



# LRBinner: Binning Long Reads in Metagenomics Datasets

Anuradha Wickramarachchi ✉ 

School of Computing, Australian National University, Canberra, Australia

Yu Lin<sup>1</sup> ✉ 

School of Computing, Australian National University, Canberra, Australia

---

## Abstract

Advancements in metagenomics sequencing allow the study of microbial communities directly from their environments. Metagenomics binning is a key step in the species characterisation of microbial communities. Next-generation sequencing reads are usually assembled into contigs for metagenomics binning mainly due to the limited information within short reads. Third-generation sequencing provides much longer reads that have lengths similar to the contigs assembled from short reads. However, existing contig-binning tools cannot be directly applied on long reads due to the absence of coverage information and the presence of high error rates. The few existing long-read binning tools either use only composition or use composition and coverage information separately. This may ignore bins that correspond to low-abundance species or erroneously split bins that correspond to species with non-uniform coverages. Here we present a reference-free binning approach, LRBinner, that combines composition and coverage information of complete long-read datasets. LRBinner also uses a distance-histogram-based clustering algorithm to extract clusters with varying sizes. The experimental results on both simulated and real datasets show that LRBinner achieves the best binning accuracy against the baselines. Moreover, we show that binning reads using LRBinner prior to assembly reduces computational resources for assembly while attaining satisfactory assembly qualities.

**2012 ACM Subject Classification** Applied computing → Bioinformatics; Applied computing → Computational genomics

**Keywords and phrases** Metagenomics binning, long reads, machine learning, clustering

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2021.11

**Supplementary Material** *Software (Source Code)*: <https://www.github.com/anuradhawick/LRBinner>; archived at [swh:1:dir:a2988a1c26fc1323580378ff00135030f1ee61a4](https://www.swh.io/dir/a2988a1c26fc1323580378ff00135030f1ee61a4)

**Acknowledgements** We would like to thank the anonymous reviewers for their valuable comments. Furthermore, this research was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI Australia), an NCRIS enabled capability supported by the Australian Government.

## 1 Introduction

Metagenomics binning is an important area of study in metagenomics analysis. Broadly, metagenomics enables the study of microbial genetic material directly from the source environment [3]. This eliminates the necessity of lab culturing thus revealing the microbial content of an environment as it is without culturing biases. Metagenomics *binning* is one key problem in metagenomics study that facilitates the clustering of sequences into different taxonomic groups. Mainly there are two approaches to address this problem; (1) reference-based binning and (2) reference-free binning. Reference-based binning tools (*e.g.*, Kraken [27], Centrifuge [8] and Kaiju [15]) bin sequences based on similarity by comparing with a database of known reference genomes and thus face challenges when the reference database

---

<sup>1</sup> To whom correspondence should be addressed.



is unavailable or incomplete. At present, reference-free binning tools have been gaining popularity over reference-based binning tools, especially in discovering novel or rare species in complex metagenomics datasets. While Next-Generation Sequencing (NGS) technologies produce short reads, existing reference-free binning tools typically rely on longer *contigs* that are assembled from short reads and contain richer information for binning. Reference-free binning tools (*e.g.*, MetaBAT [6, 7], MaxBin [29, 28], BMC3C [30], BusyBeeWeb [12, 11], SolidBin [24] and VAMB [18], *etc.*) bin contigs based on their composition and coverage information, without using any reference database. For example, a recent work VAMB [18] introduced the use of deep variational auto-encoders to perform reference-free unsupervised binning of contigs incorporating both the composition and coverage information. VAMB then uses an iterative medoid clustering algorithm which extracts clusters (bins) in a local search fashion. Thanks to the accurate composition and coverage information of contigs, reference-free approaches show promising results in binning contigs from metagenomics assemblies.

With the advent of Third Generation Sequencing (TGS) technologies such as Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), reads obtained are much longer than NGS reads (>10kbp). Therefore, more information becomes available in the reads themselves to support direct reads binning. However, contig-binning tools cannot be directly applied to bin long reads (by treating them as contigs) because there is no coverage information available for each long read. Moreover, while certain contig-binning tools make use of single-copy marker genes to estimate the number of bins in the sample, the high error rates of long reads and the varying coverages of different species make it infeasible to derive accurate estimations. Furthermore, datasets containing raw long reads are much larger in size compared to typical datasets containing contigs, and hence, demand more scalable reference-free binning tools.

Recently, a long-read binning tool MetaBCC-LR [26] was introduced to bin error-prone long reads. While MetaBCC-LR shows very promising results in binning long reads, it still suffers from accuracy and scalability issues, especially in complex metagenomics datasets. Firstly, MetaBCC-LR uses the composition and coverage information of long reads in a separate manner. This can result in the ignorance of bins for species with low abundance and incorrect bin split for species with non-uniform composition or coverage. Secondly, due to its scalability issue, MetaBCC-LR has to employ a sampling strategy to work on a subset of reads for large datasets, which affects its overall binning accuracy. In addition, binning of long read datasets requires novel algorithms to detect clusters of vastly varying sizes (hundreds to millions of reads per species), which is different from the contig-binning scenarios (few hundreds of contigs per species [14]). Therefore, it is persistently demanding better approaches to bin massive long-read datasets accurately and efficiently. The requirement is further supported by the advent of PacBio HiFi technology [25] which produces accurate and massive long-read datasets in metagenomics studies.

In this paper, we present LRBinner to bin TGS long reads without using any reference databases. LRBinner combines the composition and coverage features and eliminates the need to sub-sample large datasets. More specifically, LRBinner uses a variational auto-encoder to obtain lower dimensional representations by incorporating both composition and coverage information of the complete dataset. LRBinner further uses a distance-histogram-based clustering algorithm that can capture confident clusters of varying sizes. LRBinner finally assigns unclustered reads to obtained clusters using their statistical profiles. The experimental results of LRBinner compared against other baselines show that LRBinner achieves better binning results on both simulated and real datasets. Moreover, we show that binning long reads by LRBinner prior to assembly helps to improve genome fraction of assemblies while reducing the memory consumption for metagenomics assembly.

## 2 Methods

LRBinner consists of three main steps; (1) learning lower dimensional latent representations of composition and coverage, (2) clustering the latent representations and (3) obtaining complete clusters. The complete workflow for LRBinner is demonstrated in Figure 1. In the following sections, we will explain these three steps in details.

### 2.1 Step 1

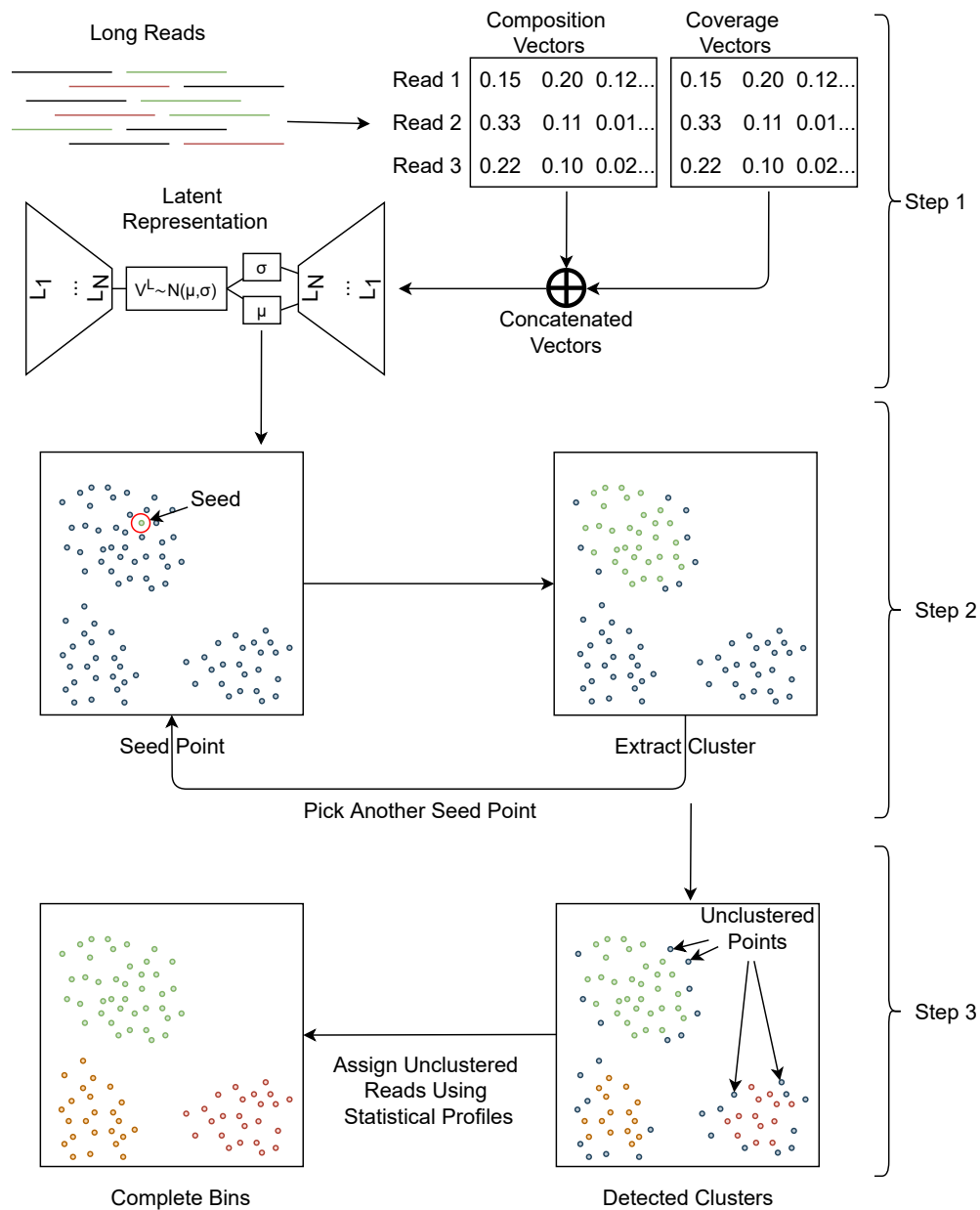
LRBinner uses two typical binning features of metagenomic sequences, composition and coverage. The composition and coverage of each long read is represented as trimer frequency vectors and  $k$ -mer coverage histograms [26], respectively.

#### 2.1.1 Computing Composition Vectors

Previous studies show that different species demonstrate unique genomic patterns [1, 4] and thus can be used in composition-based metagenomics binning. Oligonucleotide frequency vectors are one such genomic representation that can be used in metagenomics binning. Small  $k$ -mer sizes ( $k$  varying from 3-7) have been used in the past to discriminate assembled contigs of different origins [2, 7, 11, 20, 28] and 3-mers have been used in metagenomics binning of error-prone long reads [26] which shows that trinucleotide frequency vectors provide stable binning despite the noise level exist in TGS reads. Therefore in LRBinner, we utilise  $k=3$  by default which results in trinucleotide composition vectors. For each long read, we count the frequencies of all 64 3-mers in this read and merge the reverse complements to form a vector of 32 dimensions. The resulting vector is then normalised by the total number of 3-mers observed in the read. We refer to this trimer frequency vector as  $V^T$ .

#### 2.1.2 Computing Coverage Vectors

While an all-vs-all alignment of long reads may provide coverage information for each long read, it is usually too time-consuming to perform such alignments on large, long-read datasets (requires indexing hundreds of thousands of reads, index searching of read minimizers and pairwise-alignment of reads to filter false positives). Given a sufficiently large  $k$ , the frequency of a  $k$ -mer is defined as the number of occurrences of this  $k$ -mer in the entire dataset. Long reads from high-abundance species tend to contain  $k$ -mers with higher frequencies compared to long reads from low-abundance species. Hence, a  $k$ -mer frequency vector can be computed for each long read to represent coverage information without performing alignments [26] to represent read coverage. In order to obtain such coverage histograms, we first compute the  $k$ -mer counts of all long reads in the entire dataset by DSK [21] (the default value of  $k=15$ ). The counts are then indexed in memory by encoding each nucleotide in 2 bits as per the encoding (*i.e.*, A=00, C=01, T=10 and G=11) [21]. The resulting index is in the form  $(k_i, coverage(k_i))$  (as *key, value* pairs), where  $coverage(k_i)$  is the number of occurrences of the  $k$ -mer  $k_i$  in the entire dataset. Now for each  $k$ -mer  $k_i$  of a read, we obtain the frequency from the index. These frequencies are then used to build a normalised histogram,  $V^C$ . We chose a preset bin width (*bin\_width*) for the histogram and obtain a vector of *bins* dimensions. By default we set *bin\_width*=10 and *bins*=32. All the  $k$ -mers with counts exceeding the histogram limits are added into the last index of the histogram. We also normalise the histogram by the total number of  $k$ -mers observed in the read.



■ **Figure 1** Overall workflow of LRBinner. (Step 1) The feature vectors of composition and coverage information are computed from long reads. The feature vectors are fed into a variational auto-encoder to obtain low-dimensional latent representations. (Step 2) Sample a seed point (read) in the latent space and derive a confident cluster (bin) that contains this seed point. Step 2 is iterated until there is no seed point. (Step 3) The unclustered points are assigned to the clusters using a statistical model. Note that the 2-dimensional representation of points is only for the illustration purpose.

### 2.1.3 Obtaining Latent Representations

For each long read, its coverage ( $V^C$ ) and composition ( $V^T$ ) vectors are concatenated to form a single vector  $V$  of 64 dimensions. We use a variational auto-encoder to obtain lower dimensional latent representations. The key motivation for using a variational auto-encoder is to combine coverage and composition features effectively. Previous work shows that using TSNE on concatenated coverage and composition reduced the effectiveness [26]. This is mainly because TSNE do not attempt to learn meaningful weights for each feature, but rather consider neighbourhoods using spatial distances. However, the variational auto-encoder learns lower dimensional representations by meaningfully weighting features through a deep neural network such that original data can be reconstructed from the decoding layers.

Our implementation of the variational auto-encoder consists of two hidden-layers in the encoder and decoder. Each layer uses batch normalisation and dropout with  $p=0.1$  during the training phase. For each input vector  $V$ , the auto-encoder learns a latent representation  $V_i^L$ , where  $V_i^L \sim \mathcal{N}(\mu_i, \sigma_i)$ . The latent representation consists of 8 dimensions. Each layer in the encoder and decoder contains 128 neurons. Similar to previous studies [18], we use *LeakyRELU* (leaky rectified linear unit function) for  $\mu$  and *softplus* function for  $\sigma$  layers. Note that  $\mu$  and  $\sigma$  represents neural network layers intended to learn the lower dimensional means and standard deviations of each read’s distribution. We use the weighted sum of reconstruction error  $E$  (equation 1) and Kullback–Leibler divergence [10, 18]  $D_{KL}$  (equation 2) as the loss function.  $E_{cov}$  and  $E_{com}$  represent reconstruction errors of coverage and composition respectively. Equation 3 demonstrates the complete loss function used.

$$E = \sum (V_{in} - V_{out})^2 \quad (1)$$

$$D_{KL}(latent|prior) = - \sum \frac{1}{2}(1 + \ln(\sigma) - \mu^2 - \sigma) \quad (2)$$

$$Total\ Loss = w_{cov}E_{cov} + w_{com}E_{com} + w_{kld}D_{KL} \quad (3)$$

Here we set  $w_{cov}=0.1$ ,  $w_{com}=1$  and  $w_{kld}=1/500$  as determined empirically using simulated data. The decoder output was obtained through LeakyRELU activation in order to reconstruct the scaled positive inputs. We train the auto-encoder with read batches of size 10,240 for 200 epochs. Finally, we obtain the predicted latent means of the input data from the encoder for clustering. Each point in the latent mean corresponds to the relevant read in the original input.

## 2.2 Step 2

In this step, we perform clustering of the latent means learnt by the variational auto-encoder. The complete clustering algorithm of LRBinner is illustrated in Figure 2. Similar to previous studies [18], we use the cosine distance as the distance measure for clustering. Note that cosine distance between point  $a$  and  $b$  in latent space  $V^L$  is defined as  $d(a, b) = \frac{V_a^L \cdot V_b^L}{\|V_a^L\| \|V_b^L\|}$ . Given a point  $a$ , a distance histogram  $H_a$  can be generated by computing the pairwise distances between  $a$  and all other points and setting the bin width as  $\Delta$  ( $\Delta=0.005$  in our experiments). We define *peak* as the index of the first maximal of the distance histogram  $H_a$ . Similarly, the *valley* is defined as the index of the first minimal after the *peak* in the distance histogram  $H_a$ . Refer to the top right figure in Figure 2 for an example of the *peak* and *valley* in a distance histogram.

As shown in VAMB [18], a point with smaller valley-to-peak ratio  $H[\textit{valley}]/H[\textit{peak}]$  is more likely to be the medoid of a cluster, where  $H[\textit{valley}]$  and  $H[\textit{peak}]$  are the number of points corresponding to the *valley* and *peak* in the distance histogram  $H$ , respectively. Therefore, VAMB randomly samples points, searches within a distance of 0.05 (up to 25 neighbouring points) and moves to another point if  $H[\textit{valley}]/H[\textit{peak}]$  can be further reduced. This step is iterated until a local minimal point of  $H[\textit{valley}]/H[\textit{peak}]$  is inferred as a proper cluster medoid and then the corresponding cluster is extracted by removing points within a distance  $\Delta \times \textit{valley}$  of the distance histogram. However, clusters of long reads are orders of magnitude larger than clusters of contigs, thus mere local search of a cluster medoid may be inefficient. Furthermore, while most contig clusters consist of hundreds of points per species[14], the long-read clusters vary in size drastically (from hundreds of points to millions of points), which demand for a more flexible search strategy rather than sampling points within a fixed radius and up to a fixed number of neighbours. Hence, we design the following strategy to dynamically extract clusters of varying sizes. Our algorithm consists of two steps; (1) from a seed point to a candidate cluster and (2) from a candidate cluster to a confident cluster.

### 2.2.1 From a Seed Point to a Candidate Cluster

A point  $s$  is called a *seed point* if its valley-to-peak ratio  $H_s[\textit{valley}]/H_s[\textit{peak}] < 0.5$  in its distance histogram  $H_s$ . Initially, LRBinner randomly picks a seed point  $s$  from the entire dataset and obtains its distance histogram  $H_s$ . Note that a distance histogram demonstrates a *candidate cluster*. This *candidate cluster* consists of the points within the distance  $\Delta \times \textit{valley}$  in  $H_s$ , referred to as *candidate cluster points*. Compared to the seed point, some candidate cluster points may have lower valley-to-peak ratio that result in more confident clusters. However, the number of candidate cluster points may vary significantly depending on the size of the ground-truth clusters. In the next section, we will show how to use sampling strategies to find a confident cluster from a candidate cluster.

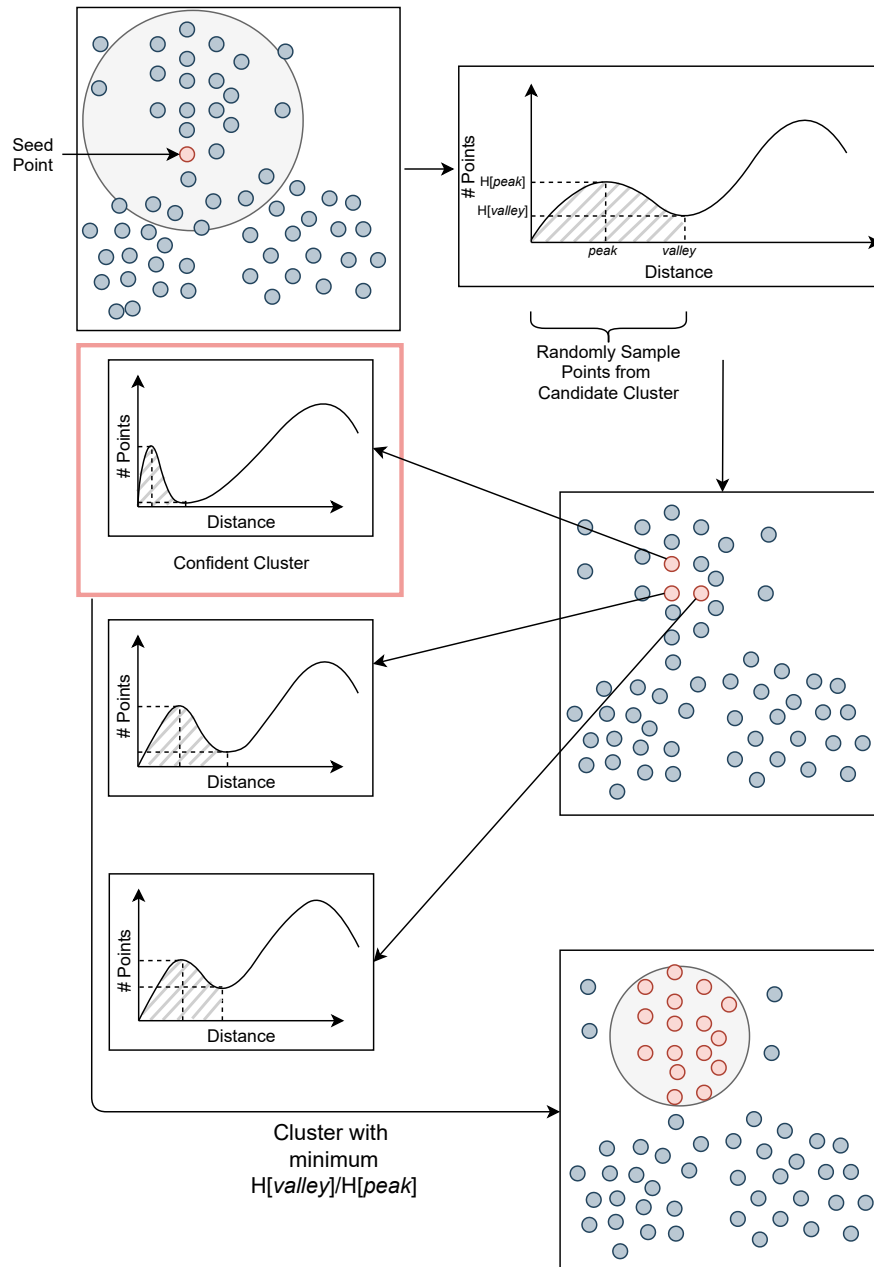
### 2.2.2 From a Candidate Cluster to a Confident Cluster

Given a *candidate cluster*, we sample 10% of candidate cluster points (up to 1,000 points) to compare their corresponding distance histograms. For each point  $p$  in *candidate cluster points*, we compute the valley-to-peak ratio  $H_p[\textit{valley}]/H_p[\textit{peak}]$  in its corresponding distance histogram  $H_p$ . We chose a point  $x$  from the sample with the minimum  $H[\textit{valley}]/H[\textit{peak}]$  value and extract a *confident cluster* which consists of points within a distance  $\Delta \times \textit{valley}$  of the distance histogram  $H_x$ . In contrast with the iterative medoid search in VAMB [18], this approach takes advantage of the rough estimation of the *candidate cluster* from a seed point and thus allows us to dynamically and efficiently discover clusters with varying sizes. This process is iterated until no further *candidate clusters* or *seed* points are observed. Please refer to Section 5 for detailed information. The resulting clusters are depicted as detected clusters in Figure 1. Note that few reads still remain unclustered due to the noise present in composition and coverage vectors of error-prone long reads and we will show how to assign them to existing bins in the next section.

## 2.3 Step 3

### 2.3.1 Obtaining Final Bins

Once all the clusters have been yielded, the points that are sparsely located are left aside. However, such points could have the potential to improve the downstream assembly processes.



■ **Figure 2** Illustration of the clustering algorithm. First select a seed point, generate its distance histogram and derive a *candidate cluster*. Sample from the *candidate cluster points* and choose a point with the minimum valley-to-peak ratio. Extract points before the *valley* to form a *confident cluster*. Note that the 2D representation of points is only for the illustration purposes.

Hence, we assign such points to the detected clusters using a statistical model similar to MetaBCC-LR [26]. For each cluster  $C_k$  the mean  $\mu_k^C$ ,  $\mu_k^T$  and standard deviation  $\sigma_k^C$ ,  $\sigma_k^T$  is computed using the coverage and composition vectors;  $V^C$  and  $V^T$  respectively.

$$PDF(\bar{v}, \bar{\mu}, \bar{\sigma}) = \prod_j \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}} \quad (4)$$

Finally the unclustered reads are assigned to the cluster  $C_i$  using a maximum likelihood computed using equation 4. The assignment of reads is performed such that equation 5 is maximised.  $V_l^C$  and  $V_l^T$  are the coverage histogram and trimer frequency vectors of the unclustered read  $l$ .

$$C_i = \operatorname{argmax}_i \left\{ PDF(V_l^C, \mu_i^C, \sigma_i^C) \times PDF(V_l^T, \mu_i^T, \sigma_i^T) \right\} \quad (5)$$

## 3 Experimental Setup

### 3.1 Datasets

We evaluated LRBinner using several simulated and real datasets containing long reads. Detailed information about the datasets and constituent species are tabulated under Tables 3 and 4 of Appendix.

#### 3.1.1 Simulated Datasets

We simulated two datasets using SimLoRD [23] to evaluate the performance of our method. The first dataset consists of 8 species and the second dataset consists of 20 species. These datasets are named as **Sim-8** and **Sim-20** respectively. We set the average read length to be 5,000bp with default error model of SimLoRD (insertion probability=0.11, deletion probability=0.04 and substitution probability=0.01).

#### 3.1.2 Real Datasets

In order to evaluate LRBinner, we use several real datasets with known ground-truth references. To determine the origins of the reads in these datasets, the reads were mapped to the respective reference species using Minimap2[13]. The information about the datasets are as follows.

- Reads from ATCC MSA-1003 Mock Microbial Community with PacBio CCS reads from NCBI BioProject number *PRJNA546278* (**MSA-1003**). For the evaluation we used the top 10 species which have more than 1% abundance.
- PacBio-HiFi reads obtained from NCBI BioProject number *PRJNA680590*. There are 3 read samples (NCBI BioSample number *SAMN16885726*) and each sample consists of 21 strains for 17 species as follows;
  - **SRX9569057**: Standard input library
  - **SRX9569058**: Low input library
  - **SRX9569059**: Ultra low input library (PCR amplified sample)

Detailed information about the simulated datasets is available in Section A of Appendix.



### 3.2 Tools for Benchmarking

There is a limited number of tools that support binning of long reads. Remind that most contig-binning tools cannot be directly applied to bin long reads (even for highly accurate PacBio HiFi reads) because there is no coverage information available for each long read. Hence, in our evaluation we use BusyBeeWeb [11] and MetaBCC-LR [26] which supports error prone long-reads as input. However, BusyBeeWeb only supports up to 200MB of FASTA formatted data. Hence, in our evaluation we have to provide BusyBeeWeb with a sub-sampled set of reads and evaluated the binning precision and recall on this sub-sampled set.

### 3.3 Evaluation Criteria

In our evaluation we report precision (equation 6), recall (equation 7) and F1-score (equation 8) of binning. We transform the binning result to a matrix  $a$  of size  $K \times S$ , where  $K$  denotes the number of bins and  $S$  denotes the number of species. Note that  $a_{ks}$  denotes the number of reads assigned to bin  $k$  with ground truth species  $s$ . In order to evaluate the quality of binning, we used AMBER [16] to obtain the completeness (defined as  $\frac{\text{true positives}_b}{\text{true positives}_b + \text{false negatives}_b}$  for each bin  $b$ ) and contamination (defined as  $1 - \frac{\text{true positives}_b}{\text{true positives}_b + \text{false positives}_b}$  for each bin  $b$ ). Please note that we only compare AMBER results of MetaBCC-LR and LRBinner as BusyBeeWeb does not bin the entire datasets due to limited input size. Furthermore, we assemble the reads before and after binning using LRBinner. Metagenomics assemblies were performed using wtdbg2 [22] and metaFlye [9]. We compare genome fractions, CPU-time and memory usage in assembly evaluation. We used MetaQUAST [17] to obtain the genome fraction (average percentage of bases aligned per reference genome) for the qualitative evaluation of assembled contigs.

$$Precision = \frac{\sum_k \max_s \{a_{ks}\}}{\sum_k \sum_s a_{ks}} \quad (6)$$

$$Recall = \frac{\sum_s \max_k \{a_{ks}\}}{\sum_k \sum_s a_{ks}} \quad (7)$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (8)$$

## 4 Results and Discussion

We first compare precision, recall, F1 score and the estimated number of bins for binning performance. We further present the completeness and contamination results of bins produced by different binners. We finally evaluate assembly results using genome fraction and recorded the resource utilisation for the chosen assembly tools.

### 4.1 Binning Results

We benchmarked the binning performance for BusyBeeWeb, MetaBCC-LR and LRBinner. Table 1 demonstrates the binning results in terms of precision, recall, F1-score and the number of inferred bins. While BusyBeeWeb, MetaBCC-LR and LRBinner perform in a comparable fashion on simulated datasets, LRBinner achieves the best estimation on the number of bins with respect to the ground truth. As BusyBeeWeb has a limitation of input

■ **Table 1** Comparison of binning results of BusyBeeWeb, MetaBCC-LR and LRBinner.

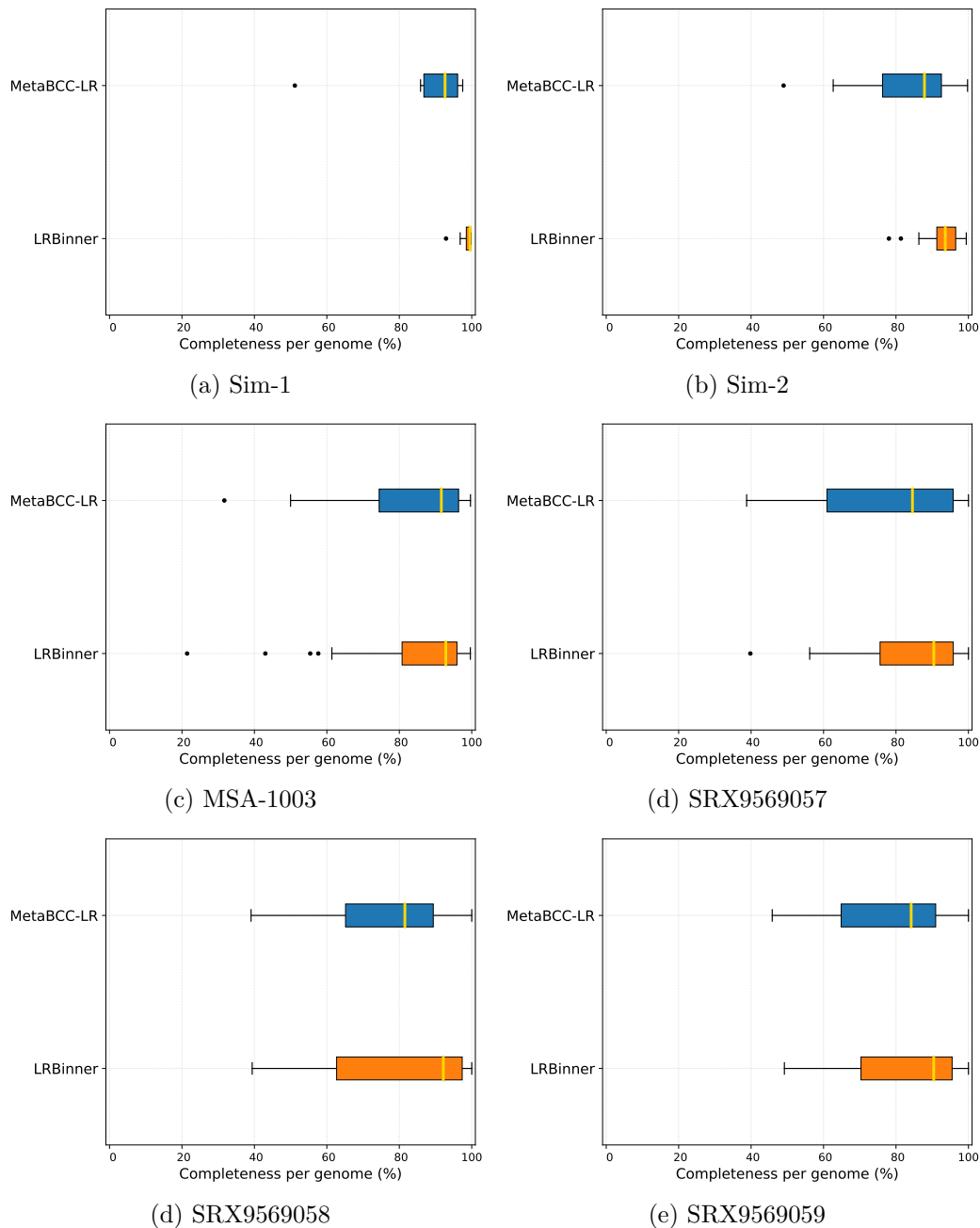
Dataset	Actual No. of Bins	Evaluation Criteria	BusyBeeWeb	MetaBCC-LR	LRBinner
Sim-8	8	Precision	90.41%	90.78%	<b>99.14%</b>
		Recall	<b>99.80%</b>	96.18%	99.14%
		F1 score	94.87%	93.40 %	<b>99.14%</b>
		Bins Detected	50	13	<b>8</b>
Sim-20	20	Precision	<b>95.88%</b>	82.97%	90.53%
		Recall	<b>97.99%</b>	81.95%	88.23%
		F1 score	<b>96.92%</b>	82.46%	89.36%
		Bins Detected	30	29	<b>18</b>
MSA-1003	10	Precision	68.30%	93.69%	<b>95.30%</b>
		Recall	81.96%	95.50%	<b>95.99%</b>
		F1 score	74.51%	94.59%	<b>95.64%</b>
		Bins Detected	87	14	<b>10</b>
SRX9569057	17	Precision	48.63%	<b>80.94</b>	80.47%
		Recall	72.68%	85.82	<b>90.68%</b>
		F1 score	58.27%	83.31	<b>85.27%</b>
		Bins Detected	111	23	<b>16</b>
SRX9569058	17	Precision	23.01%	70.18%	<b>73.72%</b>
		Recall	32.64%	86.63%	<b>91.03%</b>
		F1 score	26.99%	77.54%	<b>81.46%</b>
		Bins Detected	117	37	<b>22</b>
SRX9569059	17	Precision	65.70%	66.69%	<b>79.70%</b>
		Recall	<b>95.36%</b>	73.76%	91.25%
		F1 score	77.80%	70.05%	<b>85.08%</b>
		Bins Detected	124	<b>16</b>	20

data size (200Mb), its binning accuracy deteriorates on the real large datasets due to its limited access to the complete dataset. Note that LRBinner improves binning results for all the real datasets as indicated by the higher F1 scores.

Figure 3 illustrates the completeness of bins produced by MetaBCC-LR and LRBinner. Note that BusyBeeWeb is not included in this comparison as it cannot handle the entire dataset in most cases. LRBinner has been able to produce bins with better average completeness over MetaBCC-LR. Figure 4 also illustrates the contamination levels of bins produced by MetaBCC-LR and LRBinner. From the plots it is evident that LRBinner produces bins with lower contamination in all datasets except for **SRX9569059**. Note that the dataset **SRX9569059** has been generated from a PCR amplified sample leading to a significant deviation from the original sample abundances in contrast with **SRX9569057** and **SRX9569058** datasets. For example, in **SRX9569059**, the abundance of *Faecalibacterium prausnitzii* drops from  $\sim 16\%$  to  $\sim 8\%$  whereas the abundance of *Fusobacterium nucleatum* surges from  $\sim 4\%$  to  $\sim 7\%$ , which may result in contamination of long reads in binning results.

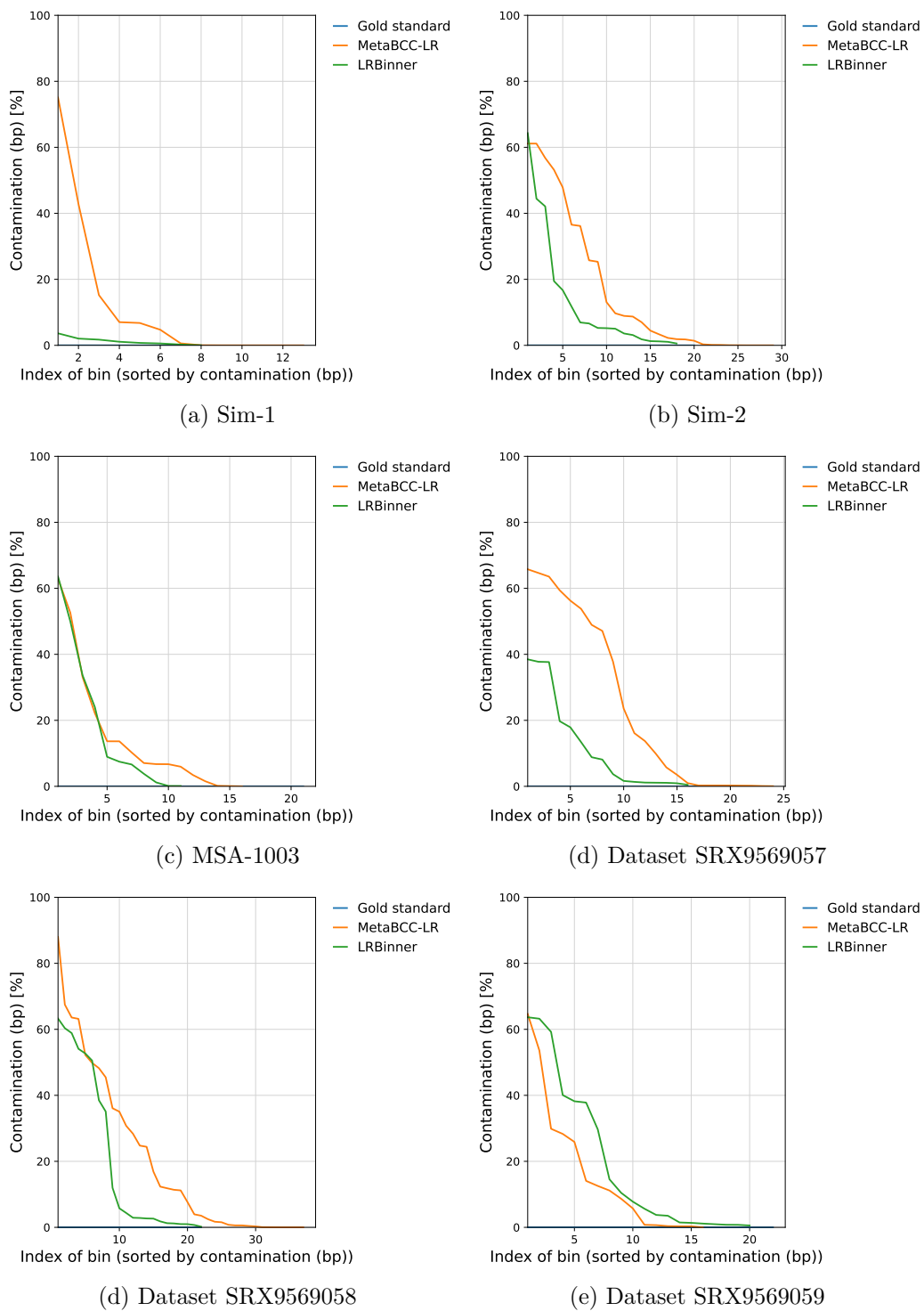
## 4.2 Assembly Results

We assembled the reads binned by LRBinner to evaluate the potential assembly quality changes. For the assembly, we chose the two state-of-the-art long-read assemblers wtdbg2 [22] and metaFlye [9]. Table 2, demonstrates that binning long reads prior to assembly by LRBinner improves the genome fraction for all wtdbg2 assemblies (up to 40%) and maintains comparable genome fractions for metaFlye assemblies. This is not surprising as metaFlye is



■ **Figure 3** Comparison of bin completeness between MetaBCC-LR and LRBinner.

a metagenomics specialised assembler in contrast with wtdbg2. For example, in the datasets SRX9569057, SRX9569058 and SRX9569059, binning via LRBinner enabled wtdbg2 to recover low-abundance species which were ignored in the assembly of the entire raw dataset, e.g., *Methanobrevibacter smithii* (from 0 to 96%), *Saccharomyces cerevisiae* (from 0 to 75%) and *Candida albican* (from 0 to 70%). This is because LRBinner allows wtdbg2 to estimate more appropriate parameters in each bin rather than applying the same parameters across the entire dataset.



■ **Figure 4** Comparison of bin contamination between MetaBCC-LR and LRBinner.

Another advantage of binning prior to assembly is the reduction of the computing resources for assembly. As demonstrated in Table 2, the peak-memory usage has been drastically reduced in both wtdbg2 (upto 10 $\times$ ) and metaFlye (upto 4 $\times$ ) assemblies. Note that the CPU time is comparable as binning long reads may not lead to significant reduction of  $k$ -mer indexing cost and the construction and simplification of assembly graphs.

■ **Table 2** Comparison of assembled genome fractions, CPU time consumed for assembly and peak memory usage of assembly before and after binning the reads.

Dataset	Assembly Tool	Genome Fraction		CPU Hours		Peak Memory (GB)	
		Raw	Binned	Raw	Binned	Raw	Binned
Sim-8	wtdbg2	98.80%	<b>98.90%</b>	<b>0.26</b>	0.84	9.28	<b>0.96</b>
	metaFlye	<b>99.90%</b>	99.85%	16.13	<b>11.64</b>	44.12	<b>10.65</b>
Sim-20	wtdbg2	97.84%	<b>99.19%</b>	<b>0.16</b>	2.28	10.60	<b>0.92</b>
	metaFlye	<b>99.80%</b>	99.75%	<b>19.44</b>	20.28	44.70	<b>11.23</b>
MSA-1003	wtdbg2	67.45%	<b>82.50%</b>	<b>0.31</b>	1.05	23.43	<b>19.61</b>
	metaFlye	91.40%	<b>91.74%</b>	<b>155.96</b>	158.59	62.28	<b>45.38</b>
SRX9569057	wtdbg2	40.40%	<b>73.02%</b>	<b>0.26</b>	1.56	21.72	<b>3.88</b>
	metaFlye	<b>77.73%</b>	73.68%	122.00	<b>116.20</b>	57.91	<b>26.31</b>
SRX9569058	wtdbg2	37.51%	<b>80.65%</b>	<b>0.30</b>	1.98	30.79	<b>3.86</b>
	metaFlye	79.16%	<b>79.63%</b>	<b>211.61</b>	212.58	87.62	<b>41.37</b>
SRX9569059	wtdbg2	41.00%	<b>80.38%</b>	<b>0.26</b>	1.82	25.63	<b>3.80</b>
	metaFlye	<b>79.69</b>	77.46%	152.64	<b>129.41</b>	62.62	<b>30.56</b>

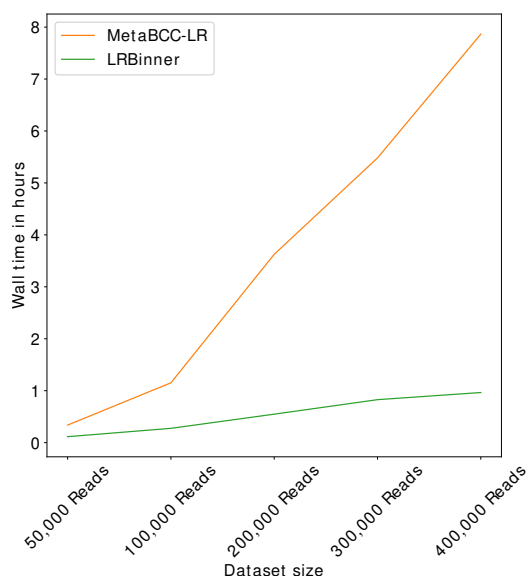
## 5 Implementation

In order to restrict the iterative search for clusters, we use early termination parameters in our algorithm. We stop drawing seed points when the remaining number of reads reaches below *min\_cluster\_size* (=5000 by default) or the number of iterations has passed *max\_iterations* (=1000 by default). We executed MetaBCC-LR and LRBinner on 5 sub-sampled datasets from **Sim-8**. We set MetaBCC-LR to skip the sampling step to make a fair comparison with LRBinner. Figure 5 illustrates the variation of time with increasing dataset size. It is evident that LRBinner scales well whereas time consumption of MetaBCC-LR grows much rapidly. Note that we do not consider BusyBeeWeb for this evaluation as it does not bin complete datasets.

LRBinner was implemented using C++ and Python version 3.7. The deep learning component is implemented using PyTorch [19] and Numpy [5]. We conducted our assemblies on NCI Australia with 2 x 24-core Intel Xeon Platinum 8274 (Cascade Lake) 3.2 GHz CPUs 192GB RAM and binning on Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 16GB RAM (4 core 8 threads) running Ubuntu Linux. We used 28 cores (with 56 threads hyper-threading) for assembly and 8 threads for binning.

## 6 Conclusion

In this paper, we presented LRBinner, a long read binner capable of binning error-prone long reads using both coverage and composition information. Our work extends the use of variational auto-encoders to combine raw features and learn a better latent representation for long-read binning. Furthermore, we presented a novel clustering strategy that can



■ **Figure 5** Wall time used by MetaBCC-LR and LRBinner without any sampling of data.

perform clustering on large datasets with varying cluster sizes. Performance of LRBinner was evaluated against existing long-read binners using simulated and real datasets. Our experimental results show that LRBinner outperforms state-of-the-art long-read binning tools and also improves resource usage of downstream assembly.

One limitation of LRBinner is the inability to distinct shared reads that arise shared genomic regions between different species. Resolution of such regions demands more experiments and significant improvements to the methodology as future work. Furthermore, we intend to introduce better noise handling to the clustering algorithm and investigate the potential of combining the binning and assembly of long reads simultaneously.

---

## References

- 1 Takashi Abe, Shigehiko Kanaya, Makoto Kinouchi, Yuta Ichiba, Tokio Kozuki, and Toshimichi Ikemura. Informatics for unveiling hidden genome signatures. *Genome Research*, 13(4):693–702, 2003. URL: <http://genome.cshlp.org/content/13/4/693.full.pdf+html>.
- 2 Johannes Alneberg, Brynjar Smári Bjarnason, Ino de Bruijn, et al. Binning metagenomic contigs by coverage and composition. *Nature Methods*, 11:1144, September 2014.
- 3 Kevin Chen and Lior Pachter. Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities. *PLOS Computational Biology*, 1(2), July 2005.
- 4 P J Deschavanne, A Giron, J Vilain, G Fagot, and B Fertil. Genomic signature: characterization and classification of species assessed by chaos game representation of sequences. *Molecular Biology and Evolution*, 16(10):1391–1399, October 1999. URL: <http://oup.prod.sis.lan/mbe/article-pdf/16/10/1391/9592103/mbe1391.pdf>.
- 5 Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi:10.1038/s41586-020-2649-2.

- 6 Dongwan D. Kang, Jeff Froula, Rob Egan, and Zhong Wang. MetaBAT, an efficient tool for accurately reconstructing single genomes from complex microbial communities. *PeerJ*, 3:e1165, 2015.
- 7 Dongwan D. Kang, Feng Li, Edward Kirton, Ashleigh Thomas, et al. MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ*, 7:e7359, 2019.
- 8 Daehwan Kim, Li Song, Florian P. Breitwieser, and Steven L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, 26(12):1721–1729, 2016. URL: <http://genome.cshlp.org/content/26/12/1721.full.pdf+html>.
- 9 Mikhail Kolmogorov, Derek M. Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy P. L. Smith, and Pavel A. Pevzner. metaflye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, November 2020. doi:10.1038/s41592-020-00971-x.
- 10 S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. doi:10.1214/aoms/1177729694.
- 11 Cedric C. Laczny, Christina Kiefer, Valentina Galata, et al. BusyBee Web: metagenomic data analysis by bootstrapped supervised binning and annotation. *Nucleic Acids Research*, 45(W1):W171–W179, May 2017. URL: <http://oup.prod.sis.lan/nar/article-pdf/45/W1/W171/18137403/gkx348.pdf>.
- 12 Cedric C. Laczny, Nicolás Pinel, Nikos Vlassis, and Paul Wilmes. Alignment-free Visualization of Metagenomic Data by Nonlinear Dimension Reduction. *Scientific Reports*, 4:4516, March 2014.
- 13 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, May 2018. URL: <http://oup.prod.sis.lan/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf>.
- 14 Hsin-Hung Lin and Yu-Chieh Liao. Accurate binning of metagenomic contigs via automated clustering sequences using information of genomic signatures and marker genes. *Scientific reports*, 6:24175–24175, April 2016. 27067514[pmid]. doi:10.1038/srep24175.
- 15 Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7:11257, April 2016. Article.
- 16 Fernando Meyer, Peter Hofmann, Peter Belmann, Ruben Garrido-Oter, Adrian Fritz, Alexander Sczyrba, and Alice C McHardy. AMBER: Assessment of Metagenome BinnerS. *GigaScience*, 7(6), June 2018. giy069. doi:10.1093/gigascience/giy069.
- 17 Alla Mikheenko, Vladislav Saveliev, and Alexey Gurevich. MetaQUAST: evaluation of metagenome assemblies. *Bioinformatics*, 32(7):1088–1090, November 2015. URL: <http://oup.prod.sis.lan/bioinformatics/article-pdf/32/7/1088/19568745/btv697.pdf>.
- 18 Jakob Nybo Nissen, Joachim Johansen, Rosa Lundbye Allesøe, Casper Kaae Sønderby, Jose Juan Almagro Armenteros, Christopher Heje Grønbech, Lars Juhl Jensen, Henrik Bjørn Nielsen, Thomas Nordahl Petersen, Ole Winther, and Simon Rasmussen. Improved metagenome binning and assembly using deep variational autoencoders. *Nature Biotechnology*, January 2021. doi:10.1038/s41587-020-00777-4.
- 19 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- 20 David Pellow, Itzik Mizrahi, and Ron Shamir. Plasclass improves plasmid sequence classification. *PLOS Computational Biology*, 16(4):1–9, April 2020. doi:10.1371/journal.pcbi.1007781.

- 21 Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, January 2013. URL: <http://oup.prod.sis.lan/bioinformatics/article-pdf/29/5/652/702231/btt020.pdf>.
- 22 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, 2020. doi:10.1038/s41592-019-0669-3.
- 23 Bianca K. Stöcker, Johannes Köster, and Sven Rahmann. SimLoRD: Simulation of Long Read Data. *Bioinformatics*, 32(17):2704–2706, May 2016. URL: <http://oup.prod.sis.lan/bioinformatics/article-pdf/32/17/2704/17346032/btw286.pdf>.
- 24 Ziyue Wang, Zhengyang Wang, Yang Young Lu, et al. SolidBin: improving metagenome binning with semi-supervised normalized cut. *Bioinformatics*, April 2019. btz253. URL: <http://oup.prod.sis.lan/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btz253/28579442/btz253.pdf>.
- 25 Aaron M. Wenger, Paul Peluso, William J. Rowell, Pi-Chuan Chang, Richard J. Hall, Gregory T. Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D. Olson, Armin Töpfer, Michael Alonge, Medhat Mahmoud, Yufeng Qian, Chen-Shan Chin, Adam M. Phillippy, Michael C. Schatz, Gene Myers, Mark A. DePristo, Jue Ruan, Tobias Marschall, Fritz J. Sedlazeck, Justin M. Zook, Heng Li, Sergey Koren, Andrew Carroll, David R. Rank, and Michael W. Hunkapiller. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37(10):1155–1162, October 2019. doi:10.1038/s41587-019-0217-9.
- 26 Anuradha Wickramarachchi, Vijini Mallawaarachchi, Vaibhav Rajan, and Yu Lin. MetaBCC-LR: metagenomics binning by coverage and composition for long reads. *Bioinformatics*, 36(Supplement\_1):i3–i11, July 2020. doi:10.1093/bioinformatics/btaa441.
- 27 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.
- 28 Yu-Wei Wu, Blake A. Simmons, and Steven W. Singer. MaxBin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets. *Bioinformatics*, 32(4):605–607, October 2015.
- 29 Yu-Wei Wu, Yung-Hsu Tang, Susannah G. Tringe, et al. Maxbin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome*, 2(1):26, 2014.
- 30 Guoxian Yu, Yuan Jiang, Jun Wang, et al. BMC3C: binning metagenomic contigs using codon usage, sequence composition and read coverage. *Bioinformatics*, 34(24):4172–4179, June 2018. URL: <http://oup.prod.sis.lan/bioinformatics/article-pdf/34/24/4172/27088792/bty519.pdf>.



## A Information About Datasets

Tables 3 and 4 demonstrate the simulated and real dataset information respectively. Note that the Table 3 tabulates the coverages used for simulation using SimLoRD [23].

■ **Table 3** Information of simulated datasets.

Dataset	Number of Reads	Total Size	Species	Coverage
Sim-8	432,333	3.5Gb	<i>Acetobacter pasteurianus</i>	25
			<i>Bacillus cereus</i>	50
			<i>Chlamydophila psittaci</i>	80
			<i>Escherichia coli</i>	125
			<i>Haemophilus parainfluenzae</i>	350
			<i>Lactobacillus casei</i>	200
			<i>Thermococcus sibiricus</i>	150
			<i>Streptomyces scabiei</i>	100
Sim-20	666,735	5.3Gb	<i>Amycolatopsis mediterranei</i>	25
			<i>Arthrobacter arilaitensis</i>	65
			<i>Brachyspira intermedia</i>	20
			<i>Corynebacterium ulcerans</i>	40
			<i>Erysipelothrix rhusiopathiae</i>	55
			<i>Enterococcus faecium</i>	50
			<i>Mycobacterium bovis</i>	80
			<i>Photobacterium profundum</i>	85
			<i>Streptococcus pyogenes</i>	100
			<i>Xanthobacter autotrophicus</i>	150
			<i>Rhizobium leguminosarum</i>	100
			<i>Francisella novicida</i>	150
			<i>Candidatus Pelagibacter ubique</i>	67
			<i>Halobacterium sp</i>	65
			<i>Lactobacillus delbrueckii</i>	60
			<i>Paenibacillus mucilaginosus</i>	90
<i>Rickettsia prowazekii</i>	100			
<i>Thermoanaerobacter brockii</i>	110			
<i>Yersinia pestis</i>	105			
<i>Nitrosococcus watsonii</i>	95			

■ **Table 4** Information of real datasets.

Dataset	Number of Reads	Total Size	Species	Abundance
MSA-1003	2,358,257	19Gb	<i>Acinetobacter baumannii</i>	0.18%
			<i>Bacillus pacificus</i>	1.80%
			<i>Bacteroides vulgatus</i>	0.02%
			<i>Bifidobacterium adolescentis</i>	0.02%
			<i>Clostridium beijerinckii</i>	1.80%
			<i>Cutibacterium acnes</i>	0.18%
			<i>Deinococcus radiodurans</i>	0.02%
			<i>Enterococcus faecalis</i>	0.02%
			<i>Escherichia coli</i>	18.0%
			<i>Helicobacter pylori</i>	0.18%
			<i>Lactobacillus gasseri</i>	0.18%
			<i>Neisseria meningitidis</i>	0.18%
			<i>Porphyromonas gingivalis</i>	18.0%
			<i>Pseudomonas aeruginosa</i>	1.80%
			<i>Rhodobacter sphaeroides</i>	18.0%
			<i>Schaalia odontolytica</i>	0.02%
			<i>Staphylococcus aureus</i>	1.80%
<i>Staphylococcus epidermidis</i>	18.0%			
<i>Streptococcus agalactiae</i>	1.80%			
<i>Streptococcus mutans</i>	18.0%			
SRX9569057	1,978,852	17Gb	<i>Faecalibacterium prausnitzii</i>	14.82%
			<i>Veillonella rogosae</i>	20.01%
			<i>Roseburia hominis</i>	12.47%
			<i>Bacteroides fragilis</i>	8.36%
			<i>Prevotella corporis</i>	6.28%
			<i>Bifidobacterium adolescentis</i>	8.86%
			<i>Fusobacterium nucleatum</i>	7.56%
SRX9569058	2,770,833	25Gb	<i>Lactobacillus fermentum</i>	9.71%
			<i>Clostridioides difficile</i>	1.10%
SRX9569059	2,480,208	20Gb	<i>Akkermansia muciniphila</i>	1.62%
			<i>Methanobrevibacter smithii</i>	0.17%
			<i>Salmonella enterica</i>	0.0065%
			<i>Enterococcus faecalis</i>	0.0011%
			<i>Clostridium perfringens</i>	0.00009%
			<i>Escherichia coli</i> (JM109)	1.83%
			<i>Escherichia coli</i> (B-3008)	1.82%
			<i>Escherichia coli</i> (B-2207)	1.65%
			<i>Escherichia coli</i> (B-766)	1.66%
			<i>Escherichia coli</i> (B-1109)	1.77%
<i>Candida albicans</i>	0.16%			
<i>Saccharomyces cerevisiae</i>	0.16%			