Quality Scores Compression of Genomic Sequencing Data: A Comprehensive Review and Performance Evaluation

Yuansheng Liu, Tao Tang, Zexuan Zhu, Xiangxiang Zeng, Quan Zou, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

(Survey/Tutorial Paper)

***Abstract*—Advanced sequencing technologies have profoundly revolutionized biology and produced vast amounts of raw sequencing data during the past decades. The enormous amount of sequencing data proposed significant challenges of data storage and transmission. Compressing a big file into a small file is an encouraging method to tackle these challenges. Howver, it has been found that traditional text data compression algorithms are not well-suited for handing the vast sequencing datasets. Therefore, several algorithms are designed specifically for the efficient compression of sequencing data. Recently, considerable research has been devoted to compressing quality scores stored in the FASTQ format file, resulting in substantial advances in compression performance. Despite these advances, there has been no systematic review and evaluation of these algorithms or software. In this review, we aim to conduct a broad review of the existing quality score compression algorithms. We mainly discuss those algorithms from two categories, i.e., lossless and lossy compression. Additionally, we benchmark the compression performance of 12 tools using 14 real datasets. We anticipate that our review will provide practical guidance for others seeking to design an appropriate algorithm for compressing quality scores.**

Received 10 October 2022; revised 29 May 2023; accepted 3 February 2025. Date of publication 7 February 2025; date of current version 3 April 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62102140, Grant 62202236, Grant 62372159, Grant 62425204, Grant 62122025, Grant U22A2037, Grant 62450002, Grant 62432011, and Grant 62272151, in part by the Science and Technology Innovation Program of Hunan Province under Grant 2022RC1100, and in part by Hunan Provincial Natural Science Foundation of China under Grant 2021JJ10020. *(Yuansheng Liu and Tao Tang contributed equally to this work.) (Corresponding author: Xiangxiang Zeng.)*

Yuansheng Liu and Xiangxiang Zeng are with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China (e-mail: yuanshengliu@hnu.edu.cn; xzeng@hnu.edu.cn).

Tao Tang is with the School of Mordern Posts, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210049, China (e-mail: tangtao@njupt.edu.cn).

Zexuan Zhu is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: zhuzx@szu.edu.cn).

Quan Zou is with the Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chengdu 610054, China (e-mail: zouquan@nclab.net).

Keqin Li is with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China, also with the National Supercomputing Center, Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of NY, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TCBBIO.2025.3539629

***Index Terms*—Lossless compression, lossy compression, quality scores compression, sequencing data compression.**

I. Introduction

SEQUENCING technology has revolutionized biological research, contributing significantly to the current genomic revolution [1], [2]. The falling costs and the increased throughput of sequencing (https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost) have driven worldwide DNA sequencing companies to produce unprecedented volumes of genomic data [3], [4]. For instance, the National Institutes of Health's Sequence Read Archive [5] and the European Nucleotide Archive (ENA) [6] have accumulated approximately ten quadrillion bases over the past years. The 100,000 Genomes Project has already stored 21 PetaBytes of data. The NovaSeq 6000 sequencing system, new sequencing system, can generate about 20 billion reads and FASTQ format files of up to 6 TeraByte within two days. Storing and sharing these large-scale sequencing data with multiple institutions is essential for gaining deeper insights into genomic structures, functions, and evolutions [7], [8]. Moreover, these sequencing data are uniquely generated as the samples used for sequencing are not allowed for resequencing. The ever-increasing amount of sequencing data presents new challenges in data storage and transmission [9]. In the past decade, the growth rate of sequencing data has exceeded the drop rate of hardware prices. Furthermore, transmitting large files takes longer time, leading to inefficient cooperation in analysis and research. Therefore, an urgent need exists for a sophisticated solution to reduce the cost of large-scale data storage and transmission time [10], [11].

The FASTQ file format is commonly used to store the raw sequencing data. Each entry in the FASTQ file comprises four lines, which include identifies, nucleotide sequence, the symbol '+', and the quality scores. An example of an entry is provided in Fig. 1. To address the challenges associated with storing and transmitting large amounts of sequencing data, traditional text data compression tools are usually used to compress sequencing data. For instance, gzip (https://www.gzip.org/) is used by the ENA. However, these tools do not effectively exploit intrinsic features such as repetitive subsequences and small alphabet size

```
@SRR635193.1 1/1
CTGCTGANGAAATATGTAAAGCTATTCATGTATTTGACTTGAAGACACTGCCTG
+
F>5E8-8#-=B?4?8@HBHBEHHHHEGGG:GGFGGBBGGGDGGBGGEGGDBAE?
@SRR635193.2 2/1
CCGGTCCNAGTCTTCCATCAGCCTTGCAATTTCAGCATCCTGCAGCTTCGAGCC
+
HHGHHB?#BCBDEDDGGG>GHHHHHHHHHHHHHBHHDHHFGD@GGDGBGGAGAB
```

Fig. 1. A real example of two entries in the FASTQ format file from the raw sequencing file "SRR635193_1.fastq". The first and fifth lines are the identifiers; the second and sixth lines are the DNA sequence; and the fourth and eighth lines are quality scores.

found in sequencing data, resulting in suboptimal compression ratios [12], [13]. In contrast, several algorithms specially designed for sequencing data have achieved high compression ratios [8], [14], [15], [16]. For instance, while the file size of "SRR5220205_1.fastq" is 41 GB, the gzip-compressed file size downloaded from ENA is 9.1 GB. Furthermore, the state-of-the-art tool SPRING [17] compressed it to 3.2 GB, offering a much better compression ratio by utilizing repeat fragments in short reads. As only one symbol is generally contained in the third line, FASTQ file compression typically involves compressing three distinct data streams [18]: identifiers, sequence, and quality scores. Advanced technologies are proposed to compress different parts of reads data, including tokenization algorithms for identifiers [19], and reference-based and *de novo* algorithms for reads sequence compression [17], [20], [21], [22]. Furthermore, over the past decade, more than 30 algorithms have been proposed to compress quality scores, resulting in significant improvements in compression performance [8], [23]. A timeline of these algorithms is depicted in Fig. 2. Despite these advancements, there is still a lack of comparative analysis of these algorithms. Therefore, this study aims to review and evaluate these algorithms.

The quality score of sequencing data indicates the probability that the nucleotide base it is related to has been incorrectly identified during the sequencing process. This probability, denoted as P, is often converted into an integer Phred score, $Q = -10 \times \log_{10} P$. The Phred scores are stored in the FASTQ file as ASCII alphabets within the range $[33 : 73]$ or $[64 : 104]$, which are typically scaled by $Q + 33$ or $Q + 64$. Some sequencing machines have introduced to reduce the resolution of the quality scores. For example, the HiSeq X uses only eight available quality scores, while the NovaSeq FASTQs have only four [21]. Lower Q scores indicate a higher probability of error, which can negatively affect downstream analysis [24]. In a FASTQ format file, as every nucleotide base has one quality score, the number of quality scores is the same as the number of nucleotide bases in the reads. However, compressing quality scores is more complicated than compressing reads sequence due to their higher entropy and larger alphabet [25]. Additionally, no reference sequence can be used for quality score compression [26], making reference-based compression algorithm unsuitable. As demonstrated in [27], the size of gzip-compressed quality scores is four times larger than that of the gzip-compressed reads.

In the field of compression, there are two categories of methods: lossless and lossy. Lossless compression algorithm, such as 7-zip (https://www.7-zip.org/) and bzip2 (http://www.bzip.org/), are commonly used for text data compression. This method produces a decompressed output that is identical to the original data, preserving enough information to restore all the compressed information. Lossless compression is widely employed for genomic data compression including reads compression [17], genome sequence compression [28], and quality score compression [29].

In contrast, lossy compression scarifies part of the data to achieve an outstanding compression ratio. In general, the compressed file size of lossy compression method is significantly smaller than that of using the lossless compression method. However, the original data cannot be recovered exactly. Multimedia data such as images and videos [30] are typically compressed using lossy compression algorithms. However, lossy compression is not suitable for nucleobases since they store essential genetic information. Quality score, which indicate the confidence level of each nucleobase, have a higher resolution than necessary for downstream analysis. Hence, numerous lossy compression methods have been developed to compress quality scores of sequencing data. However, the loss of information may affect downstream applications such as Single Nucleotide Polymorphism (SNP) genotyping [24]. The impact of existing lossy methods on downstream applications has been evaluated in previous studies [31], [32].

This article introduces the compression of four different format files in genomic sequencing data and provides a comprehensive survey of existing quality score compression algorithms. Lossless and lossy compression algorithms for quality scores are summarized, and the latter is clustered into different groups based on their rationales and features. The compression performance of these algorithms, including compression ratio, running time, and memory usage, is evaluated using open source scripts available at https://github.com/ttan6729/qsc-review.

II. Genomic Sequencing Data Compression

Genomic sequencing data is typically stored in one of four file formats, as illustrated in Fig. 3. In the next-generation sequencing technology [33], the outputs are millions of short fragments having the same lengths about hundred bases. These fragments, called reads, are generally considered to be irregularly sampled substrings of the DNA sequence. Third-generation sequencing method, on the other hand, generates fewer but longer reads, ranging from 100 to 1000 bases in length. The raw sequencing data is stored in the FASTQ format file [34].

To analyze the sequencing results, the reads in the FASTQ file are aligned to the reference genome using tools such as BWA-MEM [35] and Bowtie2 [36]. The alignment results are then saved in the SAM (short for sequence alignment map) format file [37]. Since SAM files can be very large, with a human genome SAM file at $30\times$ coverage taking up about 90 GB, the binary BAM format file is preferred in downstream applications as it is much smaller.

After alignment, variant calling is used to identify differences between the reads and the reference genome [38], and the results are stored in the VCF (short for variant call format) format file [39]. A binary BCF format file, which is more effecient for

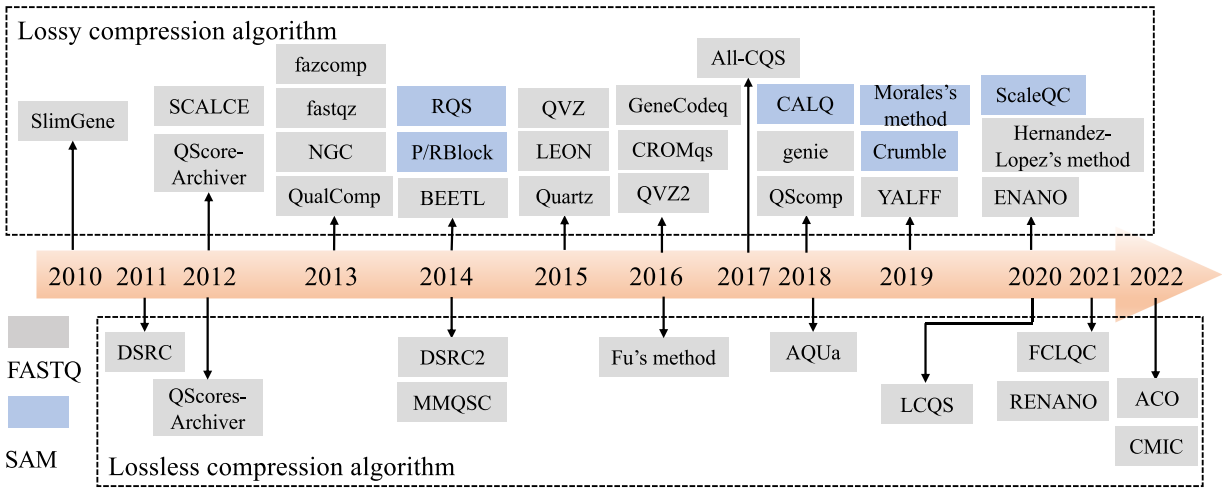


Fig. 2. A time line of algorithms for quality score compression.

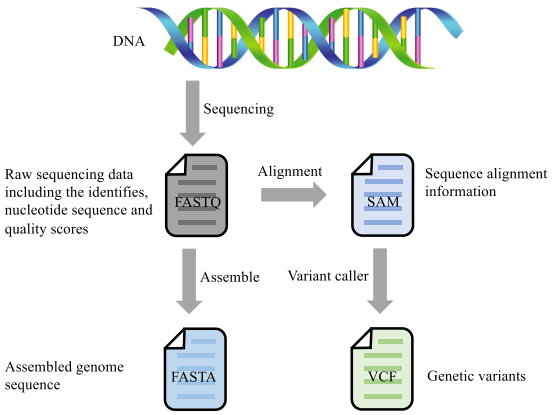


Fig. 3. The typical file formats used to store sequencing data. The raw data sequenced from DNA is stored in the FASTQ format file. The aligned data is stored in the SAM format file. The variants are stored in the VCF format file. The assembled genome sequences are stored in the FASTA format file.

storage and querying, can also be used to store the information in the VCF file.

Additionally, short reads can be assembled into the genome sequences [40], which are saved in the FASTA format file. For instance, it requires approximately 3 GB of storage to store 24 sequences of the human genome.

As alignment, variant calling, and assembly technologies evolve, the above four format files are necessary for reanalyzing genomic sequencing data. Therefore, there is a growing interest in genomic sequencing data compression from these four file formats.

A. *Raw Sequencing Data Compression*

The FASTQ format is the most commonly used format for storing sequencing data, and many algorithms have been proposed to compress FASTQ files over the past two decades [41]. DNA sequences and the quality scores contain the majority of sequencing information, and therefore, most algorithms focused on compressing these two parts. The *de novo* compression algorithm attracts more attention for DNA sequence compression becasuse it has achieved better performance than the reference-based compression algorithm [22]. Reordering reads has significantly improved the compression ratio, and most of prevalent algorithms specially designed for sequence compression are composed of read reordering and contig assembly [17], [22], [42]. For instance, PgRC [43] assembles reads into a pseudogenome and achieved the highest compression ratio on all datasets. The most recent algorithm, Mstcom [44], stores the minimum spanning tree over the reads overlap graph, and uses multiple minimizers to search similar reads [45]. Mstcom performs better than PgRC in terms of compression ratio, but neither is capable of compressing the whole FASTQ file. In practice, SPRING [17] and FaStore [21] are the most popular tools as they have stable performance in speed, memory usage, and compression ratio.

The third-generation sequencing technology produces long reads, which has several advantages over short reads [46] and are widely used in the analysis [47]. The tools designed specifically for short reads cannot be directly applied to long reads due to inherent differences in their features, such as read length, sequencing depth, and error characteristics. For example, PgRC and Minicom [22] can not compress reads having different lengths, and only SPRING supports compression of long reads by storing lengths of every read. Furthermore, some algorithms [48], [49] are developed to compress FASTQ files generated by nanopore sequencing. The reference-based tool, RENANO [50], achieved the best compression ratio.

B. *Aligned Sequencing Data Compression*

Aligned sequencing data is stored in the SAM format file, which is tab-delimited and was developed in the 1000 Genomes Project. The SAM file contains of two parts: header and alignment. The alignment part consists of 11 mandatory fields and some optional fields. Since the SAM file not only stores information in FASTQ but also stores the alignment information, its compression is more challenging. BAM is obtained by using

general-purpose compression algorithms, but its compression ratio can not resolve the major bottleneck. CRAM [51], [52] is a reference-based compression algorithm that reduces the file size by over 30% and is 1.8 times faster than BAM. Therefore, it has become the preferred format for ENA submission. In the past decade, several tools were developed to compress SAM file [27], [53], [54], [55], with most of them focusing on improving the compression performance for the SEQ and QUAL fields as they take up most of the space in the whole file. For example, GeneComp [54] and DeeZ [53] both use a reference genome to boost the compression of DNA sequences.

As the file size of the SAM file is large, querying in the compressed file without decompressing the whole compressed file is of paramount importance. For instance, CSAM [27] supports two random access operations in the compressed file. Downstream applications can selectively extract data to save resources.

C. Variants Compression

The data format downloaded from the 1000 Genomes Project is VCF. This file format stores the genetic variants of a sample, which are obtained by comparison to a reference genome. A VCF file also consists of header and body parts. In the body section, each row includes eight mandatory fields, as well as additional fields. TGC [56] was the first tool designed to compress a collection of VCF files. Genozip [57] is the most recent and achieves the best compression ratio on a VCF file when compared to other existing tools. During downstream analyses, it is often necessary to query all samples that have a common variant or all variants of specific samples. Efficient querying of variants is of utmost significance. However, TGC and genozip are not capable of querying in their compressed representation. To overcome this limitation, several compression algorithms have been proposed, such as BGT [58], GTRAC [59], GTC [60], and GTShark [61], which support random access. The compression ratio of GTC is slightly worse than TGC, TGC and GTC achieved compression sizes of 400 MB and 600 MB respectively, when compressing the collection of 1000 Genomes Project. GTC offers a fast query for variants. GTC offers a fast query for variants, which is critical for downstream analyses.

D. Assembled Sequence Compression

With the rapid development of sequencing technology and assembly algorithm, a large number of assembled genomes are expected to be presented in the near future. Special algorithms have been designed for genome compression, which can be classified into two categories: reference-free and reference-based genome compression, based on whether a reference genome is used or not.

Specialized DNA sequence compression tools have made substantial improvement compared to general text compression tools. These algorithms are divided into different types, such as substitution-based method, including GenCompress [62], DNACompress [63] and BioCompression [64] (see the reviews in [12], [65]). These tools identify repeats in genome sequence and make use of intrinsic features of genomic sequences. Recently, neural network have been widely used in many fields of bioinformatics [66], [67], [68], [69], [70], [71]. Neural networks also have been applied in the compression of DNA sequences. For instance, DeepDNA [72] uses neural networks to extract features in genome sequence and to predict the next base, while GeCo3 [73] employs a neural network to combine various context models.

Reference-based genome compression algorithm always achieve better compression ratios than reference-free algorithms since they utilize a genome of the same species or variation data as side information and only store the differences. This is based on the assumption that two genomes of the same species are very similar. For the human genome, only 1% of differences are needed to store if given the reference genome. The first reference-based algorithm is DNAZip [74], which compressed the genome of James Watson [75] to a size of 4 MB. The genome was further compressed into 2.5 MB [76]. However, they heavily rely on SNP data, which can be challenging to obtain. In recent years, several reference-based genome compression algorithms that only use a reference genome have been proposed in recent years. Notable examples GeCo [77], iDoComp [78], ERGC [79], and HiRGC [80], which achieved great performance. The idea of reference-based compression has also been applied in some other data compression, such as trajectory data [81], [82], public transit schedules [83]. The most recent tool, MemRGC, has the best compression ratio on the benchmarking genomes of various species. Its superior performance is contributed by mutation-containing matches and k-mer sampling scheme [84]. More challenging work is to compress the genome data set [85]. In particular, the two-level encoding framework proposed by GDC-2 [86] achieved a very high compression ratio and aws further investigated in [87].

III. Quality Score Compression

A. Lossless Compression Algorithms

The use of lossless compression algorithm is widespread across several fields. Traditional text data compression tools, such as 7zip (www.7-zip.org) and bzip2, can compress quality scores streams while achieving lossless compression. However, these compression algorithms are unable to take advantage of the special patterns in quality scores, thereby failling to satisfy the compression ratio requirement for large-scale datasets. Consequently, specific algorithms have been proposed to achieve a higher compression ratio. We summarize these algorithms in Table I.

Some patterns can be used in the uneven distribution of quality score symbols lines. For example, Deorowicz and Grabowski [88] analyzed the symbol distribution and proposed the tool DSRC. They found that the distribution of score symbols did not heavily rely on position, and the last quality symbol in most of the lines is '#'. In addition, strong correlations were observed between neighboring scores. DSRC employed suffix extraction, run-length encoding and different order Huffman encoding to compress quality scores according to different cases. In the improved version DSRC2, two distinct methods are proposed. The first method is similar to DSRC, using only the

TABLE I
DETAILS OF LOSSLESS COMPRESSION ALGORITHMS FOR QUALITY SCORES COMPRESSION

Categories	Method	The main novelty	Compared to	Some details
Distribution-based	DSRC [88]	Three observations: 1) symbol distribution; 2) ending with '#' symbol; 3) local correlations	gzip, bzip2, G-SQZ [89]	In case 1) and 2), detect and remove the trailing hashes; order-0 Huffman encoder. In case 3), run-length encoding is used; order-1 Huffman encoder
	DSRC2 [18]	-	pigz, bzip2, Quip, Fqzcomp, DSRC, SeqDB [90]	1) similar to DSRC but only using order-1 Huffman encoder; 2) arithmetic coding (maximum context length is 6)
	Fu's method [91]	Reorder quality score lines to change the distribution; Linear combination prediction	gzip, bzip2, KIC, SCALCE	Cluster quality score lines; the centroid is determined by sampling part of quality scores; three lines are used to predict
	LCQS [92]	k-mer frequency-based adaptive packing method	gzip, 7-zip, AQUa	ZPAQ (http://mattmahoney.net/dc/zpaq.html)
Replacement-based	QScores-Archiver [29]	Three lossless transformations	-	MinShifting: minus the minimum value; FreqOrdering: reorder scores in decreasing order and remap scores to their order; GapTranslating: gaps of neighboring scores
	MMQSC [93]	Using memetic algorithm to optimize the construction of codebook	RLE, Huffman, gzip, bzip2, LZMA	Code vector is obtained by using Huffman coding; quality score is compressed as the index of similar code vector and differences of two symbol strings
Others	AQUa [94]	adaptive binary arithmetic coding; support random access	gzip, 7-zip, SCALCE, QVZ	Difference coder; Average difference coder; Convolutional predictor; Single repeat predictor; Average predictor; Normal search predictor; Hierarchical normal search predictor
	ScaleQC [95]	Bit-plane coding	CRAM, DeeZ, TSC, DSRC, P/R-Block, Quartz, LEON, QVZ2, CALQ, Crumble	Binary arithmetic code
	LEON [96]	-	gzip, DSRC, fqzcomp, fastqz, LEON, SCALCE, Quip, Mince [97], ORCOM [98]	zlib
	NGC [99]	-	CRAM, Goby [100]	bzip2 with best compression option '-9'
	Fqzcomp [101]	-	gzip, bzip2, SCALCE, DSRC, Quip	byte-wise arithmetic coder (http://ctxmodel.net)
	Fastqz [101]	-	gzip, bzip2, SCALCE, DSRC, Quip	ZPAQ

order-1 Huffman coding in two different cases. The second one uses arithmetic coding to compress quality scores. Fu et al. [91] aimed to modify the distribution of quality score symbols by reordering the quality scores lines. They clustered similar quality score lines into the same group and obtained the centroid of each cluster by sampling part of quality score lines. To utilize the high correlation among different quality score lines within a cluster, they adopted a simple linear combination method to predict the next line. Finally, the context mixing probabilistic modeling algorithm ZPAQ (http://mattmahoney.net/dc/zpaq.html) was employed to compress each cluster. In the LCQS process [92], quality score lines are partitioned into two groups by using k-mer (k-length substring) of quality and their frequency, and a k-mer frequency-based adaptive packing method is applied to reduce content. The packed stream is then compressed by ZPAQ. ACO [102] proposed a scanning order that traverses in an adaptive direction by considering the content to visit the quality score.

Some algorithms aim to map original quality scores to new values with reduced entropy, such as the replacement-based method. For instance, QScores-Archiver [29] aimed to reduce the number of different quality score symbols by proposing three lossless transformation strategies to modify the quality scores to lower values. MMQSC [93] is a codebook-based compression tool [103] that proposed a memetic algorithm to optimize the construction of codebook that stores the Huffman coded vector. Quality score is replaced by the index of their most similar Huffman coded vector and their differences. It aims to implement encoding of the most frequency occurring short score substrings using the shorter code. First, Huffman coding is used to encode raw quality scores, and the coding bits are converted into readable ASCII characters. Then, an evolutionary algorithm is used to optimize the construction of the codebook. Finally, a matching algorithm using dynamic programming is utilized to obtain exact information about differences.

In contrast to distribution-based and replacement-based methods, AQUa [94] employs multiple encoding methods to compress various redundancy patterns among quality score lines. Within each block, one of seven different coding tools is selected based on . These tools are specially designed to accomplish this task. To facilitate random access, FCLQC [104] splited the file

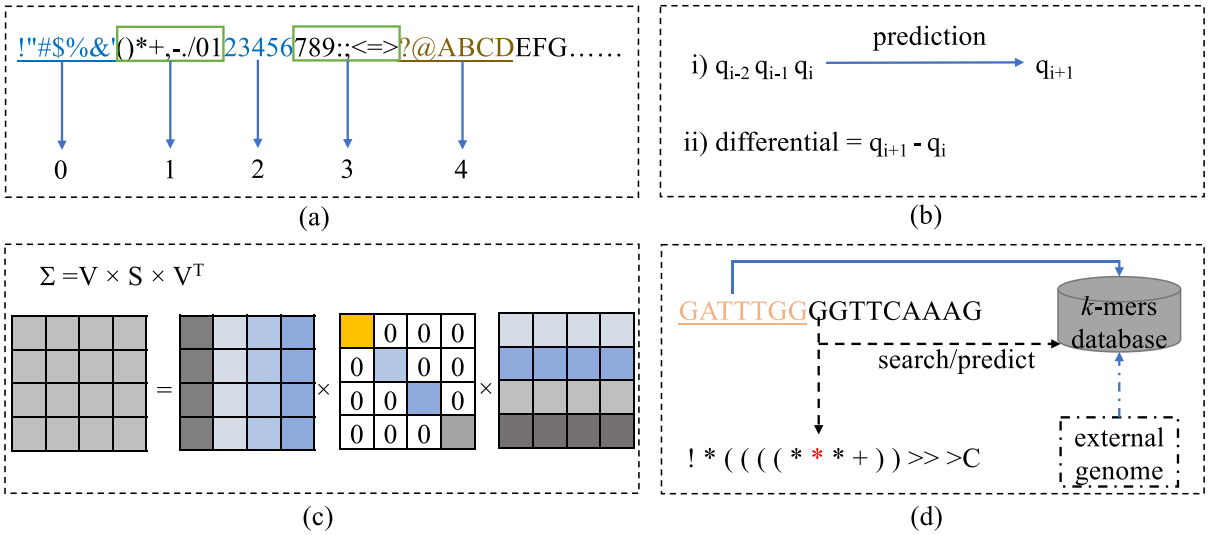


Fig. 4. Illustration of four categories lossy compression algorithm. (a) Rebinning-based method. Original quality scores are changed to small number of new scores. Some quality scores are combined into the same bin. (b) Context-base method. Neighboring values are used to predict the next one or calculate the difference for next step. (c) Singular value decomposition (SVD)-based method. SVD is performed on the covariance matrix of quality score lines. (d) Base information-based method. The quality score is determined on the prediction of the corresponding base. Also k-mers of reference genome are integrated in some methods.

into sub-files and utilizes an adaptive arithmetic coding method to encode them.

Other tools such as ScaleQC [95], LEON [96], NGC [99], and Fastqz and Fqzcomp [101], primarily concentrate on lossy compression, but they also provide support for lossless quality scores compression by using existing traditional text data compression algorithms. For instance, NGC utilizes bzip2 in lossless mode with the best compression parameter settings.

B. Lossy Compression algorithms

Given that downstream analysis typically does not require the same resolution as raw quality scores, lossy compression algorithms have become increasingly popular [31]. These algorithms often achieve much higher compression ratio than lossless alternatives, and can also improves the efficiency of downstream analyses in some cases. The basic architectures of some methods are illustrated in Fig. 4. Depending on the target file format of the tools (i.e., FASTQ and SAM), we categorize the methods into two types and introduce them separately.

1) Compression of Quality Scores in FASTQ File: The Phred score is obtained by quantizing the original probability irreversibly. This quantization is equivalent to dividing the range $[0, 1]$ into a fixed number of bins with equal sizes. QScores-Archiver [29] proposes three lossy binning transformation methods to rebin quality scores. In the UniBinning method, the range $[0, 1]$ is partitioned into a predefined number of bins. The Truncating transformation combines some bins into a single bin, where the values of quality score are considered to to α if they are larger than α, which is the well-defined parameter. The LogBinning method has a very similar idea to UniBinning, but it employs the logarithms of the probabilities to obtain new bins by regrouping the original bins.

Some algorithms leverage the strong correlations that exist among adjacent quality values (i.e., neighboring quality values are close to each other). The context of quality score is further exploited for modeling or prediction. For example, SlimGene [105] encodes the differences of adjacent quality values using the fixed-order Markov model, and the prediction result is encoded by Huffman. QVZ [25], which is short for quality values zip, first models the quality score using an order-1 Markov chain. It then uses the transition probabilities to construct a codebook, where the indexes are position and previous quantizers, and the values are quantizers. A revised version of the Lloyd–Max algorithm [106] is proposed to compute quantizers. All scores are quantized sequentially, and each quantized value forms the left context of the following quantizer. The quantized result is further compressed by applying the adaptive arithmetic coding method. Fqzcomp [101] mainly uses the public context model (http://ctxmodel.net). Some contexts, such as the quality score of the previous position and the maximum quality value of two previous positions, are used to predict the i-th quality. SCALCE [107] is a tool that mainly focuses on compressing sequence data in the FASTQ file. It also proposed a lossy transformation method to achieve the lossy compression of quality scores. A frequency table is constructed using 1 million lines of quality scores, and the local maximum of the table is found using a greedy algorithm. The quality scores in the neighborhood of the local maximum are reduced up to the defined error threshold. ALL-CQS [108] is an improved method of PBlock [109]. It constructs a difference matrix of contiguous quality score to seize the locality information, rather than operating on the initial quality score matrix. In the clustering stage of k-means algorithm, it samples a minimal subset of quality score lines to obtain the initialize centroid of the final clustering methods.

QualComp [110] and CROMqs [111] [112] are compression algorithms that rely on the singular value decomposition (SVD) [113] of a quality score matrix. QualComp is a lossy compression algorithm that allows users to set the compression rate before compression. It uses the statistical data obtained

from the quality scores. The mean and the covariance matrix are empirically worked out from the set of quality score vectors. To reduce the correlation between these vectors, SVD is performed on the covariance matrix. The new quality score vector is then created using the quality score vectors, mean matrix, and unitary matrix. Finally, the normalized new quality score vector is mapped to decision regions. The key innovation is that uses can set the compression rate. CROMqs [111], [112] performs the same SVD as QualComp, followed by an infinitesimal successive refinement lossy compressor [114] on the new quality scores vectors. The encoding algorithm compresses the quality score once with a relative high compression rate, and the decoding algorithm can iteratively recover them according to the required rate of users.

In the FASTQ file, base information is the most fundamental information for downstream analysis. Base information is utilized to assist the compression of quality score, with the aim of achieving better compression performance and improving the performance of downstream analysis. BEETL [115] introduced the idea of smoothing quality scores by assuming that a quality score can be disregarded or roughly compressed if a nucleotide base can be estimated with high probability by using the base context. Such predictions are made by using the famous algorithm Burrows–Wheeler transform (BWT) [116] and the popular data structure LCP (longest common prefix) array [117]. LEON [96] introduced a simple replacing strategy: if a nucleotide base is enveloped in some solid k-mers, it is examined as error-free base. These high quality scores have little effect on downstream analysis, so the corresponding quality scores of error-free bases are modified to the high-quality value ('@'). The new data stream of quality scores after replacing is then compressed by zlib.

External corpus is further used to improve the compression of quality scores by incorporating base information. Quartz [118] used individual sequences in the 1000 Genomes Project to build a reference corpus, which is a frequency table of 32-mers. It smooths quality score by assumping that a divergent base is likely to be an SNP or sequencing error. It stores the original quality scores that have a high probability of being a variant, and other quality scores of high-confident bases are set to a default value. GeneCodeq [119] leveraged the reference genome to derive the codewords. The k-mer corpus generated from the reference genome and Bayesian based noisy model are used to estimates the posterior probability of sequencing errors. The corresponding quality scores are then adjusted according to the posterior probability. YALFF [120], [121] employed an external corpus of k-mers to estimate the accuracy of bases by distinguishing between correct bases and sequencing errors. [122] to achieve efficient searching of k-mers. Similar to Quartz and GeneCodeq, it searches and compares k-mers of reads in the dictionary to compress quality values. But, YALFF requires all k-mers related to the quality score under compression to be discovered.

Other advanced methods are also applied in compression algorithms. In Fastqz [101], high-quality values larger than 30 are encoded using specific byte codes. In slow mode, three context models are further invented to model the previous encoded bytes. QScomp [123] decomposes the quality score into two values according to Elias gamma code. Then, the two data streams are separately compressed by bzip2. Hernandez-Lopez's approach [124] uses HISAT2's alignment data [125] to rebin the quality score, mitigating the effect of sequence alignment. ENANO [48] is a specialized tool developed for lossless compression of nanopore sequencing FASTQ file. It employs the specific context model [126] to determine the probability of the quality scores distribution, which is then compressed by an arithmetic encoder [127].

2) Compression of Quality Scores in SAM File: Several tools utilize alignment data from mapped reads stored in the SAM file to compress quality scores. NGC [99] uses a simple binning strategy similar to the UniBinning of the QScores-Archiver. It rebins the quality values of an interval to the representative value (i.e., the minimum or maximum values) of that interval. Quality values are divided into four categories according to sequence alignment information, and different quantification schemes are used for each category. CALQ [128] exploited alignment data of aligned reads to estimate the unpredictability of the genotype at each position in the genome sequence. Then, the allowable level of distortion rate is determined. For each locus, CALQ computed the genotype uncertainty and quantizer index. Finally, the quality values are represented by the quantizer index, and compressed by zero-order arithmetic coding. To minimize the effects on downstream application, Crumble [129] first identifies quality scores that are necessary for downstream analysis, after which they are further compressed using PBlock [109].

Cánovas et al. [109] utilized localized properties of the quality scores. They separate quality scores into blocks with variant sizes. In each size, a representative value and its length are stored using some measure criterion.

Morales's method [130] proposed a dynamic binning scheme, where quality scores is split into blocks. In each block, the quality alphabet is split into five bins. In each bin, all these quality alphabets are replaced by the most occurrences.

As only a part of symbols are used in the quality score part in a FASTQ file, ScaleQC [95] constructed a lookup table to modify the original quality scores into a continuous interval, with the lookup table stored for decoding. The quality values at possible variant locus are amplified by a pre-selected parameter. Bit-plane is constructed by scanning from left bit (i.e., most significant bit) to right of quality score value horizontally. Adaptive binary arithmetic code [131] is used to compress the bit stream.

Although RQS [132] was implemented for the compression of SAM file, it utilized base sequence information. In the first stage, a dictionary is generated to store high-frequency k-mers, and the quality scores of k-mers with a short Hamming distance to the elements of dictionary are discarded. Only the low-quality scores of k-mers very different from the dictionary are stored.

IV. Performance Comparison

A. Benchmarking Datasets

We conducted a comparison of the performance of 12 tools on 14 sequencing datasets that served as "benchmarking datasets"

TABLE II
DETAILS OF BENCHMARKING DATASETS

Accession	Reads Number	Reads length	Instrument model	Library strategy	Library source	Species	Alphabets
SRR032209	18, 828, 274	36	Illumina Genome Analyzer II	ChIP-Seq	Genomic	*Mus musculus*	!#%&()*+,01234567 89:;<=>?@ABC
SRR13575706_1	25, 184, 401	150	Illumina NovaSeq 6000	RNA-Seq	Transcriptomic	*Oryza sativa*	#,:F
SRR7216025_1	29, 233, 415	150	Illumina HiSeq X Ten	RNA-Seq	Transcriptomic	*Arabidopsis thaliana*	#)-7<AFJ
SRR5220205_1	129, 917, 724	149	Illumina NextSeq 500	Others	Genomic	*Homo sapiens*	#/6<AE
SRR15066805	50, 020, 584	35, 36	Illumina NovaSeq 6000	AMPLI-CON	VIRAL RNA	SARS-CoV-2	!#,:F
SRR15089532	5, 612, 042	77-148	Illumina NovaSeq 6000	AMPLI-CON	VIRAL RNA	SARS-CoV-2	#/6<AE
SRR027520_1	24, 246, 685	76	Illumina Genome Analyzer II	WGS	GENOMIC	*Homo sapiens*	!#%&'()*+,-./012345 6789:;<=>?@ABC
SRR034940_1	18, 198, 907	100	Illumina Genome Analyzer II	WGS	GENOMIC	*Homo sapiens*	!#%&'()*+,-./0123 456789:<=>; ?@AB CDEFGHIJ
SRR959239_1	2, 686, 416	98	Illumina HiSeq 2000	WGA	GENOMIC	*Escherichia coli*	#&'()*+,-./01234 56789:; <=>?@A BCDEFGHIJ
SRR1770413_1	643, 253	301	Illumina MiSeq	WGS	GENOMIC	*Escherichia coli*	#()*+,-./0123456789:; <=>?@ABCDEFG
SRR327342_1	15, 036, 699	63	Illumina Genome Analyzer II	WGS	GENOMIC	*Saccharomyces cerevisiae*	%&'()*+,-./012345 6789:; <=>?@ABC
SRR359032_1	17, 345, 097	100	Illumina HiSeq 2000	WGS	METAGENOMIC	metagenome	#%&'()*+,-./012345 6789:; <=>?@AB CDEFGHI
ERR532393_1	35, 752, 873	100	Illumina HiSeq 2000	WGS	METAGENOMIC	metagenome	#&'()*+,-./012345 6789:; <=>?@AB CDEFGHIJ
ERR1474585_1	38, 323, 142	43-97	Illumina HiSeq 2500	WGS	METAGENOMIC	metagenome	#&'()*+,-./012345 6789:; <=>?@AB CDEFGHIJ

(See Table II). These datasets include different species, such as *Homo sapiens*, *Mus musculus*, *Oryza sativa*, *Arabidopsis thaliana*, and *Escherichia coli*. The data were generated using a variety of sequencing technologies, including various instrument models, library strategies and sources. These species are widely used for evaluation the compression performance in previous studies [23], [32], [99]. Specifically, the dataset SRR032209 is often used as a benchmark in [29], [99], [110], [133]. They are available on the website of ENA (https://www.ebi.ac.uk/ena/browser/home) and NCBI (https://trace.ncbi.nlm.nih.gov/).

B. Performance Metrics

The compression performance was evaluated using compression ratio, run-time (wall-clock) and peak memory usage of compression and decompression procedures. The compression ratio is defined as the number of bits per quality (bpq) used to store those compressed scores, i.e.,

$$\text{ratio} = \frac{\text{Compressed size of quality scores}}{\text{Number of quality score}}.$$

The running time and peak memory consumption are acquired by utilizing the Unix command "/usr/bin/time -v". All tools are executed on a computing server running Ubuntu 19.10 having 2.2 GHz Intel Xeon Silver 4210 CPU (10 cores) and 128 GB RAM.

C. Performance Comparison of Lossless Compression Tools

Five tools, namely, AQUa, QScores-Archiver, LCQS, fastqz, and SCALCE, were tested, and their performances of compression and decompression are shown in Tables III and IV. These tools compress only quality score part stored in the FASTQ file. Among these tools, QScores-Archiver achieved a stable compression ratio over 0.8 bpq but was much worse than

TABLE III
COMPRESSION PERFORMANCE OF FIVE DIFFERENT LOSSLESS COMPRESSION TOOLS (COMPRESSION OF QUALITY SCORE PART)

Dataset	AQUa			QScores-Archiver			LCQS			fastqz			SCALCE (p0)		
	cratio	ct	cm	cratio	ct	cm	cratio	ct	cm	cratio	ct	cm	cratio	ct	cm
SRR032209	0.3724	75	1374	0.8189	1	7044	Error[2]			0.3236	8	1358	0.3325	5	2924
SRR13575706_1	0.0487	136	1384	0.8690	2	30556	0.0384	21	3101	0.0389	30	1460	0.0375	11	5237
SRR7216025_1	0.1196	246	1386	0.8732	3	35468	0.0167	48	6385	0.0893	54	1461	0.0988	27	5269
SRR5220205_1	0.1972	1591	1482	Out of memory			0.1315	480	9474	0.1299	149	1494	0.1413	180	5231
SRR15066805	0.0317	62	140	Error[1]			Error[2]			Error[1]			Error[1]		
SRR15089532	0.1251	48	1387	Error[1]			Error[2]			Error[1]			Error[1]		
SRR027520_1	0.4231	181	1371	0.8021	2	16099	0.2408	14	9517	0.3647	14	1462	0.3788	3	4595
SRR034940_1	0.4152	180	1393	0.8444	1	12362	0.2388	15	8615	0.3685	12	1461	0.3744	3	4244
SRR959239_1	0.2286	28	1394	0.8637	1	1829	Error[2]			0.1811	2	1290	0.2109	1	1562
SRR1770413	0.2142	3	1398	0.8599	1	1512	0.1442	1	4582	0.1709	1	1233	0.1964	1	1373
SRR327342	0.4004	120	1394	0.8349	7	6085	0.2179	9	8307	0.3532	5	1428	0.3629	2	3035
SRR359032	0.3597	121	1403	0.8465	1	11783	0.2058	26	8804	0.3070	13	14485	0.3214	3	4092
ERR532393	0.4053	361	1419	0.8534	3	24282	0.2346	2	8234	0.3337	37	1480	0.3614	4	5290
ERR1474585	0.3169	382	1391	Error[1]			0.2135	41	8885	Error[1]			Error[1]		

Notes: 'cratio' represents the compression ratio and is expressed in bpq. 'ct' represents running time of compression and is measured in minutes. 'cm' represents peak memory usage of compression and is measured in megabytes. [1]The tool requires that all input sequences have the same length. [2]Program reports "munmap chunk(): invalid pointer".

TABLE IV
DECOMPRESSION PERFORMANCE OF FIVE LOSSLESS COMPRESSION TOOLS

Dataset	AQUa		QScores-Archiver		LCQS		fastqz		SCALCE (p0)	
	dt	dmem	dt	dmem	dt	dmem	dt	dmem	dt	dmem
SRR032209	3	1325	1	7428	–	–	10	1360	2	1044
SRR13575706_1	6	1308	3	22485	33	12871	33	1460	7	1033
SRR7216025_1	9	1312	4	26099	80	16540	50	1460	9	1039
SRR5220205_1	53	1416	–	–	Error[1]		166	1478	37	1035
SRR15066805	2	1319	–	–	–	–	–	–	–	–
SRR15089532	2	1290	–	–	–	–	–	–	–	–
SRR027520_1	7	1332	2	12549	13	27431	19	1460	1	1047
SRR034940_1	9	1323	2	15621	11	5302	17	1460	1	1046
SRR959239_1	1	1323	1	1893	–	–	2	1290	1	1039
SRR1770413	1	1336	1	1293	1	4623	1	1232	1	1038
SRR327342	3	1323	1	7003	Error[1]		Error[2]		1	1045
SRR359032	4	1331	2	15621	Error[1]		Error[2]		2	1044
ERR532393	12	1333	5	31237	Error[1]		Error[2]		3	1045
ERR1474585	11	1335	–	–	Error[1]		–	–	–	–

Notes: 'dt' represents the running time of decompression and is measured in minutes. 'dmem' represents the peak memory usage of decompression and is measured in megabytes. '–' indicates that the corresponding compression process report error. [1]Program reports "munmap chunk(): invalid pointer". [2]Error caused by a mismatch between expected base number and recorded number.

TABLE V
COMPRESSION AND DECOMPRESSION PERFORMANCE OF TWO LOSSLESS COMPRESSION TOOLS (COMPRESSION OF FASTQ FILE)

Dataset	LEON					DSRC2 (q0)					DSRC2 (q2)				
	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm
SRR032209	471	10	1675	4	947	424	1	416	1	1853	419	2	1760	3	2275
SRR13575706_1	539	60	5260	17	1381	1059	2	194	1	1540	1033	3	1498	4	2770
SRR7216025_1	1074	78	4954	19	1972	1489	2	315	2	1762	1444	4	1907	5	2827
SRR5220205_1	7310	510	6726	107	7187	7440	10	337	8	1858	7102	20	2852	21	2836
SRR15066805	176	11	2016	7	513	486	2	517	1	1472	479	3	2689	3	2736
SRR15089532	148	7	2778	4	874	249	1	463	1	1507	242	1	1923	1	2773
SRR027520_1	1306	4	3583	3	1827	1203	1	1006	1	1874	1221	1	2823	1	2619
SRR034940_1	1275	4	3847	1	1754	1199	1	1284	1	1845	1198	1	2878	1	2212
SRR959239_1	92	1	1819	1	992	129	1	257	1	218	123	1	2013	1	2022
SRR1770413	55	1	1885	1	703	84	1	338	1	566	83	1	1724	1	1774
SRR327342	527	2	3078	1	1024	638	1	1179	1	2046	628	2	1652	1	2143
SRR359032	829	2	3753	1	1385	1083	1	762	1	2036	1055	1	2800	1	2165
ERR532393	2048	5	4542	2	2319	2440	1	1726	1	2097	2393	1	2783	1	2291
ERR1474585	1850	6	4644	4	2271	2132	3	563	1	1992	2017	4	1799	1	2423

Notes:'csize' represents the size of compressed file and is measured in MB. 'ct' (resp. 'dt') represents the running time of compression (resp. decompression) and is measured in minutes. 'cm' (resp. 'dm') represents peak memory usage of compression (resp. decompression) and is measured in megabytes.

TABLE VI
COMPRESSION AND DECOMPRESSION PERFORMANCES OF FOUR LOSSY COMPRESSION TOOLS (COMPRESSION OF QUALITY SCORE PART)

Dataset	QScores-Archiver					fastqz (e3)					fastqz (e5)				
	cratio	ct	cm	dt	dm	cratio	ct	cm	dt	dm	cratio	ct	cm	dt	dm
SRR032209	0.0972	1	7043	1	7429	0.1986	9	1365	10	1338	0.0467	9	1361	10	1355
SRR13575706_1	0.1674	2	30555	3	22485	0.0389	30	1460	34	1461	0.0174	30	1460	33	1460
SRR5220205_1	Error in compression					0.1299	152	1495	167	1494	0.0359	160	1495	173	1493
SRR7216025_1	0.2190	3	35467	4	26100	0.0893	29	1461	32	1460	0.0504	19	1461	22	1460
SRR15066805	Error[1]					Error[1]					Error[1]				
SRR15089532	Error[1]					Error[1]					Error[1]				
SRR027520_1	0.3469	2	16099	22	12549	0.2246	11	1460	13	1460	0.1820	12	1460	13	1460
SRR034940_1	0.4417	2	12362	2	15621	0.2380	10	1460	13	1460	0.1838	11	1461	13	1460
SRR959239_1	0.4735	1	1829	1	1893	0.1188	2	1292	2	1292	0.0555	2	1295	2	1292
SRR1770413	0.3738	1	1512	1	1293	0.1284	1	1233	Error[2]		0.0973	1	1445	Error[2]	
SRR327342	0.3661	1	6085	1	7003	0.1938	9	1435	Error[2]		0.1626	5	1460	Error[2]	
SRR359032	0.4461	2	11783	2	15621	0.1932	10	1461	Error[2]		0.1438	9	1520	Error[2]	
ERR532393	0.4407	4	24282	4	31237	0.2171	20	1511	Error[2]		0.1500	20	1514	Error[2]	
ERR1474585	Error[1]					Error[1]					Error[1]				
	QVZ					SCALCE (p30)									
	cratio	ct	cm	dt	dm	cratio	ct	cm	dt	dm					
SRR032209	0.0176	1	973	1	21	0.0459	4	2925	2	1037					
SRR13575706_1	0.0281	6	4040	5	28	0.0167	12	5237	7	1032					
SRR5220205_1	0.0280	32	20598	25	28	0.0630	177	5287	37	1039					
SRR7216025_1	0.0282	7	4686	6	28	0.0442	29	5232	9	1035					
SRR15066805	Error[1]					Error[1]									
SRR15089532	Error[1]					Error[3]									
SRR027520_1	0.0529	4	2190	3	30	0.2435	3	4595	1	1562					
SRR034940_1	0.0554	4	2076	3	33	0.2392	2	4180	1	1041					
SRR959239_1	0.0969	1	332	1	29	0.0858	1	1034	1	1562					
SRR1770413	0.0704	2	295	1	63	0.1200	1	1147	1	1035					
SRR327342	0.0602	2	1183	1	29	0.2188	2	4608	1	1040					
SRR359032	0.0557	4	1982	3	34	0.2091	2	4092	2	1039					
ERR532393	0.0565	7	4036	5	34	0.1822	5	5289	2	1038					
ERR1474585	Error[1]					Error[1]									

Notes:'cratio' represents the compression ratio and is measured in bpq. 'ct' (resp. 'dt') represents running time of compression (resp. decompression) and is measured in minutes. 'cm' (resp. 'dm') represents peak memory usage of compression (resp. decompression) and is measured in megabytes. [1]The tool requires that all input sequences have the same length. [2]Error caused by a mismatch between expected base number and recorded number. [3]Error caused invalid pointer.

other tools in compression ratio. For instance, on the dataset SRR13575706_1, QScores-Archiver was 23 times worse than the tool SCALCE. Fastqz and LCQS achieved similar compression ratio on the first three datasets, which were slightly higher than AQUa. SCALCE obtained the best compression ratio in dataset SRR13575706_1, its perforamnce is litter better than other three tools. Fastqz had the best compression ratio on two datasets and its compression ratio was only 0.02 bpq better than SCALCE. LCQS achive the best compression performance on SRR027520_1, SRR034940_1, SRR959239_1, SRR1770413, SRR327342, SRR359032, ERR532393. However, a number of its decompression process report invalid pointer error, potentially rendering the corresponding compression ratio inaccurate. The fastqz method encountered multiple instances of mismatch between the recorded and actual base number during decompression. In compression procedure, QScores-Archiver consumed more memory than AQUs and LCQS. Nothing that, the size of 128 GB RAM is not enough for QScores-Archiver to compress the dataset SRR5220205_1. LCQS consumed memory ranging from 3.0 GB to 9.3 GB. The memory usage by AQUa and fastqz was very stable, and they used less than 1.5 GB in all cases. SCALCE consumed more memory than AQUa and fastqz. In terms of compression speed, QScores-Archiver was the fastest tool. However, AQUa was much slower than other four tools. Especially, AQUa spent more than one day compressing the dataset SRR5220205_1. Fastqz and SCALCE spent less time than the tool LCQS. In lossless mode, the QScores-Archiver maps each quantized score to lower values and utilizes the minimum number of bits to store them. Consequently, it achieves a fast speed but a low compression ratio during the compression process. On the other hand, SCALCE constructs a frequency table for a selected subset of quality scores and then determines the mapping strategy for all quality scores by computing the local maxima of the table. This approach takes into account the characteristics of to-be-compressed data and achieves a balance between compression ratio and speed.

In decompression process, QScores-Archiver and LCQS spent more memory than other tools. AQUa, fastqz, and SCALCE were more fragile than other tools. For instance, LCQS consumed memory one order of magnitude larger than SCALCE in decompression. SCALCE was the fastest tool in decompression. In the decompression speed, LCQS and fastqz were worse than AQUa and SCALCE. In practice, SCALCE was better than other tools when lossless compression of quality scores was needed. Noting that, SCALCE can not preserve the order of reads.

The tools LEON and DSRC2 compress the whole FASTQ file, and their compression/decompression results are depicted in Table V. For the tool DSRC2, two modes, namely, q0 and q2, are tested. DSRC2 using q2 was always better than using q0 in compression ratio except the result of SRR027520_1. And DSRC (q2) consumes three times more memory than

TABLE VII
COMPRESSION AND DECOMPRESSION PERFORMANCE OF FOUR LOSSY COMPRESSION TOOLS (COMPRESSION OF FSATQ FILE).

Dataset	genie					fqzcomp (q1)					fqzcomp (q3)				
	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm
SRR032209	691	31	5089	10	11208	379	2	105	2	103	375	2	377	2	375
SRR13575706_1	751	34	14148	15	18469	729	4	104	6	102	Error in compression				
SRR5220205_1	7501	5672	25862	19	18643	3173	18	104	25	102	2973	21	376	28	374
SRR7216025_1	1222	51	16133	19	17414	628	4	104	6	102	590	6	376	7	374
SRR15066805	132	5	265	4	566	150	3	101	3	98	142	3	372	3	372
SRR15089532	160	1	351	1	300	98	1	374	1	316	90	1	374	1	316
SRR027520_1	1187	7	20046	2	17215	1071	2	106	3	104	1057	3	378	2	375
SRR034940_1	629	1	3993	6	20046	1045	2	107	3	104	1034	4	378	3	375
SRR959239_1	84	2	3157	1	6328	96	1	105	1	103	91	1	376	1	374
SRR1770413	43	1	1600	1	1902	59	1	106	1	103	57	1	376	1	374
SRR327342	482	8	14898	2	12569	486	1	106	3	103	480	1	377	3	374
SRR359032	760	12	16444	2	19428	753	3	106	6	103	737	3	377	3	375
ERR532393	1809	16	26637	4	18298	2048	7	107	10	104	1947	4	377	6	375
ERR1474585	1599	6	28129	2	17641	1826	6	106	7	104	1749	4	377	8	374
	LEON					DSRC2 (q0)					DSRC2 (q2)				
	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm	csize	ct	cm	dt	dm
SRR032209	385	12	1674	4	805	281	1	501	1	1663	274	1	1858	1	2816
SRR13575706_1	421	45	5275	17	1211	1059	2	373	1	1538	1037	3	1814	4	2766
SRR5220205_1	4521	219	6736	103	4428	7437	12	1692	8	1861	7161	21	2848	22	2836
SRR7216025_1	609	41	4963	19	1409	1489	2	570	2	1756	1450	4	2579	5	2827
SRR15066805	176	11	2016	7	513	486	2	2019	7	559	478	3	2428	2	2735
SRR15089532	148	7	2778	4	874	249	1	471	1	1564	239	1	1919	1	2579
SRR027520_1	1047	5	3574	2	1713	783	1	1566	5	1691	757	1	2779	1	2829
SRR034940_1	784	4	3859	1	1578	765	1	1648	1	537	746	1	2002	1	2815
SRR959239_1	30	1	951	1	902	101	1	765	1	2779	94	1	1878	1	1954
SRR1770413	17	1	1797	1	660	64	1	290	1	513	63	1	1596	1	1726
SRR327342	179	1	3076	1	971	370	1	511	1	1693	357	2	1628	1	2810
SRR359032	232	2	3753	1	947	713	1	832	1	1744	683	1	2216	1	2823
ERR532393	731	7	4520	9	1504	1619	1	1347	1	1853	1551	3	2477	1	2838
ERR1474585	676	7	4640	4	1377	1536	3	599	1	1785	1471	1	2775	1	2834

Notes:'csize' represents the size of compressed file and is measured in MB. 'ct' (resp. 'dt') represents running time of compression (resp. decompression) and is measured in minutes. 'cm' (resp. 'dm') represents peak memory usage of compression (resp. decompression) and is measured in megabytes.

DSRC (q0). DSRC2 (q2) achieved the best compression ratio on four datasets (SRR032209, SRR5220205_1, SRR027520_1,SRR034940_1), and LEON performs better than DSRC2 on the other 10 datasets. Especially, on the dataset SRR13575706_1, the compressed file by LEON was only half the size of the compressed file by DSRC2. Under both modes, DSRC2 was faster than LEON in compression/decompression speeds on all datasets.

D. Performance Comparison of Lossy Compression Tools

Four tools, i.e., QScores-Archiver, fastqz, QVZ, and SCALCE, are used to compress quality scores in FASTQ files. Their compression ratio, running time and peak memory consumption of compression and decompression process are presented in Table VI. We set '-q3' and '-q5' for the tool fastqz and tested the tool SCALCE on the model 'p30'. However, QScores-Archiver, fastqz and QVZ can not complete the compression process on some datasets. For the tool fastqz, its compression ratio under '-e5' is much better than '-e3' setting. On the dataset SRR032209, the lossy mode of SCALCE achieves about two times better compression ratio than its lossless mode. However, on three other datasets, both SCALCE models have the same compression ratio. SCALCE achieves the highest compression ratio on the dataset SRR13575706_1, it is also the only method in Table VI that can process reads with inconsistent length. On seven other datasets, QVZ achieves the best compression ratio, and it performs stably on different datasets. QScore-Archiver is the fastest tool, but it has the worst compression ratio.

QScores-Archiver consumes more memory than other tools during compression and decompression. On the largest dataset SRR5220205_1, QVZ consumes about 20 GB RAM, which is one order of magnitude large than the memory usage of fastqz. In decompression, QVZ uses less than 30 MB memory. The compression speed of LEON is much slower than that of QVZ and QScores-Archiver. QVZ employed a two-phase encoding strategy in which the quality scores were initially encoded using the mapping function generated by the Lloyd-Max quantizer. Subsequently, an arithmetic encoder was applied to the resulting data. This strategy combines the advantages of both encoding algorithms and achieves the best performance in most cases. Hence, we highly recommend choosing QVZ as the preferred method for compressing quality scores in lossy mode.

Genie, fqzcomp, LEON and DSRC2 compress all the data streams in a FASTQ file, their performance on the benchmarking datasets are shown in Table VII. DSRC2 achieves the smallest compressed file on the dataset SRR032209. On the other three datasets, fqzcomp has much better compression ratios than the tools genie and DSRC2. Fqzcomp and DSRC2 are much faster than genie. Additionally, the memory usage of genie is at least 50 times larger than fqzcomp and larger than DSRC2 from 2.7 to 9 times. In these four tools, DSRC2 outperforms

the other in almost all of the metrics. However, an error is obtained by fqzcomp on the dataset SRR13575706_1. LEON is a good choice for compression the FASTQ file in terms for compression ratio. DSRC2 achieves better compression speed but worse compression ratio than LEON in most of the cases.

V. Conclusion and Discussion

Quality scores are the main part in raw sequencing data, and compressing them poses a significant challenge. Accordingly, tremendous research efforts have been exerted into proposing high-performance algorithms to compress quality scores. This review provides a comprehensive overview of all existing quality score compression methods. In addition, we have performed benchmarking evaluation of the available tools developed for quality score compression on six datasets. We hope that this comprehensive review will offer useful guidance for developing innovative and more powerful compression methods.

Our systematic assessment of the compression tools for quality scores in sequencing data highlights both the advantages and limitations of these compression algorithms across various datasets. We discuss the challenges and future works from the following three perspectives.

Unavailability on Some Datasets: In benchmarking, some tools are unable to work well on some datasets. For instance, fqzcomp generates an error when compressing the file "SRR13575706_1.fastq". More serious, more than five tools, including some tools that only compress quality score part, fail to compress two sequencing datasets. It is hard to see their performance on various sequencing data. One of the main reasons for these errors is the inability to handle reads with unequal lengths. Developing tools that can work well on different FASTQ files is argued to address the challenges of big sequencing data. For tools that only compress quality score part, they should be easy to be integrated into those sequence compression tools [22], [42], [43].

Unstable Compression Ratio: We found that the compression ratios vary across different datasets. For example, LEON achieves the best compression on the dataset SRR13575706_1, which is more than twice as good as DSRC2 (See Table VII). However, on the dataset SRR5330305_1, the compression ratio of fqzcomp (q3) is 1.5 times better than LEON. In the future, we may try to employ novel technologies such as the error-bounded lossy [134] to design algorithm for quality score compression.

Not Lossless: By examining the decompression results, we observed that some tools, e.g., fqzcomp, cannot achieve lossless compression, i.e., the decompression result is not identical to the original one. Noting that, some of tools alter the reads order. Some analysis demonstrated that the order affects the result of downstream [21], [135]. More serious, some lossy results are not caused by the order of reads. Therefore, comprehensive verification for lossless compression is required.

There are additional tests that can be conducted for deeper insight into the compression of quality scores. First, the effect of lost quality scores on downstream analysis is worth investigating on more datasets [31], [118], [136]. Second, tools designed for compressing long reads need to be evaluated. Lastly, the compression tools for SAM format files can be tested on various dataset. In addition, several considerations should be taken into account when developing compression tools for quality scores. First, given the increasing size of sequencing data, efficient data structure are essential to reduce the computation time and memory usage during compression process. Second, the new tools should incorporate a mode that can handle reads with varying length. Lastly, the integration of sequence data and quality score compression is of utmost importance, as both components are integral to accurate data interpretation and downstream analyses.

References

[1] E. R. Mardis, "The impact of next-generation sequencing technology on genetics," *Trends Genet.*, vol. 24, no. 3, pp. 133–141, 2008.

[2] S. E. Levy and R. M. Myers, "Advancements in next-generation sequencing," *Annu. Rev. Genomic. Hum. Genet.*, vol. 17, pp. 95–115, 2016.

[3] J. Zhang, R. Chiodini, A. Badr, and G. Zhang, "The impact of next-generation sequencing on genomics," *J. Genet. Genomic.*, vol. 38, no. 3, pp. 95–109, 2011.

[4] Z. D. Stephens et al., "Big data: Astronomical or genomical?," *PLoS Biol.*, vol. 13, no. 7, 2015, Art. no. e1002195.

[5] R. Leinonen, H. Sugawara, M. Shumway, and I. N. S. D. Collaboration, "The sequence read archive," *Nucleic Acids Res.*, vol. 39, no. suppl_1, pp. D19–D21, 2010.

[6] R. Leinonen et al., "The European nucleotide archive," *Nucleic Acids Res.*, vol. 39, no. suppl_1, pp. D28–D31, 2010.

[7] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: Ten years of next-generation sequencing technologies," *Nature Rev. Genet.*, vol. 17, no. 6, 2016, Art. no. 333.

[8] M. Hernaez, D. Pavlichin, T. Weissman, and I. Ochoa, "Genomic data compression," *Annu. Rev. Biomed. Data Sci.*, vol. 2, pp. 19–37, 2019.

[9] M. Arita, I. Karsch-Mizrachi, and G. Cochrane, "The international nucleotide sequence database collaboration," *Nucleic Acids Res.*, vol. 49, no. D1, pp. D121–D124, 2021.

[10] A. Nibali and Z. He, "Trajic: An effective compression system for trajectory data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3138–3151, Nov. 2015.

[11] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng, "A novel representation and compression for queries on trajectories in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 613–629, Apr. 2018.

[12] M. Hosseini, D. Pratas, and A. J. Pinho, "A survey on data compression methods for biological sequences," *Information*, vol. 7, no. 4, 2016, Art. no. 56.

[13] Z. Zhu, Y. Zhang, Z. Ji, S. He, and X. Yang, "High-throughput DNA sequence data compression," *Brief. Bioinf.*, vol. 16, no. 1, pp. 1–15, 2015.

[14] Z. Zhu, J. Zhou, Z. Ji, and Y.-H. Shi, "DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 643–658, Oct. 2011.

[15] J. R. Almeida, A. J. Pinho, J. L. Oliveira, O. Fajarda, and D. Pratas, "GTO: A toolkit to unify pipelines in genomic and proteomic research," *SoftwareX*, vol. 12, 2020, Art. no. 100535.

[16] V. V. Cogo, J. Paulo, and A. Bessani, "GenoDedup: Similarity-based deduplication and delta-encoding for genome sequencing data," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 669–681, May 2021.

[17] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman, "SPRING: A next-generation compressor for FASTQ data," *Bioinformatics*, vol. 35, no. 15, pp. 2674–2676, 2019.

[18] Ł. Roguski and S. Deorowicz, "DSRC 2—Industry-oriented compression of FASTQ files," *Bioinformatics*, vol. 30, no. 15, pp. 2213–2215, 2014.

[19] Z.-A. Huang, Z. Wen, Q. Deng, Y. Chu, Y. Sun, and Z. Zhu, "LW-FQZip 2: A parallelized reference-based compression of FASTQ files," *BMC Bioinf.*, vol. 18, no. 1, pp. 1–8, 2017.

[20] Y. Zhang, L. Li, Y. Yang, X. Yang, S. He, and Z. Zhu, "Light-weight reference-based compression of FASTQ data," *BMC Bioinf.*, vol. 16, no. 1, pp. 1–8, 2015.

[21] Ł. Roguski, I. Ochoa, M. Hernaez, and S. Deorowicz, "FaStore: A space-saving solution for raw sequencing data," *Bioinformatics*, vol. 34, no. 16, pp. 2748–2756, 2018.

[22] Y. Liu, Z. Yu, M. E. Dinger, and J. Li, "Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression," *Bioinformatics*, vol. 35, no. 12, pp. 2066–2074, 2019.

[23] I. Numanagi ć et al. "Comparison of high-throughput sequencing data compression tools," *Nature Methods*, vol. 13, no. 12, pp. 1005–1008, 2016.
[24] M. A. DePristo et al., "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature Genet.*, vol. 43, no. 5, 2011, Art. no. 491.
[25] G. Malysa, M. Hernaez, I. Ochoa, M. Rao, K. Ganesan, and T. Weissman, "QVZ: Lossy compression of quality values," *Bioinformatics*, vol. 31, no. 19, pp. 3122–3129, 2015.
[26] D. Pavlichin, T. Weissman, and G. Mably, "The quest to save genomics: Unless researchers solve the looming data compression problem, biomedical science could stagnate," *IEEE Spectr.*, vol. 55, no. 9, pp. 27–31, Sep. 2018.
[27] R. Cánovas, A. Moffat, and A. Turpin, "CSAM: Compressed sam format," *Bioinformatics*, vol. 32, no. 24, pp. 3709–3716, 2016.
[28] Y. Liu, L. Wong, and J. Li, "Allowing mutations in maximal matches boosts genome compression performance," *Bioinformatics*, vol. 36, no. 18, pp. 4675–4681, 2020.
[29] R. Wan, V. N. Anh, and K. Asai, "Transformations for the compression of FASTQ quality scores of next-generation sequencing data," *Bioinformatics*, vol. 28, no. 5, pp. 628–635, 2012.
[30] M. Abedi, B. Sun, and Z. Zheng, "A sinusoidal-hyperbolic family of transforms with potential applications in compressive sensing," *IEEE Trans. Image Process.*, vol. 28, no. 7, pp. 3571–3583, Jul. 2019.
[31] I. Ochoa, M. Hernaez, R. Goldfeder, T. Weissman, and E. Ashley, "Effect of lossy compression of quality scores on variant calling," *Brief. Bioinf.*, vol. 18, no. 2, pp. 183–194, 2017.
[32] R. Yu, W. Yang, and S. Wang, "Performance evaluation of lossy quality compression algorithms for RNA-seq data," *BMC Bioinf.*, vol. 21, no. 1, pp. 1–15, 2020.
[33] M. P. Cox, D. A. Peterson, and P. J. Biggs, "SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data," *BMC Bioinf.*, vol. 11, no. 1, pp. 1–6, 2010.
[34] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Res.*, vol. 38, no. 6, pp. 1767–1771, 2010.
[35] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," 2013, *arXiv:1303.3997*.
[36] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, 2012.
[37] H. Li et al., "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
[38] D. C. Koboldt, "Best practices for variant calling in clinical sequencing," *Genome Med.*, vol. 12, no. 1, pp. 1–13, 2020.
[39] P. Danecek et al., "The variant call format and VCFtools," *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, 2011.
[40] M. C. Schatz, A. L. Delcher, and S. L. Salzberg, "Assembly of large genomes using second-generation sequencing," *Genome Res.*, vol. 20, no. 9, pp. 1165–1173, 2010.
[41] S. Vargas-Pérez and F. Saeed, "A hybrid MPI-OpenMP strategy to speedup the compression of big next-generation sequencing datasets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2760–2769, Oct. 2017.
[42] S. Chandak, K. Tatwawadi, and T. Weissman, "Compression of genomic sequencing reads via hash-based reordering: Algorithm and analysis," *Bioinformatics*, vol. 34, no. 4, pp. 558–567, 2018.
[43] T. M. Kowalski and S. Grabowski, "PgRC: Pseudogenome-based read compressor," *Bioinformatics*, vol. 36, no. 7, pp. 2082–2089, 2020.
[44] Y. Liu and J. Li, "Hamming-shifting graph of genomic short reads: Efficient construction and its application for compression," *PLoS Comput. Biol.*, vol. 17, no. 7, 2021, Art. no. e1009229.
[45] Y. Liu, X. Zhang, Q. Zou, and X. Zeng, "Minirmd: Accurate and fast duplicate removal tool for short reads via multiple minimizers," *Bioinformatics*, vol. 37, no. 11, pp. 1604–1606, 2021.
[46] D. J. Burgess, "Next regeneration sequencing for reference genomes," *Nature Rev. Genet.*, vol. 19, no. 3, pp. 125–125, 2018.
[47] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and challenges in long-read sequencing data analysis," *Genome Biol.*, vol. 21, no. 1, pp. 1–16, 2020.
[48] G. Dufort YÁlvarez, G. Seroussi, P. Smircich, J. Sotelo, I. Ochoa, and Á. Martín, "ENANO: Encoder for NANOpore FASTQ files," *Bioinformatics*, vol. 36, no. 16, pp. 4506–4507, 2020.
[49] G. D. y Álvarez, G. Seroussi, P. Smircich, J. Sotelo, I. Ochoa, and Á. Martín, "Compression of nanopore FASTQ files," in *Proc. Int. Work-Conf. Bioinf. Biomed. Eng.*, Springer, 2019, pp. 36–47.
[50] G. Seroussi et al., "RENANO: A REference-based compressor for NANOpore FASTQ files," *Bioinformatics*, vol. 37, pp. 4862–4864, 2021.
[51] J. K. Bonfield, "The Scramble conversion tool," *Bioinformatics*, vol. 30, no. 19, 2014, Art. no. 2818.
[52] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney, "Efficient storage of high throughput DNA sequencing data using reference-based compression," *Genome Res.*, vol. 21, no. 5, pp. 734–740, 2011.
[53] F. Hach, I. Numanagic, and S. C. Sahinalp, "DeeZ: Reference-based compression by local assembly," *Nature Methods*, vol. 11, no. 11, pp. 1082–1084, 2014.
[54] R. Long, M. Hernaez, I. Ochoa, and T. Weissman, "GeneComp, a new reference-based compressor for SAM files," in *Proc. 2017 Data Compression Conf.*, 2017, pp. 330–339.
[55] I. Ochoa, H. Li, F. Baumgarte, C. Hergenrother, J. Voges, and M. Hernaez, "AliCo: A new efficient representation for SAM files," in *Proc. 2019 Data Compression Conf.*, 2019, pp. 93–102.
[56] S. Deorowicz, A. Danek, and S. Grabowski, "Genome compression: A novel approach for large collections," *Bioinformatics*, vol. 29, no. 20, pp. 2572–2578, 2013.
[57] D. Lan, R. Tobler, Y. Souilmi, and B. Llamas, "genozip: A fast and efficient compression tool for VCF files," *Bioinformatics*, vol. 36, no. 13, pp. 4091–4092, 2020.
[58] H. Li, "BGT: Efficient and flexible genotype query across many samples," *Bioinformatics*, vol. 32, no. 4, pp. 590–592, 2016.
[59] K. Tatwawadi, M. Hernaez, I. Ochoa, and T. Weissman, "GTRAC: Fast retrieval from compressed collections of genomic variants," *Bioinformatics*, vol. 32, no. 17, pp. i479–i486, 2016.
[60] A. Danek and S. Deorowicz, "GTC: How to maintain huge genotype collections in a compressed form," *Bioinformatics*, vol. 34, no. 11, pp. 1834–1840, 2018.
[61] S. Deorowicz and A. Danek, "GTShark: Genotype compression in large projects," *Bioinformatics*, vol. 35, no. 22, pp. 4791–4793, 2019.
[62] X. Chen, S. Kwong, and M. Li, "A compression algorithm for DNA sequences," *IEEE Eng. Med. Biol. Mag.*, vol. 20, no. 4, pp. 61–66, Jul./Aug. 2001.
[63] X. Chen, M. Li, B. Ma, and J. Tromp, "DNACompress: Fast and effective DNA sequence compression," *Bioinformatics*, vol. 18, no. 12, pp. 1696–1698, 2002.
[64] S. Grumbach and F. Tahi, "Compression of DNA sequences," in *Proc. IEEE Data Compression Conf.*, 1993, pp. 340–350.
[65] N. S. Bakr et al., "DNA lossless compression algorithms," *Amer. J. Bioinf. Res.*, vol. 3, no. 3, pp. 72–81, 2013.
[66] B. Song, F. Li, Y. Liu, and X. Zeng, "Deep learning methods for biomedical named entity recognition: A survey and qualitative comparison," *Brief. Bioinf.*, vol. 22, 2021, Art. no. bbab282.
[67] X. Yang et al., "Modality-DTA: Multimodality fusion strategy for drug-target affinity prediction," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 20, no. 2, pp. 1200–1210, Mar./Apr. 2023.
[68] B. Song, X. Luo, X. Luo, Y. Liu, Z. Niu, and X. Zeng, "Learning spatial structures of proteins improves protein–protein interaction prediction," *Brief. Bioinf.*, vol. 23, no. 2, 2022, Art. no. bbab558.
[69] X. Zeng, X. Tu, Y. Liu, X. Fu, and Y. Su, "Toward better drug discovery with knowledge graph," *Curr. Opin. Struct. Biol.*, vol. 72, pp. 114–126, 2022.
[70] J. Dong, M. Zhao, Y. Liu, Y. Su, and X. Zeng, "Deep learning in retrosynthesis planning: Datasets, models and tools," *Brief. Bioinf.*, vol. 23, no. 1, 2022, Art. no. bbab391.
[71] T. Tang et al., "Machine learning on protein–protein interaction prediction: Models, challenges and trends," *Brief. Bioinf.*, vol. 24, no. 2, 2023, Art. no. bbad076.
[72] R. Wang et al., "DeepDNA: A hybrid convolutional and recurrent neural network for compressing human mitochondrial genomes," in *Proc. 2018 IEEE Int. Conf. Bioinf. Biomed.*, 2018, pp. 270–274.
[73] M. Silva, D. Pratas, and A. J. Pinho, "Efficient DNA sequence compression with neural networks," *GigaScience*, vol. 9, no. 11, 2020, Art. no. giaa119.
[74] S. Christley, Y. Lu, C. Li, and X. Xie, "Human genomes as email attachments," *Bioinformatics*, vol. 25, no. 2, pp. 274–275, 2009.
[75] D. A. Wheeler et al., "The complete genome of an individual by massively parallel DNA sequencing," *Nature*, vol. 452, no. 7189, pp. 872–876, 2008.
[76] D. S. Pavlichin, T. Weissman, and G. Yona, "The human genome contracts again," *Bioinformatics*, vol. 29, no. 17, pp. 2199–2202, 2013.
[77] D. Pratas, A. J. Pinho, and P. J. Ferreira, "Efficient compression of genomic sequences," in *Proc. IEEE 2016 Data Compression Conf.*, 2016, pp. 231–240.

[78] I. Ochoa, M. Hernaez, and T. Weissman, "iDoComp: A compression scheme for assembled genomes," *Bioinformatics*, vol. 31, no. 5, pp. 626–633, 2015.

[79] S. Saha and S. Rajasekaran, "ERGC: An efficient referential genome compression algorithm," *Bioinformatics*, vol. 31, no. 21, pp. 3468–3475, 2015.

[80] Y. Liu, H. Peng, L. Wong, and J. Li, "High-speed and high-ratio referential genome compression," *Bioinformatics*, vol. 33, no. 21, pp. 3364–3372, 2017.

[81] S. Wandelt and X. Sun, "Efficient compression of 4D-trajectory data in air traffic management," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 844–853, Apr. 2015.

[82] K. Zheng, Y. Zhao, D. Lian, B. Zheng, G. Liu, and X. Zhou, "Reference-based framework for spatio-temporal trajectory compression and query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 11, pp. 2227–2240, Nov. 2020.

[83] S. Wandelt, X. Sun, and Y. Zhu, "Lossless compression of public transit schedules," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 11, pp. 3075–3086, Nov. 2016.

[84] Y. Liu, L. Y. Zhang, and J. Li, "Fast detection of maximal exact matches via fixed sampling of query K-mers and Bloom filtering of index K-mers," *Bioinformatics*, vol. 35, no. 22, pp. 4560–4567, 2019.

[85] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[86] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Sci. Rep.*, vol. 5, no. 1, pp. 1–12, 2015.

[87] T. Tang, Y. Liu, B. Zhang, B. Su, and J. Li, "Sketch distance-based clustering of chromosomes for large genome database compression," *BMC Genomic.*, vol. 20, no. 10, pp. 1–9, 2019.

[88] S. Deorowicz and S. Grabowski, "Compression of DNA sequence reads in FASTQ format," *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.

[89] W. Tembe, J. Lowey, and E. Suh, "G-SQZ: Compact encoding of genomic sequence and quality data," *Bioinformatics*, vol. 26, no. 17, pp. 2192–2194, 2010.

[90] M. Howison, "High-throughput compression of FASTQ data with SeqDB," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 10, no. 1, pp. 213–218, Jan. 2013.

[91] J. Fu, Y. Ma, and S. Dong, "A lossless FASTQ Quality Scores file compression algorithm based on linear combination prediction," in *Proc. 2016 IEEE Int. Conf. Bioinf. Biomed.*, 2016, pp. 1894–1896.

[92] J. Fu, B. Ke, and S. Dong, "LCQS: An efficient lossless compression tool of quality scores with random access functionality," *BMC Bioinf.*, vol. 21, no. 1, pp. 1–12, 2020.

[93] J. Zhou, Z. Ji, Z. Zhu, and S. He, "Compression of next-generation sequencing quality scores using memetic algorithm," *BMC Bioinf.*, vol. 15, no. 15, pp. 1–7, 2014.

[94] T. Paridaens, G. Van Wallendael, W. De Neve, and P. Lambert, "AQUa: An adaptive framework for compression of sequencing quality scores with random access functionality," *Bioinformatics*, vol. 34, no. 3, pp. 425–433, 2018.

[95] R. Yu and W. Yang, "ScaleQC: A scalable lossy to lossless solution for NGS data compression," *Bioinformatics*, vol. 36, no. 17, pp. 4551–4559, 2020.

[96] G. Benoit et al., "Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph," *BMC Bioinf.*, vol. 16, no. 1, pp. 1–14, 2015.

[97] R. Patro and C. Kingsford, "Data-dependent bucketing improves reference-free compression of sequencing reads," *Bioinformatics*, vol. 31, no. 17, pp. 2770–2777, 2015.

[98] S. Grabowski, S. Deorowicz, and Ł. Roguski, "Disk-based compression of data from genome sequencing," *Bioinformatics*, vol. 31, no. 9, pp. 1389–1395, 2015.

[99] N. Popitsch and A. von Haeseler, "NGC: Lossless and lossy compression of aligned high-throughput sequencing data," *Nucleic Acids Res.*, vol. 41, no. 1, pp. e27–e27, 2013.

[100] Goby. [Online]. Available: https://github.com/CampagneLaboratory/goby

[101] J. K. Bonfield and M. V. Mahoney, "Compression of FASTQ and SAM format sequencing data," *PLoS One*, vol. 8, no. 3, 2013, Art. no. e59190.

[102] Y. Niu, M. Ma, F. Li, X. Liu, and G. Shi, "ACO: Lossless quality score compression based on adaptive coding order," *BMC Bioinf.*, vol. 23, no. 1, pp. 1–14, 2022.

[103] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[104] M. Cho and A. No, "FCLQC: Fast and concurrent lossless quality scores compressor," *BMC Bioinf.*, vol. 22, no. 1, pp. 1–14, 2021.

[105] C. Kozanitis, C. Saunders, S. Kruglyak, V. Bafna, and G. Varghese, "Compressing genomic sequence fragments using SlimGene," in *Proc. Annu. Int. Conf. Res. Comput. Mol. Biol.*, Springer, 2010, pp. 310–324.

[106] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[107] F. Hach, I. Numanagić, C. Alkan, and S. C. Sahinalp, "SCALCE: Boosting sequence compression algorithms using locally consistent encoding," *Bioinformatics*, vol. 28, no. 23, pp. 3051–3057, 2012.

[108] J. Fu and S. Dong, "All-CQS: Adaptive locality-based lossy compression of quality scores," in *Proc. 2017 IEEE Int. Conf. Bioinf. Biomed.*, 2017, pp. 353–359.

[109] R. Cánovas, A. Moffat, and A. Turpin, "Lossy compression of quality scores in genomic data," *Bioinformatics*, vol. 30, no. 15, pp. 2130–2136, 2014.

[110] I. Ochoa, H. Asnani, D. Bharadia, M. Chowdhury, T. Weissman, and G. Yona, "QualComp: A new lossy compressor for quality scores based on rate distortion theory," *BMC Bioinf.*, vol. 14, no. 1, pp. 1–16, 2013.

[111] I. Ochoa, A. No, M. Hernaez, and T. Weissman, "CROMqs: An infinitesimal successive refinement lossy compressor for the quality scores," in *Proc. 2016 IEEE Inf. Theory Workshop*, 2016, pp. 121–125.

[112] A. No, M. Hernaez, and I. Ochoa, "CROMqs: An infinitesimal successive refinement lossy compressor for the quality scores," *J. Bioinf. Comput. Biol.*, vol. 18, pp. 2050031–2050031, 2020.

[113] W. Wang, C. Chen, W. Yao, K. Sun, W. Qiu, and Y. Liu, "Synchrophasor data compression under disturbance conditions via cross-entropy-based singular value decomposition," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2716–2726, Apr. 2021.

[114] A. No and T. Weissman, "Rateless lossy compression via the extremes," *IEEE Trans. Inf. Theory*, vol. 62, no. 10, pp. 5484–5495, Oct. 2016.

[115] L. Janin, G. Rosone, and A. J. Cox, "Adaptive reference-free compression of sequence quality scores," *Bioinformatics*, vol. 30, no. 1, pp. 24–30, 2014.

[116] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," in *Digital SRC Research Report*. Princeton, NJ, USA: Citeseer, 1994.

[117] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM J. Comput.*, vol. 22, no. 5, pp. 935–948, 1993.

[118] Y. W. Yu, D. Yorukoglu, J. Peng, and B. Berger, "Quality score compression improves genotyping accuracy," *Nature Biotechnol.*, vol. 33, no. 3, pp. 240–243, 2015.

[119] D. L. Greenfield, O. Stegle, and A. Rrustemi, "GeneCodeq: Quality score compression and improved genotyping using a Bayesian framework," *Bioinformatics*, vol. 32, no. 20, pp. 3124–3132, 2016.

[120] Y. Shibuya and M. Comin, "Better quality score compression through sequence-based quality smoothing," *BMC Bioinf.*, vol. 20, no. 9, pp. 1–11, 2019.

[121] Y. Shibuya and M. Comin, "Indexing k-mers in linear space for quality value compression," *J. Bioinf. Comput. Biol.*, vol. 17, no. 05, 2019, Art. no. 1940011.

[122] P. Ferragina and G. Manzini, "Indexing compressed text," *J. ACM*, vol. 52, no. 4, pp. 552–581, 2005.

[123] J. Voges, A. Fotouhi, J. Ostermann, and M. O. Külekci, "A two-level scheme for quality score compression," *J. Comput. Biol.*, vol. 25, no. 10, pp. 1141–1151, 2018.

[124] A. A. Hernandez-Lopez, C. Alberti, and M. Mattavelli, "Toward a dynamic threshold for quality score distortion in reference-based alignment," *J. Comput. Biol.*, vol. 27, no. 2, pp. 288–300, 2020.

[125] D. Kim, J. M. Paggi, C. Park, C. Bennett, and S. L. Salzberg, "Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype," *Nature Biotechnol.*, vol. 37, no. 8, pp. 907–915, 2019.

[126] J. Rissanen, "A universal data compression system," *IEEE Trans. Inf. Theory*, vol. 29, no. 5, pp. 656–664, Sep. 1983.

[127] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Develop.*, vol. 20, no. 3, pp. 198–203, 1976.

[128] J. Voges, J. Ostermann, and M. Hernaez, "CALQ: Compression of quality values of aligned sequencing data," *Bioinformatics*, vol. 34, no. 10, pp. 1650–1658, 2018.

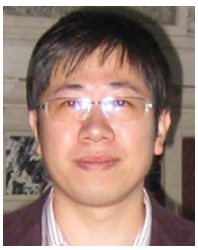
[129] J. K. Bonfield, S. A. McCarthy, and R. Durbin, "Crumble: Reference free lossy compression of sequence quality values," *Bioinformatics*, vol. 35, no. 2, pp. 337–339, 2019.

[130] V. S. Morales and S. Houghten, "Lossy compression of quality values in sequencing data," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 18, no. 5, pp. 1958–1969, Sep./Oct. 2021.

[131] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.
[132] Y. W. Yu, D. Yorukoglu, and B. Berger, "Traversing the *k*-mer landscape of NGS read datasets for quality score sparsification," in *Proc. Int. Conf. Res. Comput. Mol. Biol.*, Springer, 2014, pp. 385–399.
[133] P. Li, X. Jiang, S. Wang, J. Kim, H. Xiong, and L. Ohno-Machado, "HUGO: Hierarchical mUlti-reference Genome cOmpression for aligned reads," *J. Amer. Med. Inform. Assoc.*, vol. 21, no. 2, pp. 363–373, 2014.
[134] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. 2016 IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 730–739.
[135] C. Firtina and C. Alkan, "On genomic repeats and reproducibility," *Bioinformatics*, vol. 32, no. 15, pp. 2243–2247, 2016.
[136] M. Rivara-Espasandín et al., "Nanopore quality score resolution can be reduced with little effect on downstream analysis," *Bioinf. Adv.*, vol. 2, no. 1, 2022, Art. no. vbac054.

Yuansheng Liu received the bachelor's and master's degrees in computer science from Xiangtan University, China, in 2012 and 2015, respectively and the PhD degree from the University of Technology Sydney, Australia, in 2019. He is currently an associate professor with the College of Information Science and Engineering, Hunan University, China. In 2020, he was postdoctoral research fellow with the University of Technology Sydney. His current research interests is bioinformatics.

Tao Tang received the BS degree from the University of Sydney in 2017, and the PhD degree in computer science from the University of Sydney in 2021. His main research interests include bioinformatics, computational biology, data mining and parallel computing.



Zexuan Zhu received the BS degree in computer science and technology from Fudan University, Shanghai, China, in 2003, and the PhD degree in computer engineering from Nanyang Technological University, Singapore, in 2008. He is currently a professor with the College of Computer Science and Software Engineering, Shenzhen University, China. His research interests include computational intelligence and bioinformatics. He is an associate editor of *IEEE Transactions on Evolutionary Computation* and *IEEE transactions on Emerging Topics in Computational Intelligence*. He is also the chair of the IEEE CIS Emergent Technologies Task Force on Memetic Computing.

Xiangxiang Zeng received the BS degree in automation from Hunan University, Changsha, China, in 2005, and the PhD degree in system engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2011. Before joining Hunan University in 2019, he was with Department of Computer Science in Xiamen University. In 2019, He is a Yuelu distinguished professor with the College of Information Science and Engineering, Hunan University. His main research interests include membrane computing and bioinformatics.

Quan Zou (Senior Member, IEEE) received the BSc, MSc, and the PhD degrees in computer science from the Harbin Institute of Technology, China, in 2004, 2007, and 2009, respectively. He worked with Xiamen University and Tianjin University from 2009 to 2018 as an assistant professor, associate professor, and professor. He is currently a professor with the Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China. His research is in the areas of bioinformatics, machine learning, and parallel computing. Several related works have been published by *Science*, *Briefings in Bioinformatics*, *Bioinformatics*, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, etc. Google scholar showed that his more than 100 papers have been cited more than 10,000 times. He is the editor-in-chief of *Current Bioinformatics*, associate editor of *IEEE Access*, and an editorial board member of *Computers in Biology and Medicine*, *Genes*, *Scientific Reports*, etc. He was selected as one of the Clarivate Analytics Highly Cited Researchers, in 2018, 2019, and 2020.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, Big Data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He has published more than 630 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.