11%
Clair3	Indel	94.53%	87.18%	90.71%	67.93%	41.68%	51.66%
	SNP	99.18%	99.52%	99.35%	91.75%	57.94%	71.03%
DeepVariant	Indel	96.83%	96.56%	96.69%	66.91%	55.92%	60.92%
	SNP	99.20%	99.84%	99.52%	90.85%	94.21%	92.50%
RUN-DVC	Indel	94.65%	92.28%	93.45%	61.22%	69.42%	65.06%
	SNP	99.17%	99.53%	99.35%	88.54%	94.69%	91.51%
Full-label	Indel	96.70%	97.58%	97.14%	64.62%	86.07%	73.82%
	SNP	99.14%	99.64%	99.39%	89.33%	97.01%	93.02%

Table 5.15: Variant calling result (PASS calls) in HG002 sample of I-A and I-B under UDA setting.

		I-C			I-D		
Method	Variant type	Recall	Precision	F1-score	Recall	Precision	F1-score
BaselineBN	Indel	86.29%	92.85%	89.45%	91.42%	82.87%	86.93%
	SNP	95.50%	99.44%	97.43%	98.81%	99.65%	99.23%
Clair3	Indel	88.63%	92.93%	90.73%	94.05%	85.94%	89.81%
	SNP	95.93%	99.38%	97.62%	99.32%	99.40%	99.36%
DeepVariant	Indel	86.96%	95.91%	91.22%	96.81%	97.58%	97.19%
	SNP	89.20%	99.67%	94.15%	99.27%	99.81%	99.54%
RUN-DVC	Indel	87.35%	95.27%	91.14%	94.71%	93.85%	94.28%
	SNP	95.49%	99.52%	97.47%	99.04%	99.70%	99.37%
Full-label	Indel	89.88%	97.61%	93.59%	96.37%	97.19%	96.78%
	SNP	96.20%	99.52%	97.83%	99.22%	99.69%	99.45%

Table 5.16: Variant calling result (PASS calls) in HG002 sample of I-C and I-D under UDA setting.

		P-A			O-A		
Method	Variant type	Recall	Precision	F1-score	Recall	Precision	F1-score
BaselineBN	Indel	97.14%	96.22%	96.68%	74.01%	80.42%	77.08%
	SNP	99.39%	99.93%	99.66%	99.61%	99.59%	99.60%
Clair3	Indel	97.29%	95.34%	96.31%	71.32%	85.46%	77.75%
	SNP	99.88%	99.92%	99.90%	99.62%	99.66%	99.64%
DeepVariant	Indel	97.31%	97.21%	97.26%	73.40%	83.10%	77.95%
	SNP	99.90%	99.95%	99.93%	99.66%	99.76%	99.71%
RUN-DVC	Indel	97.44%	97.03%	97.23%	73.87%	81.21%	77.37%
	SNP	99.42%	99.93%	99.68%	99.60%	99.58%	99.59%
Full-label	Indel	98.48%	98.69%	98.58%	76.40%	83.72%	79.89%
	SNP	99.56%	99.93%	99.75%	99.63%	99.58%	99.60%

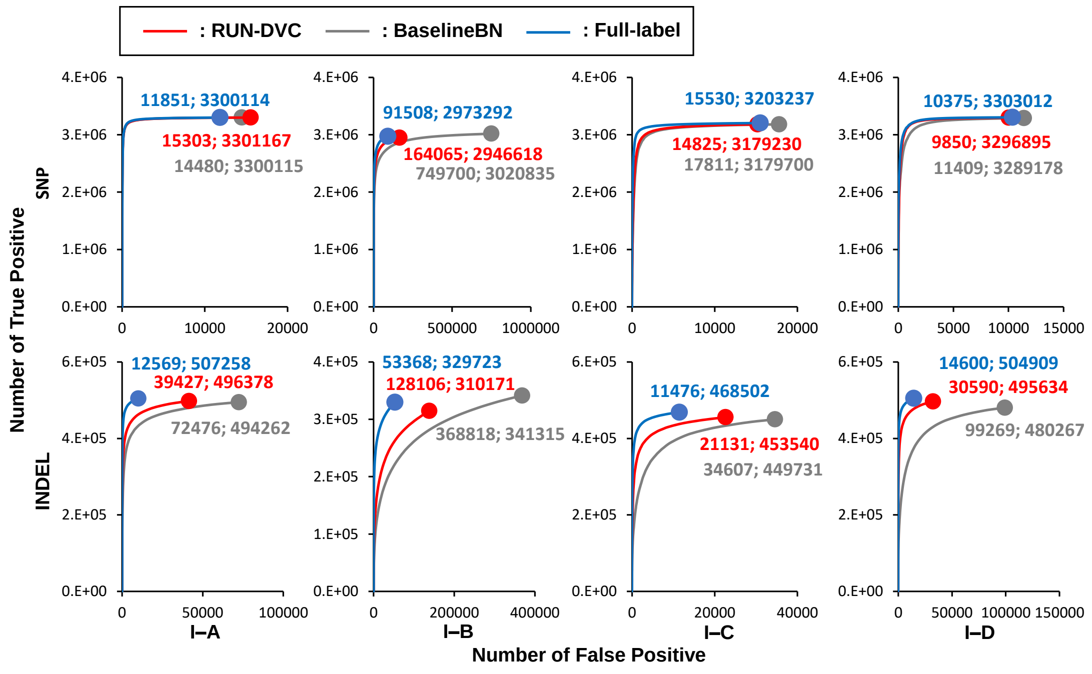
Table 5.17: Variant calling result (PASS calls) in HG002 sample of P-A and O-A under UDA setting.

Dataset	Model type	($\gamma = 0$)	($\gamma = 0.5$)	($\gamma = 1$)
I-A	RNN	99.80%	99.91%	99.91%
I-B	RNN	92.40%	97.61%	98.90%
	CNN	94.02%	97.69%	98.88%
I-C	RNN	99.64%	99.90%	99.90%
P-A	RNN	99.91%	99.92%	99.92%

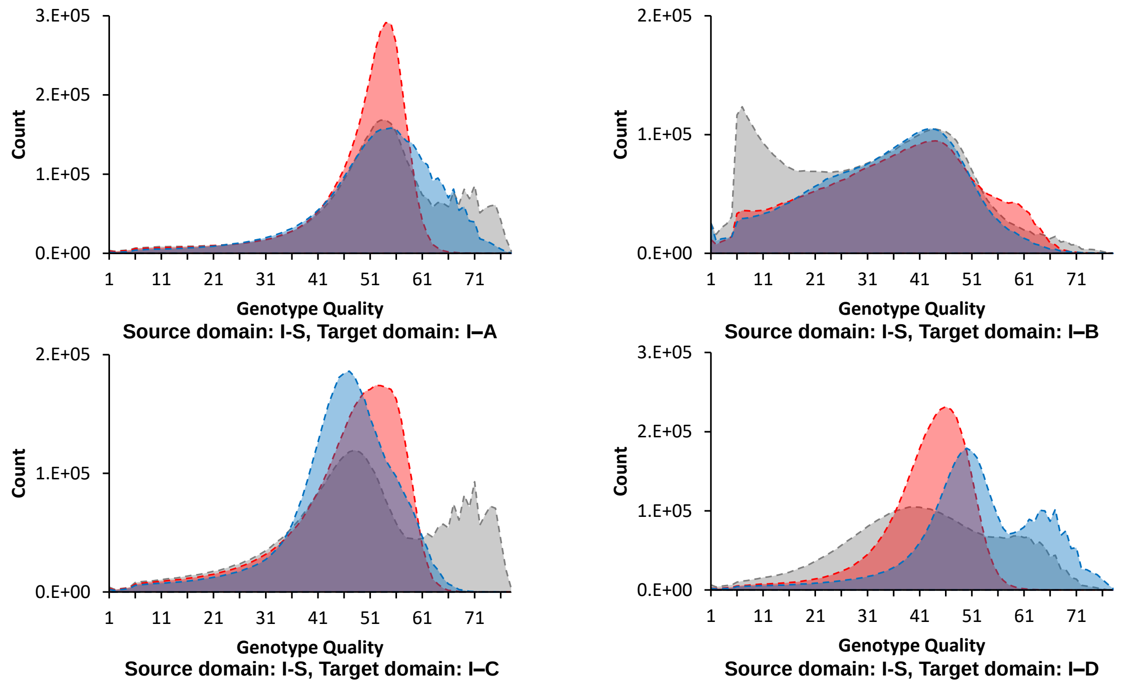
Table 5.18: Accuracy analysis of binary classification for true variant and non-variant using RNN and CNN model.

Dataset	Variant type	Recall	Precision	F1-score
I-A	Indel	83.03%	70.75%	76.40%
	SNP	99.05%	99.12%	99.08%
I-B	Indel	57.94%	49.48%	53.38%
	SNP	87.04%	87.08%	87.06%
I-C	Indel	40.80%	87.20%	55.59%
	SNP	34.16%	95.50%	50.33%
P-A	Indel	83.97%	90.42%	87.08%
	SNP	99.58%	99.63%	99.60%

Table 5.19: Accuracy of variant calling using RNN model.



(a) A number of true positives and false positives. The number of false positives and true positives for all PASS calls is marked with a solid circle and a label (the number of false positives; the number of true positives).



(b) Genotype quality distribution results.

Figure 5.4: Performance analysis of RUN-DVC, *BaselineBN*, and Full-label on I-A, I-B, I-C, and I-D datasets in UDA setting.

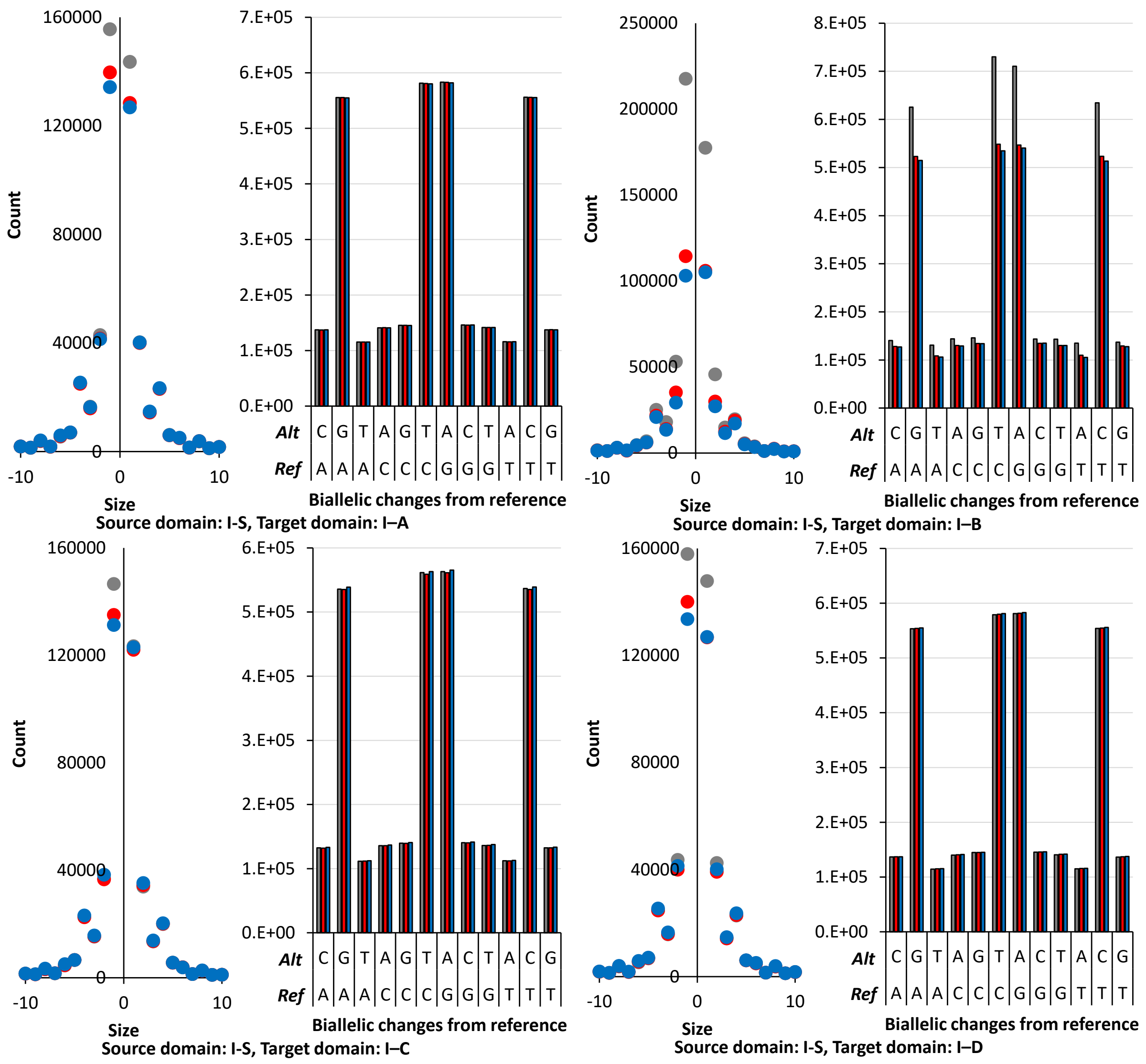


Figure 5.5: Analysis on disagreements between RUN-DVC, *BaselineBN*, and *Full-label* in UDA setting.

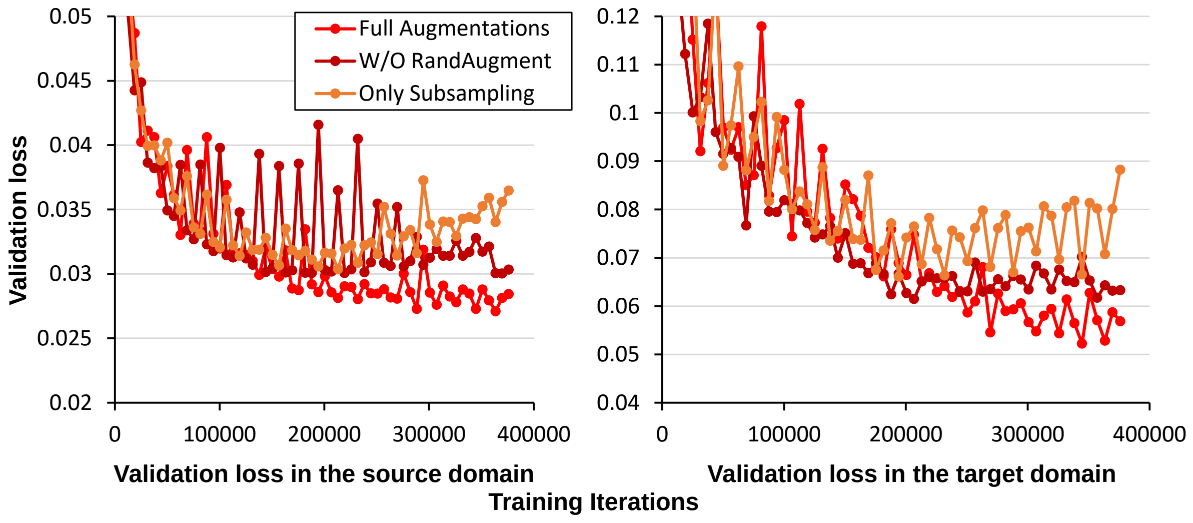


Figure 5.6: An ablation study demonstrating the impact of varying data augmentation strategies on RUN-DVC in a UDA setting (Source: I-Source, Target: I-A).
The "Only Subsampling" method applies subsampling as the sole augmentation for both weak and strong augmentation policies. "W/O RandAugment" incorporates both subsampling and vertical shifting in weak and strong augmentation policies, excluding RandAugment. "Full Augmentations" extends the "W/O RandAugment" by additionally including RandAugment in the strong augmentation policy.

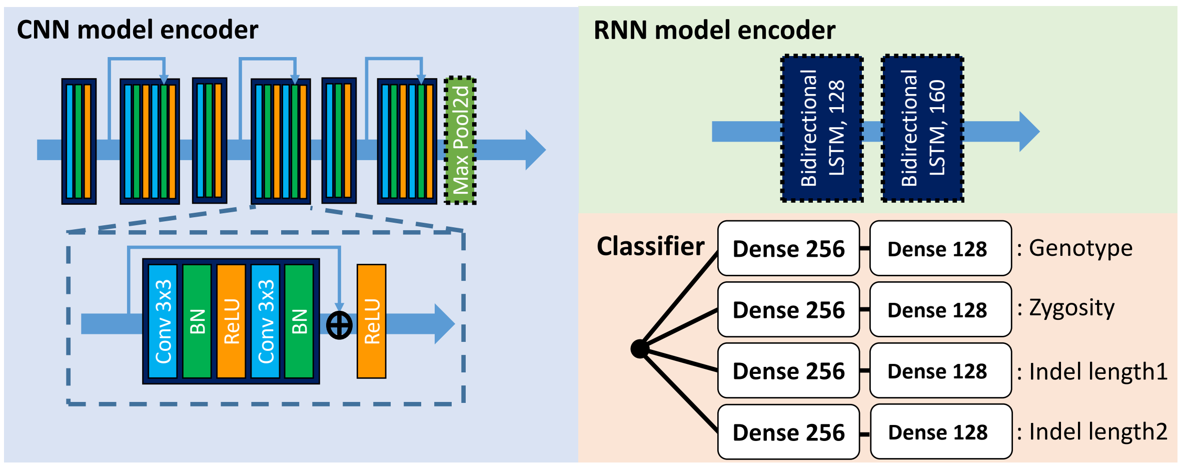
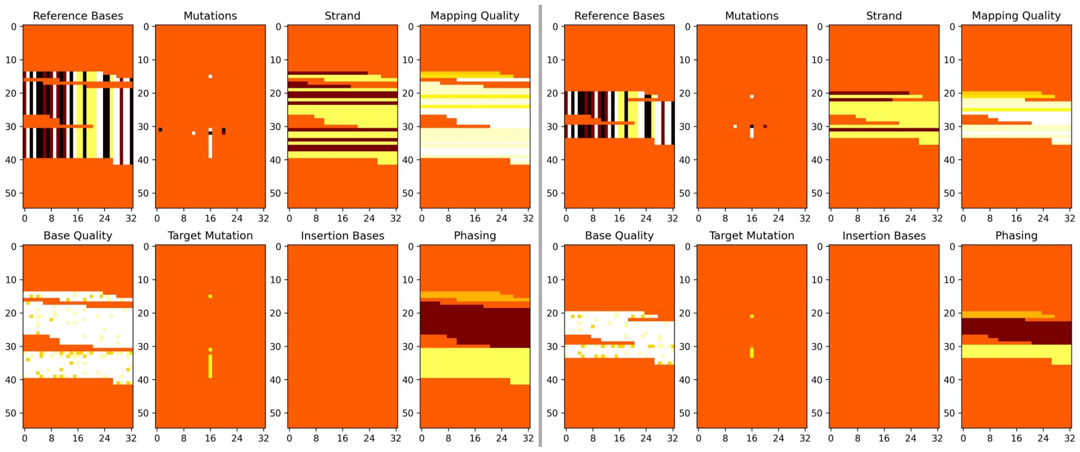


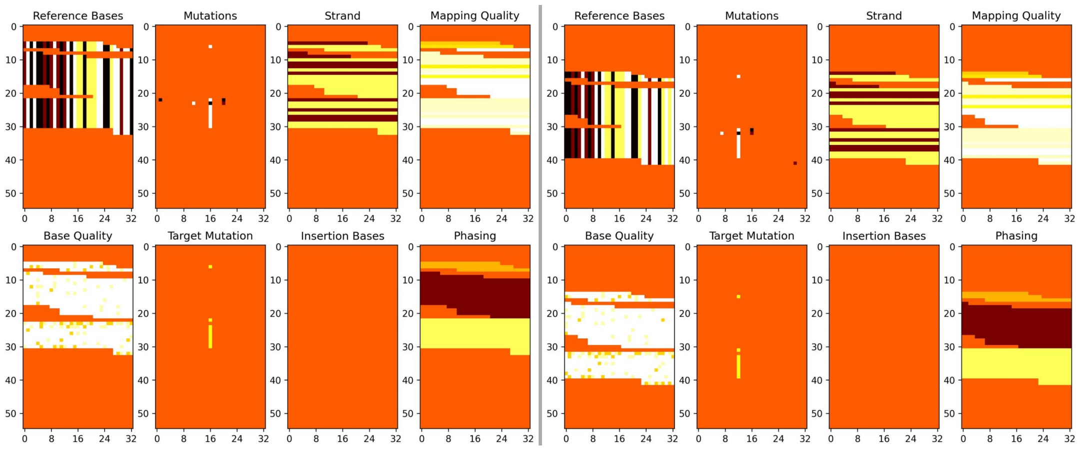
Figure 5.7: Model architecture of the encoder and classifier.

Table 5.20: Performance of RUN-DVC and Baseline methods under various quantities of labeled datasets.

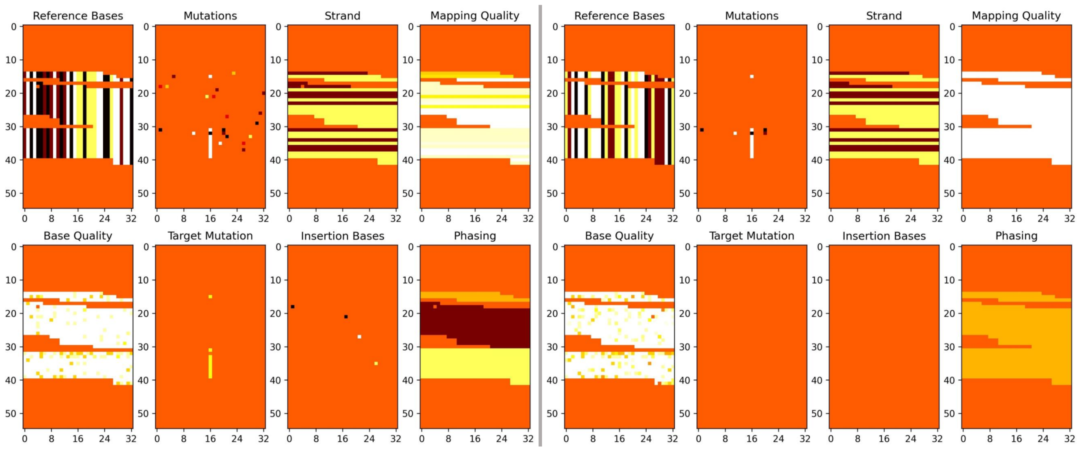
Data	Metric	Method	Number of Labeled Datasets (million)			
			0.4	0.8	1.6	3.2
I-A	Validation Loss	RUNDVC	0.03769	0.035506	0.033981	0.032647
	Validation Loss	BaselineBN	0.044177	0.040084	0.036158	0.033772
	F1-score (INDEL)	RUNDVC	0.959551	0.965844	0.967458	0.970639
	F1-score (INDEL)	Baseline	0.953517	0.959907	0.964962	0.968631
	F1-score (SNP)	RUNDVC	0.993561	0.993741	0.993702	0.993811
	F1-score (SNP)	Baseline	0.993608	0.993665	0.993693	0.993779
I-B	Validation Loss	RUNDVC	0.112521	0.106701	0.10008	0.096151
	Validation Loss	BaselineBN	0.141616	0.118686	0.104717	0.098638
	F1-score (INDEL)	RUNDVC	0.693044	0.701516	0.712803	0.717854
	F1-score (INDEL)	Baseline	0.669771	0.694079	0.710827	0.717897
	F1-score (SNP)	RUNDVC	0.919777	0.920341	0.921952	0.923228
	F1-score (SNP)	Baseline	0.90728	0.914979	0.919946	0.921779
I-A (seed2)	Validation Loss	RUNDVC	0.037763	0.035852	0.034277	0.032966
	Validation Loss	BaselineBN	0.043844	0.039584	0.036641	0.033777
	F1-score (INDEL)	RUNDVC	0.960536	0.96486	0.968501	0.970342
	F1-score (INDEL)	Baseline	0.952172	0.959439	0.965232	0.96901
	F1-score (SNP)	RUNDVC	0.993628	0.993658	0.993729	0.993809
	F1-score (SNP)	Baseline	0.993585	0.993606	0.993663	0.993806
I-B (seed2)	Validation Loss	RUNDVC	0.110411	0.105141	0.100272	0.096536
	Validation Loss	BaselineBN	0.13249	0.118803	0.103799	0.099854
	F1-score (INDEL)	RUNDVC	0.69594	0.705068	0.711312	0.718372
	F1-score (INDEL)	Baseline	0.677013	0.694128	0.710965	0.71763
	F1-score (SNP)	RUNDVC	0.919744	0.921293	0.921882	0.922887
	F1-score (SNP)	Baseline	0.909666	0.91393	0.919964	0.921768



(a) Left: Original, Right: Down sampling



(b) Left: Vertical shift, Right: Horizontal shift



(c) Left: SNP/Indel variants, Right: Distortions to Reference bases, Mapping quality, Base quality, Target variant, and Phasing

Figure 5.8: Overview of data augmentations

Bibliography

[1] Nvidia parabricks retraining tool, 2023. https://catalog.ngc.nvidia.com/orgs/nvidia/collections/claraparabricks/entities.

[2] Nauman Ahmed et al. Heterogeneous hardware/software acceleration of the bwa-mem dna alignment algorithm. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 240–246. IEEE, 2015.

[3] Mian Umair Ahsan, Qian Liu, Li Fang, and Kai Wang. Nanocaller for accurate detection of snps and indels in difficult-to-map regions from long-read sequencing by haplotype-aware deep neural networks. *Genome biology*, 22(1):1–33, 2021.

[4] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[5] Shanika L Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E Ritchie, and Quentin Gouil. Opportunities and challenges in long-read sequencing data analysis. *Genome biology*, 21(1):1–16, 2020.

[6] Gunjan Baid, Maria Nattestad, Alexey Kolesnikov, Sidharth Goel, Howard Yang, Pi-Chuan Chang, and Andrew Carroll. An extensive sequence dataset of gold-standard samples for benchmarking and development. *bioRxiv*, 2020.

[7] Madeleine P Ball, Jason R Bobe, Michael F Chou, Tom Clegg, Preston W Estep, Jeantine E Lunshof, Ward Vandewege, Alexander Wait Zaranek, and George M Church. Harvard personal genome project: lessons from participatory public research. *Genome medicine*, 6(2):1–7, 2014.

[8] David Berthelot, Rebecca Roelofs, Kihyuk Sohn, Nicholas Carlini, and Alex Kurakin. Adamatch: A unified approach to semi-supervised learning and domain adaptation. *arXiv preprint arXiv:2106.04732*, 2021.

[9] Mark J Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC bioinformatics*, 13(1):1–18, 2012.

[10] Nae-Chyun Chen, Alexey Kolesnikov, Sidharth Goel, Taedong Yun, Pi-Chuan Chang, and Andrew Carroll. Improving variant calling using population data and deep learning. *BMC bioinformatics*, 24(1):1–15, 2023.

[11] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[12] Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

[13] Isidro Cortés-Ciriano, Doga C Gulhan, Jake June-Koo Lee, Giorgio EM Melloni, and Peter J Park. Computational analysis of cancer genome sequencing data. *Nature Reviews Genetics*, 23(5):298–314, 2022.

[14] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[15] Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, Adam Gudyś, and Szymon Grabowski. Whisper: read sorting allows robust mapping of dna sequencing data. *Bioinformatics*, 35(12):2043–2050, 2019.

[16] Sebastian Deorowicz and Adam Gudyś. Whisper 2: indel-sensitive short read mapping. *SoftwareX*, 14:100692, 2021.

[17] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. Star: ultrafast universal rna-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[18] Zirui Dong, Xia Zhao, Qiaoling Li, Zhenjun Yang, Yang Xi, Andrei Alexeev, Hanjie Shen, Ou Wang, Jie Ruan, Han Ren, et al. Development of coupling controlled polymerizations by adapter-ligation in mate-pair sequencing for detection of various genomic variants in one single assay. *DNA Research*, 26(4):313–325, 2019.

[19] Michael A Eberle et al. A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *Genome research*, 27(1):157–164, 2017.

[20] Paolo Ferragina and Giovanni Manzini. An experimental study of an opportunistic index. In *SODA*, pages 269–278, 2001.

[21] Steven R Head, H Kiyomi Komori, Sarah A LaMere, Thomas Whisenant, Filip Van Nieuwerburgh, Daniel R Salomon, and Phillip Ordoukhanian. Library construction for next-generation sequencing: overviews and challenges. *Biotechniques*, 56(2):61–77, 2014.

[22] Darryl Ho et al. Lisa: towards learned dna sequence search. *arXiv preprint arXiv:1910.04728*, 2019.

[23] Darryl Ho et al. Lisa: Learned indexes for sequence analysis. *bioRxiv*, pages 2020–12, 2021.

[24] Carl Maximilian Hommelsheim, Lamprinos Frantzeskakis, Mengmeng Huang, and Bekir Ülker. Pcr amplification of repetitive dna: a limitation to genome editing technologies and many other applications. *Scientific reports*, 4(1):5052, 2014.

[25] Ernst Joachim Houtgast et al. Hardware acceleration of bwa-mem genomic short read mapping for longer read lengths. *Computational biology and chemistry*, 75:54–64, 2018.

[26] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[27] W James Kent. Blat—the blast-like alignment tool. *Genome research*, 12(4):656–664, 2002.

[28] Gelana Khazeeva, Karolis Sablauskas, Bart van der Sanden, Wouter Steyaert, Michael Kwint, Dmitrijs Rots, Max Hinne, Marcel van Gerven, Helger Yntema, Lisenka Vissers, et al. Denovocnn: a deep learning approach to de novo variant calling in next generation sequencing data. *Nucleic acids research*, 50(17):e97–e97, 2022.

[29] Hyunbin Kim, Andy Jinseok Lee, Jongkeun Lee, Hyonho Chun, Young Seok Ju, and Dongwan Hong. Firevat: finding reliable variants without artifacts in human cancer samples using etiologically relevant mutational signatures. *Genome Medicine*, 11:1–17, 2019.

[30] Jong-Il Kim, Young Seok Ju, Hansoo Park, Sheehyun Kim, Seonwook Lee, Jae-Hyuk Yi, Joann Mudge, Neil A Miller, Dongwan Hong, Callum J Bell, et al. A highly annotated whole-genome sequence of a korean individual. *nature*, 460(7258):1011–1015, 2009.

[31] Sangtae Kim, Konrad Scheffler, Aaron L Halpern, Mitchell A Bekritsky, Eunho Noh, Morten Källberg, Xiaoyu Chen, Yeonbin Kim, Doruk Beyter, Peter Krusche, et al. Strelka2: fast and accurate calling of germline and somatic variants. *Nature methods*, 15(8):591–594, 2018.

[32] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. Sosd: A benchmark for learned indexes. *NeurIPS Workshop on Machine Learning for Systems*, 2019.

[33] Melanie Kirsche et al. Sapling: accelerating suffix array queries with learned data models. *Bioinformatics*, 37(6):744–749, 2021.

[34] Daniel C Koboldt. Best practices for variant calling in clinical sequencing. *Genome Medicine*, 12(1):1–13, 2020.

[35] Alexey Kolesnikov, Sidharth Goel, Maria Nattestad, Taedong Yun, Gunjan Baid, Howard Yang, Cory Y McLean, Pi-Chuan Chang, and Andrew Carroll. Deeptrio: variant calling in families using deep learning. *bioRxiv*, 2021.

[36] Tim Kraska et al. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[37] Kiran Krishnamachari, Dylan Lu, Alexander Swift-Scott, Anuar Yeraliyev, Kayla Lee, Weitai Huang, Sim Ngak Leng, and Anders Jacobsen Skanderup. Accurate somatic variant detection using weakly supervised deep learning. *Nature Communications*, 13(1):4248, 2022.

[38] Peter Krusche, Len Trigg, Paul C Boutros, Christopher E Mason, Francisco M De La Vega, Benjamin L Moore, Mar Gonzalez-Porta, Michael A Eberle, Zivana Tezak, Samir Lababidi, et al. Best practices for benchmarking germline small-variant calls in human genomes. *Nature biotechnology*, 37(5):555–560, 2019.

[39] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[40] Heng Li. wgsim-read simulator for next generation sequencing. *Github repository*, 2011.

[41] Heng Li. Exploring single-sample snp and indel calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012.

[42] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[43] Heng Li. Toward better understanding of artifacts in variant calling from high-coverage samples. *Bioinformatics*, 30(20):2843–2851, 2014.

[44] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[45] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[46] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[47] Ruiqiang Li et al. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[48] Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. debga: read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.

[49] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[50] Yongchao Liu et al. Cushaw: a cuda compatible short read aligner to large genomes based on the burrows–wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[51] Yongchao Liu and Bertil Schmidt. Long read alignment based on maximal exact match seeds. *Bioinformatics*, 28(18):i318–i324, 2012.

[52] Ruibang Luo, Fritz J Sedlazeck, Tak-Wah Lam, and Michael C Schatz. A multi-task convolutional deep neural network for variant calling in single molecule sequencing. *Nature communications*, 10(1):1–11, 2019.

[53] Ruibang Luo, Chak-Lim Wong, Yat-Sing Wong, Chi-Ian Tang, Chi-Man Liu, Chi-Ming Leung, and Tak-Wah Lam. Exploring the limit of using a deep neural network on pileup data for germline variant calling. *Nature Machine Intelligence*, 2(4):220–227, 2020.

[54] Guillaume Marçais, Arthur L Delcher, Adam M Phillippy, Rachel Coston, Steven L Salzberg, and Aleksey Zimin. Mummer4: A fast and versatile genome alignment system. *PLoS computational biology*, 14(1):e1005944, 2018.

[55] Ryan Marcus et al. Cdfshop: Exploring and optimizing learned index structures. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2789–2792, 2020.

[56] Jing Meng, Brandon Victor, Zhen He, Hongde Liu, and Taijiao Jiang. Deepssv: detecting somatic small variants in paired tumor and normal sequencing data with convolutional neural network. *Briefings in Bioinformatics*, 22(4):bbaa272, 2021.

[57] Michael L Metzker. Sequencing technologies—the next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

[58] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, et al. A universal snp and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10):983–987, 2018.

[59] Alon Rashelbach et al. A computational approach to packet classification. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 542–556, 2020.

[60] Sayed Mohammad Ebrahim Sahraeian, Ruolin Liu, Bayo Lau, Karl Podesta, Marghoob Mohiyuddin, and Hugo YK Lam. Deep convolutional neural networks for accurate somatic mutation detection. *Nature communications*, 10(1):1–10, 2019.

[61] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, Trevor Darrell, and Kate Saenko. Semi-supervised domain adaptation via minimax entropy. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8050–8058, 2019.

[62] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, and Kate Saenko. Universal domain adaptation through self supervision. *Advances in neural information processing systems*, 33:16282–16292, 2020.

[63] Kishwar Shafin, Trevor Pesout, Pi-Chuan Chang, Maria Nattestad, Alexey Kolesnikov, Sidharth Goel, Gunjan Baid, Mikhail Kolmogorov, Jordan M Eizenga, Karen H Miga, et al. Haplotype-aware variant calling with pepper-margin-deepvariant enables high accuracy in nanopore long-reads. *Nature methods*, 18(11):1322–1332, 2021.

[64] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33:596–608, 2020.

[65] Arun Subramaniyan et al. Accelerated seeding for genome sequence alignment with enumerated radix trees. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 388–401. IEEE, 2021.

[66] Adrian Tan, Gonçalo R Abecasis, and Hyun Min Kang. Unified representation of genetic variants. *Bioinformatics*, 31(13):2202–2204, 2015.

[67] Ole Tange et al. Gnu parallel-the command-line power tool. *The USENIX Magazine*, 36(1):42–47, 2011.

[68] Joaquín Tárraga et al. Acceleration of short and long dna read mapping without loss of accuracy using suffix array. *Bioinformatics*, 30(23):3396–3398, 2014.

[69] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in neural information processing systems*, 30, 2017.

[70] Brett Trost, Susan Walker, Syed A Haider, Wilson WL Sung, Sergio Pereira, Charly L Phillips, Edward J Higginbotham, Lisa J Strug, Charlotte Nguyen, Akshaya Raajkumar, et al. Impact of dna source on genetic variant detection from human whole-genome sequencing data. *Journal of medical genetics*, 56(12):809–817, 2019.

[71] Md. Vasimuddin et al. Efficient architecture-aware acceleration of bwa-mem for multicore systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 314–324, 2019.

[72] Sergey Vilov and Matthias Heinig. Deepsom: a cnn-based approach to somatic variant calling in wgs samples without a matched normal. *Bioinformatics*, 39(1):btac828, 2023.

[73] Michael Vyverman, Bernard De Baets, Veerle Fack, and Peter Dawyndt. Prospects and limitations of full-text index structures in genome analysis. *Nucleic acids research*, 40(15):6993–7015, 2012.

[74] Justin Wagner, Nathan D. Olson, Lindsay Harris, Ziad Khan, Jesse Farek, Medhat Mahmoud, Ana Stankovic, Vladimir Kovacevic, Byunggil Yoo, Neil Miller, Jeffrey A. Rosenfeld, Bohan Ni, Samantha Zarate, Melanie Kirsche, Sergey Aganezov, Michael C. Schatz, Giuseppe Narzisi, Marta Byrska-Bishop, Wayne Clarke, Uday S. Evani, Charles Markello, Kishwar Shafin, Xin Zhou, Arend Sidow, Vikas Bansal, Peter Ebert, Tobias Marschall, Peter Lansdorp, Vincent Hanlon, Carl-Adam Mattsson, Alvaro Martinez Barrio, Ian T. Fiddes, Chunlin Xiao, Arkarachai Fungtammasan, Chen-Shan Chin, Aaron M. Wenger, William J. Rowell, Fritz J. Sedlazeck, Andrew Carroll, Marc Salit, and Justin M. Zook. Benchmarking challenging small variants with linked and long reads. *Cell Genomics*, 2(5):100128, 2022.

[75] Ting Wang, Lucinda Antonacci-Fulton, Kerstin Howe, Heather A Lawson, Julian K Lucas, Adam M Phillippy, Alice B Popejoy, Mobin Asri, Caryn Carson, Mark JP Chaisson, et al. The human pangenome project: a global resource to map genomic diversity. *Nature*, 604(7906):437–446, 2022.

[76] Youyun Wang et al. Sindex: a scalable learned index for string keys. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 17–24, 2020.

[77] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, 33:6256–6268, 2020.

[78] Zhenxian Zheng, Shumin Li, Junhao Su, Amy Wing-Sze Leung, Tak-Wah Lam, and Ruibang Luo. Symphonizing pileup and full-alignment for deep learning-based long-read variant calling. *Nature Computational Science*, 2(12):797–803, 2022.

[79] Justin M Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E Mason, Noah Alexander, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific data*, 3(1):1–26, 2016.

[80] Justin M Zook, Brad Chapman, Jason Wang, David Mittelman, Oliver Hofmann, Winston Hide, and Marc Salit. Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls. *Nature biotechnology*, 32(3):246–251, 2014.

[81] Justin M Zook, Nancy F Hansen, Nathan D Olson, Lesley Chapman, James C Mullikin, Chunlin Xiao, Stephen Sherry, Sergey Koren, Adam M Phillippy, Paul C Boutros, et al. A robust benchmark for detection of germline large deletions and insertions. *Nature biotechnology*, 38(11):1347–1355, 2020.

[82] Justin M Zook, Jennifer McDaniel, Nathan D Olson, Justin Wagner, Hemang Parikh, Haynes Heaton, Sean A Irvine, Len Trigg, Rebecca Truty, Cory Y McLean, et al. An open resource for accurately benchmarking small variant and reference calls. *Nature biotechnology*, 37(5):561–566, 2019.