NOVEL ALGORITHMS TO ACCOUNT FOR UNCERTAINTIES IN THE SEQUENCING OF GENETIC MATERIAL WITH SKEWED ABUNDANCE

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I AT MĀNOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

AUGUST 2022

By
Cédric Arisdakessian

Dissertation Committee:

Guylaine Poisson, Chairperson
Mahdi Belcaid, Chairperson
Henri Casanova
Kyungim Baek
Scott Robertson
Monique Chyba

Acknowledgements

Taking the Ph.D. route was a life changing experience, that would not have been successful without my advisors, Dr. Guylaine Poisson and Dr. Mahdi Belcaid. Guylaine has become a very close friend and mentor during those five years. Her guidance and advice were critical to my success in my PhD journey. She made me feel welcome in the ICS community and valued as a researcher. Mahdi has been a close mentor from whom I owe most of my current data science skills. At his contact, I learned how to think more critically and be a better researcher. I will miss our encounters at coffee shops and our discussions on project ideas.

I would like to thank this dissertation's committee for taking the time to carefully review the manuscript and for their responsiveness whenever I was short on deadlines.

I wish to express my gratitude to Dr. Henri Casanova, for his insights in my everyday research efforts. He is an infinite source of knowledge that made me grow both as a computer scientist and as a person.

I wish to thank Dr. Kiana Frank for her positivity, selflessness and support in all my endeavours, as well as for sharing with me her research with the local community. Working in the Loʻi at Ulupō Heiau was an enriching experience I am glad I took part in.

I would like to express my thanks to the entire computer science department for making this journey possible, and in particular I would like to thank Ms. Janice Oda-Ng for making me feeling welcome, and for her help in all of my administrative matters.

Lastly, I would like to thank my friends and family for their continuous encouragement during those five years, and especially my mother Fabienne for her unwavering support in the last year of this PhD.

Abstract

The sequencing of genetic material (microbial DNA or RNA) is essential in biological experiments. However, while the cost of sequencing has decreased substantially, the highly skewed distribution of genetic material makes it challenging to accurately represent the genetic content of a sample. For instance, in DNA-based metagenomic experiments, DNA fragments are randomly sampled and used to identify and quantify organisms present in an environmental sample. Rare species are sampled less frequently, thus challenging subsequent bioinformatic analyses. Given the prevalence and the drastic implications of the uneven distribution of genetic material on bioinformatic analyses, our research focuses on new graph- and deep learning-based methods to address these issues in three different contexts. Specifically, we propose (1) an imputation method that can accurately recover the abundance of under-represented genetic material in single-cell RNA-seq experiments (2) a binning method to reduce genome fragmentation in viral metagenome sequencing experiments, and (3) a tool to explore and cluster viral populations based on their genomic structure. Our contributions focus on three popular biological contexts for which the issue of abundance hampers the bioinformatic analyses. Furthermore, the last two chapters focus on understanding viral diversity and modeling the genesis of novel virus strains through recombinations. Despite being at the core of the current COVID-19 crisis, the issue of recombination remains understudied, and few tools exist to model how viral populations evolve through recombination.

Table of Contents

Acknowledgements iii

Abstract iv

List of Tables vii

List of Figures viii

Chapter 1: Introduction 1

1.1 Background 2

1.1.1 The basic units of life 2

1.1.2 DNA and RNA sequencing 4

1.2 Problem statement 4

1.3 Contribution 7

1.4 Structure of the Dissertation 7

Chapter 2: DeepImpute: an accurate and efficient deep learning method for single-cell RNA-seq data imputation 10

2.1 Introduction 11

2.2 Results 13

2.2.1 Overview of the DeepImpute algorithm 13

2.2.2 DeepImpute is the most accurate among imputation methods on scRNA-seq data 14

2.2.3 DeepImpute improves the gene distribution similarity with FISH experimental data 17

2.2.4 DeepImpute improves downstream functional analysis 19
2.2.5 DeepImpute is a fast and memory efficient package 22
2.2.6 DeepImpute is a scalable machine learning method 22
2.3 Discussion 25
2.4 Methods 27
2.4.1 The workflow of DeepImpute 27
2.4.2 Evaluation metrics 28
2.4.3 Downstream functional analysis 29
2.4.4 RNA FISH validation 31
2.5 Availability of data and materials 32
2.5.1 scRNA-seq Datasets 32
2.5.2 Third party software 32
2.5.3 DeepImpute's material 33
Chapter 3: CoCoNet: an efficient deep learning tool for viral metagenome binning . 44
3.1 Introduction 45
3.2 Methods 46
3.3 Results 57
3.4 Discussion 64
3.5 Conclusion 66
3.6 Supplementary material 73
3.6.1 Supplementary methods 73
3.6.2 Supplementary figures 75
3.6.3 Supplementary tables 87
Chapter 4: Module painting 92
4.1 Introduction 92
4.2 Methods 95
4.2.1 Definitions 95
4.2.2 Module-painter: main steps and parameters 96

4.2.3 Clustering metrics 101
4.2.4 Data collection 102
4.3 Results 103
4.3.1 Module-painter reconstructs subpopulations in a simulated dataset . 103
4.3.2 Module-painter identifies recombinations in complete genomes 104
4.3.3 Module painter identifies recombinations in whole genome sequencing experiments 106
4.4 Discussion and conclusion 108
Chapter 5: Conclusion 113
5.1 Scientific contributions 113
5.2 Future work 114
5.3 Other contributions 114

List of Tables

S2.1 Single-cell datasets summary 42

3.1 Simulation parameters summary 58

S3.1 Dataset filtering summary 87

S3.2 Bin count summary for each method 87

S3.3 Hyperameter optimization of the parameters γ_1, γ_2, θ, `max_neighbors`. 91

List of Figures

1.1 Recombination process between 2 phages 3

1.2 Sequencing methods and impact of skewed genetic material distribution. . . 6

2.1 Neural network architecture of DeepImpute 14

2.2 Accuracy comparison between DeepImpute and other competing methods . 16

2.3 Comparison among imputation methods using RNA FISH data 18

2.4 Effect of imputation on downstream functional analysis on experimental data (GSE102827) 20

2.5 Effect of imputation on downstream functional analysis on simulated data using Splatter 21

2.6 Scalability comparison between imputation methods 24

S2.1 Preprocessing steps for DeepImpute 41

S2.2 Masking experiment in single cell RNA-Seq data 42

S2.3 Effect of dropout rate on imputation accuracy 43

S2.4 Accuracy comparison between DeepImpute and two other variant architectures 43

3.1 CoCoNet's neural network architecture and learning 49

3.2 CoCoNet clustering approach 52

3.3 Distribution of k-mer distances 59

3.4 Neural network performance on simulated data 60

3.5 Clustering performance on simulated data 62

3.6 Clustering performance on Station ALOHA 63
S3.1 Template length histogram 76
S3.2 Classification accuracy for simulated datasets 77
S3.3 Importance of coverage variability 78
S3.4 Distances in composition and coverage spaces 79
S3.5 Effect of contig size on binning performance 80
S3.6 Heatmap view of hyperparameter optimization 81
S3.7 Taxonomy of RefSeq's genomes 82
S3.8 Bin size histogram for simulated data 83
S3.9 Binning performance comparison 86

4.1 Recombination mechanism in phages 93
4.2 Example of coverage 95
4.3 Coverage computation steps in module-painter 98
4.4 Missing fragments matching 99
4.5 Parent selection 100
4.6 Module-painter on simulated data 104
4.7 Clustering of dairy phages 105
4.8 Recombination analysis on WGS dataset 107

Chapter 1

Introduction

DNA sequencing is the process of identifying the DNA sequence of the genetic material in a sample. Its popularity has kept increasing over the years, and sequencing facilities now provide fast and high-throughput services to sequence samples. However, the sequencing process cannot yet produce ready-to-use data, but instead generate fragmented and sometimes erroneous sequences. With the help of computational tools, it is possible to improve the quality of the sequencing output and help in the most challenging part, the data analysis.

The field of metagenomics, which consists in studying microbial communities in the environment, has strongly benefited from this technology. By studying the nature and abundance of microbes in an environment, we can learn more about its properties. Experimentally, the composition and abundance of microbes in an environmental sample (e.g., earth, water, dust, ...) can be estimated using whole-genome shotgun sequencing technologies, and consists of sequencing the entire genomes of the microbial populations. The sequencing process is complex and results in fragmented genomes that need to be stitched together using computational tools. Another popular use of sequencing technologies is the study of the activity of specific genes in a cell. In that case, we do not seek to capture different genomes, but rather identify individual RNA copies, which are much shorter than the whole genomes. For this reason, the genetic material can be sequenced without fragmentation.

Whether we are sequencing a whole genome or the RNA in single cells, the sequencing process amounts to random sampling of genetic material, and is therefore highly affected by its underlying distribution. Our research focuses on new graph- and deep learning-based methods to address these issues in three different contexts. Specifically, we propose (1) an imputation method that can accurately recover the abundance of under-represented genetic material in single-cell RNA-seq experiments (2) a binning method to reduce genome fragmentation in viral metagenome sequencing experiments and (3) a tool to explore and cluster viral populations based on their genomic structure.

1.1 Background

In this section, we provide the necessary bioinformatics and biology background for understanding the research presented in the following chapters.

1.1.1 The basic units of life

The functions of living organisms are encoded in their DeoxyriboNucleic Acid (DNA), which is made of two interleaved strands. A strand is a succession of molecules called nucleotides. A nucleotide can be identified by its base which is one of A (Adenosine), C (Cytosine), G (Guanine) or T (Thymine). At each position along the DNA, there are two complementary bases (on each strand), called a base pair (or "bp"), where the complement relationship is a mapping that maps A$\leftrightarrow$ T and C$\leftrightarrow$G. Because of this complementary relationship, a DNA sequence can be uniquely described with the sequence of its bases on one of the two strands. In the following, we interchangeably use the words base, base pair, or nucleotide since, for this dissertation, they correspond to the same entity.

The DNA can be seen as an efficient structure to compress the cell's information: the molecule itself is very stable and is preserved in the most secure part of the cell, the nucleus. Similar to a software library, it provides functions that can be used upon request. Each function is delineated in a region of the DNA (called a "gene") and can be transcribed into a transient form, the RNA. Finally, the RNA can be translated into its active form, the

protein, using a translation code mapping every three nucleotides (called a codon) to one of 22 amino acids. Therefore, the DNA can be viewed as a static structure showing the potential of a cell (all the possible things it can do), while the RNA and the proteins provide a dynamic view of the cell's activity at a certain time.

Every organism, including viruses, needs DNA or RNA to function. However, unlike other organisms, viruses lack the necessary material to survive on their own. Thus, they need to infect other living organisms (called "hosts") and use their machinery to replicate. Viruses are often categorized according to the type of host they infect. Bacteriophages, or simply phages, are the most common viruses on earth and, as their name suggests, infect bacteria. Viruses are ubiquitous throughout the earth [1], which makes encounters hard to avoid. Thus, bacteria have naturally evolved to fight viral infection through various mechanisms (such as the CRISPR-Cas system [2, 3], which is a bacterial mechanism that cleaves and saves a fragment of the infecting virus's genome in order to better target it the next time). In turn, viruses also evolved to bypass those mechanisms. However, viruses evolve at a much faster pace than other organisms, partly because of the process of recombination [4, 5], which consists of exchanging part of their DNA with other members of their species to produce two complementary children (see figure 1.1). This high sequence variability can challenge DNA recognition mechanisms in bacteria (e.g. CRISPR) if the saved fragment is altered.

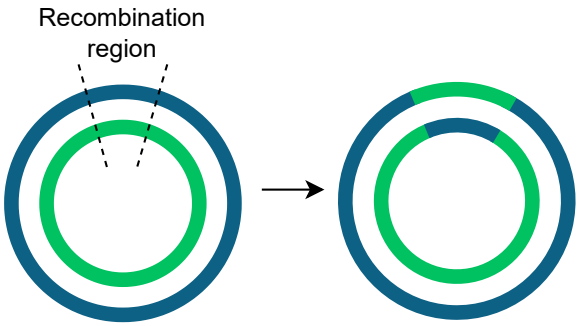


Figure 1.1: Recombination process between 2 phages

Both chapter 2 and 3 focus on viral metagenomes, and chapter 3 provides a more in-depth study of viral recombinations and how it affects the structure of viral genomes.

1.1.2 DNA and RNA sequencing

In this work, we focus on DNA and RNA sequencing experiments, which involve measuring the genetic content of a sample. In the case of DNA, our goal is to identify the genomes of the living organisms in the sample, whereas for RNA, our goal is to identify the genes being actively transcribed in a given context. Chapter 2 of this thesis focuses on single-cell RNA sequencing experiments, which consists in measuring the (transcription) activity of each individual cell in a sample.

DNA or RNA sequencing can today be easily achieved for a reasonable price [6]. Most sequencing technologies require the genetic fragments to be relatively short, which is rarely the case in practice for whole genome sequencing (a bacterial genome can easily reach up to several millions of base pairs). Therefore, long strands of DNA need to be sheared in smaller fragments before sequencing. After sequencing, we are left with short, overlapping sequences called "reads". Read assembly consists in reconstructing the original DNA using the overlap between the reads. This assembly step can be more or less challenging depending on the size of the DNA we sequence and its unique characteristics, the main one being repeated sequences of bases. The contiguous sequences, or contigs, resulting from the assembly can identify the organisms in our samples, and their abundance can simply be estimated by counting the number of times the reads span the length of the contigs.

1.2 Problem statement

Sequencing platforms can identify the genetic material in a sample by randomly selecting its fragments, amplifying and synthesizing them (in the case of the Illumina technology). In order to minimize sequencing errors and ensure that no fragments were missed, the platforms often sequence a sample multiple times. The number of times a sample's DNA is sequenced,

called the sequencing depth, directly affects the cost of the sequencing. Therefore, biologists have to find a trade-off between the number of samples they want to sequence and the level of details they need for their analysis given a fixed budget. Even with a very high sequencing depth, identifying all organisms in a metagenomic sample is not feasible in practice. Cost left aside, the main issue is that organisms' abundance is rarely uniformly distributed, but instead tends to follow a log-normal distribution [7]. This means that an even higher sequencing depth is required to detect low abundance organisms. For example, if a sample contains two distinct organisms of length m with the first one being 100 times more abundant than the second, then we need on average to sequence $101 \times m$ bases to measure the low abundance organism in its entirety. Furthermore, we often want to measure the same organism multiple times in order to reduce potential errors in sequencing. It is therefore rarely possible to fully sequence an environment with the currently available technologies.

The incomplete sequencing of a sample has different consequences depending on the type of experiment being carried out (see figure 1.2). In the case of whole-genome sequencing experiments, missing fragments is the cause of genome fragmentation. The assembly process reconstructs a metagenome by identifying overlapping fragments. If a fragment binding two genomic regions is missing, it results in two different genomes that cannot be reconstructed with approaches based on sequence overlap. Several tools have been developed to group split genomes by relying on other characteristics, such as the unique DNA patterns contigs contain. For the time being, the field is biased towards bacteria and is currently lacking in virus-specific tools.

In the case of RNA sequencing however, since the measured genetic region is much smaller (a single RNA), there is generally no issue with fragmentation. However, the log-normal distribution of counts makes it difficult to measure low abundance genes. A RNA physically present in multiple cells can be detected in one cell and not the other, artificially creating differences between the two. In practice, it is common to observe more than 50% of zeros in an abundance table (see figure 1.2).

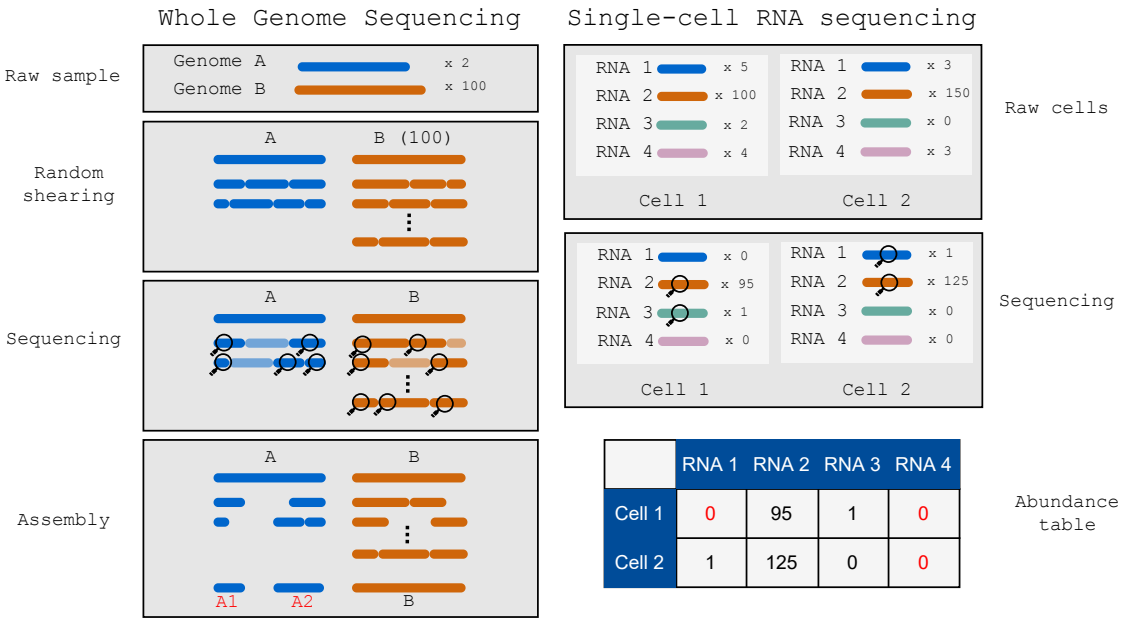


Figure 1.2: Sequencing methods and impact of skewed genetic material distribution.: (A) Whole genome sequencing for 2 genomes and a single sample. All the genomes are fragmented and sequenced at random. Because there are few copies of A, missed fragments cause the assembly to be fragmented into A1 and A2. B is however not fragmented because it is in high abundance, and other fragments can account for missing information. (B) RNA sequencing for 2 cells and 4 RNA types: The genetic material (RNA) we sequence is much shorter, so we do not need to shear it. However, some low abundance RNA remain unsequenced, which results in a zero count in the final abundance table.

1.3 Contribution

As previously mentioned, issues related to the sequencing of a sample with uneven genetic material distribution cannot be completely solved by simply increasing the sequencing depth. Efficient methods that account for missing fragments are therefore desperately needed. The first two contributions of this dissertation make use of the advances in the deep learning field to learn patterns in the data to identify missing data. First in chapter 2, we developed a Deep Learning-based missing value imputation tool for scRNA-seq data. Second, in chapter 3, we tackle the issue of genome fragmentation in viral metagenome sequencing experiments. Finally, in chapter 4 in an effort to improve our approach, we investigate the unique combinatorial properties of viruses in order to fine-tune species-level bins.

1.4 Structure of the Dissertation

This dissertation will be presented as a collection of peer-reviewed published (or submitted) papers.

Chapter 2 [8] presents a deep-learning based approach to impute missing values in single-cell RNA-seq experiments.

Chapter 3 [9] presents a deep-learning based approach for inferring split contigs after assembling a viral metagenome.

Chapter 4 investigates viral recombination as a potential feature to improve the approach described in chapter 3.

References

[1] A. G. Cobián Güemes, M. Youle, V. A. Cantú, B. Felts, J. Nulton, and F. Rohwer, "Viruses as winners in the game of life," *Annual Review of Virology*, vol. 3, pp. 197–214, 2016.

[2] F. J. Mojica, C. Díez-Villaseñor, J. García-Martínez, E. Soria, *et al.*, "Intervening sequences of regularly spaced prokaryotic repeats derive from foreign genetic elements," *Journal of molecular evolution*, vol. 60, no. 2, pp. 174–182, 2005.

[3] A. F. Andersson and J. F. Banfield, "Virus population dynamics and acquired virus resistance in natural microbial communities," *Science*, vol. 320, no. 5879, pp. 1047–1050, 2008.

[4] D. Botstein, "A theory of modular evolution for bacteriophages," *Annals of the New York Academy of Sciences*, vol. 354, no. 1, pp. 484–491, 1980.

[5] A. A. Hossain, J. McGinn, A. J. Meeske, J. W. Modell, and L. A. Marraffini, "Viral recombination systems limit crispr-cas targeting through the generation of escape mutations," *Cell Host & Microbe*, 2021.

[6] K. A. Wetterstrand, "Dna sequencing costs: data from the nhgri genome sequencing program (gsp). 2013," *URL http://www. genome. gov/sequencingcosts*, 2016.

[7] W. Ulrich, M. Ollik, and K. I. Ugland, "A meta-analysis of species–abundance distributions," *Oikos*, vol. 119, no. 7, pp. 1149–1155, 2010.

[8] C. Arisdakessian, O. Poirion, B. Yunits, X. Zhu, and L. X. Garmire, "Deepimpute: an accurate, fast, and scalable deep neural network method to impute single-cell rna-seq data," *Genome biology*, vol. 20, no. 1, pp. 1–14, 2019.

[9] C. G. Arisdakessian, O. D. Nigro, G. F. Steward, G. Poisson, and M. Belcaid, "Coconet: an efficient deep learning tool for viral metagenome binning," *Bioinformatics*, 2021.

Chapter 2

DeepImpute: an accurate and efficient deep learning method for single-cell RNA-seq data imputation

Abstract

Single-cell RNA sequencing (scRNA-seq) offers new opportunities to study gene expression of tens of thousands of single cells simultaneously. We present DeepImpute, a deep neural network-based imputation algorithm that uses dropout layers and loss functions to learn patterns in the data, allowing for accurate imputation. Overall, DeepImpute yields better accuracy than other six publicly available scRNA-seq imputation methods on experimental data, as measured by the mean squared error or Pearson's correlation coefficient. DeepImpute is an accurate, fast, and scalable imputation tool that is suited to handle the ever-increasing volume of scRNA-seq data, and is freely available at `https://github.com/lanagarmire/DeepImpute`.

Published: Cédric Arisdakessian, Olivier Poirion, Breck Yunits, Xun Zhu, and Lana X. Garmire. "DeepImpute: an accurate, fast, and scalable deep neural network method to impute single-cell RNA-seq data." Genome biology 20, no. 1 (2019): 1-14.

2.1 Introduction

The RNA sequencing technologies keep evolving and offering new insights to understand biological systems. In particular, single-cell RNA sequencing (scRNA-seq) represents a major breakthrough in this field. It brings a new dimension to RNA-seq studies by zooming in to the single-cell level. Currently, various scRNA-seq platforms are available such as Fluidigm- and Drop-Seq-based methods. While Drop-Seq can process thousands of cells in a single run, Fluidigm generally processes fewer cells but with a higher coverage. In particular, 10X Genomics' platform is gaining popularity in the scRNA-seq community due to its high yield and low cost per cell. Consequently, an increasing number of studies have taken advantage of these technologies to discover new cell types [1, 2], new markers for specific cell types [1, 3, 4], and cellular heterogeneity [4, 5, 6, 7, 8, 9].

Despite these advantages, scRNA-seq data are very noisy and incomplete [10, 11, 12] due to the low starting amount of mRNA copies per cell. Datasets with more than 70% missing (zero) values are frequently observed in an scRNA-seq experiment. These apparent zero values could be truly zeros or false negatives. The latter phenomenon is called "dropout" [13] and is due to failure of amplification of the original RNA transcripts. Among genes of various lengths, shorter genes are more likely to be dropped out [14]. Such bias may increase further during the subsequent amplification steps. As a result, dropout can affect downstream bioinformatics analysis significantly, such as clustering [15] and pseudo-time reconstruction [16], as it decreases the power of the studies and introduces biases in gene expression. To correct such issue, analysis platforms such as Granatum [17] have included an imputation step, in order to improve the downstream analysis.

Several imputation algorithms have been proposed, based on different principles and models. MAGIC [18] focuses on cell/cell interactions to build a Markov transition matrix and smooth the data. ScImpute [19] builds a LASSO regression model for each cell and imputes them iteratively. SAVER [20] is a Bayesian-based model using various prior probability functions. DrImpute [21] is a clustering-based method and uses a consensus

strategy: it estimates a value with several cluster priors or distance matrices and then imputes by aggregation. VIPER is a recent published statistical method that looks at cell/cell interaction to fit a linear model for each cell. Instead of using a LASSO regression as for scImpute, the authors use a hard thresholding approach to limit the number of predictors [22]. Most recently, DCA builds an auto-encoder to model the genes distribution using a zero inflated negative binomial prior. To this end, the auto-encoder tries to predict the genes' mean, standard deviation, and dropout probability [23]. As the low quality of the scRNA-seq datasets continues to be a bottleneck while the measurable cell counts keep increasing, the demand for faster and scalable imputation methods also keeps increasing [23, 24, 25]. While some of these earlier algorithms may improve the quality of original datasets and preserve the underlying biological variance [26], most of them demand extensive running time, impeding their adoption in the ever-increasing scRNA-seq data space.

In this chapter, we present a novel algorithm, DeepImpute, as the next generation imputation method for scRNA-seq data. DeepImpute is short for "Deep neural network Imputation". As reflected by the name, it belongs to the class of deep neural-network models [27, 28, 29]. Recent years, deep neural network algorithms have gained much interest in the biomedical field [30], ranging from applications from extracting stable gene expression signatures in large sets of public data [31] to stratify phenotypes [32] or impute missing values [33] using electronic health record (EHR) data. In this report, we construct DeepImpute models by splitting the genes into subsets and build sub-networks to increase its efficacy and efficiency. Using accuracy metrics, we demonstrate that DeepImpute performs better than the six other recently published imputation methods mentioned above (MAGIC, DrImpute, ScImpute, SAVER, VIPER, and DCA). It also improves the downstream analysis results, on clustering using both real and simulated datasets, as well as on differential expression using a simulated dataset. We additionally show the superiority of DeepImpute over the other methods in terms of computational running time and memory use. Moreover, DeepImpute allows to train the model with a subset of the data to save computing time, with little detriment to prediction accuracy. In summary, DeepImpute is a fast, scalable,

and accurate next generation imputation method capable of handling the ever-increasing scRNA-seq data.

2.2 Results

2.2.1 Overview of the DeepImpute algorithm

DeepImpute is a deep neural network model that imputes genes in a divide-and-conquer approach, by constructing multiple sub-neural networks (Fig. S2.1). Doing so offers the advantage of reducing the complexity by learning smaller problems and fine-tuning the sub-neural networks [34]. For each dataset, we select to impute a list of genes, which have a certain variance over mean ratio (default=0.5). Each sub-neural network aims to understand the relationship between the input genes (input layer) and a subset of target genes (output layer) (Fig. 2.1). Users can set the size of the target genes, and we set 512 as the default value, as it offers a good trade-off between speed and stability. As shown in Fig.2.1, each sub-neural network is composed of four layers. The input layer consists of genes that are highly correlated with the target genes, followed by a 256-neuron dense hidden layer, a dropout layer with 20% dropout rate (note: not the dropout rate in the single cell data matrix) of neurons which avoid overfitting (Fig. S2.3), and the output neurons made of the above mentioned target genes. We use rectified linear unit (ReLU) as the default activation function and train each sub-model in parallel by splitting the data to train (95% of the cells) and test (5%) data. We stop the training if the test loss does not improve for 5 consecutive epochs or the number of epochs exceeds 500, whichever is smaller. Because of the simplicity of each sub-network, we observe very low variability due to hyperparameter tuning. As a result, we set the default parameters for batch size at 64 and learning rate at 0.0001. Further information about the network parameters are described in the Methods, section 2.4. In the following sections, we describe the comprehensive evaluations of DeepImpute.

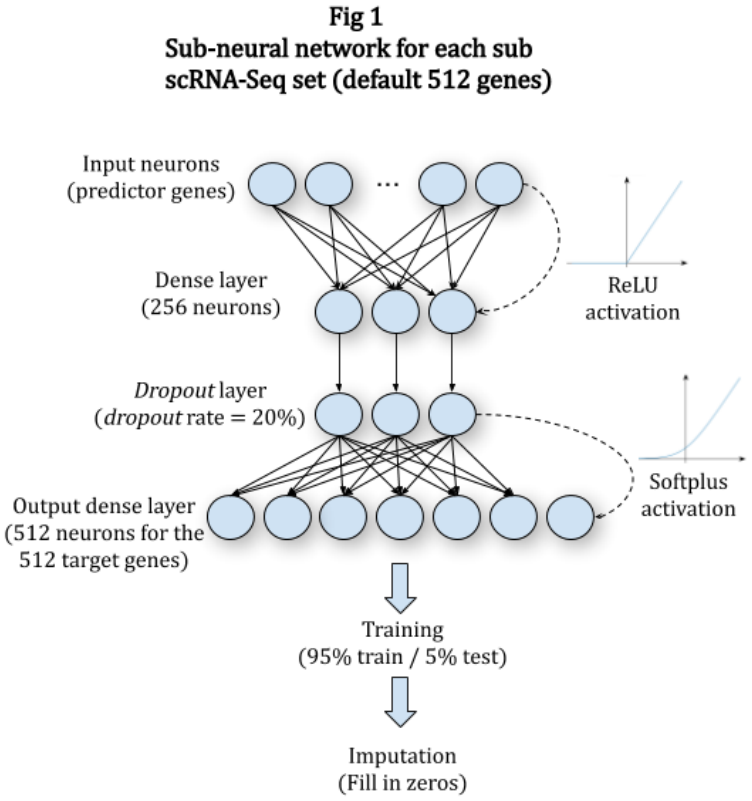


Figure 2.1: Neural network architecture of DeepImpute: Each sub-neural network is composed of four layers. The input layer is genes that are highly correlated with the target genes in the output layer. It is followed by a dense hidden layer of 256 neurons and a dropout layer (dropout rate=20%). The output layer consists of a subset of target genes (default N=512), whose zero values are to be imputed.

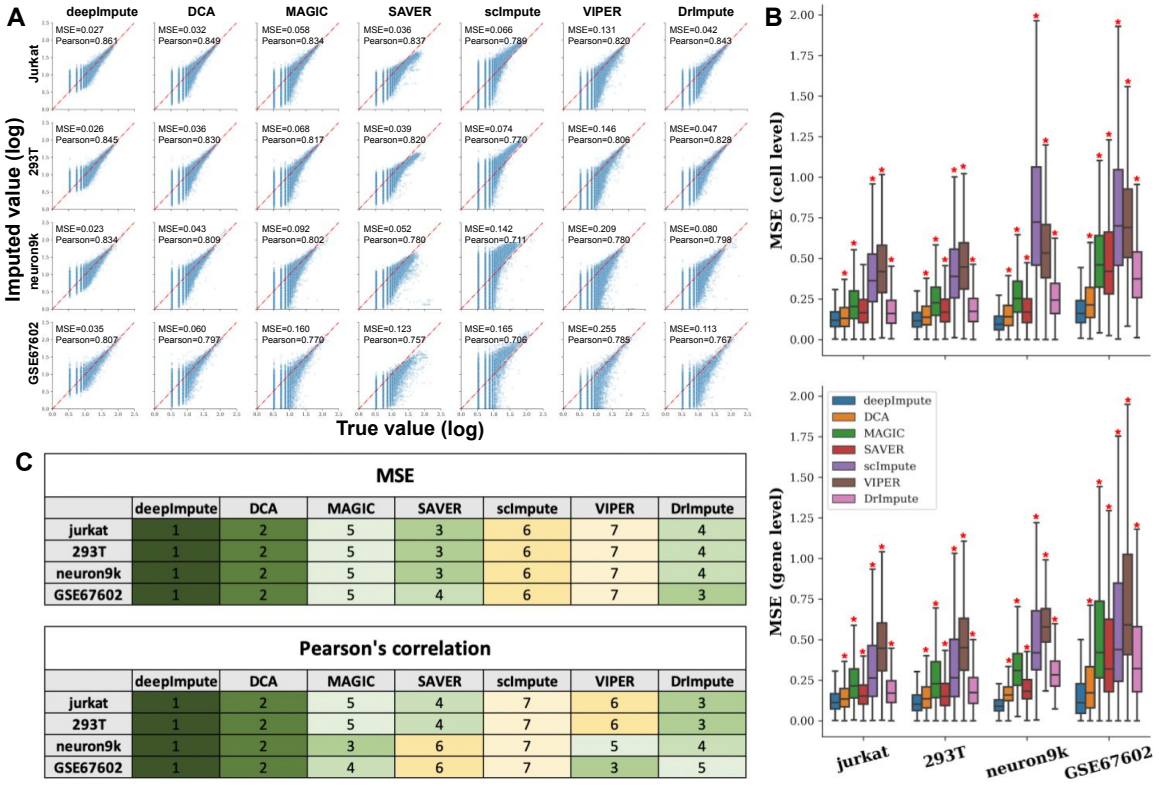
2.2.2 DeepImpute is the most accurate among imputation methods on scRNA-seq data

We tested the accuracy of imputation on four publicly available scRNA-seq datasets (Table S2.1): two cell lines, Jurkat and 293T (10X Genomic); one mouse neuron cells dataset (10X Genomics); and one mouse interfollicular epidermis dataset deposited in GSE67602. We compared DeepImpute with six other state-of-the-art, representative algorithms: MAGIC, DrImpute, ScImpute, SAVER, VIPER, and DCA. Since the real dropout values are unknown, we evaluated the different methods by randomly masking (replacing with zeros) a part of the expression matrix of a scRNA-seq dataset (Fig. S2.2) and then measure the

differences between the inferred and actual values of the masked data. In order to mimic a more realistic dropout distribution, we estimated the masking probability function from the data (see Methods, section 2.4). We measured the accuracies using the two metrics on the masked values: Pearson's correlation coefficient and mean squared error (MSE), as done earlier [20, 35].

Figure 2.2 shows all the results of imputation accuracy metrics on the masked data. DeepImpute successfully recovers dropout values from all ranges, introduces the least distortions and biases to the masked values, and yields both the highest Pearson's correlation coefficient and the best (lowest) MSE in all datasets (Fig. 2.2A and C). DCA, another neural-network-based method, has the second best performance after DeepImpute, based on both MSE and Pearson's correlation coefficient. By contrast, other methods present various issues: VIPER tends to underestimate the original values, as reflected by the largest MSEs. scImpute has the widest range of variations among imputed data and generates the lowest Pearson's correlations. MAGIC, SAVER, and DrImpute have intermediate performances compared to other methods. However, SAVER persistently underestimate the values, especially among the highly expressed genes. We further examined MSE distributions calculated on the gene and cell levels (Fig. 2.2B). DeepImpute is the clear winner with consistently the best (lowest) MSEs both at gene and cell levels on all datasets, which are significantly lower than all other imputation methods ($p<0.05$). scImpute and VIPER give the two highest MSEs at the cell level, whereas VIPER consistently has the highest MSE at the gene level (Fig. 2.2B). Other methods are ranked in between, with varying rankings depending on the datasets and gene or cell level. As internal controls, we also compared DeepImpute (with ReLU activation) with 2 variant architectures: the first one with no hidden layers and the second one with the same hidden layers but using linear activation function (instead of ReLU). As shown in Fig.S2.4, DeepImpute (with ReLU activation) yields Pearson's correlation coefficients and MSEs that are either on par or better than the two other variant architectures. For GSE67602 and neuron9k datasets generated from complex animal primary tissues, DeepImpute (with ReLU activation) performs better;

for Jurkat and 293T datasets generated from cell lines, the results are comparable. This suggests that DeepImpute (with ReLU activation) handles complex datasets better than its competitors. In summary, DeepImpute yields the highest accuracy in the datasets studied, among the imputation methods in comparison.



C

MSE

	deepImpute	DCA	MAGIC	SAVER	scImpute	VIPER	DrImpute
jurkat	1	2	5	3	6	7	4
293T	1	2	5	3	6	7	4
neuron9k	1	2	5	3	6	7	4
GSE67602	1	2	5	4	6	7	3

Pearson's correlation

	deepImpute	DCA	MAGIC	SAVER	scImpute	VIPER	DrImpute
jurkat	1	2	5	4	7	6	3
293T	1	2	5	4	7	6	3
neuron9k	1	2	3	6	7	5	4
GSE67602	1	2	4	6	7	3	5

Figure 2.2: Accuracy comparison between DeepImpute and other competing methods: (A) Scatter plots of imputed vs. original data masked. The x-axis corresponds to the true values of the masked data points, and the y-axis represents the imputed values. Each row is a different dataset, and each column is a different imputation method. The mean squared error (MSE) and Pearson's correlation coefficients (Pearson) are shown above each dataset and method. (B) Bar graphs of cell-cell and gene-gene level MSEs between the true (masked) and imputed values, based on those in (A). Asterisk indicates statistically significant difference ($P<0.05$) between DeepImpute and the imputation method of interest using the Wilcoxon rank-sum test. Color labels for all imputation methods are shown in Figure (C). Ranking of each method for all four datasets for both overall MSE and Pearson's correlation coefficient

2.2.3 DeepImpute improves the gene distribution similarity with FISH experimental data

Another way to assess imputation efficiency is through experimental validation on scRNA-Seq data. Single-cell RNA FISH is such a method that directly detects a small number of RNA transcripts in a single cell. Torre et al. measured the gene expression of a melanoma cell line using both RNA FISH and Drop-Seq and compared their distribution using their GINI coefficients (see the "Methods" section) [36]. Similarly, we compared the same list of genes using their GINI coefficients of RNA FISH vs. those after imputation (or raw scRNA-seq data). DrImpute could not handle the large cell size and was omitted from comparison. Comparing to Pearson's correlation coefficient between RNA FISH and the raw scRNA-seq data (-0.260), three methods, DeepImpute, SAVER, and DCA, have the top 3 most improved and positive correlation coefficients, with values of 0.984, 0.782, and 0.732, respectively. VIPER barely changed the GINI coefficients, whereas scImpute had a correlation coefficient (-0.451) even lower than the raw scRNA-seq dataset (Fig. 2.3A). For MSE, all other imputation methods achieved better (smaller) MSEs compared to the raw scRNA-seq results (MSE=0.178), except VIPER, which gives the same MSE as raw data. Echoing the results of correlation coefficient, three methods, SAVER, DeepImpute, and DCA, give the lowest MSEs. DeepImpute is the second most accurate method with an MSE (MSE=0.0259), closely after SAVER (MSE=0.0152) and followed by DCA (MSE=0.0436). Additionally, we compared the distributions of each gene before and after various imputation methods, as well as in FISH experiments (Fig. 2.3B). Overall, DeepImpute (blue curves) yields the most similar distributions to those of FISH experiments (gray curves) for three of five genes (LMNA, MITF, and TXNRD1), with K-S test statistics of 0.08, 0.15, and 0.18, respectively. For KDM5A, it achieved 2nd best K-S statistics 0.18, almost the same as DCA (0.17). It does not perform as well for gene VGF (K-S statistic of 0.44), which has over 40% zero values even in RNA-FISH data (56% in raw Drop-Seq data). Altogether, the FISH validation results clearly show that DeepImpute improves the data quality by imputation.

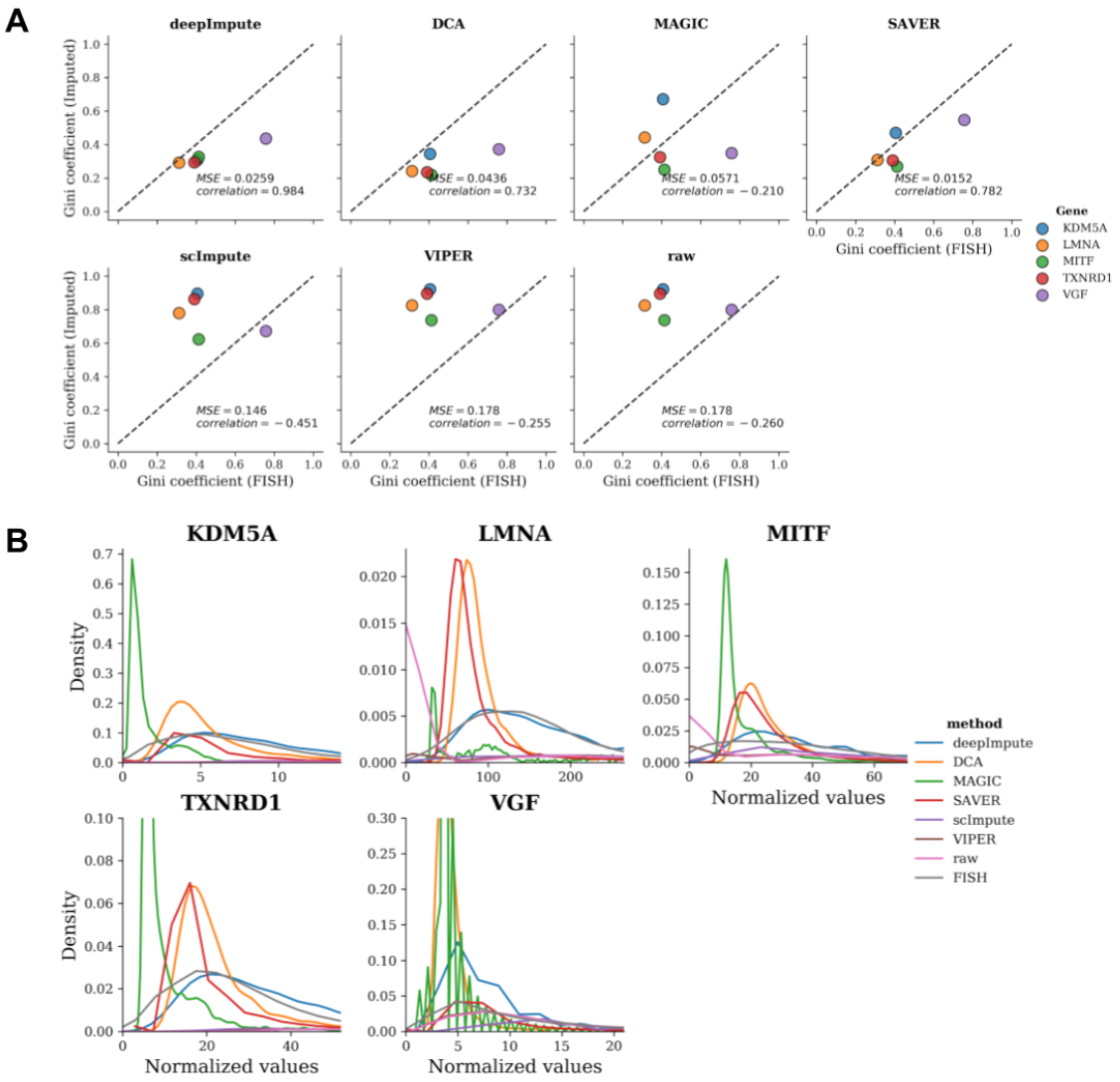


Figure 2.3: Comparison among imputation methods using RNA FISH data: (A) Scatter plots of GINI coefficients from the imputed (or raw) vs. FISH data. The x-axis is the "true" GINI coefficient as determined by FISH experiments, and the y-axis is the imputed (or raw) GINI coefficient. The Pearson's correlation coefficients (Pearson) and mean squared error (MSE) are shown for each method. Colors represent different genes: KDM5A (blue), LMNA (orange), MITF (green), TXNRD1 (red), and VGF (purple). (B) Gene distributions for seven imputation methods: DeepImpute (blue), DCA (yellow), MAGIC (green), SAVER (red), scImpute (purple), VIPER (brown), raw (pink), and FISH (gray) data

2.2.4 DeepImpute improves downstream functional analysis

Another way to assess possible benefits of imputation is to conduct downstream functional analysis. To this end, we utilized additional experimental and simulation datasets. We use an experimental dataset (Hrvatin) from GSE102827, composed of 48,267 annotated primary visual cortex cells from mice and which had 33 prior cell type labels [37]. Using the Seurat pipeline implemented in Scanpy, we extracted the UMAP [38] components (Fig. 2.4A). We then performed cell clustering using the Leiden clustering algorithm [39], an improved version of the Louvain algorithm [40]. We measure the accuracy of clustering assignments using various metrics, including the Adjusted Rand Index (ARI), the Adjusted Mutual Score (AMS), the Fowlkes-Mallow Index (FMI), and Silhouette Index (SI) to exam UMAP cluster shapes (Fig. 2.4B). Due to the size of the Hrvatin dataset, we could not run DrImpute and VIPER (speed issues) as well as scImpute (speed and memory issues), but only DeepImpute, DCA, MAGIC, and SAVER. DeepImpute manages to disentangle many clusters (Fig. 2.4A), resulting in the most improved clustering metrics compared to the scenario without imputation (Fig. 2.4B). DCA, the other deep neural-network-based method, also slightly improves the clustering metrics (Fig. 2.4B). On the contrary, MAGIC and SAVER decrease, rather than improve the clustering outcome. Notably, MAGIC manages to split many cell types but also highly distorts the data (Fig. 2.4A). SAVER disentangles some clusters, but also splits some clusters beyond the original cell type labels (Fig. 2.4A).

Given lack of absolute truth of class labels in experimental data, we next generated a simulation data using Splatter. This simulation dataset (sim) is composed of 4000 genes and 2000 cells, which are split into 5 cell types (proportions: 5%/10%/20%/20%/40%). DeepImpute successfully separates cell types on the simulation, closely followed by scImpute (Fig. 2.5A). These observations are confirmed by the evaluation metrics, where DeepImpute achieves almost perfect scores for ARI, AMS, and FMI and significantly increases the Silhouette score compared to the raw data (Fig. 2.5B). Next, we compare all seven imputation methods for their capabilities to recover differentially expressed genes in the

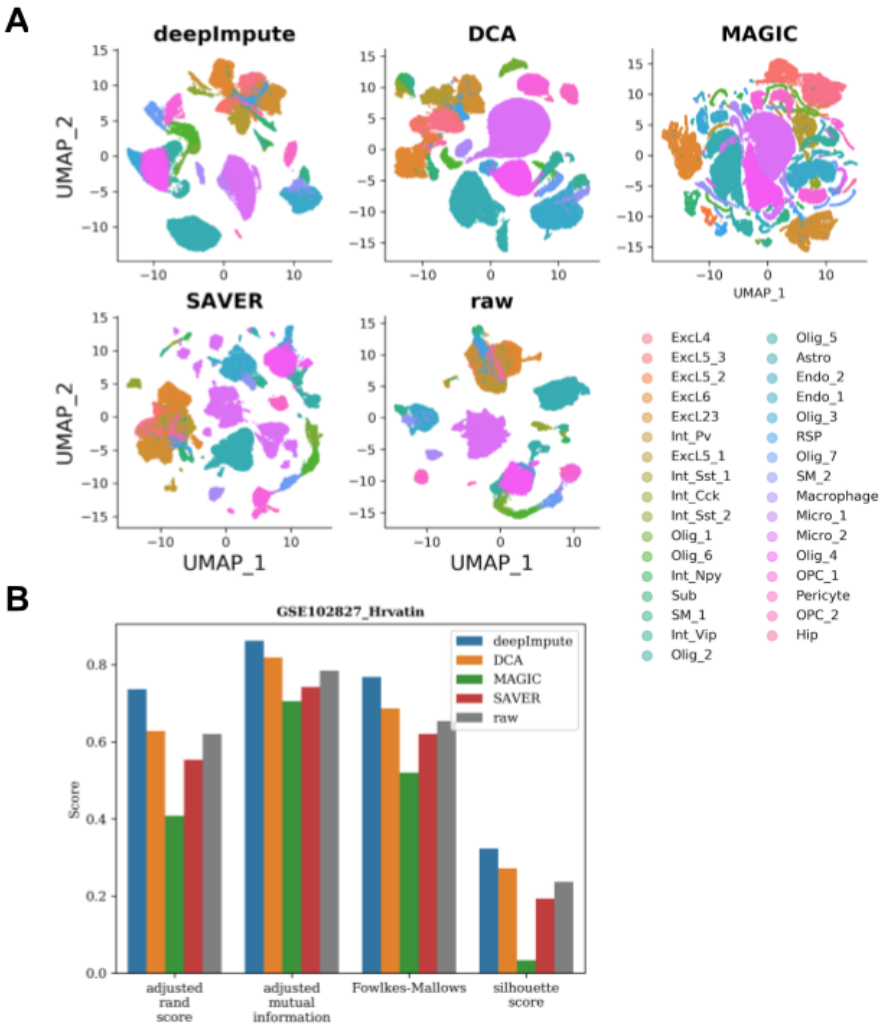


Figure 2.4: Effect of imputation on downstream functional analysis on experimental data (GSE102827): (A) UMAP plots of DeepImpute, DCA, MAGIC, SAVER, and raw data (scImpute, DrImpute, and VIPER) failed to run due to the large cell size of 48,267 cells. Colors represent original cell type labels as annotated. (B) Accuracy measurements of clustering using various metrics: adjusted Rand index (adjusted_rand_score), adjusted mutual information, Fowlkes–Mallows Index (Fowlkes-Mallows), and Silhouette coefficient (Silhouette score). Higher values indicate better clustering accuracy. Bar colors represent different methods: DeepImpute (blue), DCA (orange), MAGIC (green), SAVER (red), and raw data (gray)

simulation data (Fig. 2.5C). For each method, we extracted the top 500 differentially expressed genes in each cell type and compared with the true differentially expressed genes. Overall, DeepImpute has the highest precision (AUC=0.894) at detecting differentially

expressed genes, compared to those of no imputation and other imputation methods. Altogether, these results from both experimental and simulation data show unanimously that DeepImpute improves downstream functional analysis.

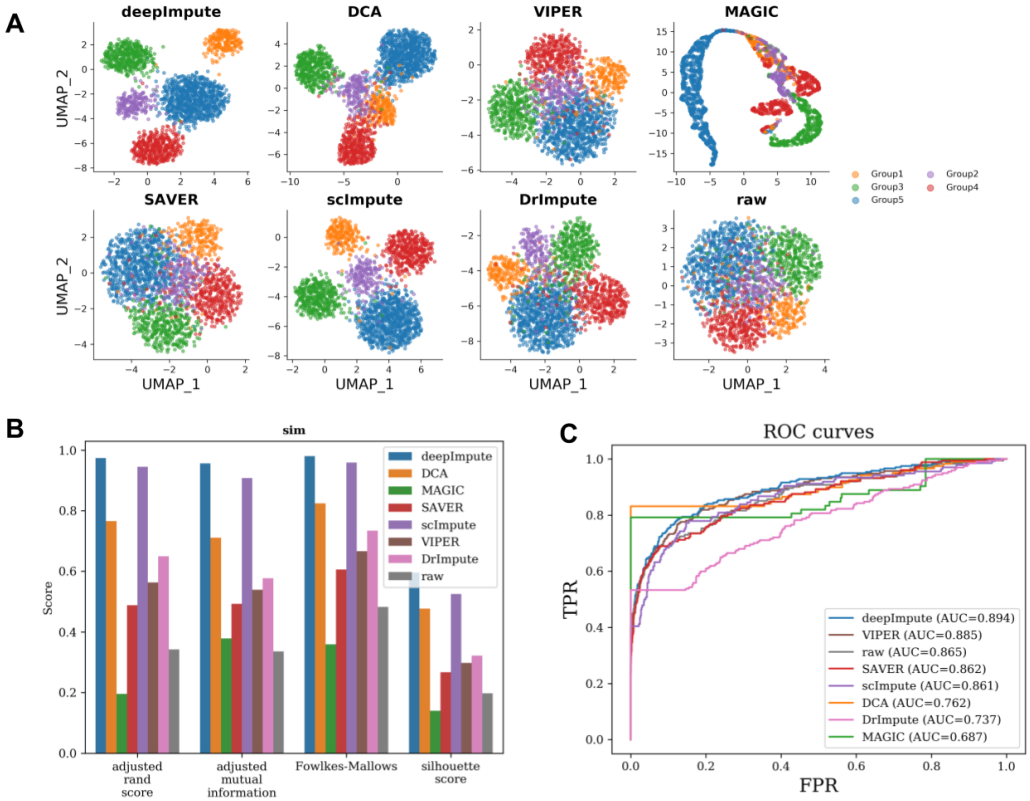


Figure 2.5: Effect of imputation on downstream functional analysis on simulated data using Splatter: This simulation dataset is composed of 4000 genes and 2000 cells, split into 5 cell types (proportions: 5%/10%/20%/20%/40%). (A) UMAP plots of DeepImpute, DCA, MAGIC, SAVER, scImpute, VIPER, DrImpute, and raw data. Each color represents one of the 5 cell types. (B) Accuracy measurements of clustering using the same metrics as in Fig. 2.4B. Bar colors represent different methods as shown in the figure. (C) Accuracy measurements of differentially expressed genes by different imputation methods. The top 500 differentially expressed genes in each cell type are used to compare with the true differentially expressed genes in the simulated data, over a range of adjusted p values for each method. Colors represent different methods as shown in the figure.

2.2.5 DeepImpute is a fast and memory efficient package

As scRNA-seq becomes more popular and the number of sequenced cells scales exponentially, imputation methods will have to be computationally efficient to be widely adopted. With such a goal in mind, we choose the Mouse1M dataset to evaluate the computational speed and memory usage among different imputation methods. We use Mouse1M dataset as it has the highest number of cells to assess how adaptive each method is.

We downsampled the Mouse1M data, ranging in size from 100 to 50k cells (100, 500, 1k, 5k, 10k, 30k, 50k). We ran the imputations three times and measured the runtime (for both training and testing steps) and memory load on an 8-core machine with 30 GB of memory. DeepImpute, DCA, and MAGIC outperformed the other four packages on speed (Fig. 2.6A), and DCA and DeepImpute are the most advantageous when the cell counts get large (>30k). DCA is consistently and slightly faster than DeepImpute through all tests. The other four imputation methods (scImpute, DrImpute, VIPER, and SAVER) are significantly slower and consume significantly more memory (Fig. 2.6B). The slow computation time of VIPER and DrImpute are due to lack of parallelization. VIPER is unable to scale beyond 1k cells within 24h, while scImpute exceeded the 30 GB of memory available and failed to run on more than 10k cells. For memory, DeepImpute and DCA, two neural-network-based methods, are the most efficient, and their merits are much more pronounced on large datasets (Fig. 2.6B). MAGIC uses a similar amount of memory as DeepImpute and DCA on smaller datasets; however, as the dataset size increases beyond 10k cells, it requires significantly more memory. It hits an out of memory error and is unable to finish the 50k cell imputation on our 30GB machine. In all, judging by both computation speed and memory efficiency on larger datasets, DeepImpute and DCA outperform the other five methods.

2.2.6 DeepImpute is a scalable machine learning method

Unlike all of the other imputation methods (except DCA), DeepImpute first fits a predictive model and then performs imputation separately. The model fitting step uses most of the

computational resources and time, while the prediction step is very fast. We then asked the question: What is the minimal fraction of the dataset needed to train DeepImpute and obtain efficient imputation without extensive training time? To answer this question, we used the neuron9k dataset and evaluated the effect of different subsampling fraction (5%, 10%, 20%, 40%, 60%, 80%, 90%, 100%) in the training phase on the imputation prediction phase. We randomly picked a subset of the samples for the training step and computed the accuracy metrics (MSE, Pearson's correlation coefficient) on the whole dataset, with 10 repetitions under each condition. Model performance improvement begins to slow down at around 40% of the cells (Fig. 2.6C). Specifically, from 40 to 100% fraction of data as the training set, the MSE decreases slightly from 0.121 to 0.116, and Pearson's coefficient score marginally improves from 0.880 to 0.884. These experiments demonstrate another advantage of DeepImpute over the other competing methods, that is, the use of only a fraction of the data set reduces the running time even more with little detriment to the accuracy of the imputed results.

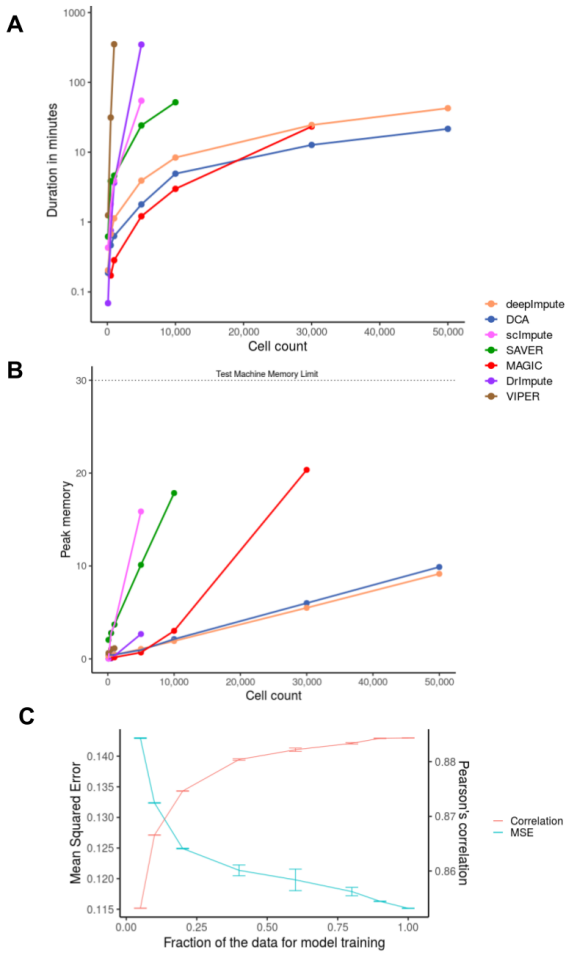


Figure 2.6: Scalability comparison between imputation methods: Speed and memory usage comparison among imputation methods, as well as the effect of subsampling training data on DeepImpute accuracy. For deep learning models, the time reported corresponds to training and testing combined. (A, B) Speed and memory comparisons on the Mouse1M dataset. This dataset is chosen for its largest cell numbers. Color labels different imputation methods. (A) Speed average over 3 runs. The x-axis is the number of cells, and the y-axis is the running time in minutes (log scale) of the imputation process. (B) RAM memory usage. The x-axis is the number of cells, and the y-axis is the maximum RAM used by the imputation process. Because of the limited amount of memory or time, scImpute, SAVER, and MAGIC exceeded the memory limit respectively at 10k, 30k, and 50k cells, thus no measurements at these and higher cell counts. VIPER and DrImpute each exceeded 24h on 5k and 10k cells; therefore, they too do not have measurements at these and higher cell counts. (C) The effect of subsampling training data on DeepImpute accuracy. Neuron9k dataset is masked and measured for performance as in Fig. 2.2. x-axis is the fraction of cells in the training data set, and y-axis labels are values for mean squared error (left) and Pearson's correlation coefficient (right). Color labels are as indicated in the graph. Error bars represent the standard deviations over the 10 repetitions

2.3 Discussion

Dropout values in scRNA-seq experiments represent a serious issue for bioinformatic analyses, as most bioinformatics tools have difficulty handling sparse matrices. In this chapter, we present DeepImpute, a new algorithm that uses deep neural networks to impute dropout values in scRNA-seq data. We show that DeepImpute not only has the highest overall accuracy using various metrics and a wide range of validation approaches, but also offers faster computation time with less demand on the computer memory. In both simulated and experimental datasets, DeepImpute shows benefits in increasing clustering results and identifying significantly differentially expressed genes, even when other imputation methods are not desirable. Furthermore, it is a very "resilient" method. The model trained on a fraction of the input data can still yield decent predictions, which can further reduce the running time. Together, these results demonstrate consistently and robustly that DeepImpute is an accurate and highly efficient method, and it is likely to withstand the tests of time, given the rapid growth of scRNA-Seq data volume.

Through systematic comparisons, two deep-learning-based methods, DeepImpute and DCA, show overall advantages over other methods, between which DeepImpute performs even better. Several unique properties of DeepImpute contribute to its superior performance. One of them is using a divide-and-conquer approach. This approach has several benefits. First, contrary to an auto-encoder as implemented in DCA, the subnetworks are trained without using the target genes as the input. It reduces overfitting while enforcing the network to understand true relationships between genes. Second, splitting the genes into subsets results in a lower complexity in each sub-model and stabilizing neural networks. As a result, a small change in the hyperparameters has little effect on the result. Using a single set of hyperparameters, DeepImpute achieves the highest accuracies in all four experimental datasets (Fig. 2.2A). Third, splitting the training into sub-networks results in increased speed as there are fewer input variables in

each subnetwork. Also, training of each sub-network is done in parallel on different threads, which is more difficult to do with one single neural network.

Unlike some other imputation algorithms, DeepImpute is a machine learning method. The training and the prediction processes of DeepImpute are separate, and this may provide more flexibility when handling large datasets. Moreover, we have shown that using only a fraction of the overall samples, one can still obtain decent imputation results without sacrificing the accuracy of the model much, thus further reducing the running time. Perhaps, another advantage of DeepImpute over other methods is that it can pre-train a dataset of a cell type (or cell state) on another cell type (or cell state) decently. This pre-training process is very valuable in some cases, such as when the number of cells in the dataset is too small to construct a high-quality model. Pre-training can also largely reduce the overall computation time, since DeepImpute spends most of the time on training the samples. Thus, it is also a good strategy when the new, large dataset is very similar to the dataset used in pre-training.

An enduring imputation method has to adapt to the ever-increasing volume of scRNA-seq data. DeepImpute is such a method, implemented in a deep learning framework where new solutions for speed improvements keep appearing. One example is the development of neural network-specific hardware (such as tensor processing units [41], or TPUs) which are now available on Google Cloud. TPU can dramatically accelerate the tensor operations and thus the imputation process. We were already able to deploy DeepImpute in a Google Cloud environment where TPUs are already available. Another example is the development of frameworks that efficiently use computer clusters to parallelize tasks such as Apache-Spark [42] or Dask [43]. Such resources will help DeepImpute and similar deep-learning methods, such as scDeepCluster designed for clustering analysis [44], achieve even higher speed over time and keep up with the development of scRNA-seq technologies.

2.4 Methods

2.4.1 The workflow of DeepImpute

DeepImpute is a deep neural network-based imputation workflow, implemented with the Keras [45] framework and TensorFlow [46] in the backend. Below, we describe the workflow in four steps: preprocessing, architecture, training procedure, and imputation.

Preprocessing

The first step of DeepImpute is selecting the genes for imputation, based on the variance over mean ratio (default=0.5), which are deemed interesting for downstream analyses [47, 48]. For efficiency, we adopt a divide-and-conquer strategy in our deep learning imputation process. We split the genes into N random subsets, each with S numbers of genes, which we call "target genes." By default, S is set as 512. If the number of target genes is not a multiple of this number, we round the number of genes to impute in N+1 subsets of deep neural networks. The details of this step are illustrated in Figure S2.1.

Network architecture

For each subset, we train a neural network of four layers: the input layer of genes that are correlated to the target genes, a 256-neuron fully connected hidden layer with a rectified linear unit (ReLU) activation function, a dropout layer (note: different from dropout data in scRNA-Seq), and an output layer of S target genes. A gene is selected to the input layer, if it satisfies these conditions: (1) it is not one of the target genes and (2) it has top 5 ranked Pearson's correlation coefficient with a target gene. The dropout layer is included after the hidden layer, as a common strategy to prevent overfitting [49]. We optimized the dropout rate as 20%, after experimenting the dropout rates from 0 to 90% (Fig.S2.3). The other default parameters of the networks include a learning rate of 0.0001, a batch size of 64, and a subset size of 512. As internal controls, we also experimented two alternative setups for

DeepImpute: one with the same architecture but with a linear activation function and the other one without hidden layers of neurons.

Training procedure

The training starts by splitting the cells between a training (95%) and a test set (5%). The test set is used at each epoch to measure overfitting. We use a weighted mean squared error (MSE) loss function that gives higher weights to genes with higher expression values. This emphasizes accuracy on high confidence values and avoids over penalizing genes with extremely low values (e.g., zeros). For a given cell c, the loss is calculated as follows:

$$Loss_c = \sum Y_i(Y_i - \hat{Y}_i)^2$$

where Y_i is the value of gene i for cell c and $\hat{Y}_i$ is the corresponding estimated value at a given epoch. For the gradient descent algorithm, we choose an adaptive learning rate method, the Adam optimizer [50], since it is known to perform very efficiently over sparse data [51]. The training stops if it reaches 500 epochs or if the training does not improve for 5 epochs.

Imputation

Once the network weights are properly trained, we impute the data by filling zeros in the original matrix with the imputed values.

2.4.2 Evaluation metrics

Accuracy comparison on real datasets

To evaluate the accuracy of imputation, we apply a random mask to the real single-cell datasets. The masking probability function is estimated in a similar fashion as in Splatter [14]. For each gene, we extract the proportion of zeros vs. the mean of those positive values. As done in Splatter, we fit a logistic function to these data points. Next, for each gene in the

dataset, we mask ten cells at random using a multinomial distribution: each cell $c_1, ..., c_n$ has a dropout probability $p_1, ..., p_n$ given by the logistic function previously fitted. The masked cells are sampled from a multinomial distribution with parameters $(q_1, q_2, ..., q_n)$, where $q_i = p_i / \sum_i p_i$ are the normalized probability such that $\sum_i q_i = 1$

These original values are used as "truth values" to evaluate the performance of imputation methods. We used two types of performance metrics: the overall Pearson correlation coefficient and MSE, both on log transformed counts. When needed, we also computed MSE between cells c_j and between genes g_i.

Speed and memory comparison

We run comparisons on a dedicated 8-core, 30-GB RAM, 100-GB HDD, Intel Skylake machine running Debian 9.4. We record process memory usage at 60-s intervals. For testing data, we use the Mouse1M dataset since it has the largest number of single cells (Table S2.1). We filter out genes that are expressed in less than 20% of cells, leaving 3,205 genes in our sample. From this dataset, we generate 7 subsets ranging in size (100, 500, 1k, 5k, 10k, 30k, 50k cells). We run each package 3 times per subset to estimate the average computation time. Some packages (VIPER, DrImpute, SAVER, scImpute, and MAGIC) are not able to successfully handle the larger files either due to out-of-memory errors (OOM) or exceedingly long run times ($>$ 24h).

2.4.3 Downstream functional analysis

Clustering

We perform cell clustering using the Seurat pipeline implemented in Scanpy. After preprocessing the data, we extract the UMAP components [38] and cluster the cells using the Leiden algorithm recommended in the Scanpy documentation. To assess the quality of the clusters, we use four metrics. For all of them, a value of 1 indicates a perfect clustering, while 0 corresponds to random assignments.

Adjusted mutual information[16]

It is an entropy-based metric that calculates the shared entropy between two clustering assignments, and is adjusted for chance. The mutual information is calculated by $MI(C,K) = \sum_{c\in C}\sum_{k\in K} P(c,k)\cdot\log\left(\frac{P(c,k)}{P(c)P(k)}\right)$, where $P(c,k)$ is the probability of a random cell belonging to both cluster c and k, and $P(c)$ (resp. $P(k)$) the probability of a random cell belonging to cluster c (resp. k).

Adjusted Rand index[16]

It is the ratio of all cell pairs that are either correctly assigned together or correctly not assigned together, among all possible pairs. It is also adjusted for chance.

Fowlkes–Mallows index

It is a metric derived from the true positives (TP), false positives (FP), and false negatives (FN) as follows: $FMI = \sqrt{\frac{TP}{TP+FP}\cdot\frac{TP}{TP+FN}}$

Silhouette coefficient[29]

It is a clustering metric derived by comparing the mean intra-cluster distance and the mean inter-cluster distance.

Differential expression analysis

We perform the differential expression analysis using the Scanpy package on the simulation as the groups are pre-defined. For each method, we extracted the differentially expressed genes for each cell group by performing a t-test of one group against the other groups. We used Benjamini-Hochberg correction for multiple hypothesis testing to obtain adjusted p value (pval_{adj}). Since each method has generated different differentially expressed genes, we extracted the top 500 differentially expressed genes for each group and pooled the differentially expressed genes for all of the groups. Using $1-\text{pval}_{adj}$ as the differential expression calling probability and the true differentially expressed genes (by Splatter) as

the truth measure, we calculated the area under the curve (AUC) for the ROC curve for each method using the scikit-learn python package.

2.4.4 RNA FISH validation

We obtain a Drop-Seq dataset (GSE99330) and its RNA FISH dataset from a melanoma cell line, as described by Torre et al. [36]. The summary of the dataset is listed in table S2.1. For the comparison between RNA FISH and the corresponding Drop-Seq experiment, we keep genes with a variance over mean ratio>0.5, the same as other datasets in this study, leaving six genes in common between the FISH and the Drop-Seq datasets.

For GINI coefficient calculation, we first normalize the cells in each dataset using a housekeeping gene (glyceraldehyde 3-phosphate dehydrogenase, or GAPDH)-based factor, as done by others [20]. We remove GAPDH outlier cells (defined here as the cells below the 10th and above the 90th percentiles). Then, we rescale each data point by a GAPDH-based factor, as follows:

$\text{data}[\text{cell}, \text{gene}] = \text{data}[\text{cell}, \text{gene}] \times \text{factor}(\text{cell})$

where $\text{factor}(\text{cell}) = \text{mean}(\text{data}[:,\text{GAPDH}])/\text{data}[\text{cell},\text{GAPDH}]$

Then, we compute GINI coefficient, as done in SAVER [20]. For distribution normalization, the procedure is the same except that we first normalize each gene by an efficiency factor (defined as the ratio between its mean value for FISH and its value for the imputation method). We calculate the MSEs and Pearson's coefficients with the following formulas:

$$\text{MSE(gene, method)} = \sum_{\text{cell}} (X_{\text{FISH}}(\text{gene}, \text{cell}) - X_{\text{method}}(\text{gene}, \text{cell}))^2$$

$$\text{corr(gene, method)} = \frac{\text{Cov}\left[X_{\text{FISH}}(\text{gene}), X_{\text{method}}(\text{gene})\right]}{\text{Var}[X_{\text{FISH}}(\text{gene})] \cdot \text{Var}[X_{\text{method}}(\text{gene})]}$$

where X is the input matrix of gene expression from RNA-FISH or Drop-Seq, Cov is the covariance, and Var is the variance.

2.5 Availability of data and materials

2.5.1 scRNA-seq Datasets

In this chapter, we evaluate imputation metrics on nine datasets. Four of them (Jurkat, 293T, neuron9k, and Mouse1M) are downloaded from the 10X Genomics support website (`https://support.10xgenomics.com/single-cell-gene-expression/datasets`). Briefly, the Jurkat dataset is extracted from the Jurkat cell line (human blood). 293T is a blood cell line derived from HEK293T that expresses a mutant version of the SV40 large T antigen. The neuron9k dataset contains brain cells from an E18 mouse. Mouse1M also contains brain cells from an E18 mouse. The FISH and GSE99330 data were both extracted from the same melanoma cell line WM989-A6 [36]. Two other datasets are taken from GSE67602 [52], composed of mouse interfollicular epidermis cells and the Hrvatin dataset GSE102827 [37] dataset, extracted from primary visual cortex of C57BL6/J mice. Additionally, we simulate a dataset with the Splatter package [14] with parameters dropout.shape=-0.5, dropout.mid=1, 4000 genes and 2000 cells split into 5 groups with proportions 10%, 10%, 20%, 20%, and 40%. Each gene in each group is automatically assigned a differential expression (DE) factor, where 1 is not differentially expressed, a value less than 1 is downregulated, and more than 1 is upregulated.

2.5.2 Third party software

For comparison, we use the latest version of SAVER (v1.1.1) at `https://github.com/mohuangx/SAVER`, ScImpute (v0.0.9) at https://github.com/Vivianstats/scImpute, DrImpute (v1.0) available as a CRAN package, MAGIC (v1.4.0) at `https://github.com/KrishnaswamyLab/magic`, VIPER (v1.0) at `https://github.com/ChenMengjie/VIPER/releases`, and DCA (0.2.2) at `https://github.com/theislab/dca`. We preprocess the datasets according to each method's standard: using a square root transformation for MAGIC, log transformation for DeepImpute (with a pseudo count of 1), but raw counts for scImpute, DrImpute, SAVER, and DCA. For VIPER, we remove all genes with a null

total count and rescale each cell to a library size of one million (RPM normalization) as recommended.

2.5.3 DeepImpute's material

The DeepImpute package and its documentation are freely available on GitHub `https://github.com/lanagarmire/DeepImpute` under the MIT license. The software as well as the source code to reproduce the figures of this chapter was deposited on Zenodo `https://doi.org/10.5281/zenodo.3459902` [53].

References

[1] D. Usoskin, A. Furlan, S. Islam, H. Abdo, P. Lönnerberg, D. Lou, J. Hjerling-Leffler, J. Haeggström, O. Kharchenko, P. V. Kharchenko, and Others, "Unbiased classification of sensory neuron types by large-scale single-cell RNA sequencing," *Nat. Neurosci.*, vol. 18, no. 1, p. 145, 2015.

[2] A.-C. Villani, R. Satija, G. Reynolds, S. Sarkizova, K. Shekhar, J. Fletcher, M. Griesbeck, A. Butler, S. Zheng, S. Lazo, and Others, "Single-cell RNA-seq reveals new types of human blood dendritic cells, monocytes, and progenitors," *Science*, vol. 356, no. 6335, p. eaah4573, 2017.

[3] A. Zeisel, A. B. Muñoz-Manchado, S. Codeluppi, P. Lönnerberg, G. La Manno, A. Juréus, S. Marques, H. Munguba, L. He, C. Betsholtz, and Others, "Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq," *Science*, vol. 347, no. 6226, pp. 1138–1142, 2015.

[4] D. A. Jaitin, E. Kenigsberg, H. Keren-Shaul, N. Elefant, F. Paul, I. Zaretsky, A. Mildner, N. Cohen, S. Jung, A. Tanay, and Others, "Massively parallel single-cell RNA-seq for marker-free decomposition of tissues into cell types," *Science*, vol. 343, no. 6172, pp. 776–779, 2014.

[5] A. Kriegstein, A. A. Pollen, T. J. Nowakowski, J. Shuga, X. Wang, A. A. Leyrat, J. H. Lui, N. Li, L. Szpankowski, B. Fowler, and Others, "Low-coverage single-cell

mRNA sequencing reveals cellular heterogeneity and activated signaling pathways in developing cerebral cortex," 2014.

[6] B. Treutlein, D. G. Brownfield, A. R. Wu, N. F. Neff, G. L. Mantalas, F. H. Espinoza, T. J. Desai, M. A. Krasnow, and S. R. Quake, "Reconstructing lineage hierarchies of the distal lung epithelium using single-cell RNA-seq," *Nature*, vol. 509, no. 7500, p. 371, 2014.

[7] I. Tirosh, A. S. Venteicher, C. Hebert, L. E. Escalante, A. P. Patel, K. Yizhak, J. M. Fisher, C. Rodman, C. Mount, M. G. Filbin, and Others, "Single-cell RNA-seq supports a developmental hierarchy in human oligodendroglioma," *Nature*, vol. 539, no. 7628, p. 309, 2016.

[8] A. K. Shalek, R. Satija, X. Adiconis, R. S. Gertner, J. T. Gaublomme, R. Raychowdhury, S. Schwartz, N. Yosef, C. Malboeuf, D. Lu, and Others, "Single-cell transcriptomics reveals bimodality in expression and splicing in immune cells," *Nature*, vol. 498, no. 7453, p. 236, 2013.

[9] F. Tang, C. Barbacioru, S. Bao, C. Lee, E. Nordman, X. Wang, K. Lao, and M. A. Surani, "Tracing the derivation of embryonic stem cells from the inner cell mass by single-cell RNA-Seq analysis," *Cell Stem Cell*, vol. 6, no. 5, pp. 468–478, 2010.

[10] J. K. Kim, A. A. Kolodziejczyk, T. Ilicic, S. A. Teichmann, and J. C. Marioni, "Characterizing noise structure in single-cell RNA-seq distinguishes genuine from technical stochastic allelic expression," *Nat. Commun.*, vol. 6, p. 8687, 2015.

[11] A. A. Kolodziejczyk, J. K. Kim, V. Svensson, J. C. Marioni, and S. A. Teichmann, "The technology and biology of single-cell RNA sequencing," *Mol. Cell*, vol. 58, no. 4, pp. 610–620, 2015.

[12] C. Jia, D. Kelly, J. Kim, M. Li, and N. Zhang, "Accounting for technical noise in single-cell RNA sequencing analysis," *bioRxiv*, p. 116939, 2017.

[13] T. S. Andrews and M. Hemberg, "Modelling dropouts allows for unbiased identification of marker genes in scRNASeq experiments." July 2016.

[14] L. Zappia, B. Phipson, and A. Oshlack, "Splatter: simulation of single-cell RNA sequencing data," *Genome Biol.*, vol. 18, no. 1, p. 174, 2017.

[15] X. Zhu, T. Ching, X. Pan, S. M. Weissman, and L. Garmire, "Detecting heterogeneity in single-cell RNA-Seq data by non-negative matrix factorization," *PeerJ*, vol. 5, p. e2888, Jan. 2017.

[16] O. Poirion, X. Zhu, T. Ching, and L. X. Garmire, "Using single nucleotide variations in single-cell RNA-seq to identify subpopulations and genotype-phenotype linkage," *Nat. Commun.*, vol. 9, p. 4892, Nov. 2018.

[17] X. Zhu, T. K. Wolfgruber, A. Tasato, C. Arisdakessian, D. G. Garmire, and L. X. Garmire, "Granatum: a graphical single-cell RNA-Seq analysis pipeline for genomics scientists," *Genome Med.*, vol. 9, no. 1, p. 108, 2017.

[18] D. van Dijk, R. Sharma, J. Nainys, K. Yim, P. Kathail, A. J. Carr, C. Burdziak, K. R. Moon, C. L. Chaffer, D. Pattabiraman, B. Bierie, L. Mazutis, G. Wolf, S. Krishnaswamy, and D. Pe'er, "Recovering gene interactions from Single-Cell data using data diffusion," *Cell*, vol. 174, pp. 716–729.e27, July 2018.

[19] W. V. Li and J. J. Li, "An accurate and robust imputation method scimpute for single-cell RNA-seq data," *Nat. Commun.*, vol. 9, p. 997, Mar. 2018.

[20] M. Huang, J. Wang, E. Torre, H. Dueck, S. Shaffer, R. Bonasio, J. I. Murray, A. Raj, M. Li, and N. R. Zhang, "SAVER: gene expression recovery for single-cell RNA sequencing," *Nat. Methods*, vol. 15, pp. 539–542, July 2018.

[21] W. Gong, I.-Y. Kwak, P. Pota, N. Koyano-Nakagawa, and D. J. Garry, "DrImpute: imputing dropout events in single cell RNA sequencing data," *BMC Bioinformatics*, vol. 19, p. 220, June 2018.

[22] M. Chen and X. Zhou, "VIPER: variability-preserving imputation for accurate gene expression recovery in single-cell RNA sequencing studies," 2018.

[23] G. Eraslan, L. M. Simon, M. Mircea, N. S. Mueller, and F. J. Theis, "Single-cell RNA-seq denoising using a deep count autoencoder," *Nat. Commun.*, vol. 10, p. 390, Jan. 2019.

[24] P. Lin, M. Troup, and J. W. K. Ho, "CIDR: Ultrafast and accurate clustering through imputation for single-cell RNA-seq data," *Genome Biol.*, vol. 18, p. 59, Mar. 2017.

[25] J. Ronen and A. Akalin, "netsmooth: Network-smoothing based imputation for single cell RNA-seq," *F1000Res.*, vol. 7, p. 8, Jan. 2018.

[26] L. Zhang and S. Zhang, "Comparison of computational methods for imputing single-cell RNA-sequencing data," *bioRxiv*, p. 241190, 2017.

[27] T. Ching, X. Zhu, and L. X. Garmire, "Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data," *PLoS Comput. Biol.*, vol. 14, p. e1006076, Apr. 2018.

[28] F. M. Alakwaa, K. Chaudhary, and L. X. Garmire, "Deep learning accurately predicts estrogen receptor status in breast cancer metabolomics data," *J. Proteome Res.*, vol. 17, pp. 337–347, Jan. 2018.

[29] K. Chaudhary, O. B. Poirion, L. Lu, and L. X. Garmire, "Deep Learning-Based Multi-Omics integration robustly predicts survival in liver cancer," *Clin. Cancer Res.*, vol. 24, pp. 1248–1259, Mar. 2018.

[30] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, C. A. Lavender, S. C. Turaga, A. M. Alexandari, Z. Lu, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, S. M.

Boca, S. J. Swamidass, A. Huang, A. Gitter, and C. S. Greene, "Opportunities and obstacles for deep learning in biology and medicine," *J. R. Soc. Interface*, vol. 15, Apr. 2018.

[31] J. Tan, G. Doing, K. A. Lewis, C. E. Price, K. M. Chen, K. C. Cady, B. Perchuk, M. T. Laub, D. A. Hogan, and C. S. Greene, "Unsupervised extraction of stable expression signatures from public compendia with an ensemble of neural networks," *Cell Syst*, vol. 5, pp. 63–71.e6, July 2017.

[32] B. K. Beaulieu-Jones, C. S. Greene, and Pooled Resource Open-Access ALS Clinical Trials Consortium, "Semi-supervised learning of the electronic health record for phenotype stratification," *J. Biomed. Inform.*, vol. 64, pp. 168–178, Dec. 2016.

[33] B. K. Beaulieu-Jones and J. H. Moore, "MISSING DATA IMPUTATION IN THE ELECTRONIC HEALTH RECORD USING DEEPLY LEARNED AUTOENCODERS," *Pac. Symp. Biocomput.*, vol. 22, pp. 207–218, 2017.

[34] C.-C. Chiang and H.-C. Fu, "A divide-and-conquer methodology for modular supervised neural network design," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 1, pp. 119–124 vol.1, June 1994.

[35] L. X. Garmire and S. Subramaniam, "Evaluation of normalization methods in mammalian microRNA-Seq data," *RNA*, vol. 18, pp. 1279–1288, June 2012.

[36] E. Torre, H. Dueck, S. Shaffer, J. Gospocic, R. Gupte, R. Bonasio, J. Kim, J. Murray, and A. Raj, "Rare cell detection by single-cell RNA sequencing as guided by single-molecule RNA FISH," *Cell systems*, vol. 6, no. 2, pp. 171–179, 2018.

[37] S. Hrvatin, D. R. Hochbaum, M. A. Nagy, M. Cicconet, K. Robertson, L. Cheadle, R. Zilionis, A. Ratner, R. Borges-Monroy, A. M. Klein, B. L. Sabatini, and M. E. Greenberg, "Single-cell analysis of experience-dependent transcriptomic states in the mouse visual cortex," *Nat. Neurosci.*, vol. 21, pp. 120–129, Jan. 2018.

[38] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," Feb. 2018.

[39] V. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: guaranteeing well-connected communities," Oct. 2018.

[40] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, vol. 2008, p. P10008, Oct. 2008.

[41] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and Others, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

[42] J. Shanahan and L. Dai, "Large scale distributed data science from scratch using apache spark 2.0," in *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, (Republic and Canton of Geneva, Switzerland), pp. 955–957, International World Wide Web Conferences Steering Committee, 2017.

[43] P. Mehta, S. Dorkenwald, D. Zhao, T. Kaftan, A. Cheung, M. Balazinska, A. Rokem, A. Connolly, J. Vanderplas, and Y. AlSayyad, "Comparative evaluation of big-data systems on scientific image analytics workloads," *Proceedings VLDB Endowment*, vol. 10, pp. 1226–1237, Aug. 2017.

[44] Tian, T. Tian, J. Wan, Q. Song, and Z. Wei, "Clustering single-cell RNA-seq data with a model-based deep learning approach," 2019.

[45] F. Chollet, "Keras," 2015.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and Others, "TensorFlow: A system for Large-Scale machine learning," in *OSDI*, vol. 16, pp. 265–283, 2016.

[47] R. Satija, J. A. Farrell, D. Gennert, A. F. Schier, and A. Regev, "Spatial reconstruction of single-cell gene expression data," *Nat. Biotechnol.*, vol. 33, pp. 495–502, May 2015.

[48] F. A. Wolf, P. Angerer, and F. J. Theis, "SCANPY: large-scale single-cell gene expression data analysis," *Genome Biol.*, vol. 19, p. 15, Feb. 2018.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412. 6980*, 2014.

[51] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609. 04747*, 2016.

[52] S. Joost, A. Zeisel, T. Jacob, X. Sun, G. La Manno, P. Lönnerberg, S. Linnarsson, and M. Kasper, "Single-Cell transcriptomics reveals that differentiation and spatial signatures shape epidermal and hair follicle heterogeneity," *Cell Syst*, vol. 3, pp. 221–237.e9, Sept. 2016.

[53] Arisdakessian, Poirion, Yunits, Zhu, and Garmire, "Deepimpute," Sept. 2019.

Supplementary information

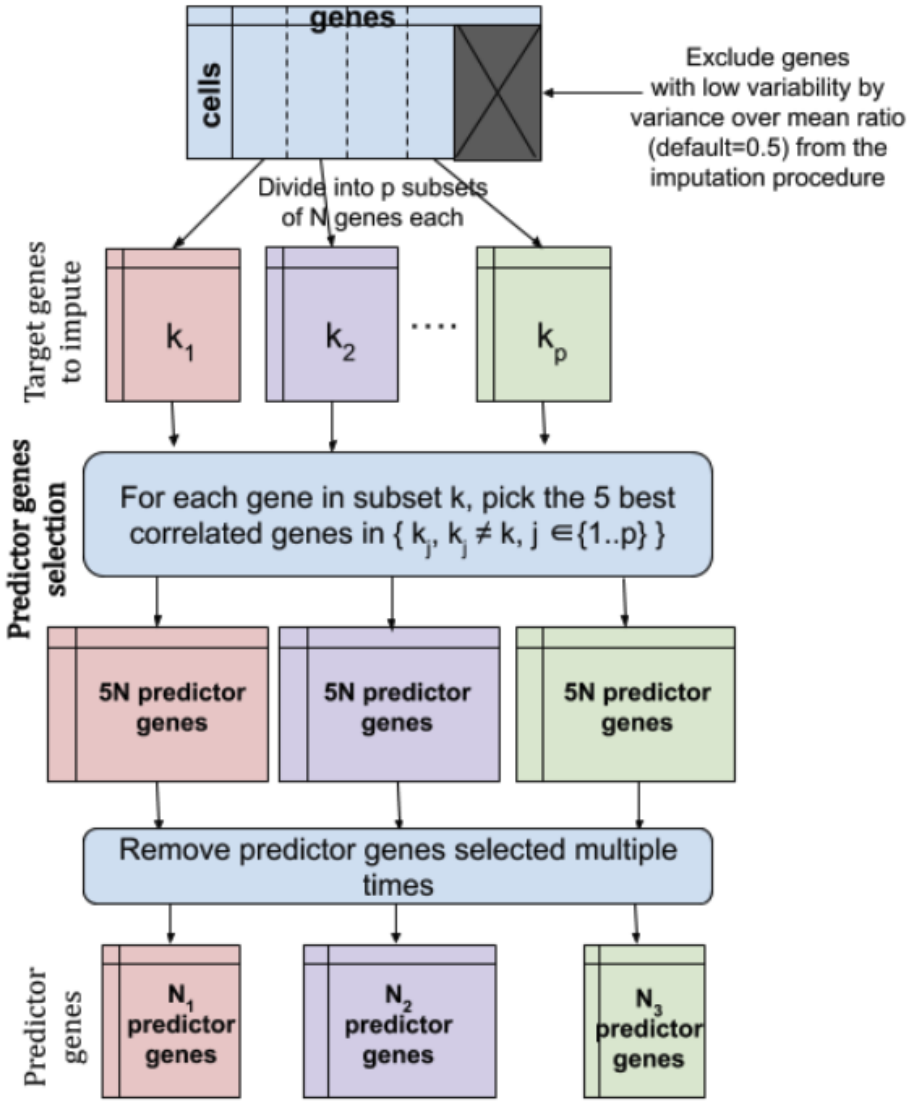


Figure S2.1: Preprocessing steps for DeepImpute: DeepImpute starts by selecting "target genes" that meet a variability criteria based on variance over mean ratio (default: 0.5), and then splits them into subsets with the same number of genes (default: N = 512). For the last subset of genes less than the default value, they are rounded into the next sub-neural network model. These target genes make up the output layer of the sub-neural network, whose zero values are imputed. For each each target gene g_i in subset k, we select the 5 best correlated genes, or predictor genes, which are not part of the target genes in the subset k. Finally, we remove those predictor genes that were selected multiple times in all the sub-neural networks. We use the remaining predictor genes as the input layer for the sub-neural network.

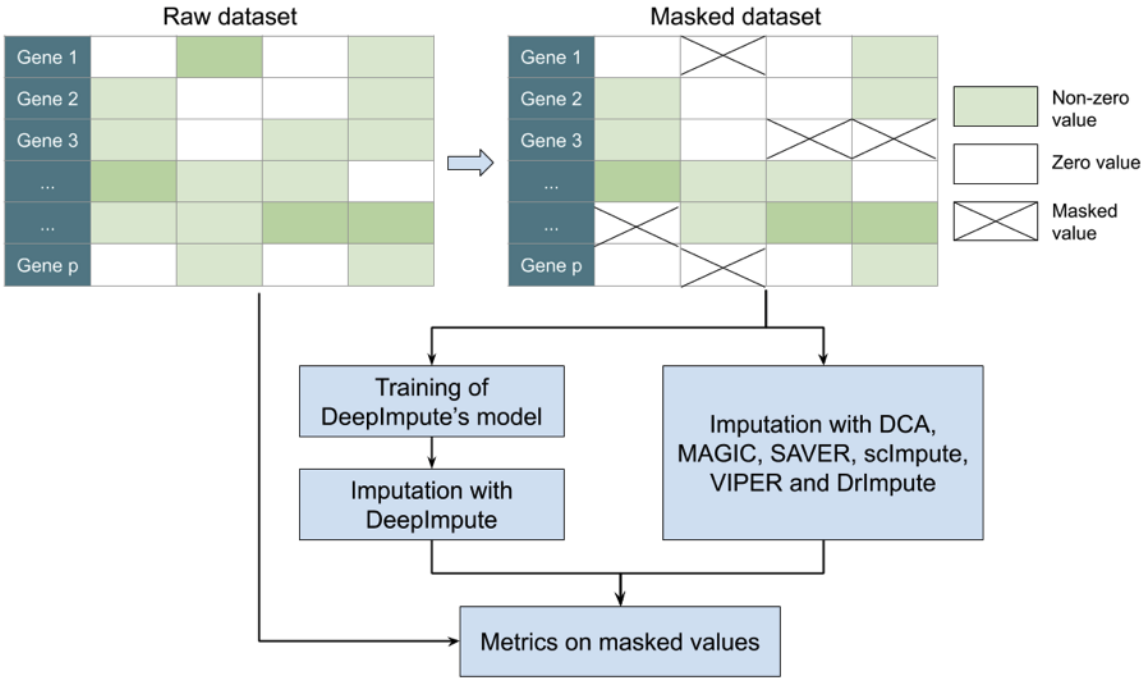


Figure S2.2: Masking experiment in single cell RNA-Seq data: (Left) Raw dataset. Darker colors represent higher values. (Right) Masked dataset. Masked values are barred with an "X".

Dataset	**Cells**	**Sample type**	**Organism**	**Source**
Jurkat	3,258	Blood cell line	Homo Sapiens	10X Genomics*
293T	2,885	Blood cell line	Homo Sapiens	10X Genomics*
Neuron9k	9,128	Brain cells	Mus Musculus	10X Genomics*
GSE67602	1,422	Interfollicular epidermis cells	Mus Musculus	GSE67602
Mouse1M	1,306,127	Brain cells	Mus Musculus	10X Genomics*
FISH	88,040	Melanoma cell line	Homo Sapiens	Torre et al. [36]
GSE99330	8,641	Melanoma cell line	Homo Sapiens	GSE99330
Sim	2,000	NA	NA	NA
Hrvatin	48,267	Primary Visual Cortex	Mus Musculus	GSE102827

Table S2.1: Single-cell datasets summary: *: the URL to access the dataset is: https://support.10xgenomics.com/single-cell-gene-expression/datasets

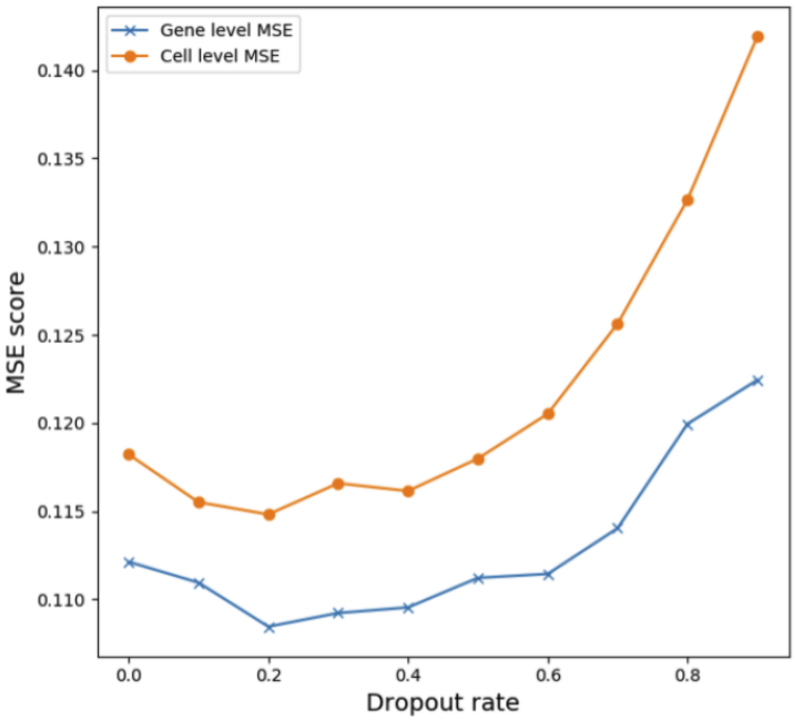


Figure S2.3: Effect of dropout rate on imputation accuracy: The MSE scores for dropout rates varying from 0 to 90% are shown. The blue and orange lines are gene-level and cell-level MSEs, respectively.

Figure S2.4: Accuracy comparison between DeepImpute and two other variant architectures: Bar plots of Pearson's correlation coefficients (left) and mean squared error (right) are shown for the masked data points of each dataset in Figure 2. Three DeepImpute variants are compared: default DeepImpute with ReLU activation function (blue), DeepImpute with linear activation function (orange), and DeepImpute without hidden layers (green).

Chapter 3

CoCoNet: an efficient deep learning tool for viral metagenome binning

Abstract

Metagenomic approaches hold the potential to characterize microbial communities and unravel the intricate link between the microbiome and biological processes. Assembly is one of the most critical steps in metagenomics experiments. It consists of transforming overlapping DNA sequencing reads into sufficiently accurate representations of the community's genomes. This process is computationally difficult and commonly results in genomes fragmented across many contigs. Computational binning methods are used to mitigate fragmentation by partitioning contigs based on their sequence composition, abundance, or chromosome organization into bins representing the community's genomes. Existing binning methods have been principally tuned for bacterial genomes and do not perform favorably on viral metagenomes.

In this chapter, we propose CoCoNet (Composition and Coverage Network), a new binning

Published: Cédric Arisdakessian, Olivia D. Nigro, Grieg F. Steward, Guylaine Poisson, and Mahdi Belcaid. "CoCoNet: an efficient deep learning tool for viral metagenome binning." Bioinformatics 37, no. 18 (2021): 2803-2810.

method for viral metagenomes that leverages the flexibility and the effectiveness of deep learning to model the co-occurrence of contigs belonging to the same viral genome and provide a rigorous framework for binning viral contigs. Our results show that CoCoNet substantially outperforms existing binning methods on viral datasets.

3.1 Introduction

Shotgun metagenomics plays a critical role in investigating the composition and function of microbial communities in diverse environments, from the human gut [1, 2] to the deep sea [3, 4]. Metagenomic assembly is particularly challenging for virome data [5]. Viral shotgun assemblies are commonly plagued by short contigs, leading to poor characterization of the underlying species diversity, richness [6] and functional capacity [7]. Binning contigs arising from a single species is a process routinely used to compensate for incomplete microbial metagenome assemblies. In the presence of reference genome sequences, binning is trivial and can be achieved by clustering contigs that align with high-confidence against the same reference genome. However, in samples containing species with unsequenced genomes, binning is more complicated since contigs need to be clustered *de novo*. While binning microbial contigs has seen significant advances recently, few programs have considered the unique set of challenges associated with viruses.

Existing methods for binning contigs can be divided into three classes: 1) methods based on sequence composition [8], 2) methods based on sequencing coverage correlation [9], and 3) a combination of both [10, 11, 12]. The first class of methods leverages the fact that bacterial species have predominantly different distributions of k-mers, or words of size k, and use that evidence to bin together contigs with similar k-mer distributions. This approach works best on species with sufficiently divergent k-mer distributions but is less effective for resolving closely related species for which the k-mer distributions may be indistinguishable, particularly over short contigs. The second class of methods uses the sequencing coverage, i.e., the number of reads aligning to each contig, and reports contigs that consistently share

similar coverage values across multiple samples as belonging to the same genome. This approach is accurate but works best when many samples are available [12]. The third class of methods leverages both k-mer and coverage profiles, typically using statistical models, to infer clusters. This class of methods combines the advantages of composition- and coverage-based solutions but is computationally more complex.

In this chapter, we introduce CoCoNet, a new method that leverages deep learning to model the k-mer composition and the coverage for binning contigs assembled from viral metagenomic data. Specifically, our method uses a neural network trained using contigs' subsequences, or fragments, to learn a flexible function for predicting the probability that any pair of contigs originated from the same genome. These probabilities are subsequently combined to infer bins representing the genomes of species present in the sequenced samples. Our approach was specifically optimized for viral metagenomes with large diversity, such as those found in environmental samples (e.g., oceans, soil, etc.). Such samples require sophisticated modeling methods that account for several sources of bias [13, 5]. We tested CoCoNet on both simulated and experimental viral metagenome data, and our results show that CoCoNet outperforms existing tools optimized for binning bacterial data. The CoCoNet source code and documentation are available at: `https://github.com/Puumanamana/CoCoNet`.

3.2 Methods

The CoCoNet algorithm

We developed CoCoNet, a *Co*mposition and *Co*verage *Ne*twork that uses deep learning in conjunction with clustering to bin assembly contigs into homogeneous clusters representing the species present in the samples. Our approach works in two phases. The first phase trains a deep neural network to estimate the probability that two fragments belong to the same genome, given their composition and coverage information. The second phase uses a

computationally tractable heuristic to bin the contigs using the co-occurrence probabilities inferred in the previous phase.

Preprocessing

To account for differences in contig lengths when extracting composition and coverage features, we divide contigs longer than 2048 bp into regularly spaced fragments of length 1024 bp (step: 128 bp).

Composition feature: We compute the composition feature by summing the number of occurrences of each k-mer, where $k = 4$, on each fragment's forward and reverse strands. This makes the distribution invariant to the reverse complement, therefore accounting for missing strand orientation information. Because this transformation induces k-mer redundancies (e.g. observing ACCG is the same as observing its reverse complement, CGGT), we are left with 136 unique k-mers.

Coverage feature: The coverage feature is computed by aligning the raw reads against the fragments, and by counting the number of reads aligning at each position of the fragment. To reduce artifacts due to erroneous read mapping and mis-assembly, we filtered the alignments to discard those that were either partial (represented 50% or less of the query's length), had a quality score lower than 30, or where one of the read pairs was unmapped (SAMflag=3596). We also removed reads that had more than one high-quality alignment or PCR/optical duplicates. We excluded contigs occurring in only one sample (prevalence < 2) since our model does not extract relevant co-occurrence information from one sample. Finally, we smoothed the coverage using an averaging window of size 64 and sub-sampled the resulting vector every 32 bases to minimize computation and data storage requirements. These steps yielded 31 coverage values for each fragment in each sample.

Filtering complete genomes: Complete genomes were used in the training but were not included in the binning. We consider a contig to be complete if it contained Direct Terminal Repeats [14]. Specifically, we flagged a contig as complete if its first and last 300 bp aligned over at least 10 bp with at least 95% identity.

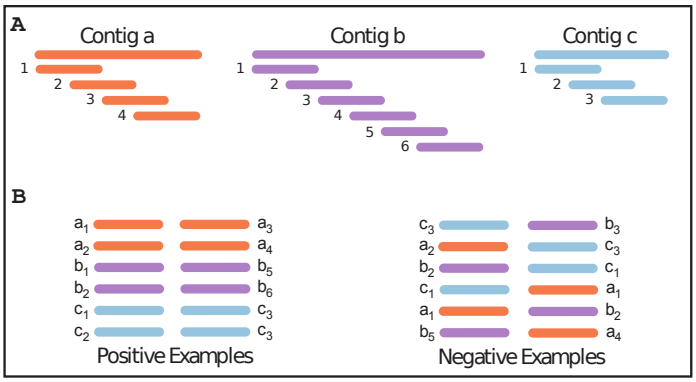
Deep learning model

This section describes how we train a neural network *de novo* to learn to predict the probability that any pair of fragments belong to the same genome.

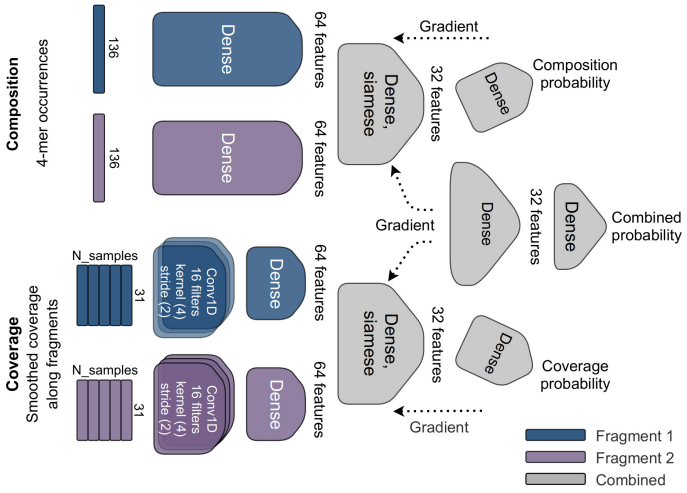
Training and test sets' construction: We divide the initial set of contigs into 90% training and 10% testing. Both sets contain positive and negative examples (see Figure 3.1a). The positive examples are pairs of fragments from the same contig. They teach the model the similarities in fragments from the same genome. The negative examples are pairs of fragments from distinct contigs. They teach the model the differences between fragments from different genomes. When selecting positive training instances, we try to maximize the distance between the fragments to avoid trivial examples where fragments are similar due to their overlap. We generate the same number of positive examples from each contig in the training data to avoid biasing the training set in favor of long contigs. Using the approach described above, some mislabeling can occur with negative examples since different contigs can belong to the same bin. However, we show that the proportion of mislabeled pairs should be negligible for diverse metagenomes (see Supplementary Methods). Other studies have also shown that neural networks are robust to label noise and mislabeling, which can be mitigated with large batch sizes [15].

Network architecture and training procedure: The network processes the composition and coverage inputs in three main steps (See Figure 3.1b).

In the first step, we process both composition vectors separately using two 64-neurons dense layers that share the same weights. Similarly, we process both coverage vectors using two pairs of layers (1D convolution layer with 16 filters, a `kernel_size`=4 and a `stride`=2, and a 64-neurons dense layer) that share the same weights. Since the network outputs a probability of the two fragments belonging to the same genomes, the output should be symmetrical, i.e. $P(\texttt{frag}_1, \texttt{frag}_2) = P(\texttt{frag}_2, \texttt{frag}_1)$. The merging layer enforces this symmetricity using Siamese networks [16]. In short, given two vectors x_1 and x_2 and a dense layer D, we compute both $D(x_1, x_2)$ and $D(x_2, x_1)$, and return the element-wise maximum between the two outputs. In the second step, the information from the four original inputs



(a) Training set construction



(b) Network architecture

Figure 3.1: CoCoNet's neural network architecture and learning: (a) Training set construction: Each contig is split into 1024 bp fragments spaced with a 128 bp step. The training set is composed of two classes, positive (resp. negative) composed of fragment pairs from the same (resp. different) contigs. The positive class is constructed by extracting a constant number of fragment pairs from each contig. For a given contig, we pick the most distant pairs. The negative class is composed of random fragments from random contig pairs. (b) Deep learning model: The neural network computes the fragments co-binning probability. First, a latent representation is computed for each composition and coverage features separately by sharing the weights of the network between the two inputs. A 64-neurons dense layer is used to process the composition vectors and a 1D convolution layer followed by a shared 64-neurons dense layer to process the coverage vectors. For a pair of fragments, the composition and the coverage features are merged with a siamese layer composed of a 32-neurons dense layer, followed by an element-wise maximum. Two 1-neuron layers are used to compute the probabilities of the observed composition and coverage values. The latent representations of the composition and coverage are also used to compute the probability of the combined coverage and composition. This is done by concatenating the latent feature representations (32-neuron layer) and running them through a dense layer with 32-neurons, followed by a layer with 1-neuron and a sigmoid activation.

is aggregated into two vectors of size 32, representing the composition and coverage (Figure 3.1b). Finally, the third step computes a composition probability, coverage probability, and a combined probability that two fragments originated in the same genome.

We train the network using a batch size of 256 and a single epoch. We use the Adam [17] optimizer with a learning rate of 10^{-3} and compute the overall loss as the weighted sum of the three binary cross-entropy (BCE) losses:

$$\text{Loss} = \text{BCE}(\text{composition}) + \text{BCE}(\text{coverage}) + 2 \cdot \text{BCE}(\text{combined})$$

The network accuracy is evaluated on the test data every 400 batches. The training stops if the test loss does not improve for after five consecutive evaluations of the test set.

Clustering

We bin the contigs using a clustering graph $G(v, e)$, where the nodes v represent the contigs and the edges, e link contigs belonging to the same bin. The edges computation, the comparison policy, and the graph clustering steps are described in what follows.

1. Edges computation: During the clustering stage, we split each contig into 30 regularly spaced and maximally distant fragments of length 1024. To compare two contigs v_i and v_j, we compute the 900 probabilities that any fragment from v_i overlaps with fragments in v_j. We sum these probabilities to yield an expected number of hits between the contigs, which lies between 0 and 900:

$$\#\text{expected\_hits}(v_i, v_j) = \sum_{l=1}^{m} \sum_{k=1}^{m} P(v_i^l, v_j^k) \tag{3.1}$$

where v_i^l represents the l^{th} fragment in contig v_i and m is the number of fragments per contig (default is 30).

We subsequently assign an edge in G between contigs v_i and v_j if the number of expected hits is higher than a given threshold θ (by default, 80% of the maximum possible value of expected_hits, m^2).

2. Comparison policy: Given the large number of possible contig pairs in a dataset (about 0.5×10^9 comparisons for 30k contigs), we use a heuristic to evaluate only pairs that are more likely to belong to the same genome. In this heuristic, we hypothesize that for two contigs v_i and v_j, $\#\text{expected\_hits}(v_i, v_j) > \theta$ when v_i and v_j's fragments are close in both the composition and coverage spaces (see supplementary figure S3.4). Given the high-dimensionality of the original features, we choose to investigate the spatial proximity of fragments in their latent representation space. We compute the latent representation of each contig in the composition and coverage spaces using the trained neural network and use that information to subsequently represent each contig in latent space as a ball centered at its fragments' center and with a radius R defined as:

$$R = \text{percentile}(\{||f - c_f||_2, f \in \text{fragments}\}, 90),$$

where $||f - c_f||_2$ is the euclidean distance of a fragment f to its center of mass c_f.

We initially limit the comparison of a query contig to its 250 closest neighbors, which we define as the contigs whose center of mass falls within the query's ball (See Figure 3.2). Further, our approach emphasizes new comparisons rather than ones previously examined. Specifically, wherever contig v_j is selected for comparison to v_i, then contig v_i will not be selected for comparison to v_j. Since the radius is the same for all contigs, shorter contigs are not penalized due to the fact that their fragments are less variable. The number of neighbors was set to 250 after hyperparameter optimization. Our tests indicated no improvement in the results when considering more neighbors (see supplementary table S3.3).

3. Graph clustering: We use the Leiden algorithm [18] to identify the bins in the graph representing the species. This clustering algorithm infers its clusters by optimizing the Constant Potts Model (CPM) quality function H defined as:

$$H = \sum_{c \in \text{clusters}} \text{edges}(c) - \gamma \cdot \binom{\text{vertices}(c)}{2},$$

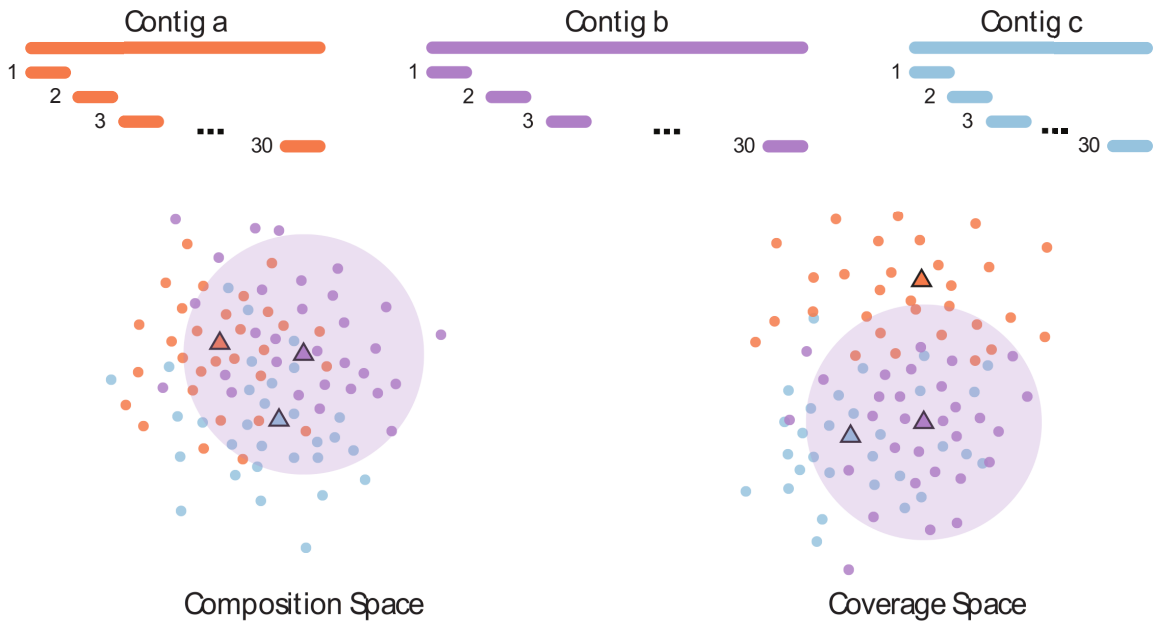


Figure 3.2: CoCoNet clustering approach: We split each contig in 30 regularly spaced and maximally distant fragments and used the deep neural network to compute the latent representation for each fragment's composition and coverage features. Considering each feature type separately, we draw for each contig a ball centered at the center of mass of its fragments' latent representations, and with a radius equal to the 90th percentile of all distances between each fragments and their respective center of mass. We compare a contig to the 250 closest contigs whose center lies within the balls derived from both feature types. The bins are finally determined using the Leiden clustering algorithm.

where edges(·) (resp. vertices(·)) counts the number of edges (resp. vertices) in a given cluster. The parameter γ controls how dense clusters are in terms of edges. Initially, each edge belongs to a different cluster. Then, the Leiden algorithm repeatedly 1) moves nodes between clusters to improve H, 2) refines the clusters into sub-communities, and 3) aggregates all the edges of a cluster into a single aggregated network. The algorithm stops when H cannot be improved further.

Since the clustering graph has potentially missing edges resulting from our heuristic, we perform the clustering in two steps. First, we run the Leiden algorithm with a small resolution parameter ($\gamma = 0.3$) to delineate unpolished clusters. Then, within each cluster, we fill the adjacency matrix with the remaining comparisons and re-run the Leiden clustering algorithm within each cluster with a higher resolution parameter ($\gamma = 0.4$). The CoCoNet implementation includes a second community detection algorithm, spectral clustering [19], which is ideal when the number of bins is known.

Metrics

Classification metrics

We measured the network's ability to classify contig pairs using the accuracy, the Area Under the Receiver Operating Characteristics (ROC) Curve (AUC), and the F1 score. These measures are defined as follows:

- The accuracy is the proportion of good predictions (either true positives or true negatives) among all predictions.

- AUC: Area under the ROC curve defined as (TPR(thresh), FPR(thresh)) at various classification thresholds.

- The F1 score is a summary value that takes into account both precision and recall. It is defined as:

$$F_1 = 2 \cdot \frac{precision \times recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

Clustering metrics

To compare the binning predicted by CoCoNet, CONCOCT [11], and Metabat2[10] against the ground truth, we used the Adjusted Rand Index (ARI), the homogeneity, and the completeness. These measures are defined as follows:

- The ARI[20] is the ratio of all pairs that are assigned correctly as belonging either together or not together, among all possible pairs. It is adjusted for chance.

- The homogeneity measures whether clusters contain only members of a single class. It is derived from the ratio of $H(C|K)$, the conditional entropy of the classes given the cluster assignments and the entropy of the cluster assignments.

$$\text{homogeneity} = 1 - \frac{H(C|K)}{H(C)}$$

- The completeness measures whether all members of a given class are assigned to the same cluster. It is derived from the ratio of $H(K|C)$, the conditional entropy of the cluster assignments given the classes and the entropy of the cluster assignments.

$$\text{completeness} = 1 - \frac{H(K|C)}{H(K)}$$

CheckV analysis

We used CheckV v0.7.0 [21] to analyze the quality of the bins resulting from the Station ALOHA dataset. We concatenated the contigs in each bin into an artificial contig as required by CheckV. We also ensured that any Direct Terminal Repeats found in the bin occur at the ends of the resulting construct. We report our results using CheckV's categories (Complete, high-quality, medium-quality, low-quality, undetermined).

Simulations and experimental datasets

Simulations

Reference genomes were downloaded from the NCBI RefSeq viral database [22] (`ftp://ftp.ncbi.nlm.nih.gov/refseq/release/viral/`; dataset version: September 12, 2019). This dataset contains 13,274 viruses, 9,316 of which had a length greater than 3 kb. The dataset was first preprocessed to convert ambiguous IUPAC codes into A, C, G, T by randomly sampling among the possible values of each ambiguous code. Unresolved nucleotides, or Ns, were removed from the k-mer counts to avoid biasing the computation. Finally, genomes shorter than 3 kb were discarded from the simulations as they were too small to yield at least two contigs of 2048 bp.

We used the preprocessed viral dataset with CAMISIM [23] to simulate four synthetic metagenomes. Each of the synthetic metagenomes had either 4 or 10 samples and a coverage of either 3X or 10X. For each simulation, we randomly sampled 500 or 2,000 genomes and used a log-normal distribution ($\mu = 1$, $\sigma = 3$) to simulate each genome's relative abundance. Each of the four experimental conditions was repeated ten times to account for sampling variability. This resulted in a total of 80 simulations.

We simulated the sequencing by fragmenting each genome into chunks with a mean length of 400 bp and a standard deviation of 10 bp. Each subsequence was then used to simulate paired-end reads with errors introduced using Illumina's MiSeq error profile. Contigs were inferred deterministically using CAMISIM's simulated reads' location in the reference genome. In this approach, which CAMISIM refers to a gold standard assembly, a contig is simply a consensus sequence of the reads simulated from the region it spans.

Experimental datasets

We used three viral metagenomes from Beaulaurier et al. [24]. The data was collected from Station ALOHA in the North Pacific Subtropical Gyre and sequenced with both Oxford Nanopore Technologies (ONT) and Illumina technologies. Raw Illumina reads

were downloaded from SRA (SRR10378148, SRR8811962, SRR8811963) and trimmed with Fastp [25]: reads were truncated when the quality on a sliding window of length 10 bp dropped below 25. Leading and trailing bases with quality below 5 were also trimmed. All reads shorter than 20 bp were filtered out. Trimmed reads were assembled with MetaSPAdes [26]. The resulting assembly contained 59,027 contigs longer than 2048 bp. Sequencing reads were aligned against the contigs using the bwa-mem algorithm [27] with default parameters. The alignments were filtered using *pysam* [28] to drop those that were partial (less than 50% of the query), had a low quality (30 or less) or contained an unmapped mate-pair (SAM flag=3596). We also filtered out alignments where reads had multiple alignments or PCR/optical duplicates. Finally, we removed paired alignments with mapping distance less than 200 or greater than 500 bp (see template length distribution in Supplementary Figure S3.1). The number of contigs filtered at each step is provided in Supplementary Table S3.1. After filtering, 42% of the contigs in each sample had a coverage above 1X.

These genomes were assembled and polished using the ONT reads as described in [24] (accession: PRJNA529454). This step resulted in 567, 96, and 1217 high-quality draft genomes in the respective samples. We pooled all the contigs and dereplicated them using Cluster-Genomes [29] using the default parameter values (minimum nucleotide identity=95%, min coverage=80%). Only 1,322 unique contigs remained after the dereplication.

Given that ONT and Illumina data were sequenced from the same biological samples, we aligned the shorter Illumina contigs against the ONT reference assembly to obtain the ground truth needed to verify our predictions. Thus, two contigs that align against the same genome reference must bin together in the ground truth solution. Conversely, pairs of contigs aligning to different genomes must appear in separate bins in the ground truth solution. We used minimap2 [30] to align the Illumina contigs against the ONT references and filter out potentially erroneous alignments if their minimum nucleotide identity was less than 95% (same threshold as Cluster-Genomes). We also discarded alignments where

the Illumina contigs were longer than the ONT reference. Such alignments don't convey information conducive to validating our binning predictions.

Other algorithms

We used CONCOCT's latest version 1.1.0 and set the maximum number of genomes to 2000 (maximum number of species in our simulations). We ran Metabat2 v2.15 with the default parameters.

3.3 Results

Viral datasets

Simulated datasets

We used a simulated dataset constructed using the NCBI's complete set of viral genomes (see Methods for more details). 38% of the viruses in the database belong to the *Caudovirales* order, a majority of which are of the *Siphoviridae* family (See supplemental Figure S3.7). As expected, the simulations were less fragmented as the number of samples or the coverage increased. For example, with 4 samples and 500 genomes, increasing the coverage from 4X to 10X decreases the number of contigs by 26% (23,258 to 17,303). Similarly, with 4X coverage and 500 genomes, increasing the samples from 3 to 15 leads to an 80% decrease in the number of contigs (see Supplementary Table S3.1). Table 3.1 displays some relevant properties of the simulated dataset in terms of average bin size, prevalence (number of samples in which a contig occurs), and number of contigs longer than 2kb. The bin size distribution for each setting is available in Supplementary Figure S3.8.

Experimental dataset "Station ALOHA"

A total of 2,332 Illumina contigs (longer than 2048bp) mapped against 988 ONT references with 95% similarity or higher. These alignments are valuable for providing the binning ground truth, which we relied on in our tests.

Dataset	Number of genomes	Coverage	Number of samples	Average bin size	Average prevalence	Number of contigs (>2kb)
Sim-1	500	3X	4	3.16 ±0.74	2.25 ±0.07	678 ±165
Sim-2	500	3X	15	1.89 ±0.15	6.30 ±0.37	893 ±70
Sim-3	500	10X	4	2.54 ±0.39	2.61 ±0.12	838 ±173
Sim-4	500	10X	15	1.17 ±0.12	8.54 ±0.32	582 ±57
Sim-5	2000	3X	4	3.25 ±0.32	2.18 ±0.07	2554 ±195
Sim-6	2000	3X	15	2.07 ±0.09	5.97 ±0.10	3871 ±163
Sim-7	2000	10X	4	2.48 ±0.08	2.56 ±0.08	3209 ±93
Sim-8	2000	10X	15	1.27 ±0.14	7.96 ±0.29	2520 ±271
SA	> 1300	6.6X ±2.2	3	N/A	1.51 ±0.61	59,027

Table 3.1: Simulation parameters summary: For each experimental condition, the means and standard deviations of ten replicates are shown. The number of bins for the Station ALOHA dataset is unknown but is estimated to be at least the number of curated contigs assembled from the Oxford Nanopore long reads. The table provides the number of contigs in each simulation, the average bin size, and the average prevalence (number of samples in which a contig occurs). The variance for each of these values across the ten replicate simulations is provided after the "±" symbol. SA refers to the experimental dataset "Station ALOHA"

Effect of fragment length on composition separation

We compared the distributions of cosine distances between the k-mer profiles of pairs of subsequences within vs across 9,316 publicly available viral genomes. Differences between intra- vs. inter-genome distance distributions were significant (t-test) at all fragment lengths tested although the distributions were very similar for the smallest fragment sizes (256 bp). The cosine distances progressively diverged (range 0.10 to 0.48) as a function of fragment length (Figure 3.3), suggesting that k-mer composition may represent a robust feature for binning when computed on fragments of at least 1024 bp. The high prevalence of viral genetic recombinations [31] and host to virus horizontal transfers [32] render binning using k-mer composition alone impractical. This point is evidenced by the overlap between the across- and within-species distributions in Figure 3.3. Other binning methods have suggested that coverage can facilitate recovery of more complete bins [11]. As such, it is crucial to include that information to enhance the signal conveyed by the k-mer composition.

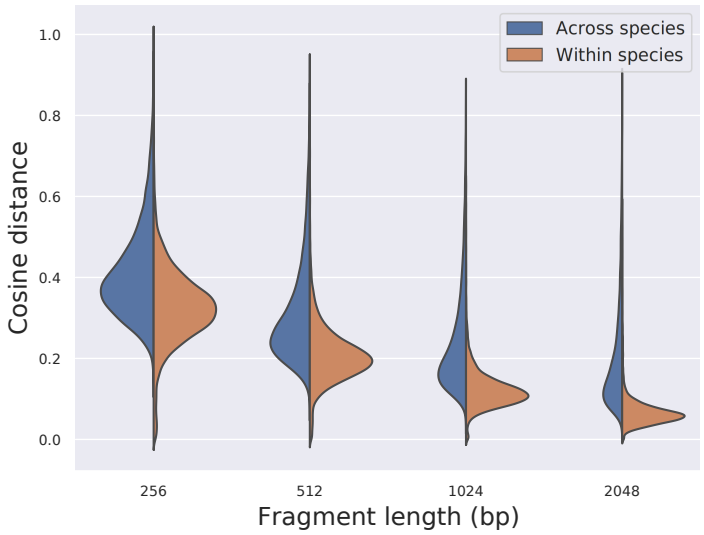


Figure 3.3: Distribution of k-mer distances: The k-mer cosine distance between fragments from the same genome (orange) and different genomes (blue) for fragments of length 256, 512, 1024, and 2048 bp. The variance of the cosine distance within and across species decreases gradually as the fragment length increases. The overlap between the two distributions is much smaller for fragments of 1024 bp or longer. This suggests that the best power for distinguishing within and across species requires fragments of at least 1024 bp.

CoCoNet performance on simulated viral data

For each simulated dataset, we trained the deep neural network using the 4-mer frequencies and coverage vectors as inputs. We measured the network learning performance using the accuracy, the F1 score, and the Area Under the ROC Curve (see Methods section 3.2 for details about these metrics).

Overall, the accuracy, the F1-score, and the AUC are all consistently above 95% (Figure 3.4). All of the metrics increase with the number of samples. Indeed, the larger the number of samples, the easier and more accurate it becomes for the model to learn the expected variability in coverage across contigs from the same genome. The computed quality metrics are also correlated with the sequencing depth since a higher coverage can lead to fewer contigs, more complete genomes, and less coverage variance between contigs

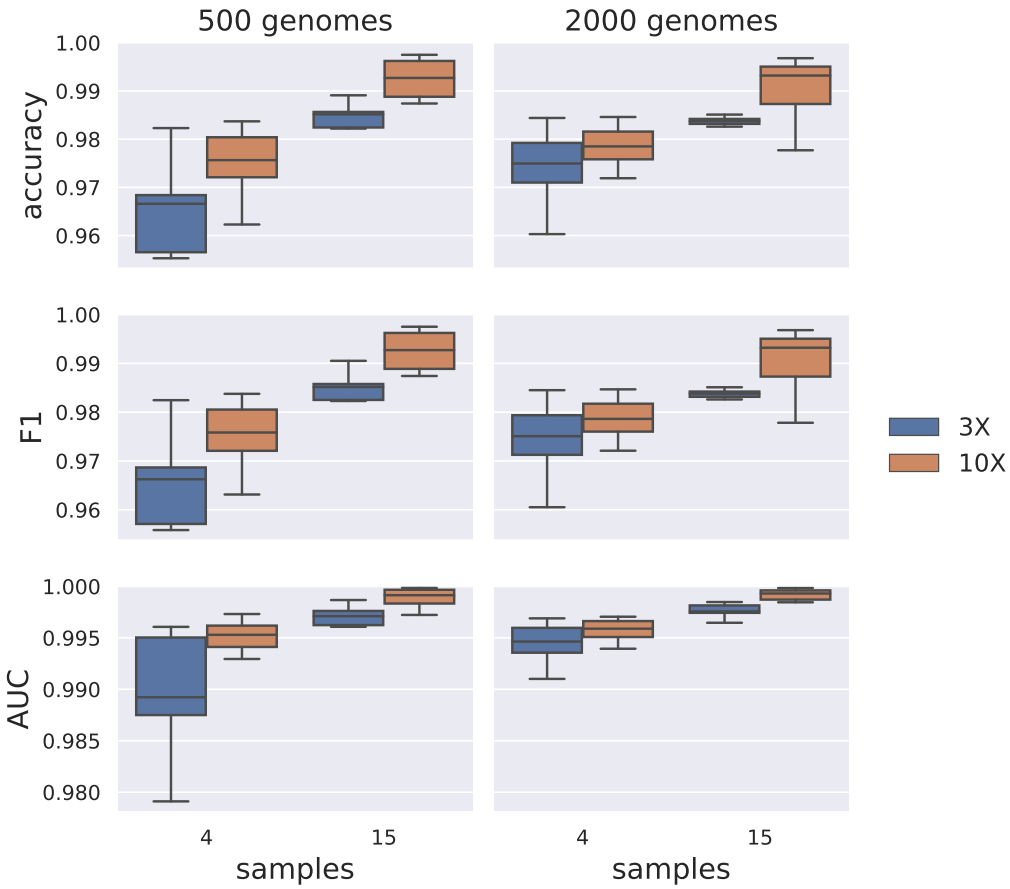


Figure 3.4: Neural network performance on simulated data: We vary the number of genomes (500, 2000), samples (4, 15) and coverage (blue: 3, orange: 10). The distribution of the scores is shown for each of the 3 metrics (AUC, Accuracy, F1 score).

of the same genome. We also observe a slight decrease in the variance and improved metrics when 2000 genomes were included in the simulations. This larger number of species allows for more diversity in training examples and more possibilities for generalization. The total counts for true positives, true negatives, false positives, false negatives show a similar trend (see Supplementary Figure S3.2).

The binning performances of CoCoNet, CONCOCT, and Metabat2 were compared for each of the simulated datasets using the ARI [20], the completeness and the homogeneity (Figure 3.5). The completeness measures true positives, or events where contigs were correctly binned together, while homogeneity measures true negatives, or events where

contigs were correctly assigned to separate bins. ARI is a more general metric that captures both types of events (see Methods section for more details). CoCoNet accurately reconstructs, on average, 14% of the bins that contain at least two contig, compared to 2.1% and 2.0% for Metabat2 and CONCOCT, respectively. Overall, CONCOCT assigned all the contigs to a small number of bins, yielding high levels of false positives. For instance, the maximum number of bins observed was 70 and occurred for a coverage of 3X, 15 samples, and 2,000 genomes. On the other hand, Metabat2 creates a small number of homogeneous, non-singleton bins (low number of false positives). In summary, CONCOCT has low homogeneity (median=0.418) and high completeness (median=0.984) while Metabat2, has high homogeneity (median=0.999) and lower completeness (median=0.817). In contrast, CoCoNet achieves high scores on both metrics, with a median homogeneity and completeness values of 0.944 and 0.942, respectively. Since the completeness measures whether contigs from the same virus are binned together, having a lower number of clusters artificially inflates this metric, i.e., decreasing the number of clusters can only increase the chance of grouping contigs. The ARI is a more generic metric since it measures both correct and erroneous binning events by considering all pairs of contigs. The ARI scores for bins produced by CONCOCT and Metabat2 were low due to their high false positive and false negative rates, respectively. Specifically, CONCOCT and Metabat2 have median ARI scores of 0.0794 and 0.200, whereas CoCoNet achieves a median score of 0.617.

Altogether, CoCoNet outperforms both CONCOCT and Metabat2 on the simulated data as it yields more bins containing contigs from the same genome and fewer bins containing contigs from different genomes.

CoCoNet accurately identifies contigs in the experimental dataset "Station ALOHA"

We processed all Illumina assembly contigs using CoCoNet, CONCOCT, and Metabat2, and computed the ARI, completeness, and homogeneity scores using only the subset of

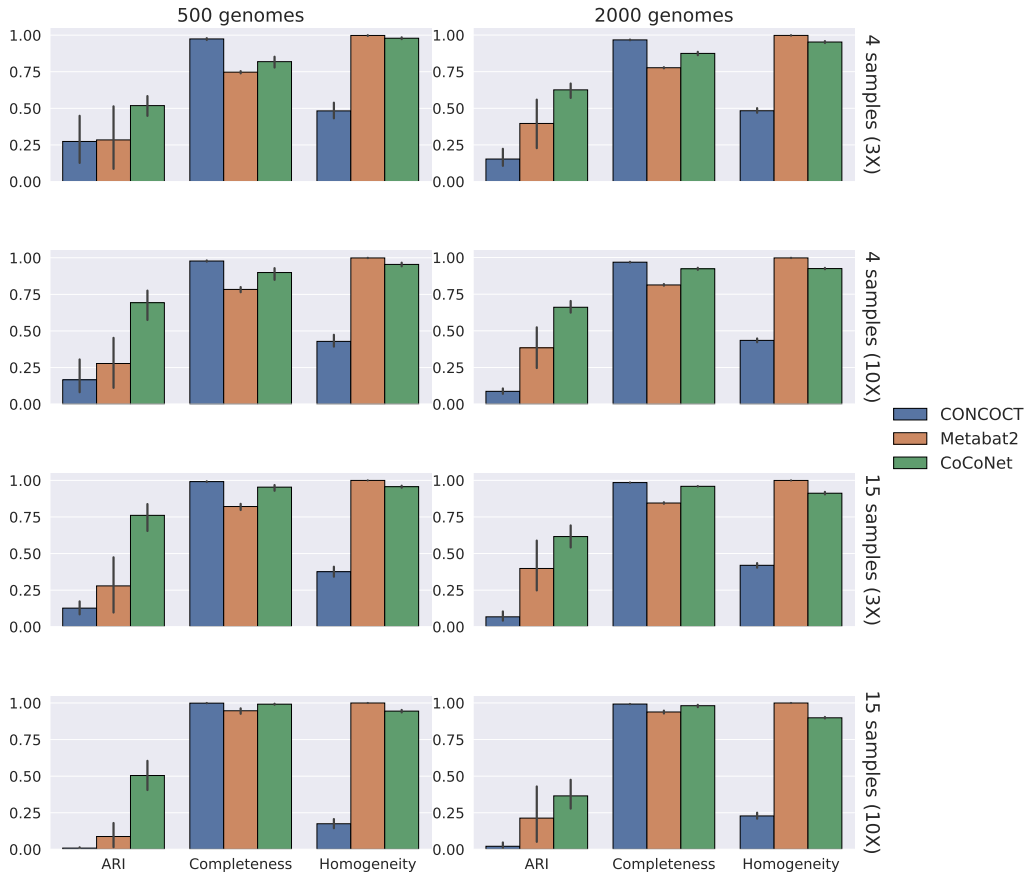


Figure 3.5: Clustering performance on simulated data: We vary the number of genomes (500, 2000), samples (4, 15) and coverage (3, 10). For each of the 3 metrics (ARI, Completeness, Homogeneity), the distribution of the scores is provided for each method (green: CoCoNet, blue: CONCOCT, orange: Metabat2).

contigs mapping against the draft genomes. Figure 3.6 compares the scores of the three metrics (ARI, Homogeneity, and completeness) for all three methods.

Similar to the results observed in the simulation, CONCOCT generates only 61 bins in total. Nineteen (19) of these bins are homogeneous, and only 9 were non-singleton. Metabat2 generated 2,212 bins, with 2,196 singletons, one homogeneous but partial, and 16 with erroneously merged contigs. In contrast, CoCoNet generates 1,452 bins, 1,177 were homogeneous, and 127 were homogeneous and non-singletons (see Supplementary Table S3.2).

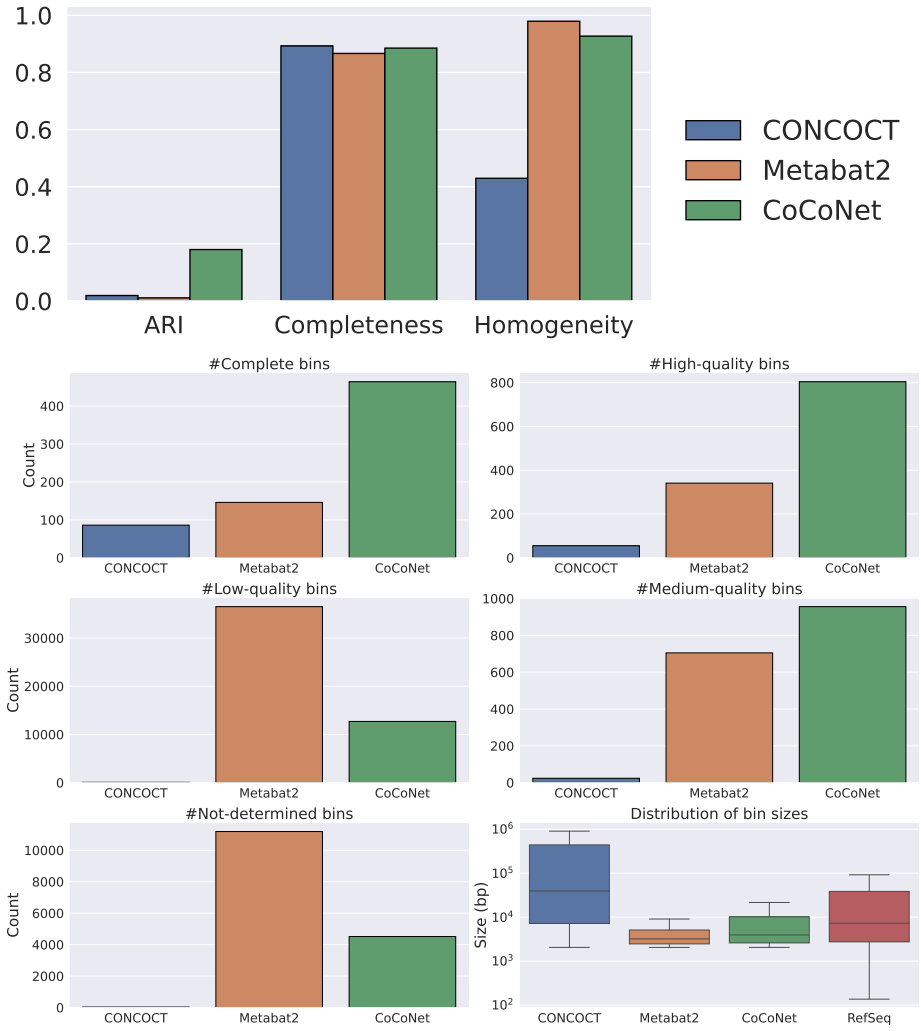


Figure 3.6: Clustering performance on Station ALOHA: (Top) The scores are provided for each of the 3 metrics (ARI, Completeness, Homogeneity) and each method (green: CoCoNet, blue: CONCOCT, orange: Metabat2). (Bottom) Bin classification by CheckV. Each barplot corresponds to the number of contigs in the given category. The last facet (boxplot) is the distribution of bin sizes for each method. The genome length distribution of the viral RefSeq database is shown as a reference.

Metabat2 generated many bins, resulting in a very high homogeneity (0.979) and completeness (0.869). However, the ARI value was low (0.00108) since only one non-singleton bin was correct but small (2 contigs). CONCOCT achieved a slightly higher completeness score (0.893) but a low homogeneity (0.421) and ARI (0.0103). In contrast, CoCoNet results in a completeness value on par with CONCOCT and Metabat2 (0.885)

but has substantially higher homogeneity (0.927) than CONCOCT and an ARI (0.180) an order of magnitude higher than the two other methods.

To further evaluate bin quality, we used CheckV to classify bins into 5 categories (complete, high-quality, medium-quality, low-quality, and not-determined). CheckV's results mirror the ARI scores, with 464 complete bins and 804 high-quality bins for CoCoNet, 146 complete bins and 341 high-quality bins for Metabat2 and 86 complete bins and 55 high-quality bins for CONCOCT (see Figure 3.6). We also compared the bin size distribution of each method with the RefSeq database to provide another qualitative assessment of bin completeness and contamination. Indeed, if bins are complete, then their size (in bp) should be similar to what can be found in reference viral databases. If bins are significantly larger, it could be a sign of contamination. For example, CONCOCT's median bin size (39,194) is much higher than in RefSeq (12,155 for genomes longer than 3kb, 7,253 overall). CoCoNet and Metabat2 are more comparable to RefSeq with medians equal to 3,926 and 3,204, respectively.

3.4 Discussion

Our results show that CoCoNet performs substantially better than existing methods for binning contigs assembled from viral metagenomes. Using CAMISIM, we simulated 80 datasets with varying number of bins, coverage, and number of samples. Overall, CoCoNet retrieved the correct bins with fewer errors compared to either CONCOCT or Metabat2. On a dataset that includes a high-quality ONT assembly and Illumina shotgun sequencing reads of the same samples, we showed that CoCoNet was able to retrieve substantially more correct bins than CONCOCT and Metabat2. This is particularly noteworthy since the dataset contained only three samples.

Unlike binning algorithms that summarize the coverage across the complete contig using one or two values (e.g. mean and standard deviation in the case of Metabat2), CoCoNet learns to model the inherent coverage variability within samples by considering the coverage

in a sliding window. This distinction is critical in viral metagenomes where the DNA amplification methods used to increase the typically low input material [5, 13] can yield uneven coverage depths [33, 34] and, therefore, confound methods that rely on summary statistics. We further highlight the importance of coverage variability in Supplementary Figure S3.3, which shows that the neural network's AUC generally decreases when the coverage variability is subtracted (i.e., coverage is set to the mean value).

CoCoNet's default behavior is to drop contigs shorter than 2048. CONCOCT, and Metabat2 also have minimum contig length thresholds. For example, Metabat2 uses a minimum contig length of 2.5kb in its model and relies on an ad-hoc heuristic to incorporate short contigs into the inferred bins. Here, we decided to drop contigs shorter than 2048 bases for two reasons. First, shorter contigs are difficult to classify (See Supplementary Figure S3.5) and their inclusion in the computation is unlikely to be without impact on the false-positive rate. Second, the more positive pairs we predict across two contigs (Figure 3.1a), the more likely we can assign those contigs to the same bin. To generate training instances from short contigs, we need to either 1- allow the fragments to overlap more or 2- take shorter fragments. The first solution results in training instances that are highly correlated. The second option results in training instances that suffer from high variance. In our tests, both solutions resulted in poor generalization power of the model.

We generate our negative examples by considering pairs of fragments that belong to different contigs. Our assumption here is that two randomly selected contigs are most likely to belong to separate rather than the same genome. Specifically, for large metagenomes such as those found in environmental samples (e.g., oceans, soil, etc.), we expect contig pairs originating from different genomes to outnumber contig pairs originating from the same species by a few orders of magnitude. As such, label noise would only represent a negligible fraction of the negative examples generated (See Supplementary Methods). Studies have also shown that neural networks are robust to massive label noise, which can be mitigated with large batch sizes [15]. CoCoNet uses a batch size of 256, which is reasonably large.

The coverage variability resulting from amplification bias [13] or coverage heteroskedasticity, where the coverage's standard deviation varies along a contig [35] are other sources of noise that can confound the binning. Deep learning models are also relatively robust to such sources of noise [36]. This makes CoCoNet more suitable for handling potential variability in sequencing coverage than alternative methods that summarize a contig's coverage as a single value.

CoCoNet's clustering approach is very conservative and emphasizes bin homogeneity. Three of CoCoNet's parameters can be adjusted to improve bin completeness at the cost of possibly decreased homogeneity. Those are 1- fragment length, 2- the minimum number of matches between two contigs connected by an edge in the contig-contig graph (θ), and 3- the minimum edge density required for considering a cluster as a bin (γ). Decreasing the values of θ or γ (respectively 80% and 75% by default) decreases the binning stringency.

Similarly, increasing the fragment length can minimize the variance in the k-mer and coverage distributions between contigs of the same species and, consequently, improve completeness (see Figure 3.3). Nevertheless, a longer fragment length can result in more contigs being assigned to singleton bins simply because they were not long enough to be processed. Naturally, decreasing the values of θ, γ, or the fragment length can result in more homogeneous but less complete bins (see Supplementary Figure S3.6).

Finally, CoCoNet's accuracy can be bolstered by combining multiple related experiments. As was shown with the simulated dataset, CoCoNet's performance strongly increases as we increase the number of samples.

3.5 Conclusion

CoCoNet is a novel, deep learning-based approach to bin viral metagenome assemblies. Our strategy leverages deep neural networks' flexibility to learn the similarity in composition and coverage across co-occurring contigs. Contrary to other methods that rely on coverage and k-mer composition, our method models these features as distributions rather than

summarizing them in a single statistic. Our results show that CoCoNet outperforms other tools on both simulated and real viral datasets. Our model is implemented in a self-contained easy to install Python program that requires reasonable computational resources to train the underlying deep neural network or use it to predict bins.

Software availability

CoCoNet was implemented in Python and is available for download on PyPi (`https://pypi.org/`). The source code is hosted on GitHub at `https://github.com/Puumanamana/CoCoNet` and the documentation is available at `https://coconet.readthedocs.io/en/latest/index.html`. CoCoNet does not require extensive resources to run. For example, binning 100k contigs took about 4 hours on 10 Intel CPU Cores (2.4GHz), with a memory peak at 27 GB (see Supplementary Figure S3.9). To process a large dataset, CoCoNet may need to be run on a high RAM capacity server. Such servers are typically available in high-performance or cloud computing settings.

Funding

This work was supported by funding from the National Science Foundation Division of Ocean Sciences (Grant #1636402—Investigation of viruses and microbes circulating deep in the seafloor) and the Office of Integrative Activities (Grants #1557349—'Ike Wai: Securing Hawaii's Water Future and #1736030–G2P in VOM: An experimental and analytical framework for genome to phenome connections in viruses of microbes).

Author contributions statement

MB envisioned the project, CA implemented the project and conducted the analysis with the help of MB. CA, MB, GP, ON and GS discussed the analyses and the results and

wrote the manuscript. All authors have read and agreed to the published version of the manuscript.

Additional information

Competing interests

The authors declare that they have no competing interests.

References

[1] H. Xie, R. Guo, H. Zhong, Q. Feng, Z. Lan, B. Qin, K. J. Ward, M. A. Jackson, Y. Xia, X. Chen, and others, "Shotgun metagenomics of 250 adult twins reveals genetic and environmental impacts on the gut microbiome," *Cell systems*, vol. 3, no. 6, pp. 572–584, 2016.

[2] A. Tyagi, B. Singh, N. K. B. Thammegowda, and N. K. Singh, "Shotgun metagenomics offers novel insights into taxonomic compositions, metabolic pathways and antibiotic resistance genes in fish gut microbiome," *Archives of microbiology*, vol. 201, no. 3, pp. 295–303, 2019.

[3] B. L. Hurwitz and M. B. Sullivan, "The Pacific Ocean Virome (POV): a marine viral metagenomic dataset and associated protein clusters for quantitative viral ecology," *PloS one*, vol. 8, no. 2, p. e57355, 2013.

[4] F. Angly, B. Felts, M. Breitbart, P. Salamon, R. Edwards, C. Carlson, A. Chan, M. Haynes, S. Kelley, H. Liu, J. Mahaffy, J. Mueller, J. Nulton, R. Olson, R. Parsons, S. Rayhawk, C. Suttle, and F. Rohwer, "The marine viromes of four oceanic regions," *PLoS biology*, vol. 4, p. e368, 12 2006.

[5] T. D. Sutton, A. G. Clooney, F. J. Ryan, R. P. Ross, and C. Hill, "Choice of assembly software has a critical impact on virome characterisation," *Microbiome*, vol. 7, no. 1, p. 12, 2019.

[6] R. García-López, J. F. Vázquez-Castellanos, and A. Moya, "Fragmentation and coverage variation in viral metagenome assemblies, and their effect in diversity calculations," *Frontiers in bioengineering and biotechnology*, vol. 3, p. 141, 2015.

[7] J. F. Vázquez-Castellanos, R. García-López, V. Pérez-Brocal, M. Pignatelli, and A. Moya, "Comparison of different assembly and annotation tools on analysis of simulated viral metagenomic communities in the gut," *BMC genomics*, vol. 15, no. 1, p. 37, 2014.

[8] M. Strous, B. Kraft, R. Bisdorf, and H. Tegetmeyer, "The binning of metagenomic contigs for microbial physiology of mixed cultures," *Frontiers in microbiology*, vol. 3, p. 410, 2012.

[9] M. Imelfort, D. Parks, B. J. Woodcroft, P. Dennis, P. Hugenholtz, and G. W. Tyson, "GroopM: an automated tool for the recovery of population genomes from related metagenomes," *PeerJ*, vol. 2, p. e603, 2014.

[10] D. D. Kang, F. Li, E. Kirton, A. Thomas, R. Egan, H. An, and Z. Wang, "Metabat 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies," *PeerJ*, vol. 7, p. e7359, 2019.

[11] J. Alneberg, B. S. Bjarnason, I. de Bruijn, M. Schirmer, J. Quick, U. Z. Ijaz, L. Lahti, N. J. Loman, A. F. Andersson, and C. Quince, "Binning metagenomic contigs by coverage and composition," *Nature Methods*, vol. 11, pp. 1144–1146, Nov. 2014.

[12] V. Popic, V. Kuleshov, M. Snyder, and S. Batzoglou, "GATTACA: lightweight metagenomic binning with compact indexing of kmer counts and minhash-based panel selection," *bioRxiv*, p. 130997, 2017.

[13] M. Parras-Moltó, A. Rodríguez-Galet, P. Suárez-Rodríguez, and A. López-Bueno, "Evaluation of bias induced by viral enrichment and random amplification protocols in metagenomic surveys of saliva dna viruses," *Microbiome*, vol. 6, no. 1, p. 119, 2018.

[14] S. R. Casjens and E. B. Gilcrease, "Determining dna packaging strategy by analysis of the termini of the chromosomes in tailed-bacteriophage virions," in *Bacteriophages*, pp. 91–111, Springer, 2009.

[15] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," *arXiv preprint arXiv:1705.10694*, 2017.

[16] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, p. 737–744, 1993.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] V. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, p. 5233, Dec. 2019. arXiv: 1810.08473.

[19] M. E. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical review E*, vol. 74, no. 3, p. 036104, 2006.

[20] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[21] S. Nayfach, A. P. Camargo, E. Eloe-Fadrosh, S. Roux, and N. Kyrpides, "Checkv: assessing the quality of metagenome-assembled viral genomes," *BioRxiv*, 2020.

[22] N. A. O'Leary, M. W. Wright, J. R. Brister, S. Ciufo, D. Haddad, R. McVeigh, B. Rajput, B. Robbertse, B. Smith-White, D. Ako-Adjei, *et al.*, "Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation," *Nucleic acids research*, vol. 44, no. D1, pp. D733–D745, 2016.

[23] A. Fritz, P. Hofmann, S. Majda, E. Dahms, J. Dröge, J. Fiedler, T. R. Lesker, P. Belmann, M. Z. DeMaere, A. E. Darling, *et al.*, "Camisim: simulating metagenomes and microbial communities," *Microbiome*, vol. 7, no. 1, pp. 1–12, 2019.

[24] J. Beaulaurier, E. Luo, J. M. Eppley, P. Den Uyl, X. Dai, A. Burger, D. J. Turner, M. Pendleton, S. Juul, E. Harrington, and others, "Assembly-free single-molecule sequencing recovers complete virus genomes from natural microbial communities," *Genome Research*, pp. gr–251686, 04 2020.

[25] S. Chen, Y. Zhou, Y. Chen, and J. Gu, "fastp: an ultra-fast all-in-one fastq preprocessor," *Bioinformatics*, vol. 34, no. 17, pp. i884–i890, 2018.

[26] S. Nurk, D. Meleshko, A. Korobeynikov, and P. A. Pevzner, "metaspades: a new versatile metagenomic assembler," *Genome research*, vol. 27, no. 5, pp. 824–834, 2017.

[27] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.

[28] S. Anders, P. T. Pyl, and W. Huber, "Htseq—a python framework to work with high-throughput sequencing data," *Bioinformatics*, vol. 31, no. 2, pp. 166–169, 2015.

[29] S. Roux and B. Bolduc, *ClusterGenomes*, 2009 (Last commit October 26, 2017).

[30] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[31] M. Lai, "Genetic recombination in rna viruses," in *Genetic Diversity of RNA Viruses*, pp. 21–32, Springer, 1992.

[32] C. Gilbert, J. Peccoud, A. Chateigner, B. Moumen, R. Cordaux, and E. A. Herniou, "Continuous influx of genetic material from host to virus populations," *PLoS genetics*, vol. 12, no. 2, p. e1005838, 2016.

[33] O. E. Karlsson, S. Belák, and F. Granberg, "The effect of preprocessing by sequence-independent, single-primer amplification (SISPA) on metagenomic detection

of viruses," *Biosecurity and bioterrorism: biodefense strategy, practice, and science*, vol. 11, no. S1, pp. S227–S234, 2013.

[34] T. Rosseel, S. Van Borm, F. Vandenbussche, B. Hoffmann, T. van den Berg, M. Beer, and D. Höper, "The origin of biased sequence depth in sequence-independent nucleic acid amplification and optimization for efficient massive parallel sequencing," *PloS one*, vol. 8, p. e76144, 09 2013.

[35] L. W. Hugerth and A. F. Andersson, "Analysing microbial community composition through amplicon sequencing: from sampling to hypothesis testing," *Frontiers in Microbiology*, vol. 8, p. 1561, 2017.

[36] S. D'Souza, K. Prema, and S. Balaji, "Machine learning in drug–target interaction prediction: current state and future directions," *Drug Discovery Today*, 2020.

[37] L. Hertel, J. Collado, P. Sadowski, and P. Baldi, "Sherpa: hyperparameter optimization for machine learning models," 2018.

3.6 Supplementary material

3.6.1 Supplementary methods

Mislabeling in CoCoNet training set

To construct a neural network's training example, we assume that two fragments that originate from different contigs are a true negative pair, i.e., belong to different genomes. This assumption is false when the two contigs belong to the same genome. Here, we show that the number of false negatives (contigs from the same genome) occurring in the training dataset are relatively low compared to the total amount of true negative examples.

First, we can observe that since CoCoNet chooses the same number of fragments for each contig, the proportion of false negative examples is the same as the proportion of false

negative contig pairs:

$$\frac{\text{False negative fragment pairs}}{\text{Negative fragment pairs}} = \frac{\text{False negative contig pairs} \times \text{Fragments per contig}}{\text{Negative contig pairs} \times \text{Fragments per contig}} = \frac{\text{False negative contig pairs}}{\text{Negative contig pairs}}$$

We consider a set of genomes G that have been sequenced and assembled into a set of contigs. Let n_{genomes} be the total number of genomes and n_{contigs} the total number of contigs in the dataset. For a given genome $g \in G$, let N_g be the number of contigs associated with g (in other words, N_g is the size of the bin defined by g).

The number of negative contig pairs is simply the number of combinations of any two (distinct) contigs:

$$N = \binom{n_{\text{contigs}}}{2}$$

The total number of possible False Negatives (FN) contig pairs corresponds to all pairs between two contigs that belong to the same genome:

$$FN = \sum_{g \in G} \binom{N_g}{2}$$

Therefore, the proportion of false negative examples depends on the fragmentation of the assembly. If we define $M = \max_{g \in G} N_g$ An upper bound of the false negative ratio is:

$$r = \frac{FN}{N} \leq \frac{n_{\text{genomes}} \times \binom{M}{2}}{\binom{n_{\text{contigs}}}{2}} \tag{3.2}$$

This upper bound is an equality if all the bins are of the same size and equation 3.2 can be simplified using $n_{\text{contigs}} = M \times n_{\text{genomes}}$:

$$r \leq \frac{M - 1}{2 \cdot (n_{\text{contigs}} - 1)} \tag{3.3}$$

Therefore, the condition to have a low false negative rate (in the simplified case) is:

$$M << n_{\text{contigs}}$$

which seems reasonable. For example, if $M = 100$ contigs/bin and $n_{\text{contigs}} = 10^4$, $r \leq 0.5\%$.

3.6.2 Supplementary figures

Station ALOHA preprocessing

We used BWA to align each of the Station ALOHA samples against the metaSPAdes contigs. We filtered the alignments to discard those that were either partial (represented 50% or less of the query's length), had a quality score lower than 30, or where one of the read pairs was unmapped (SAMflag=3596).

We also removed reads that had more than one high-quality alignment, or PCR/optical duplicates. As Figure S3.1 shows, the distribution of template length appears to follow a normal distribution centered around 300-350 bp. Based on this observation, we filtered any alignments shorter than 200 bp or longer than 500 bp.

Neural network performance: TP, TN, FP and FN

Figure S3.2 provides the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) values obtained by the neural network for each simulation setting. The TP and TN increase as we increase the coverage and the number of samples. The FP and FN decrease as a function of the number of samples and the coverage.

Importance of coverage variability

The objective here was to evaluate the effect of minimizing the coverage variation on the performance of the neural network. We removed a fragment's coverage variability by setting the new coverage as the mean of the fragment's coverage over all the samples (See Figure S3.3).

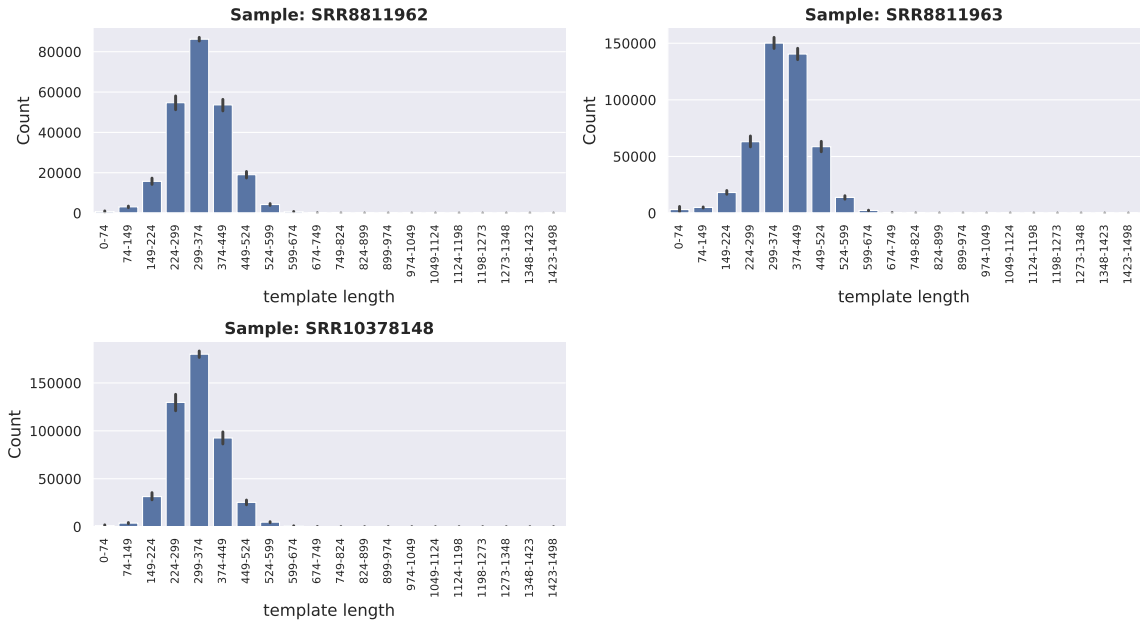


Figure S3.1: Template length histogram: Each facet is a different sample from the Aloha Station dataset. The x-axis is a fragment length bin and the y-axis is the bin count

Figure S3.3 shows that including the variation leads to a slight improvement in the AUC.

Distances in latent spaces

The clustering step creates the graph by comparing contigs in a pairwise manner. Since it is computationally intractable to compute all pairwise comparisons, we use a heuristic to choose contigs that are likely to cluster together into the same bin. In our heuristic, two such contigs must exist nearby in the composition and coverage latent spaces, i.e., the two contigs much be similar in their embeddings for composition and coverage. Figure S3.4 provides a representation of the composition and coverage spaces for a simulation containing 4 samples of 500 genomes at a 3X coverage.

Figure S3.4 shows that contigs in the same bin (orange dots) are closer to each other in both spaces than they are to objects from different bins (blue dots).

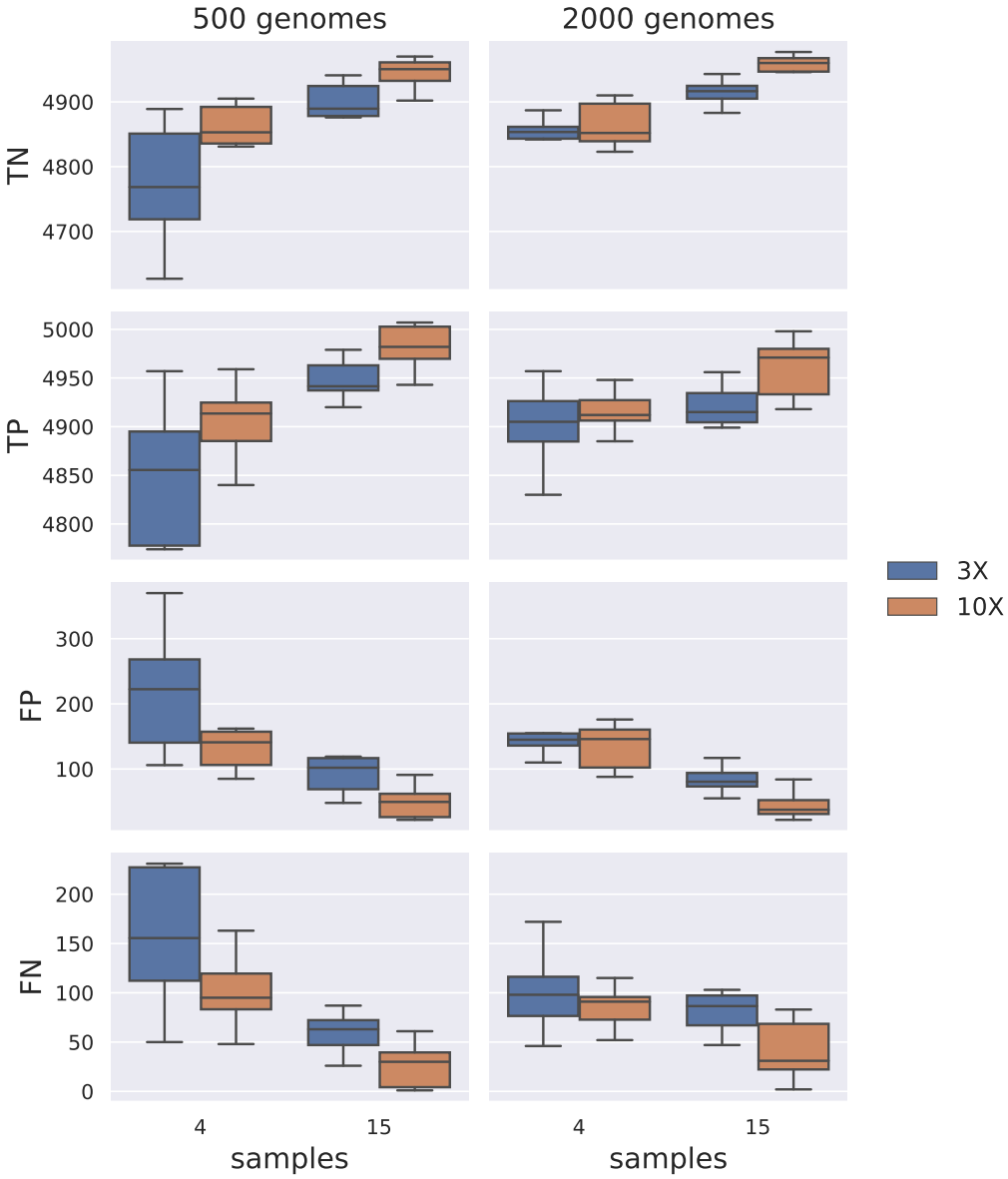


Figure S3.2: Classification accuracy for simulated datasets: Each row is a different metric (TN/TP/FP/FN) and each column corresponds to the number of genomes (500 or 2000). The number of samples (4 or 15) is on the x-axis and the y-axis is the metric's value. Colors represent sequencing depth (3X or 10X)

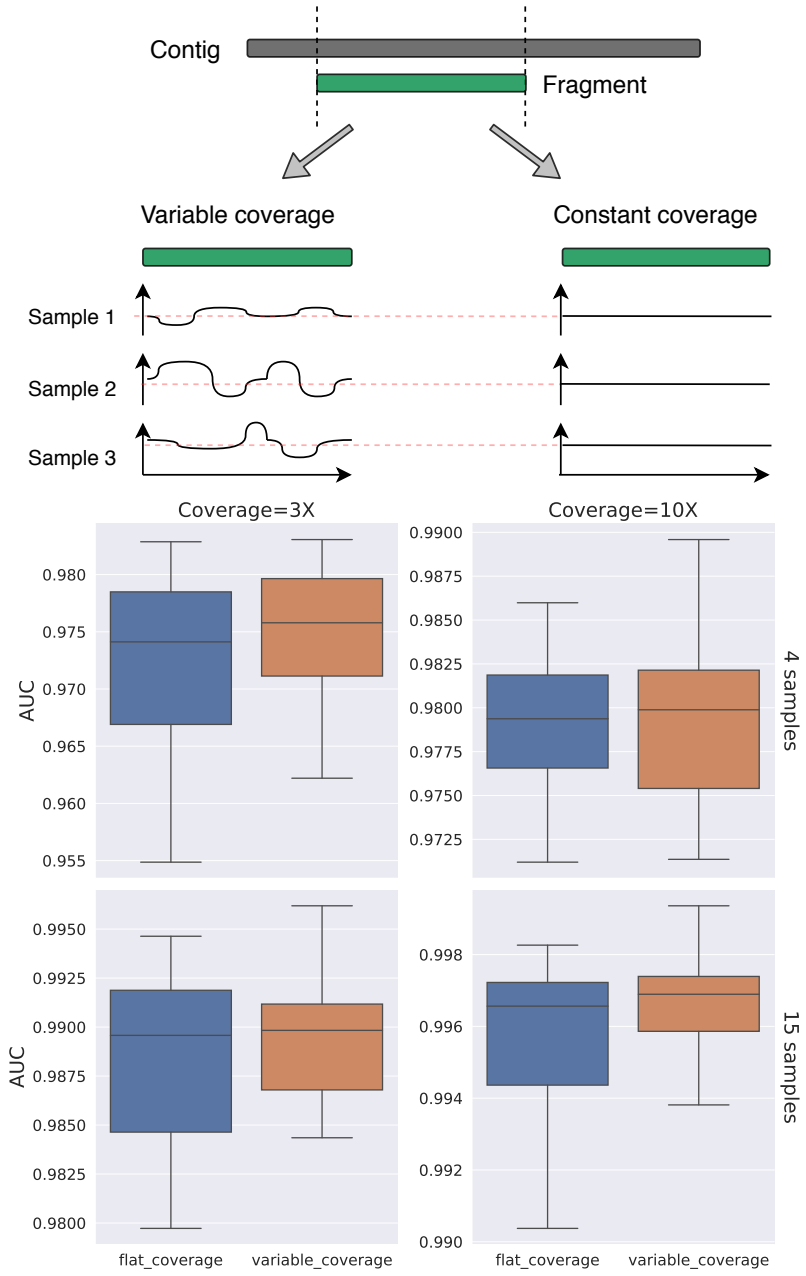


Figure S3.3: Importance of coverage variability: (Top) Coverage flattening procedure: For each fragment, we flatten the coverage by setting the coverage of each sample to its mean value. (Bottom) Comparison of the neural network's performance when taking the coverage variability into account. Each facet represents a different simulation setting.

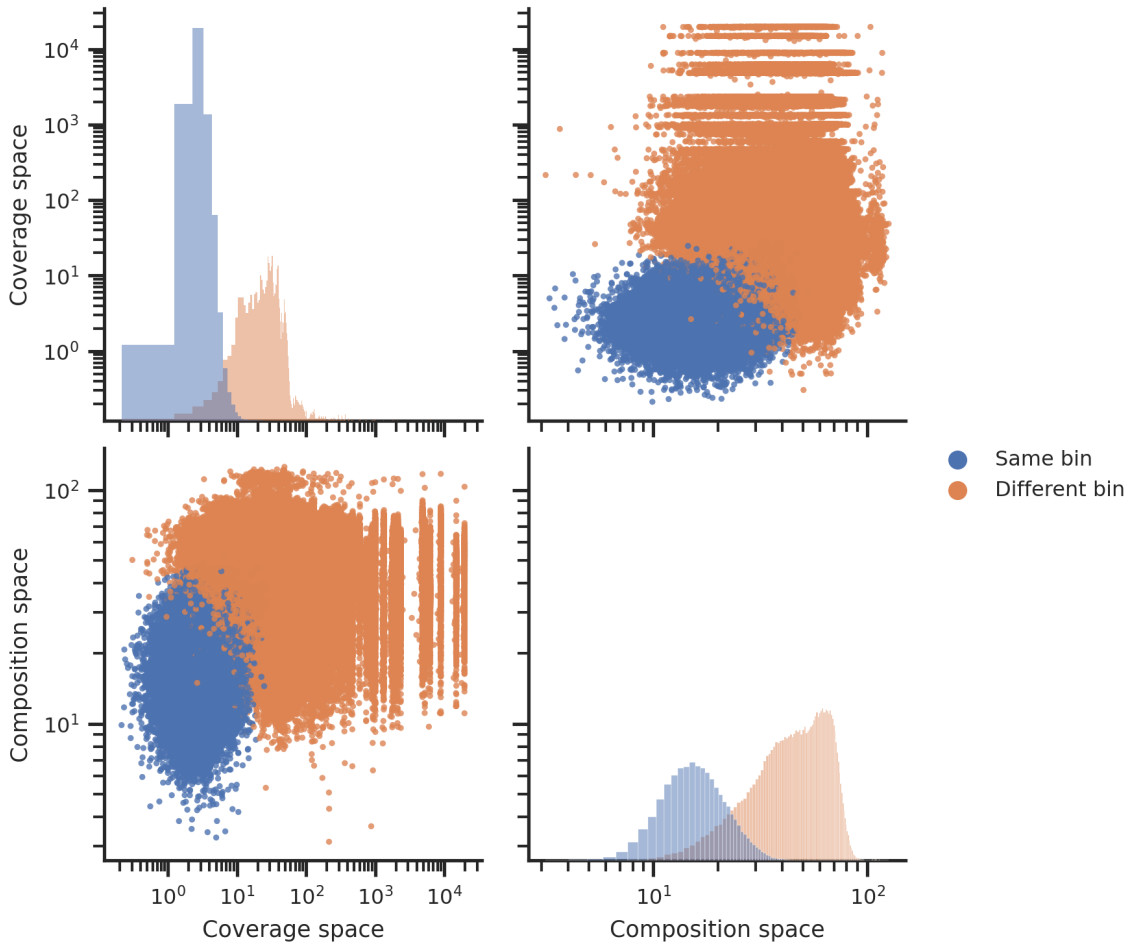


Figure S3.4: Distances in composition and coverage spaces: Top left and bottom right plots are the kernel density functions of the distribution of euclidean distances for fragment pairs in the same bin (orange) or different bins (blue). The x-axis corresponds to distance between fragments, while the y-axis is the corresponding frequency. The top right and bottom left plot show the correlation between the distances in each space. Each data point is a fragment pair colored in orange if they belong to the same bin, and blue otherwise. In the top right subplot, the x and y axes are the distances in the composition and coverage spaces, respectively. The lower left subplot is the same as the top right with axes inverted.

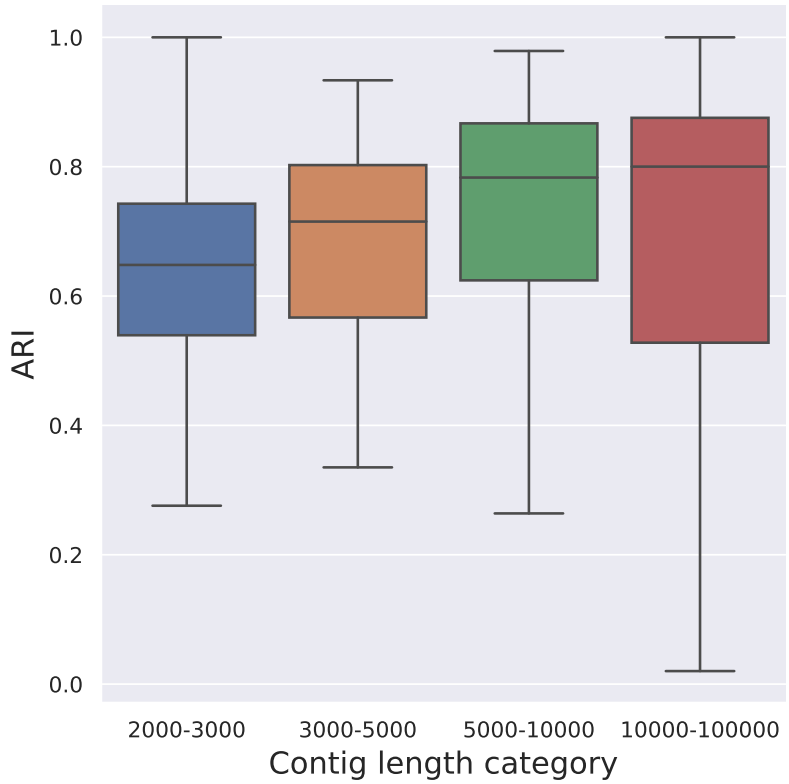


Figure S3.5: Effect of contig size on binning performance: The x-axis is a different contig size category and the y-axis is the binning performance for this category

Influence of contig length on clustering performance

To test whether contig length affects clustering performance, we ran CoCoNet on all the simulation data. We then split the contigs into groups based on their lengths and used the simulation results to compute each group's adjusted rand index. The objective here was to identify any statistically significant differences in ARI due to contig length. We assigned contigs based on their lengths to 4 groups, 2kb-3kb, 3kb-5kb, 5kb-10kb and 10kb-100kb:

Figure S3.5 show that the binning performance seems to improve as contig length increases. These differences were statistically significant at the 0.05 level (Kruskal-Wallis test).

Hyper-parameters tuning

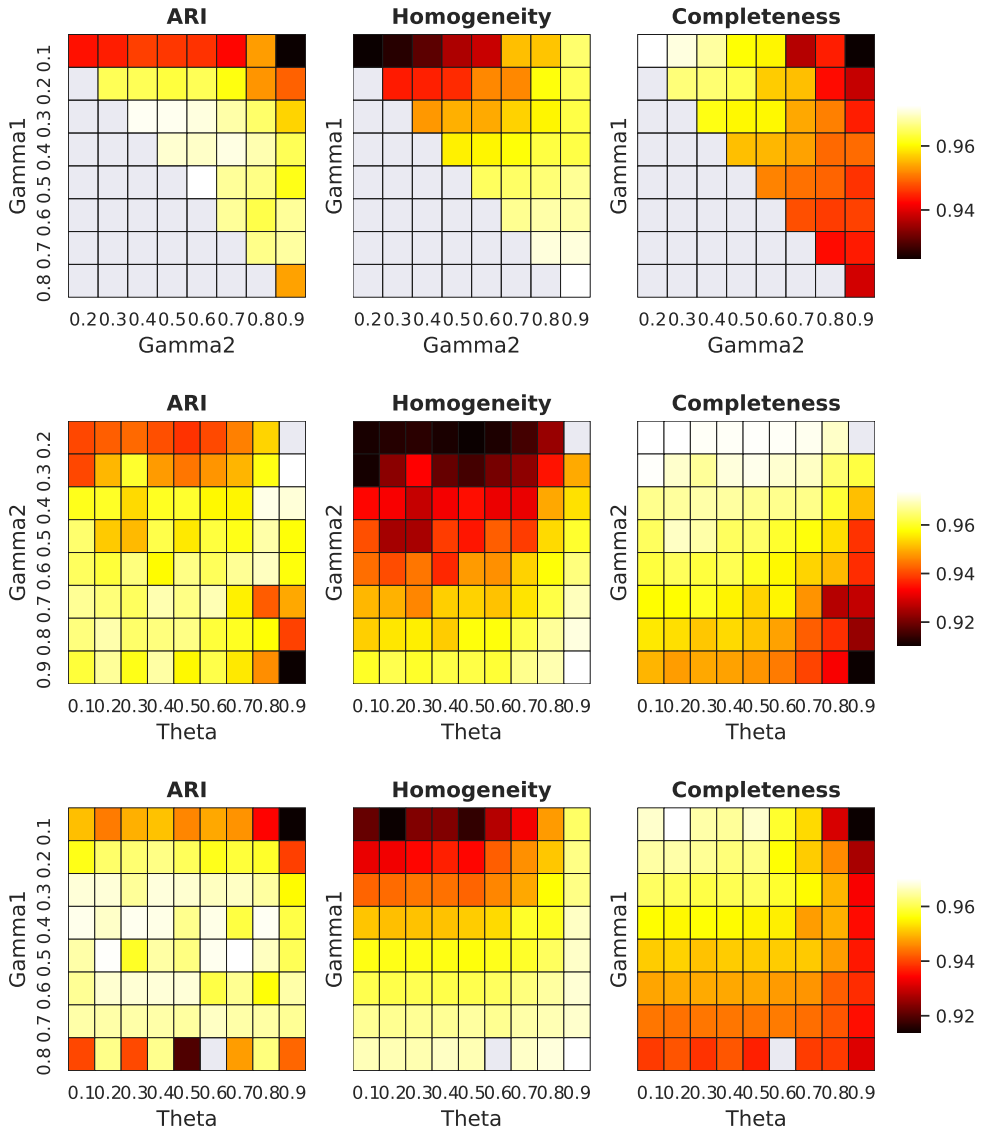


Figure S3.6: Heatmap view of hyperparameter optimization: Each row shows a different factor combination (in order (γ_1, γ_2), (γ_2, θ) and (γ_1, θ)) and each column shows a different metric (ARI, Homogeneity and Completeness). The color indicates the score (from dark red to light yellow). Grey entries in the lower triangles of the matrices comparing γ_1 and γ_2 were not evaluated since our heuristic requires that γ_1 be smaller than γ_2, i.e., the final clusters need to be more densely connected that the unpolished clusters. The remaining missing entries are parameter combinations that were not sampled in the Bayesian optimization procedure.

We explored the hyperparameter space with a Bayesian optimization approach (implemented in the python package `parameter-sherpa` [37]) to maximize the adjusted rand index (ARI). We evaluated the parameters γ_1, γ_2, and θ in the range $(0, 1)$ with a 0.1 increment, and the parameter `max_neighbors` in the range $[50, 500]$ with a 50 point increment. The parameters were tested on a CAMISIM simulation with 4,000 genomes, 5 samples and a coverage of 6X.

Figure S3.6 shows the average scores for all combinations of γ_1, γ_2, and θ for all three clustering metrics described in the paper (ARI, homogeneity and completeness). The optimal adjusted rand index was obtained for $\theta = 0.8, \gamma_1 = 0.3, \gamma_2 = 0.4$ and `max_neighbors`$= 250$. The hyper-parameter search table is available in supplementary table S3.3.

RefSeq taxonomy

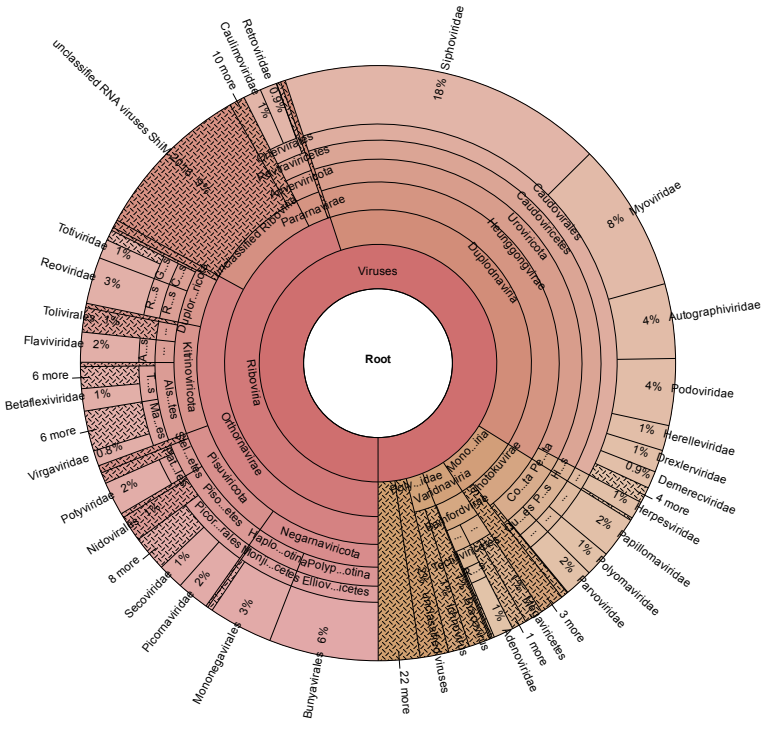


Figure S3.7: Taxonomy of RefSeq's genomes: Each disk is a different taxonomic level, starting from Kingdom down to Family level.

Simulation fragmentation

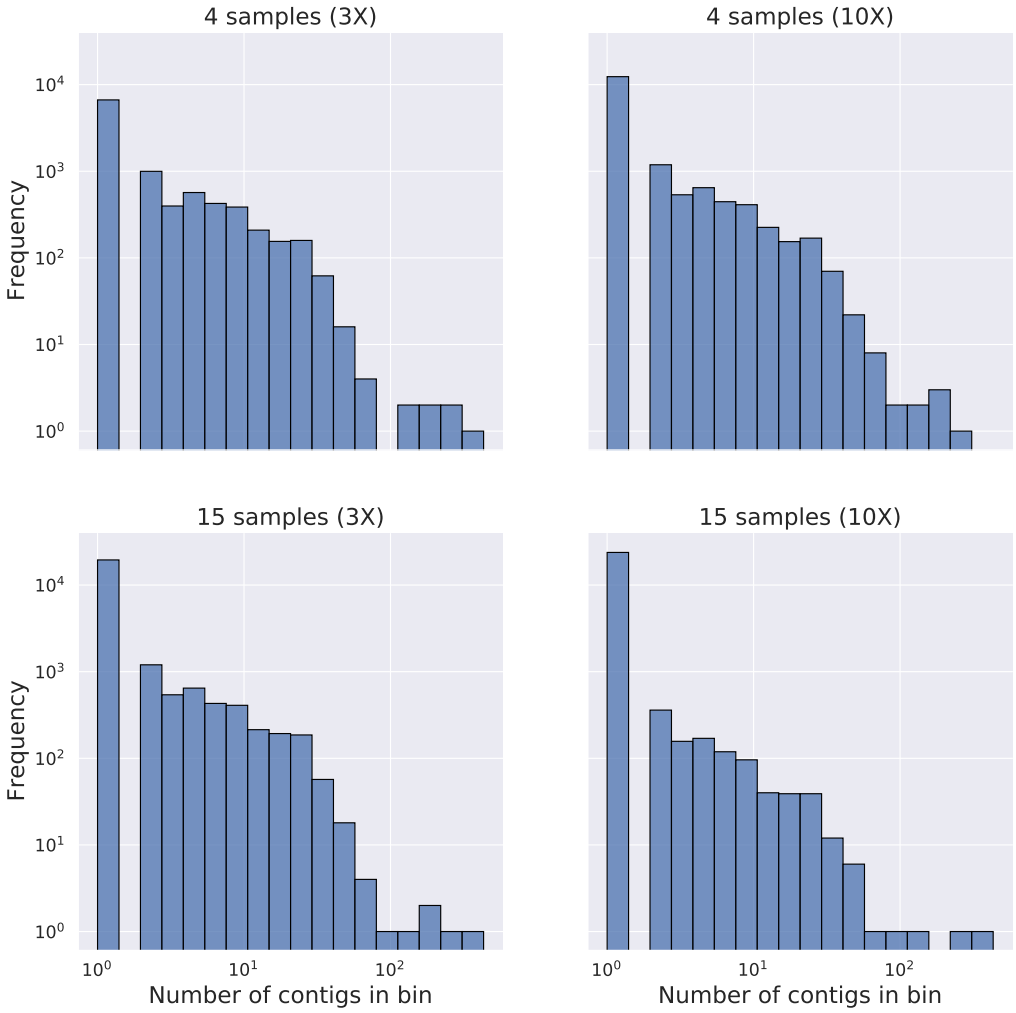


Figure S3.8: Bin size histogram for simulated data: The top row (resp. bottom) corresponds to simulations with 4 (resp. 15) samples. The left column (resp. right) corresponds to the simulations with a sequencing depth of 3X (resp. 10X).

Scalability

Dataset simulation

We explore the scalability of our approach using 7 new datasets with 5 samples each. We construct a dataset by iteratively selecting a genome and splitting it into a number of contigs we randomly select from the range $[1, \frac{\text{genome\_length}}{2048}]$. We only keep contigs that

are longer than 2048 bp. We then use a log-normal distribution where $\log \mu = 1$ and $\log \sigma = 2$ to sample the genome's mean coverage in each sample. We generate each sample's actual coverage by sampling from a Poisson distribution parameterized by the genome's mean coverage from the previous step.

Parameters affecting the runtime

The time required to train the neural network depends on the number of training examples (`--n-train`, default is 4M) and the patience parameter for early stopping (`--patience`, default is 5).

The time required during the clustering phase depends critically on the number of contigs passed as input. Our heuristics works in two phases. First, we initialize the clustering graph by comparing each contig with its closest neighbors in coverage and composition spaces (up to 250) and inserting an edge between two contigs if they meet our similarity criteria. We then use the contigs (nodes) and the newly added edges with the Leiden clustering algorithm to detect unpolished bins.

In the second step, we compute all previously uncomputed pairwise comparisons within an unpolished bin. The number of comparisons is therefore $\sum_{b \in bins} \binom{|\text{contigs} \in b|}{2}$. In the simplified case where all the bins have the same size, the number of comparisons becomes:

$$n_{\text{comparisons}} \simeq \#\text{bins} \times \left(\frac{\#\text{contigs}}{\#\text{bins}}\right)^2 = \frac{\#\text{contigs}}{\#\text{bins}} \cdot \#\text{contigs} = \#\text{contigs per bin} \times \#\text{contigs}$$

If we further assume that each bin contains less than a few hundred contigs, this step becomes linear in the number of contigs. However, the runtime can increase significantly when an unpolished bin contains thousands of contigs. To remedy this issue, we impose a hard limit of 100k comparisons in each unpolished bin. The user can also increase the resolution parameter γ_1 used in the first round of clustering, which will create more densely connected (i.e., smaller) unpolished bins.

In summary, clustering is the bottleneck step since its computational efficiency depends on the number of contigs processed. However, in most cases, the runtime scales linearly with the number of contigs, making CoCoNet a viable tool to handle large metagenomes.

Parameters affecting memory usage

The deep learning training phase is the most memory-intensive step in CoCoNet. We need to load the complete assembly into RAM during the training. We also use multiple processes to compute k-mer composition, which duplicates the data across processes. As such, CoCoNet's RAM requirements scale linearly with the number of contigs and the number of CPU used.

Results

Figure S3.9 shows CoCoNet, Metabat2 and CONCOCT's runtime and memory usage on a server with 10 Intel CPU Cores (2.4GHz). We performed the comparison on the 7 simulated datasets previously described (Dataset simulation) with 100, 500, 1k, 5k, 10k, 50k, and 100k contigs and 5 replicates each (total of 35 simulations). CoCoNet's runtime scales linearly with the number of contigs, and takes 240 minutes to bin 100k contigs. The peak memory consumption has a sublinear trend and reaches 27GB for 100k contigs. Metabat2 and CONCOCT require $\sim$ 3 GB of RAM and cluster 100k contigs in 26 and 400 minutes, respectively.

Despite its higher resources requirement, CoCoNet can handle large datasets with reasonable computational resources. In our experience, most viral metagenomes contain less than a million contigs longer than 2 kb. Since CoCoNet scales linearly with the number of contigs, it could bin metagenomes with a million contigs in less than a day on a deca-core server with 100 GB RAM. Such a configuration is typically available in most academic high-performance computing centers or can be leased for a few dollars in a cloud computing setting.

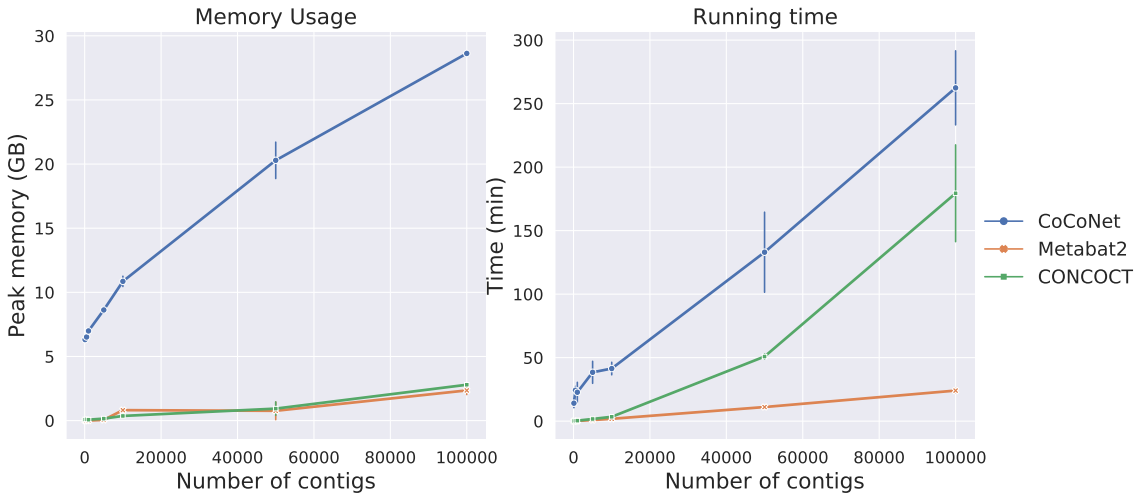


Figure S3.9: Binning performance comparison: Time and memory usage comparison between CoCoNet (blue), Metabat2 (orange) and CONCOCT (green) for increasing number of contigs (100, 500, 1k, 5k, 10k, 50k, and 100k). (Left) Memory usage in GB, (Right) Running time in minutes

3.6.3 Supplementary tables

Dataset	Number of genomes	Coverage	Number of samples	Raw	L> 2048bp	Prevalence> 1
Sim-1	500	3X	4	23,258	678	544
Sim-2	500	3X	15	4,375	894	892
Sim-3	500	10X	4	17,303	838	751
Sim-4	500	10X	15	941	582	582
Sim-5	2000	3X	4	92,516	2,555	1,992
Sim-6	2000	3X	15	20,756	3,872	3,865
Sim-7	2000	10X	4	68,699	3,210	2,883
Sim-8	2000	10X	15	5,192	2,521	2,520
SA	>1300	6.6X	3	2,882,543	56,844	56,565

Table S3.1: Dataset filtering summary: The "Raw" column represents the total number of contigs. The ">2048bp" column represents the number of contigs longer than 2048 bp. The "Prevalence>1" represents the number of contigs that both are longer than 2048 bp and appear in more than 1 sample. The last row is the filtering summary for the Station ALOHA (SA) dataset.

Method	Total bins	Homogeneous bins	Total bins (non-singleton)	Homogeneous bins (non-singleton)
CoCoNet	1452	1177	402	127
CONCOCT	61	19	51	9
Metabat2	2212	2197	16	1

Table S3.2: Bin count summary for each method: Homogeneous bins are the predicted bins with no false positives

γ_1	γ_2	θ	neighbors	ARI	completeness	homogeneity
0.1	0.2	0.1	250	0.71274	0.97344	0.92237
0.1	0.2	0.2	350	0.71893	0.97363	0.92339
0.1	0.2	0.3	300	0.72141	0.97296	0.92368
0.1	0.2	0.4	200	0.71543	0.97328	0.92179
0.1	0.2	0.4	200	0.7118	0.97279	0.92342
0.1	0.2	0.4	250	0.71456	0.97258	0.92256
0.1	0.2	0.4	450	0.71867	0.97285	0.92291
0.1	0.2	0.5	200	0.70722	0.97323	0.92134
0.1	0.2	0.6	200	0.71661	0.97251	0.92307
0.1	0.2	0.6	400	0.71013	0.97329	0.92263
0.1	0.2	0.7	250	0.73074	0.97292	0.92582
0.1	0.2	0.7	250	0.72586	0.97263	0.92629
0.1	0.2	0.7	400	0.72617	0.97205	0.92549
0.1	0.2	0.8	250	0.74896	0.97	0.93208
0.1	0.3	0.1	100	0.71336	0.97308	0.92216
0.1	0.3	0.1	300	0.71219	0.97338	0.92216
0.1	0.3	0.2	150	0.71342	0.97336	0.92207
0.1	0.3	0.2	450	0.71607	0.97318	0.92217
0.1	0.3	0.4	50	0.71549	0.97319	0.9221
0.1	0.3	0.4	100	0.7153	0.97359	0.92254
0.1	0.3	0.5	50	0.71948	0.97369	0.92364
0.1	0.3	0.5	100	0.70617	0.97347	0.92165
0.1	0.3	0.5	100	0.71793	0.97309	0.92311
0.1	0.3	0.5	350	0.71495	0.97251	0.92343
0.1	0.3	0.5	400	0.71707	0.97313	0.92317
0.1	0.3	0.6	150	0.71277	0.97208	0.92313
0.1	0.3	0.6	350	0.71373	0.97188	0.92407
0.1	0.3	0.6	400	0.70783	0.97161	0.9231
0.1	0.3	0.6	500	0.72522	0.97225	0.92594
0.1	0.3	0.7	150	0.72645	0.9707	0.92801
0.1	0.3	0.7	300	0.72957	0.97149	0.92744
0.1	0.3	0.7	350	0.73157	0.97119	0.92812
0.1	0.3	0.8	250	0.75136	0.96443	0.9376
0.1	0.3	0.8	300	0.73276	0.96254	0.9355
0.1	0.3	0.8	500	0.75104	0.96587	0.93761
0.1	0.3	0.8	500	0.74688	0.96514	0.93605
0.1	0.4	0.1	400	0.70694	0.97392	0.92238
0.1	0.4	0.2	50	0.71809	0.97396	0.9229
0.1	0.4	0.2	200	0.71613	0.97267	0.92317
0.1	0.4	0.3	100	0.71371	0.9739	0.92234
0.1	0.4	0.3	450	0.70743	0.97325	0.92147
0.1	0.4	0.3	450	0.70803	0.97327	0.92209
0.1	0.4	0.4	500	0.71836	0.97324	0.92395
0.1	0.4	0.5	300	0.71736	0.97217	0.92403
0.1	0.4	0.6	50	0.71487	0.97075	0.9259
0.1	0.4	0.6	200	0.71778	0.96944	0.92693
0.1	0.4	0.6	300	0.72247	0.97106	0.92582
0.1	0.4	0.7	50	0.73883	0.96769	0.93241
0.1	0.4	0.7	200	0.7323	0.9661	0.93155
0.1	0.4	0.7	300	0.74152	0.96834	0.93338
0.1	0.4	0.7	300	0.73596	0.96642	0.93201
0.1	0.4	0.7	400	0.74608	0.96782	0.9353
0.1	0.4	0.9	250	0.76307	0.94309	0.95288
0.1	0.4	0.9	450	0.79665	0.95001	0.95638
0.1	0.5	0.1	400	0.71492	0.97333	0.92241
0.1	0.5	0.2	50	0.70749	0.97312	0.92173
0.1	0.5	0.2	350	0.7152	0.97237	0.92342
0.1	0.5	0.2	400	0.715	0.97342	0.92333
0.1	0.5	0.3	200	0.713	0.97206	0.92294
0.1	0.5	0.3	450	0.7157	0.97306	0.92386
0.1	0.5	0.3	450	0.71636	0.97212	0.9233
0.1	0.5	0.4	250	0.71428	0.97148	0.92371
0.1	0.5	0.5	250	0.7217	0.9691	0.92832
0.1	0.5	0.5	250	0.72141	0.96864	0.92865
0.1	0.5	0.5	250	0.72293	0.9706	0.92785
0.1	0.5	0.5	350	0.7199	0.96949	0.92684
0.1	0.5	0.5	350	0.71368	0.96809	0.9269
0.1	0.5	0.5	500	0.72138	0.96904	0.92749
0.1	0.5	0.6	200	0.73317	0.96675	0.93225
0.1	0.5	0.6	300	0.7259	0.96327	0.93504
0.1	0.5	0.6	450	0.72556	0.96297	0.9345
0.1	0.5	0.7	100	0.7387	0.95713	0.94065
0.1	0.5	0.7	100	0.73033	0.95228	0.94272
0.1	0.5	0.7	200	0.76454	0.95691	0.94377
0.1	0.5	0.7	400	0.7579	0.95917	0.93958
0.1	0.5	0.7	450	0.75552	0.95778	0.94154
0.1	0.5	0.8	250	0.73309	0.94574	0.94839
0.1	0.5	0.9	200	0.77952	0.93395	0.96374
0.1	0.5	0.9	400	0.70768	0.9258	0.95991
0.1	0.5	0.9	500	0.74512	0.93309	0.96054
0.1	0.5	0.9	500	0.70835	0.92814	0.95909
0.1	0.6	0.1	300	0.71094	0.97186	0.92257
0.1	0.6	0.2	100	0.72024	0.97256	0.92552
0.1	0.6	0.2	250	0.70409	0.97201	0.92368
0.1	0.6	0.3	50	0.72318	0.97109	0.9267
0.1	0.6	0.4	50	0.73058	0.9698	0.92867
0.1	0.6	0.4	50	0.72018	0.96947	0.92856
0.1	0.6	0.4	150	0.72397	0.96892	0.92943
0.1	0.6	0.4	150	0.72325	0.96924	0.92827
0.1	0.6	0.4	250	0.71628	0.96889	0.92871
0.1	0.6	0.4	500	0.72542	0.9695	0.92792
0.1	0.6	0.6	50	0.7408	0.95333	0.94165
0.1	0.6	0.6	150	0.7506	0.95924	0.9412
0.1	0.6	0.6	300	0.74043	0.95917	0.94011
0.1	0.6	0.6	300	0.74275	0.95895	0.94067
0.1	0.6	0.6	450	0.73582	0.95565	0.94074
0.1	0.6	0.6	500	0.75877	0.96014	0.94267
0.1	0.6	0.7	50	0.74408	0.95124	0.94704
0.1	0.6	0.7	450	0.76721	0.95008	0.94803
0.1	0.6	0.9	50	0.6255	0.92236	0.96064
0.1	0.6	0.9	200	0.72119	0.92271	0.96664
0.1	0.7	0.1	100	0.73589	0.96836	0.93207
0.1	0.7	0.2	250	0.72688	0.96776	0.93108
0.1	0.7	0.3	300	0.72776	0.96555	0.93456
0.1	0.7	0.3	450	0.74249	0.96695	0.93671
0.1	0.7	0.5	50	0.77341	0.95922	0.945
0.1	0.7	0.5	400	0.76126	0.95634	0.94555
0.1	0.7	0.5	450	0.75803	0.95575	0.94622
0.1	0.7	0.6	150	0.77773	0.95282	0.9506
0.1	0.7	0.7	250	0.77089	0.93985	0.95833
0.1	0.7	0.7	300	0.76617	0.94035	0.95728
0.1	0.7	0.8	100	0.69447	0.91734	0.96247
0.1	0.7	0.8	100	0.69265	0.91489	0.96487
0.1	0.7	0.8	150	0.68722	0.91588	0.96303
0.1	0.7	0.8	200	0.70522	0.91995	0.96208
0.1	0.7	0.8	300	0.69733	0.91965	0.96242
0.1	0.7	0.8	450	0.70927	0.91827	0.96224
0.1	0.7	0.9	350	0.60962	0.90465	0.96925
0.1	0.7	0.9	400	0.64454	0.90814	0.96875
0.1	0.7	0.9	450	0.65555	0.90983	0.97042
0.1	0.8	0.1	150	0.76582	0.96118	0.94313
0.1	0.8	0.1	500	0.77951	0.96265	0.94598
0.1	0.8	0.1	500	0.76971	0.96207	0.94424
0.1	0.8	0.2	150	0.77599	0.95958	0.94499
0.1	0.8	0.3	150	0.77084	0.95924	0.94687
0.1	0.8	0.3	150	0.77624	0.95736	0.94813
0.1	0.8	0.3	250	0.78163	0.95853	0.94899
0.1	0.8	0.3	250	0.77734	0.95853	0.94743
0.1	0.8	0.3	500	0.77695	0.95825	0.94807
0.1	0.8	0.4	100	0.7764	0.95436	0.95181
0.1	0.8	0.4	200	0.77953	0.95301	0.95179
0.1	0.8	0.4	200	0.78203	0.95463	0.95112
0.1	0.8	0.4	200	0.78313	0.95615	0.95131
0.1	0.8	0.4	300	0.78085	0.95476	0.95246
0.1	0.8	0.4	300	0.77376	0.95452	0.95041
0.1	0.8	0.4	500	0.77568	0.95396	0.95066
0.1	0.8	0.4	500	0.77511	0.95227	0.95118
0.1	0.8	0.5	450	0.7689	0.94563	0.95671
0.1	0.8	0.6	300	0.76231	0.94264	0.95826
0.1	0.8	0.6	300	0.75019	0.93526	0.96028
0.1	0.8	0.6	350	0.78171	0.9448	0.95917
0.1	0.8	0.7	50	0.63252	0.92709	0.96187
0.1	0.8	0.7	300	0.73723	0.92914	0.96268
0.1	0.8	0.7	400	0.7351	0.92611	0.96207
0.1	0.8	0.8	150	0.64431	0.90823	0.96527
0.1	0.8	0.8	150	0.66784	0.90925	0.96641
0.1	0.8	0.8	350	0.62531	0.90401	0.9661
0.1	0.8	0.9	450	0.58267	0.8975	0.9714
0.1	0.9	0.1	100	0.78995	0.95715	0.95476
0.1	0.9	0.1	500	0.79087	0.95696	0.95331
0.1	0.9	0.2	150	0.79252	0.95475	0.95459
0.1	0.9	0.3	50	0.69353	0.95075	0.95628
0.1	0.9	0.3	150	0.78197	0.95244	0.95456
0.1	0.9	0.4	250	0.78315	0.94942	0.95788
0.1	0.9	0.6	200	0.74137	0.93058	0.96226
0.1	0.9	0.6	300	0.75982	0.9379	0.96208
0.1	0.9	0.6	350	0.74393	0.93183	0.96153
0.1	0.9	0.7	100	0.63147	0.91671	0.96529
0.1	0.9	0.7	350	0.71942	0.92064	0.96723
0.1	0.9	0.7	500	0.69967	0.9182	0.96611
0.1	0.9	0.8	400	0.57783	0.89817	0.96829
0.1	0.9	0.9	200	0.49021	0.88963	0.97326
0.1	0.9	0.9	200	0.45163	0.87863	0.973
0.1	0.9	0.9	300	0.48101	0.88903	0.97214
0.1	0.9	0.9	450	0.48017	0.88756	0.97131
0.2	0.3	0.2	150	0.76734	0.96799	0.94055
0.2	0.3	0.2	350	0.76912	0.9667	0.94013
0.2	0.3	0.3	250	0.76262	0.96662	0.9388
0.2	0.3	0.3	400	0.76673	0.96682	0.93973
0.2	0.3	0.3	400	0.76533	0.9671	0.93997
0.2	0.3	0.3	450	0.7755	0.96742	0.94116
0.2	0.3	0.4	200	0.77165	0.96715	0.94053
0.2	0.3	0.5	500	0.77259	0.96816	0.94013
0.2	0.3	0.6	100	0.76783	0.96728	0.94037
0.2	0.3	0.6	100	0.76867	0.96757	0.94075
0.2	0.3	0.7	300	0.7747	0.96607	0.94228
0.2	0.3	0.8	150	0.78867	0.96365	0.94841
0.2	0.3	0.8	250	0.78196	0.96497	0.94618
0.2	0.3	0.8	300	0.78874	0.96457	0.94824
0.2	0.3	0.9	500	0.80298	0.96162	0.95301
0.2	0.4	0.1	200	0.77069	0.96725	0.94078
0.2	0.4	0.2	100	0.77497	0.96753	0.94142
0.2	0.4	0.2	350	0.77278	0.9671	0.94099
0.2	0.4	0.2	350	0.77111	0.96704	0.94051
0.2	0.4	0.4	50	0.77648	0.96756	0.94186
0.2	0.4	0.4	500	0.76723	0.96661	0.9396
0.2	0.4	0.4	500	0.77109	0.96711	0.94114
0.2	0.4	0.5	50	0.77295	0.96763	0.94098

γ_1	γ_2	θ	neighbors	ARI	completeness	homogeneity
0.2	0.4	0.5	150	0.76636	0.96669	0.94084
0.2	0.4	0.5	250	0.76933	0.96759	0.94044
0.2	0.4	0.6	350	0.77183	0.96646	0.94193
0.2	0.4	0.7	150	0.77415	0.96562	0.94362
0.2	0.4	0.7	300	0.77675	0.96591	0.94218
0.2	0.4	0.7	400	0.7744	0.96444	0.94418
0.2	0.4	0.8	100	0.78164	0.96227	0.94796
0.2	0.4	0.9	50	0.80815	0.95666	0.9586
0.2	0.5	0.1	400	0.77017	0.96713	0.94038
0.2	0.5	0.1	500	0.7718	0.96708	0.94108
0.2	0.5	0.2	50	0.77779	0.96779	0.94128
0.2	0.5	0.2	350	0.76915	0.96747	0.94089
0.2	0.5	0.2	450	0.76944	0.96728	0.94143
0.2	0.5	0.2	450	0.76696	0.96689	0.94049
0.2	0.5	0.3	200	0.7692	0.96705	0.94051
0.2	0.5	0.4	150	0.77143	0.96677	0.94085
0.2	0.5	0.4	450	0.77172	0.96753	0.94136
0.2	0.5	0.5	350	0.76912	0.96647	0.94143
0.2	0.5	0.6	100	0.77184	0.96489	0.94256
0.2	0.5	0.8	450	0.77898	0.95486	0.95278
0.2	0.5	0.9	100	0.78237	0.94566	0.96288
0.2	0.6	0.1	100	0.76491	0.967	0.9404
0.2	0.6	0.1	300	0.76661	0.96713	0.94016
0.2	0.6	0.2	400	0.7709	0.96738	0.94189
0.2	0.6	0.3	100	0.77311	0.96684	0.94136
0.2	0.6	0.3	300	0.77073	0.96695	0.94141
0.2	0.6	0.3	400	0.77264	0.96705	0.94126
0.2	0.6	0.4	350	0.76902	0.96683	0.94185
0.2	0.6	0.5	150	0.77045	0.96579	0.94219
0.2	0.6	0.5	500	0.77195	0.96563	0.94294
0.2	0.6	0.6	150	0.76305	0.96297	0.94495
0.2	0.6	0.7	250	0.77954	0.95775	0.95046
0.2	0.6	0.8	100	0.78305	0.94995	0.95812
0.2	0.6	0.8	200	0.78958	0.9519	0.95789
0.2	0.6	0.8	250	0.78402	0.94826	0.95663
0.2	0.6	0.8	250	0.76289	0.94643	0.95594
0.2	0.6	0.8	250	0.79028	0.94781	0.95855
0.2	0.6	0.8	300	0.7833	0.94808	0.95685
0.2	0.6	0.8	500	0.77438	0.94839	0.95721
0.2	0.6	0.9	450	0.77416	0.93689	0.96454
0.2	0.6	0.9	500	0.78989	0.93823	0.96706
0.2	0.7	0.1	250	0.77081	0.96601	0.94261
0.2	0.7	0.2	400	0.77404	0.96591	0.94344
0.2	0.7	0.2	450	0.76566	0.96613	0.94235
0.2	0.7	0.3	150	0.77093	0.9656	0.94249
0.2	0.7	0.3	500	0.77462	0.9662	0.94321
0.2	0.7	0.4	500	0.77596	0.96375	0.94497
0.2	0.7	0.5	500	0.77061	0.96133	0.94609
0.2	0.7	0.6	500	0.77603	0.95935	0.94832
0.2	0.7	0.7	50	0.6946	0.94766	0.95268
0.2	0.7	0.7	100	0.74965	0.94452	0.95435
0.2	0.7	0.7	150	0.7874	0.95358	0.95585
0.2	0.7	0.7	350	0.79171	0.95531	0.95501
0.2	0.7	0.8	150	0.76905	0.93802	0.96175
0.2	0.7	0.9	250	0.75691	0.92881	0.96953
0.2	0.8	0.2	300	0.77845	0.96171	0.94748
0.2	0.8	0.2	400	0.77853	0.96113	0.94724
0.2	0.8	0.2	400	0.77945	0.96078	0.94797
0.2	0.8	0.4	350	0.78566	0.95854	0.9521
0.2	0.8	0.5	400	0.77549	0.95574	0.95207
0.2	0.8	0.6	100	0.78144	0.95211	0.95709
0.2	0.8	0.6	100	0.76215	0.95007	0.95617
0.2	0.8	0.7	150	0.78113	0.9453	0.96181
0.2	0.8	0.7	150	0.77442	0.94362	0.96098
0.2	0.8	0.7	200	0.77102	0.94153	0.96127
0.2	0.8	0.8	150	0.73361	0.92893	0.96624
0.2	0.8	0.9	50	0.58818	0.91696	0.96919
0.2	0.8	0.9	250	0.67133	0.91341	0.97048
0.2	0.8	0.9	450	0.66489	0.91264	0.97161
0.2	0.9	0.1	50	0.70653	0.95364	0.95433
0.2	0.9	0.2	100	0.78634	0.95589	0.95522
0.2	0.9	0.3	100	0.78779	0.95523	0.95602
0.2	0.9	0.3	150	0.78962	0.95467	0.95595
0.2	0.9	0.3	450	0.78903	0.95592	0.95593
0.2	0.9	0.4	150	0.7891	0.95386	0.95748
0.2	0.9	0.4	150	0.78966	0.95332	0.95629
0.2	0.9	0.4	500	0.79166	0.95272	0.95735
0.2	0.9	0.5	100	0.76883	0.94879	0.95804
0.2	0.9	0.6	50	0.6629	0.93933	0.9595
0.2	0.9	0.6	250	0.77932	0.94394	0.9628
0.2	0.9	0.6	400	0.76683	0.94153	0.96142
0.2	0.9	0.7	200	0.74352	0.93405	0.96478
0.2	0.9	0.7	450	0.75282	0.93743	0.96442
0.2	0.9	0.8	400	0.69866	0.92392	0.96873
0.2	0.9	0.8	450	0.6964	0.92198	0.96726
0.2	0.9	0.8	450	0.6938	0.92192	0.96848
0.2	0.9	0.9	200	0.63974	0.90409	0.97364
0.2	0.9	0.9	250	0.60675	0.9039	0.97272
0.2	0.9	0.9	300	0.66121	0.90739	0.97218
0.3	0.4	0.1	50	0.7875	0.96206	0.94906
0.3	0.4	0.1	200	0.79101	0.96242	0.94897
0.3	0.4	0.2	50	0.79343	0.9631	0.94941
0.3	0.4	0.2	400	0.79135	0.9634	0.94937
0.3	0.4	0.2	450	0.78969	0.96307	0.94906
0.3	0.4	0.3	50	0.78974	0.9629	0.94918
0.3	0.4	0.3	400	0.78661	0.96208	0.94856
0.3	0.4	0.3	450	0.79391	0.96301	0.94991
0.3	0.4	0.4	50	0.79377	0.96362	0.94955
0.3	0.4	0.5	50	0.78595	0.96276	0.94915
0.3	0.4	0.5	500	0.79125	0.963	0.94997
0.3	0.4	0.6	100	0.79036	0.96176	0.94949
0.3	0.4	0.6	200	0.79566	0.9628	0.95059
0.3	0.4	0.6	450	0.79203	0.96238	0.94947
0.3	0.4	0.7	50	0.79434	0.96207	0.95138
0.3	0.4	0.8	150	0.79865	0.96005	0.9532
0.3	0.4	0.8	250	0.8095	0.96115	0.95503
0.3	0.4	0.8	450	0.80617	0.96019	0.95475
0.3	0.4	0.9	100	0.80788	0.953	0.95956
0.3	0.4	0.9	300	0.80444	0.95406	0.95842
0.3	0.5	0.1	300	0.78899	0.96239	0.94898
0.3	0.5	0.1	300	0.79192	0.96249	0.95004
0.3	0.5	0.1	400	0.78862	0.96233	0.94907
0.3	0.5	0.1	400	0.78563	0.96235	0.9486
0.3	0.5	0.4	500	0.78786	0.96194	0.94892
0.3	0.5	0.5	300	0.79363	0.96312	0.94991
0.3	0.5	0.5	350	0.78608	0.96226	0.94921
0.3	0.5	0.5	450	0.78675	0.96213	0.94925
0.3	0.5	0.6	450	0.79243	0.9619	0.95096
0.3	0.5	0.7	50	0.79667	0.96147	0.95198
0.3	0.5	0.7	350	0.79955	0.96222	0.95236
0.3	0.5	0.8	150	0.80455	0.95758	0.95803
0.3	0.5	0.8	200	0.8077	0.95856	0.95788
0.3	0.5	0.9	50	0.80559	0.95332	0.96256
0.3	0.5	0.9	450	0.79867	0.9476	0.96331
0.3	0.6	0.1	150	0.79084	0.9628	0.94927
0.3	0.6	0.1	200	0.78937	0.9629	0.94935
0.3	0.6	0.2	200	0.78893	0.96252	0.94975
0.3	0.6	0.2	250	0.78721	0.96268	0.94882
0.3	0.6	0.2	250	0.7858	0.96261	0.94888
0.3	0.6	0.2	450	0.792	0.96304	0.9498
0.3	0.6	0.3	500	0.79294	0.96226	0.94988
0.3	0.6	0.4	250	0.78992	0.96224	0.94931
0.3	0.6	0.5	150	0.787	0.96199	0.94958
0.3	0.6	0.5	450	0.78825	0.96204	0.95003
0.3	0.6	0.5	500	0.79145	0.96168	0.94987
0.3	0.6	0.6	100	0.78929	0.96178	0.95045
0.3	0.6	0.6	150	0.79585	0.96243	0.95106
0.3	0.6	0.6	300	0.7912	0.96114	0.95152
0.3	0.6	0.7	200	0.79467	0.95967	0.95293
0.3	0.6	0.8	150	0.8052	0.95623	0.96051
0.3	0.6	0.8	250	0.8015	0.95528	0.95907
0.3	0.6	0.8	400	0.80647	0.95628	0.95957
0.3	0.6	0.9	100	0.78381	0.94105	0.96578
0.3	0.7	0.1	100	0.79142	0.96208	0.94979
0.3	0.7	0.1	350	0.78795	0.96244	0.9497
0.3	0.7	0.1	400	0.7863	0.96121	0.94984
0.3	0.7	0.2	350	0.78806	0.96156	0.94963
0.3	0.7	0.3	250	0.78921	0.96197	0.95025
0.3	0.7	0.3	350	0.78839	0.96242	0.95068
0.3	0.7	0.4	350	0.79356	0.96166	0.95099
0.3	0.7	0.4	350	0.7934	0.96149	0.95074
0.3	0.7	0.4	450	0.79198	0.96127	0.95089
0.3	0.7	0.6	150	0.79014	0.95978	0.9521
0.3	0.7	0.6	450	0.79235	0.95882	0.95281
0.3	0.7	0.6	450	0.78753	0.95886	0.95276
0.3	0.7	0.9	200	0.77164	0.9335	0.96806
0.3	0.7	0.9	300	0.77157	0.93466	0.96771
0.3	0.7	0.9	350	0.77368	0.93283	0.96854
0.3	0.7	0.9	400	0.77139	0.93304	0.96813
0.3	0.8	0.1	200	0.79111	0.96063	0.95166
0.3	0.8	0.1	300	0.78976	0.95983	0.95237
0.3	0.8	0.1	400	0.79329	0.96155	0.95218
0.3	0.8	0.1	500	0.79159	0.96068	0.952
0.3	0.8	0.2	350	0.79463	0.96061	0.95262
0.3	0.8	0.4	300	0.79164	0.95972	0.9534
0.3	0.8	0.4	350	0.78831	0.9587	0.95259
0.3	0.8	0.4	400	0.7937	0.96042	0.95272
0.3	0.8	0.5	200	0.79198	0.95904	0.95377
0.3	0.8	0.6	50	0.72169	0.95383	0.95442
0.3	0.8	0.6	350	0.79597	0.95511	0.9568
0.3	0.8	0.6	500	0.79044	0.95589	0.95479
0.3	0.8	0.7	450	0.77397	0.94648	0.95996
0.3	0.8	0.8	100	0.75413	0.93943	0.96528
0.3	0.8	0.8	350	0.79014	0.94463	0.96471
0.3	0.8	0.8	450	0.78895	0.93995	0.96806
0.3	0.8	0.9	250	0.722	0.92417	0.96951
0.3	0.8	0.9	400	0.73116	0.92489	0.97124
0.3	0.9	0.1	300	0.79454	0.95703	0.95629
0.3	0.9	0.1	450	0.79401	0.95585	0.95601
0.3	0.9	0.2	150	0.79283	0.95645	0.95722
0.3	0.9	0.2	200	0.78691	0.95505	0.95563
0.3	0.9	0.2	400	0.79388	0.95675	0.95625
0.3	0.9	0.3	50	0.69802	0.95169	0.95641
0.3	0.9	0.3	200	0.79135	0.95566	0.95705
0.3	0.9	0.3	500	0.79281	0.95603	0.95699
0.3	0.9	0.4	100	0.78316	0.95318	0.95789
0.3	0.9	0.5	450	0.78684	0.95205	0.95809
0.3	0.9	0.6	300	0.78356	0.94842	0.96001

γ_1	γ_2	θ	neighbors	ARI	completeness	homogeneity
0.3	0.9	0.6	300	0.78937	0.95048	0.95931
0.3	0.9	0.7	350	0.7648	0.94154	0.963
0.3	0.9	0.8	50	0.63621	0.93354	0.96719
0.3	0.9	0.8	200	0.76834	0.93627	0.96725
0.3	0.9	0.8	500	0.76527	0.93628	0.96713
0.3	0.9	0.9	500	0.67418	0.91493	0.97184
0.3	0.9	0.9	500	0.69365	0.91522	0.97156
0.3	0.9	0.9	500	0.67918	0.91579	0.97206
0.4	0.5	0.1	350	0.7955	0.95715	0.9574
0.4	0.5	0.1	450	0.79303	0.95673	0.95639
0.4	0.5	0.3	350	0.79528	0.95642	0.95662
0.4	0.5	0.4	350	0.79477	0.95612	0.9565
0.4	0.5	0.4	350	0.7933	0.95599	0.95578
0.4	0.5	0.5	50	0.72052	0.95364	0.95604
0.4	0.5	0.5	100	0.79317	0.9566	0.95657
0.4	0.5	0.5	200	0.794	0.95711	0.95659
0.4	0.5	0.5	250	0.79284	0.95694	0.95625
0.4	0.5	0.6	100	0.79431	0.95628	0.95697
0.4	0.5	0.6	150	0.79542	0.95619	0.95746
0.4	0.5	0.6	400	0.79837	0.95601	0.95784
0.4	0.5	0.6	450	0.79352	0.95539	0.95634
0.4	0.5	0.8	100	0.79864	0.9551	0.96022
0.4	0.5	0.8	300	0.79293	0.95406	0.95845
0.4	0.5	0.8	400	0.80192	0.95475	0.96011
0.4	0.6	0.2	50	0.75738	0.95509	0.95562
0.4	0.6	0.2	500	0.79623	0.95698	0.95687
0.4	0.6	0.2	500	0.79134	0.95592	0.95625
0.4	0.6	0.3	250	0.79381	0.95708	0.95707
0.4	0.6	0.3	350	0.79612	0.95647	0.95666
0.4	0.6	0.3	400	0.79602	0.95677	0.95722
0.4	0.6	0.3	500	0.79287	0.95614	0.95615
0.4	0.6	0.4	200	0.78745	0.95594	0.95546
0.4	0.6	0.4	500	0.79768	0.95652	0.95662
0.4	0.6	0.5	50	0.74409	0.95426	0.95765
0.4	0.6	0.5	150	0.79505	0.95713	0.95644
0.4	0.6	0.5	200	0.79519	0.95634	0.95704
0.4	0.6	0.6	300	0.79391	0.95618	0.95675
0.4	0.6	0.6	300	0.79482	0.95574	0.95807
0.4	0.6	0.8	300	0.79926	0.95301	0.96052
0.4	0.6	0.9	500	0.79626	0.94711	0.96645
0.4	0.7	0.1	100	0.78826	0.95592	0.95625
0.4	0.7	0.1	150	0.79559	0.95601	0.95723
0.4	0.7	0.1	250	0.79219	0.95621	0.95636
0.4	0.7	0.1	350	0.78933	0.95666	0.95581
0.4	0.7	0.1	500	0.79392	0.95534	0.95687
0.4	0.7	0.2	300	0.79328	0.95659	0.95675
0.4	0.7	0.2	400	0.79472	0.9559	0.95694
0.4	0.7	0.3	200	0.79175	0.95612	0.95713
0.4	0.7	0.6	300	0.79287	0.95513	0.95806
0.4	0.7	0.6	350	0.79694	0.95621	0.95801
0.4	0.7	0.6	450	0.7982	0.95629	0.95864
0.4	0.7	0.7	400	0.79197	0.95114	0.95991
0.4	0.7	0.8	200	0.79861	0.94898	0.96321
0.4	0.7	0.9	450	0.78272	0.93846	0.96944
0.4	0.8	0.2	100	0.78938	0.95532	0.95697
0.4	0.8	0.3	250	0.79276	0.95559	0.9568
0.4	0.8	0.3	400	0.79244	0.95551	0.95701
0.4	0.8	0.4	250	0.79156	0.95515	0.95791
0.4	0.8	0.4	250	0.79463	0.95564	0.95801
0.4	0.8	0.4	250	0.79396	0.95563	0.95691
0.4	0.8	0.5	300	0.79487	0.95561	0.95794
0.4	0.8	0.5	400	0.79282	0.95499	0.95804
0.4	0.8	0.5	500	0.79699	0.95611	0.95878
0.4	0.8	0.5	500	0.79557	0.95549	0.95866
0.4	0.8	0.5	500	0.79704	0.95544	0.9584
0.4	0.8	0.6	100	0.79232	0.95445	0.9598
0.4	0.8	0.7	150	0.79458	0.95164	0.96205
0.4	0.8	0.8	250	0.79069	0.94381	0.96534
0.4	0.8	0.8	350	0.78769	0.94284	0.96525
0.4	0.8	0.9	200	0.74737	0.93024	0.9714
0.4	0.8	0.9	300	0.77071	0.93312	0.97116
0.4	0.8	0.9	450	0.74066	0.92782	0.97059
0.4	0.9	0.1	300	0.79596	0.95478	0.95952
0.4	0.9	0.1	400	0.79472	0.95445	0.95894
0.4	0.9	0.2	250	0.79209	0.95379	0.95807
0.4	0.9	0.2	450	0.79008	0.95433	0.95843
0.4	0.9	0.3	200	0.79258	0.95408	0.95898
0.4	0.9	0.4	100	0.78723	0.95347	0.95885
0.4	0.9	0.5	50	0.6942	0.94779	0.96019
0.4	0.9	0.5	350	0.79148	0.95246	0.95945
0.4	0.9	0.6	100	0.77426	0.95024	0.96193
0.4	0.9	0.6	150	0.79062	0.95039	0.96131
0.4	0.9	0.6	350	0.79358	0.94956	0.9618
0.4	0.9	0.7	50	0.66984	0.94272	0.96361
0.4	0.9	0.7	100	0.77703	0.94601	0.96416
0.4	0.9	0.7	200	0.78088	0.94538	0.9655
0.4	0.9	0.7	400	0.78795	0.94885	0.96327
0.4	0.9	0.8	450	0.78821	0.94054	0.96809
0.5	0.6	0.1	100	0.79168	0.95267	0.96226
0.5	0.6	0.1	250	0.79465	0.95297	0.96141
0.5	0.6	0.1	500	0.79622	0.95318	0.96201
0.5	0.6	0.3	350	0.79393	0.95256	0.96167
0.5	0.6	0.4	350	0.79545	0.95241	0.96117
0.5	0.6	0.4	400	0.79323	0.95298	0.9611
0.5	0.6	0.4	400	0.79181	0.95267	0.96108
0.5	0.6	0.5	400	0.7927	0.95179	0.96086
0.5	0.6	0.6	200	0.79473	0.95253	0.96162
0.5	0.6	0.6	200	0.79556	0.952	0.96148
0.5	0.6	0.6	450	0.79297	0.95168	0.96096
0.5	0.6	0.6	450	0.79465	0.95268	0.96113
0.5	0.6	0.6	500	0.79483	0.95231	0.96133
0.5	0.6	0.7	150	0.79501	0.95143	0.96195
0.5	0.6	0.7	250	0.79933	0.9525	0.96233
0.5	0.6	0.7	350	0.79802	0.95215	0.96312
0.5	0.6	0.7	400	0.79753	0.95241	0.96233
0.5	0.6	0.8	150	0.80271	0.95169	0.96425
0.5	0.6	0.8	300	0.79783	0.95077	0.9637
0.5	0.6	0.8	350	0.79894	0.95059	0.96309
0.5	0.6	0.8	400	0.80187	0.95168	0.96382
0.5	0.6	0.9	200	0.7991	0.94637	0.96705
0.5	0.6	0.9	250	0.80339	0.94718	0.96764
0.5	0.7	0.2	350	0.79485	0.95274	0.96229
0.5	0.7	0.3	100	0.78685	0.95167	0.96112
0.5	0.7	0.3	100	0.78796	0.95163	0.96131
0.5	0.7	0.5	350	0.79068	0.95221	0.96053
0.5	0.7	0.5	350	0.79599	0.95233	0.96178
0.5	0.7	0.5	450	0.79602	0.95262	0.962
0.5	0.7	0.6	200	0.79623	0.95186	0.96186
0.5	0.7	0.8	50	0.71216	0.94631	0.9653
0.5	0.7	0.9	200	0.78926	0.94109	0.96877
0.5	0.8	0.1	150	0.79178	0.95131	0.96143
0.5	0.8	0.1	200	0.79327	0.95184	0.96146
0.5	0.8	0.1	300	0.79422	0.95224	0.96173
0.5	0.8	0.1	450	0.79426	0.95201	0.96218
0.5	0.8	0.2	350	0.79919	0.9523	0.96259
0.5	0.8	0.2	350	0.79713	0.95215	0.96233
0.5	0.8	0.2	400	0.79717	0.95213	0.9625
0.5	0.8	0.2	400	0.79161	0.95135	0.96143
0.5	0.8	0.2	500	0.79474	0.95186	0.96193
0.5	0.8	0.3	50	0.70765	0.94965	0.9628
0.5	0.8	0.3	300	0.79398	0.95201	0.96174
0.5	0.8	0.4	50	0.70813	0.94863	0.96229
0.5	0.8	0.4	300	0.79681	0.95238	0.96246
0.5	0.8	0.5	50	0.70983	0.94917	0.96254
0.5	0.8	0.6	100	0.79095	0.95167	0.96329
0.5	0.8	0.7	100	0.79327	0.94947	0.96406
0.5	0.8	0.7	500	0.79738	0.95019	0.96454
0.5	0.8	0.8	150	0.79136	0.94705	0.96601
0.5	0.8	0.8	300	0.79204	0.94663	0.96605
0.5	0.8	0.8	350	0.79376	0.94613	0.96632
0.5	0.8	0.8	400	0.79579	0.94673	0.96695
0.5	0.8	0.9	300	0.7606	0.93032	0.97134
0.5	0.9	0.1	50	0.70174	0.94818	0.96283
0.5	0.9	0.1	100	0.78756	0.95054	0.96237
0.5	0.9	0.2	500	0.79548	0.95154	0.96254
0.5	0.9	0.3	50	0.67637	0.94635	0.96385
0.5	0.9	0.3	500	0.79451	0.95113	0.96286
0.5	0.9	0.4	200	0.79511	0.95132	0.96236
0.5	0.9	0.4	500	0.79619	0.95123	0.96277
0.5	0.9	0.6	100	0.78586	0.95059	0.96322
0.5	0.9	0.6	250	0.79719	0.95013	0.96393
0.5	0.9	0.6	450	0.79901	0.95083	0.96444
0.5	0.9	0.8	500	0.78005	0.94025	0.96761
0.5	0.9	0.9	100	0.71575	0.92533	0.97274
0.5	0.9	0.9	350	0.75177	0.92872	0.97253
0.6	0.7	0.1	100	0.78957	0.94906	0.96516
0.6	0.7	0.1	450	0.78857	0.94866	0.96458
0.6	0.7	0.1	500	0.79027	0.94871	0.96394
0.6	0.7	0.2	100	0.78344	0.94883	0.96419
0.6	0.7	0.2	200	0.79003	0.94908	0.96415
0.6	0.7	0.2	400	0.79178	0.94924	0.96445
0.6	0.7	0.3	400	0.79065	0.94846	0.9646
0.6	0.7	0.4	150	0.7892	0.9482	0.96392
0.6	0.7	0.4	150	0.79251	0.94931	0.96462
0.6	0.7	0.4	250	0.78961	0.94851	0.96406
0.6	0.7	0.5	150	0.79193	0.94888	0.96435
0.6	0.7	0.5	250	0.79159	0.94853	0.96398
0.6	0.7	0.5	300	0.78898	0.94865	0.96397
0.6	0.7	0.7	50	0.69509	0.94535	0.96577
0.6	0.7	0.9	150	0.78477	0.93931	0.9699
0.6	0.8	0.1	50	0.69054	0.94507	0.96484
0.6	0.8	0.1	150	0.7925	0.94824	0.96432
0.6	0.8	0.1	500	0.79371	0.94871	0.96441
0.6	0.8	0.2	300	0.79528	0.94955	0.9658
0.6	0.8	0.2	350	0.78918	0.94863	0.96372
0.6	0.8	0.3	450	0.79243	0.9493	0.96465
0.6	0.8	0.5	100	0.78754	0.94814	0.96533
0.6	0.8	0.5	150	0.79042	0.94806	0.96436
0.6	0.8	0.5	450	0.79132	0.94915	0.96438
0.6	0.8	0.6	50	0.69328	0.94512	0.96525
0.6	0.8	0.6	200	0.79314	0.94929	0.96533
0.6	0.8	0.7	150	0.79278	0.94814	0.96567
0.6	0.8	0.7	200	0.79185	0.94764	0.96586
0.6	0.8	0.8	50	0.6782	0.93946	0.96882
0.6	0.8	0.8	200	0.79278	0.9454	0.96751
0.6	0.8	0.8	400	0.78917	0.94528	0.96762
0.6	0.8	0.8	500	0.7929	0.94597	0.96767
0.6	0.8	0.9	450	0.78138	0.93731	0.97122

γ_1	γ_2	θ	neighbors	ARI	completeness	homogeneity
0.6	0.9	0.1	300	0.79092	0.9492	0.96461
0.6	0.9	0.1	350	0.7918	0.94897	0.96484
0.6	0.9	0.1	350	0.79088	0.94838	0.96491
0.6	0.9	0.2	100	0.78219	0.94865	0.9645
0.6	0.9	0.2	150	0.79166	0.94932	0.96446
0.6	0.9	0.2	350	0.78917	0.94805	0.96416
0.6	0.9	0.3	100	0.78607	0.94869	0.96522
0.6	0.9	0.4	250	0.79007	0.94881	0.96438
0.6	0.9	0.4	350	0.7902	0.94848	0.96455
0.6	0.9	0.4	400	0.79153	0.94855	0.96451
0.6	0.9	0.5	100	0.78681	0.94808	0.96542
0.6	0.9	0.5	300	0.79173	0.94863	0.96515
0.6	0.9	0.6	250	0.79631	0.94805	0.96551
0.6	0.9	0.6	500	0.7888	0.94718	0.96504
0.6	0.9	0.7	100	0.78147	0.94674	0.96696
0.6	0.9	0.7	250	0.79628	0.94751	0.96732
0.6	0.9	0.7	250	0.793	0.94752	0.96711
0.6	0.9	0.7	300	0.7926	0.9464	0.96716
0.6	0.9	0.7	400	0.79065	0.94759	0.96679
0.6	0.9	0.8	50	0.65224	0.93543	0.96926
0.6	0.9	0.8	350	0.78462	0.94277	0.96789
0.6	0.9	0.8	350	0.78835	0.94239	0.96842
0.6	0.9	0.8	500	0.78347	0.94223	0.9688
0.7	0.8	0.2	200	0.78111	0.94394	0.96692
0.7	0.8	0.3	100	0.77168	0.94341	0.96756
0.7	0.8	0.3	400	0.78295	0.94489	0.9677
0.7	0.8	0.3	400	0.78642	0.94425	0.96775
0.7	0.8	0.3	400	0.78383	0.94478	0.96786
0.7	0.8	0.4	300	0.78117	0.94462	0.96724
0.7	0.8	0.4	450	0.78256	0.94436	0.96768
0.7	0.8	0.5	100	0.77552	0.94451	0.96785
0.7	0.8	0.6	300	0.78714	0.94491	0.96763
0.7	0.8	0.7	100	0.77854	0.94417	0.96848
0.7	0.8	0.8	100	0.77336	0.94182	0.9693
0.7	0.8	0.8	200	0.78285	0.94181	0.96892
0.7	0.8	0.8	300	0.78645	0.94181	0.96939
0.7	0.8	0.8	400	0.78122	0.94157	0.9686
0.7	0.8	0.8	450	0.78645	0.94203	0.96945
0.7	0.8	0.9	450	0.77988	0.9357	0.97021
0.7	0.9	0.1	150	0.78309	0.94426	0.96776
0.7	0.9	0.2	250	0.78468	0.94408	0.96816
0.7	0.9	0.3	200	0.78541	0.94455	0.96778
0.7	0.9	0.3	200	0.7842	0.94443	0.9679
0.7	0.9	0.3	300	0.78642	0.94501	0.96817
0.7	0.9	0.3	400	0.7812	0.94431	0.96747
0.7	0.9	0.4	150	0.78454	0.94407	0.96779
0.7	0.9	0.4	300	0.78446	0.9443	0.96802
0.7	0.9	0.4	350	0.78362	0.94458	0.96779
0.7	0.9	0.5	450	0.78823	0.9445	0.96855
0.7	0.9	0.7	300	0.78331	0.94369	0.9685
0.7	0.9	0.7	350	0.78492	0.94368	0.96891
0.7	0.9	0.7	450	0.78416	0.94295	0.96869
0.8	0.9	0.1	50	0.64664	0.93643	0.96898
0.8	0.9	0.1	300	0.78088	0.94238	0.96954
0.8	0.9	0.2	150	0.78195	0.94184	0.9694
0.8	0.9	0.2	300	0.77617	0.94106	0.9691
0.8	0.9	0.2	300	0.77677	0.94134	0.96864
0.8	0.9	0.2	300	0.78101	0.94133	0.9694
0.8	0.9	0.2	450	0.77695	0.94165	0.96897
0.8	0.9	0.2	500	0.77916	0.94175	0.96921
0.8	0.9	0.2	500	0.7766	0.94144	0.96918
0.8	0.9	0.3	50	0.65341	0.93688	0.96903
0.8	0.9	0.3	300	0.77503	0.9405	0.96882
0.8	0.9	0.4	150	0.78015	0.94188	0.96992
0.8	0.9	0.4	400	0.77781	0.94174	0.96956
0.8	0.9	0.5	50	0.6561	0.9371	0.96919
0.8	0.9	0.7	50	0.65733	0.93682	0.96981
0.8	0.9	0.7	100	0.76477	0.9402	0.97023
0.8	0.9	0.7	200	0.77934	0.94169	0.96983
0.8	0.9	0.8	250	0.77518	0.93906	0.97052
0.8	0.9	0.8	400	0.77711	0.93944	0.97112
0.8	0.9	0.9	50	0.62967	0.92692	0.97347
0.8	0.9	0.9	250	0.77018	0.93488	0.97252
0.8	0.9	0.9	450	0.76437	0.93351	0.97159

Table S3.3: Hyperameter optimization of the parameters γ_1, γ_2, θ, `max_neighbors`.

Chapter 4

Module painting

Abstract

With the rapid expansion of microbiome sequencing, hundreds of bacteriophage population samples are now available in public databases. Pervasive recombinations within and across populations reveal a large share of homologous recombinations, and recombinations between genes that share little similarity but perform the same biological function, called modular recombinations. We adapted the idea of chromosome painting to this particular problem, and use recombinations as a feature to describe bacteriophage populations. We propose module-painter, a tool that can quickly quantify and describe the exchange of modules between populations. We show that module-painter can identify recombinations in both simulated and real datasets and identify meaningful clusters based on this feature. The code is publicly available at: `https://github.com/Puumanamana/module-painter.git`

4.1 Introduction

Unlike other organisms, viruses lack the necessary material to survive on their own. Thus, they need to infect other living organisms and use their machinery to replicate. Viruses

Work in progress. Collaboration with Guylaine Poisson, Anne Bergeron and Mahdi Belcaid

infecting bacteria, or bacteriophages, are the most common viruses on earth [1], which makes them hard to avoid. Bacteria have naturally adapted to fight viral infection through various mechanisms (such as the CRISPR-Cas system [2, 3]). In turn, viruses also evolve to bypass those mechanisms. However, viruses evolve at a much faster pace than other organisms, partly because of the process of recombination [4, 5], which consists of exchanging part of their DNA with other members of their species to produce two complementary children (see figure 4.1). This high sequence variability can challenge DNA recognition mechanisms in bacteria.

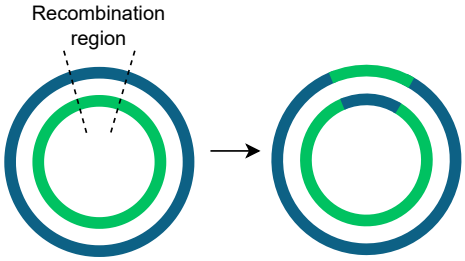


Figure 4.1: Recombination mechanism in phages: (left) Two parent phages (blue and green) recombine to produce two children (right).

DNA recombination is not unique to viruses and is observed in most organisms. For diploid organisms (with pairs of chromosomes), recombination can happen during reproduction, since the progeny receives one chromosome from each parent. Other mechanisms, such as crossing-over, also play an important role in increasing the genetic diversity in a population. Since the same recombination rarely occurs twice, it produces descendants with similar genomic structure compared to other individuals of the same generation. This leads to small variations in specific DNA sequences, called variants, to be propagated within the population when they provide some kind of survival benefits. When a region of DNA is present in a population with two or more variants, it is called polymorphic. By studying the distribution of polymorphisms, we can hope to unravel a population's history of recombination. The most popular idea consists in decomposing (or "painting") the chromosomes of an individual as a set of acquired DNA segments from its ancestors, and is now known as in silico "chromosomal painting". In [6], the authors focus on humans and

aim at identifying the most recent "donor" for multiple DNA segments in an individual's DNA. The probability that a segment was received from a given donor is modeled using a Hidden Markov Model, and the most likely scenario leading to the observed individual is computed. Using this model, they build a "co-ancestry" matrix showing each individual's relatedness with the others across all segments of DNA considered. The authors further expand their work and use this co-ancestry matrix as a baseline for subsequent analyses to cluster the individuals into subpopulations and infer their history of recombination. Although this approach has been used in [7] for highly conserved phage genes, it is in general not well suited for viruses because of their high mosaicism (i.e. their genome is an assemblage of DNA modules exchanged with their environment) [8]. It is indeed common to observe DNA segments in phages that serve a similar function, but for which the DNA sequences are drastically different. In comparison, the genetic variations in humans are most commonly observed at individual DNA positions. High dissimilarity between individuals complicates genome comparisons and the identification of polymorphism. Furthermore, the propagation of mutation in a population is believed to be the driving force of evolution for eukaryotes [9, 10], while in the case of phages, previous research seems to indicate that recombinations play a major role in their adaptation [10]. The recombination process of viruses is vastly different from most organisms and is still actively researched. A popular paradigm for recombination was proposed by Botstein [4], who hypothesized in his modular theory of phage recombination that viral genomes recombine by exchanging DNA segments from variable regions (called modules) that are flanked by conserved regions. Based on this theory, the authors in [11, 12, 13] align viral genomes and successfully identify viral modules and their variants. These variants are further used to analyze a population of Staphylococcus aureus and phages found in dairy factories in order to reconstruct the recombination history. Based on the observed recombination in the population, they were able to identify a minimal set of viruses that can generate the whole population using viral recombination operations. Despite their success in recovering the recombination history, these approaches are ill-suited for environmental viral metagenomes, since they are often

very fragmented and therefore do not fit in the authors' mathematical model (which requires a complete and ordered set of modules).

Here, we aim to reproduce the approach from this earlier work while reducing the assumptions about genome completeness. Our finding are implemented in module-painter, a tool for identifying the traces of a parent viral population on a target population. By identifying recombination patterns, we are able to cluster the viruses into subpopulations based on their shared history. We tested our method on both simulated datasets and real metagenomic data and showed that module-painter identifies recombinations between phages and produces meaningful phage clusters.

4.2 Methods

4.2.1 Definitions

We consider two phage populations P and Q with circular genomes, where the phages in P are referred to as the parents and the phages in Q the children.

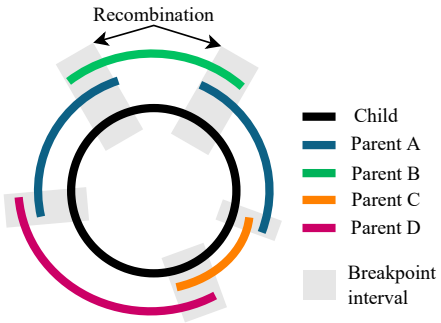


Figure 4.2: Example of coverage: Coverage of a phage (black) by 4 parents, A (blue), B (green), C (orange) and D (pink). There are five breakpoints (shaded regions) that include parents A/B (twice), A/C, A/D and C/D. There is only one recombination (A/B + A/B).

The coverage of a phage q in Q is a set of overlapping DNA segments from phages in P aligning on q. When two successive segments overlap, they define a breakpoint interval (or breakpoint for simplicity). A breakpoint is characterized by the start and end positions of the overlap, and the IDs of the corresponding phages. A recombination may occur when

a pair of parents are involved in two breakpoints with different (start, end) positions (see example in figure 4.2).

4.2.2 Module-painter: main steps and parameters

The module-painter algorithm is composed of two main steps. First, we determine a minimal set of segments from the parent population covering each child's genome. Second, we identify all recombinations shared by multiple children in order to cluster them into subpopulations. These steps are explained in further details below.

Coverage computation

The coverage computation is done for each child separately, therefore in this section we only consider one child sequence (and many possible parents).

Initial coverage: We perform an initial alignment with Minimap2 [14] to identify high confidence alignments with little to no gaps (parameters: `-r 100 -z 50 -U 100 --no-long-join -c -P`) and discard hits with less than 90% identity. The relative orientation of a parent and a child's genome can differ, thus we only keep hits from a single orientation, chosen as the orientation of the hit with the most base pair matches.

Interval synchronization (figure 4.3A): To make the coverage computation more robust to small variations in alignment accuracy, we start by "synchronizing" segments boundaries. It consists in adjusting the boundary of intervals with similar start/end positions, and is done through agglomerative clustering of all the intervals' start positions. Intervals in the same cluster are extended to the cluster's minimum on the left. We do the same for the end position of each interval (but this time, we extend intervals to the cluster's maximum on the right). This step yields a more homogeneous coverage of the children population, with differences in the boundaries of the intervals larger than 20 bp.

Gap filling (figure 4.3B-C): At this stage, it is common to observe regions on the child genome that are not covered by any parent. This step aims at filling those gaps in coverage. First, we reduce the fragmentation of each parent by refining their alignment. We consider

all consecutive intervals, and fuse them together either if they are close (distance <100 bp) or if the region that separates them in the parent and the child is more than 90% highly similar with a Needleman-Wunsch alignment (figure 4.3B). Then, once the parents' segments are maximally extended, we investigate the remaining holes in coverage. If the missing portion is small (distance < 100 bp) we extend the segments flanking this region to touch, and otherwise we add a putative parent in the parent set, whose sequence is the missing portion on the child's genome ("NA" in figure 4.3C).

Coverage simplification (figure 4.3D-E): In order to focus on the more meaningful alignments, we remove any interval embedded in a larger one (figure 4.3D). We further simplify the coverage using the algorithm in [15] to find the minimal set of arcs covering each child (figure 4.3E).

Identification of shared missing segments

The same missing segment can occur in multiple children, and identifying them can help us detect additional recombinations involving this shared putative missing parent. Here, we perform an all-vs-all global alignment (using minimap2 [14]) between all missing segments and build a homology graph where the vertices are missing segments and an edge links two segments with >90% identity (defined as #matches over the length of the smallest segment). In this graph, matching segments correspond to the connected components and will be considered as the same putative missing parents (see figure 4.4).

Identification of shared recombinations

If we assume that two phages can recombine with each other multiple times at different positions, then a recombination involving the same parents in multiple children does not necessarily mean it is the same recombination. In order to ascertain whether a recombination was inherited by two children, we need to check whether the DNA sequence of the breakpoint intervals in the recombination is the same for both. To this end, we use a similar approach as before by doing an all-vs-all alignment and building a homology

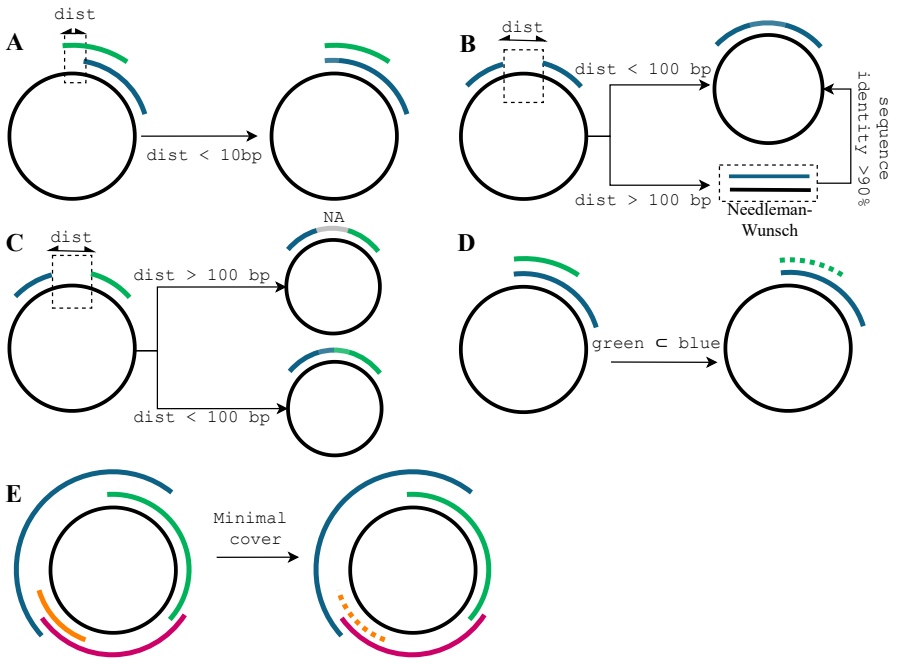


Figure 4.3: Coverage computation steps in module-painter: (A) Since the start position of both parents is close (<10bp), we extend the blue parent to match the start of the green. (B) There is a gap between two consecutive segments of the same parent. We close it if the distance is short or if the exact alignment of the unaligned region with the Needleman-Wunsch algorithm yields a high similarity. (C) Gap in coverage between the blue and the green segments. If the distance between them is small (default is shorter than 100bp), we extend both parents to touch. Otherwise, we add a putative parent (gray) to fill the hole. (D) One parent (green) is embedded in the other (blue), therefore we remove the smaller one. (E) The coverage contains 4 parents, with no embedding relationship. We can still simplify the coverage by removing the orange parent (using the circle-cover algorithm in [15]).

graph where vertices are breakpoints and edges link vertices with >90% identity with a length difference of at most 50%. Identical breakpoints are the connected components in this graph. We can then detect shared recombination whenever two shared breakpoints with the same parents are identified.

Parent selection

Multiple parents can cover the exact same region on a child genome if they all have the same variant. This complicates the identification of relatedness between children, as we

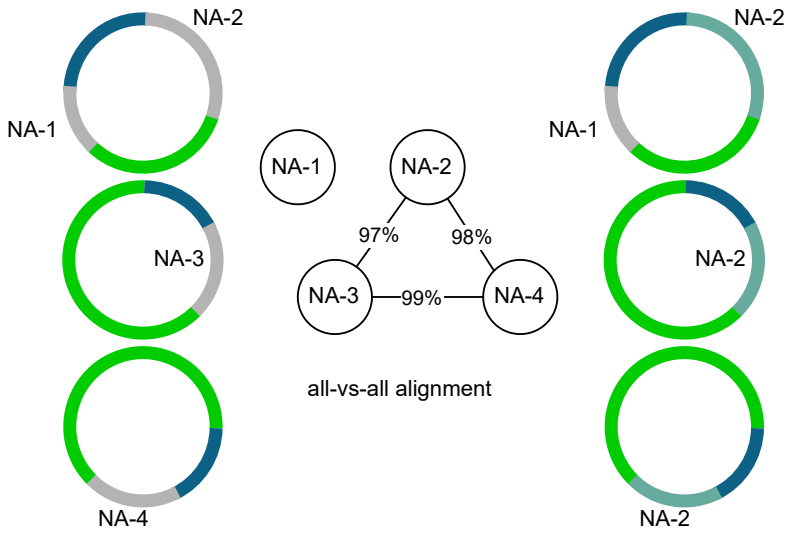
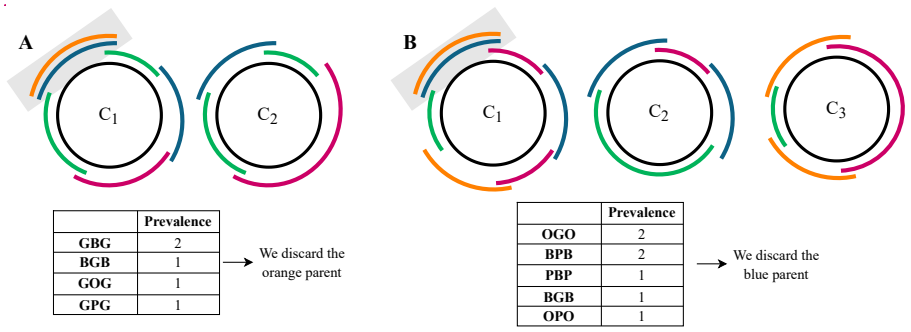


Figure 4.4: Missing fragments matching: We start with 3 children with various degrees of missing data. We align each missing fragment (labeled as NA) against all the others and build a homology graph, where the vertices are the missing fragments and edges link fragments with >=90% identity. Finally, we give the same label to the fragments belonging to the same connected components.

do not know which parent should be chosen to define the breakpoint. For example, in figure 4.5A, if we chose the blue parent as the ancestor of phage C1, then we define a recombination between the blue and green parents, whereas if we choose the orange parent, we have a recombination between the green and orange parents. We solve this issue by selecting the parents that maximize the number of shared recombinations between children. This is however a very complex problem since for p undetermined regions, each of them with $n_1, ..., n_p$ possible parents, we have $n_1 \times ... \times n_p$ combinations of parents. Thus, we choose to adopt a greedy approach (figure 4.5). We start by enumerating all possible recombinations and rank them by prevalence, defined here as the number of children in which the recombination occurs. For each recombination from best to worst, we discard alternative parents that occur at the same place (see figure 4.5A). It is however fairly common to have recombinations with the same prevalence, in which case we need to break ties. First, we prioritize recombinations for which breakpoints cannot be reused. For example, if a pair of parents define three breakpoints bk1, bk2, bk3, then we can only define one recombination

with two of the three breakpoints. Thus, if the selection of a parent leads to erasing one of them, we can still define a recombination with the remaining two. This is depicted on figure 4.5B, where there are three breakpoints involving the blue and pink parents. In this case, if we choose to pick the orange parent, we still have a recombination between pink and blue. If this criteria still leads to a tie, we pick the recombination involving the most common parents overall, and pick at random to break any remaining ties.



	Prevalence
GBG	2
BGB	1
GOG	1
GPG	1

	Prevalence
OGO	2
BPB	2
PBP	1
BGB	1
OPO	1

Figure 4.5: Parent selection: Shaded areas are segments where multiple parents are covering the same region. Parents are referred to with the initial of their color (e.g. P for pink). (A) The recombination GBG appears on both C1 and C2 whereas GOG appears only in C1. Therefore, we discard the orange parent. (B) Both OGO and BPB appear on two children (C1+C3 and C1+C2 respectively). However, if we pick BPB, we erase the OGO recombination whereas if we choose OGO, we still have the PBP recombination. Therefore, we discard the blue parent.

If multiple parents cover a segment not involved in any recombination, we also need to make a decision. We take a similar approach and focus on breakpoint prevalence: we keep the parent for which the breakpoint occurs in the most children, and break ties by selecting the most common parent pair, ultimately selecting at random for the remaining ties.

Population clustering

We construct a co-occurrence graph where vertices are children and edges link vertices with shared recombinations. We identify clusters using the Leiden community detection algorithm [16]. When multiple overlapping recombinations occur on the same phage, some breakpoints (and thus recombinations) can be erased, therefore erasing edges in the graph. Instead, we can hope to identify more links between phages by simply looking at shared

breakpoints instead of recombinations. Therefore, we propose two clustering options, based on either recombination or breakpoint.

Rotating the parent population

In some datasets, there is no natural choice for the parent population —any sample could be chosen to be the parent. The module-painter algorithm can still be used in this case to cluster phages according to recombination patterns. For a dataset with n samples, we use a "leave-one-out" strategy, where we iteratively select each sample to be the parent population and the rest to be the children. Each iteration identifies recombinations shared between phages in n-1 samples. We use the recombinations from all n runs to cluster the phages as described in the previous paragraph.

4.2.3 Clustering metrics

To evaluate the clustering accuracy, we used the Adjusted Rand Index (ARI), the homogeneity and the completeness. These measures are defined as follows: The ARI [17] is a measure based on the relative cluster assignment of any pair of samples. The Rand Index is the proportion of phage pairs that are correctly clustered together or correctly not clustered together among all possible pairs. The ARI is a modified version of the Rand Index that accounts for chance. The homogeneity measures whether the predicted clusters are "pure", or in other words, if they only group phages from the same population. It is based on the conditional entropy of the subpopulations given the predicted clusters, $H(S \mid K)$, as follows:

$$\text{Homogeneity} = 1 - \frac{H(S \mid K)}{H(S)}$$

The completeness is complementary to the homogeneity, and measures whether phages from the same population are split apart. It is based on the conditional entropy of the predicted

clusters given the subpopulations, $H(K \mid S)$, as follows:

$$\text{Completeness} = 1 - \frac{H(K \mid S)}{H(K)}$$

4.2.4 Data collection

Simulation

Our simulation relies on a modular recombination model. Each simulation can be split into 3 phases. First, we generate a genetic reservoir by defining p ordered modules, each with two or more variants. The number of variants per module is uniformly sampled in {5,...,10}, and the size of each variant (in bp) is uniformly sampled in {200,...,500}. Second, we generate a parent phage by selecting a variant at random for each of the p modules. We repeat this process multiple times to generate a parent population. In order to enforce a certain population structure, we divide the parent population into K clusters, each with a size sampled from a Poisson distribution of mean 5. Third, we generate the children population within each cluster by randomly recombining parents. To simulate a recombination, we pick two parents at random and one or more consecutive modules to exchange. We produce two children with each parents' variant of the chosen module exchanged (as in figure 4.1). In general, only one of the children benefits from the exchange, and we therefore discard one of them at random. For a given subpopulation, the total number of recombination is the product of the recombination rate, the population size, and the number of generations. In our simulations, we use a recombination rate of 20% and vary the number of generations. Since phages in a population recombine randomly, the true clusters do not necessarily correspond to the K initial populations, but are further divided into subpopulations. In these subpopulations, any two phages share a common ancestor.

Dairy bacteriophages

The authors in [18, 13] sampled phages around the world and performed a hierarchical clustering of their genomic features to ascertain whether the phages cluster according to

where they come from. Their attempts were overall successful except for 4 Dutch dairy factories that we use in this paper to determine if recombination patterns are more adapted in identifying each factory. We collected the phages using the accession provided in [18] and divided the sequences between the 4 factories.

Human gut metagenome dataset

The dataset from [19] contains the metagenomes of 24 individuals (with 6 controls and 18 exposed) before, 7 days after, and 3 months after administering an antibiotic. We collected assemblies provided from the authors on NCBI (bioproject `PRJEB8094`) and we kept contigs larger than 20kb, while filtering for viral contigs using DeepVirFinder [20].

4.3 Results

4.3.1 Module-painter reconstructs subpopulations in a simulated dataset

We first validate our method on simulated datasets with known population structure. The simulation process is explained in the Methods section. We generated a total of 600 datasets, with 50 replicates for each combination of the number of clusters (2, 5 or 10) and number of generations (2, 4, 6, or 8). We evaluated the clustering accuracy based on three metrics, the Adjusted Rand Index (ARI), the completeness and the homogeneity (see figure 4.6). The completeness measures true positives, or events where phages were correctly grouped together, while homogeneity measures true negatives, or events where phages were correctly assigned to separate subpopulations. ARI is a more general metric that captures both types of events (see Methods for more details). We can see that the homogeneity is very high (>0.99) across all simulation settings, meaning that very few clusters group phages from different populations. Both the completeness and ARI are negatively correlated with the number of generations, which tends to make the population structure more complex. The completeness is relatively high, going from more than 0.9 for 2 generations to 0.5-0.6 for 6 and 8 generations. The ARI follows a similar trend across all settings, starting at 0.5

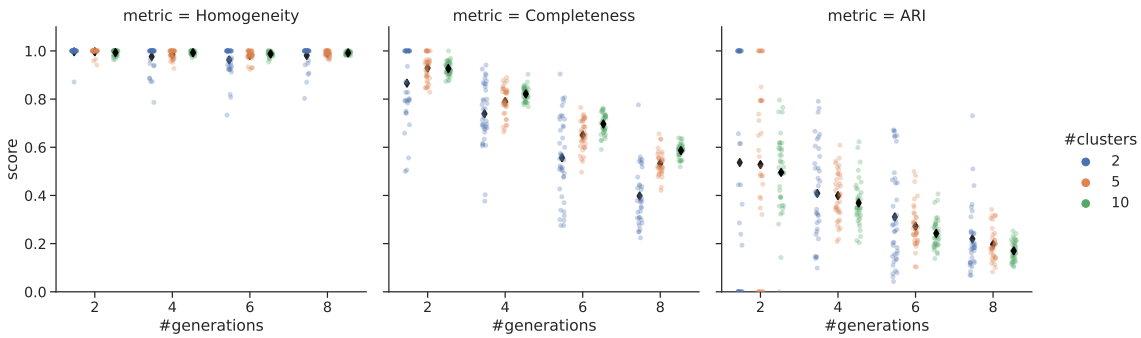


Figure 4.6: Module-painter on simulated data: For all figures, the x-axis identifies the number of generations (2, 4, 6 or 8), which is subdivided based on the number of clusters (2, 5 or 10). (A) Clustering accuracy on simulated datasets. Each column is a different metric (left: Homogeneity, middle: Completeness, right: ARI) and each point corresponds to a dataset. Colors correspond to the number of clusters (blue: 2, orange: 5, green: 10). The y-axis measures the score for each given metric, and the black diamonds symbols represent the median of the metric in each category.

for 2 generations to 0.2 for 8 generations. Almost all the identified recombinations link phages from the same population (data not shown) which corroborates our observations with the homogeneity metric. The number of identified recombinations also grows with the number of generations, as expected. Thus, although module-painter does not identify the entire populations, it can identify many recombinations and use them to define robust subpopulations.

4.3.2 Module-painter identifies recombinations in complete genomes

We evaluate the performance of module-painter on phages from four Dutch dairy factories. In their paper [18], the authors were not able to separate phages of each factory by looking solely at the phages' genome organization. We show here that recombination patterns can help identify similarities between phages. For this experiment, there is no clear choice of who should be selected as the parent population. Therefore, we use a "leave-one-out" strategy (see Methods) by iteratively selecting one factory as the parent population and the others to be the children.

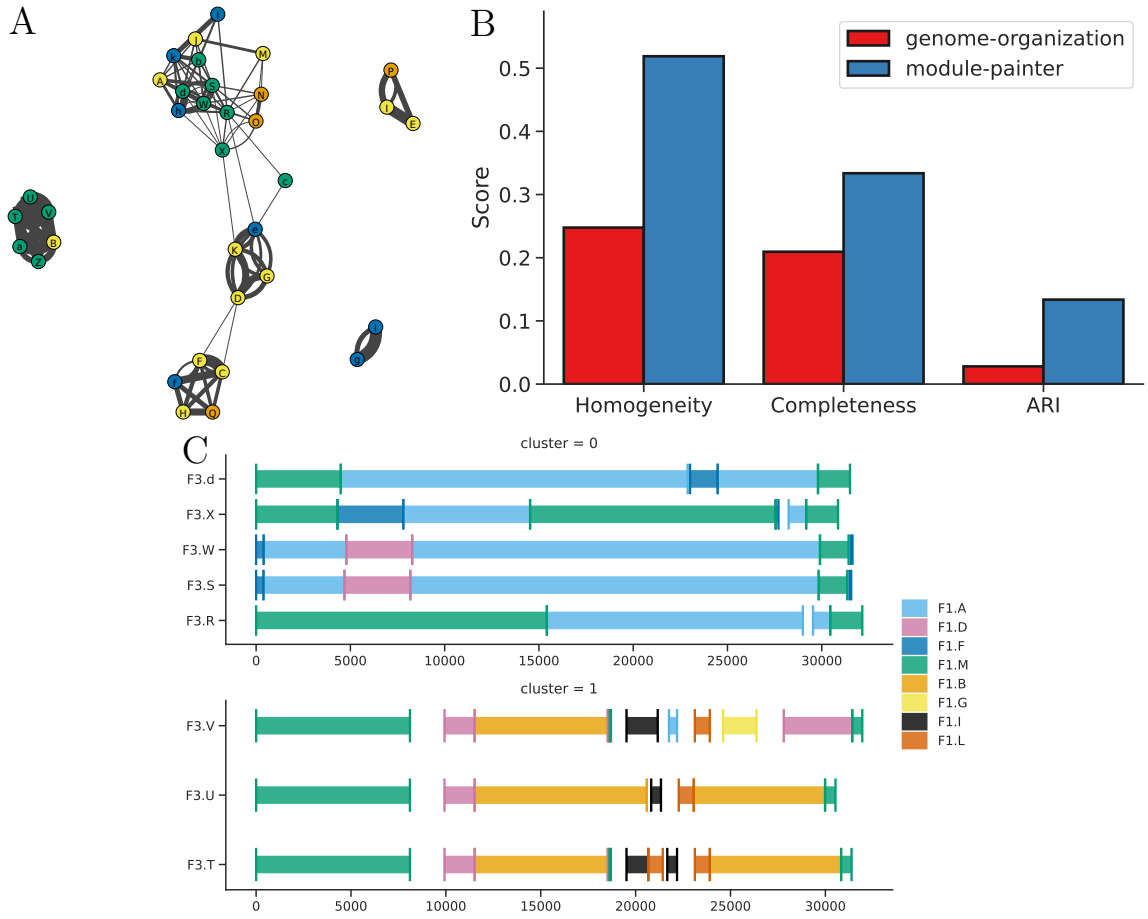


Figure 4.7: Clustering of dairy phages: (A) Recombination graph identified by module-painter. Vertices represent phages, and edges are recombination between them. Colors identify different factories (blue: 1, orange: 2, yellow: 3, green: 4). (B) Clustering accuracy on dairy factories dataset. The x-axis identifies the clustering metric and the clustering method used (also visible with the coloring, red for genome organization and blue for module-painter), while the y-axis displays the score of each metric. (C) Coverage of phages in factory 3 by phages in factory 1. Each subplot represents a different subpopulation, and colors identify different parents. The x-axis is the position along the child genome, while the y-axis separates multiple children in the same cluster.

We identified a total of 65 recombinations, 15% of which (10) are shared by multiple phages. We used the shared recombinations to build a graph (see figure 4.7A) that we cluster as described in the Methods. We compare the accuracy of the clusters of phages with the ones found with genome organization features in [18]. In order to emphasize recombinations

as a clustering feature, we exclude any phage for which no recombinations were found. Figure 4.7B shows that module-painter is more accurate than the clusters based on genome organization: recombinations lead to less false positive groupings (homogeneity value of 0.51 for module-painter versus 0.26 for genome-organization) and more true positives (completeness of 0.33 for module-painter vs 0.23 for genome-organization). The ARI is relatively low, still in favor of module-painter (0.13 versus 0.033).

Next, we show that module-painter can identify subpopulations within phages from the same factory. Figure 4.7C shows 8 phages from factory 3 divided into 2 clusters when studied through the lens of factory 1. For example, phages d, X, W, S and R are put in the same cluster because they share breakpoints involving A, M and F, while V, U and T are grouped because they share breakpoints involving D and B. Overall, we show that recombinations are a powerful feature for studying the interactions between phages in a population, and be an interesting alternative to more commonly used methods.

4.3.3 Module painter identifies recombinations in whole genome sequencing experiments

Finally, we use a metagenome dataset in which 24 patients (8 controls and 16 exposed) are followed at three different time points (before antibiotic treatment, 7 days after, 90 days after). For each patient, we follow the evolution of the phage population at each time point. First, we use module-painter to track phage recombinations over time for each patient separately. We pick t_0 (i.e. before treatment) as the parent population, and we look for recombinations shared by phages at t_7 and t_{90}. As shown in figure 4.8A,B, we found two examples for patients 13 and 21 of phages identified at t_7 and t_{90} who share the same recombination. Used in concordance with abundance data, this could give us insights about the adaptation of phages when they are subject to changes in their environment.

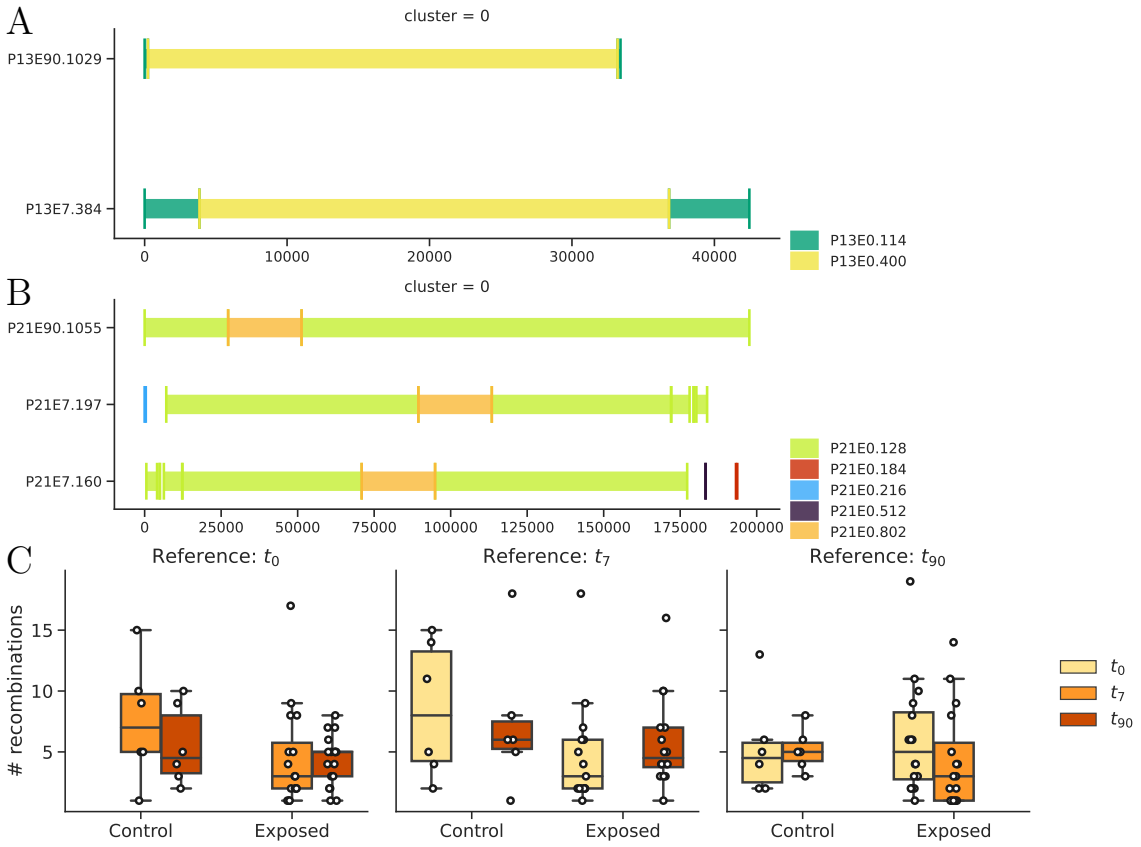


Figure 4.8: Recombination analysis on WGS dataset: (A,B) Coverage examples for two patients. The x-axis represents the position on the genome, while the y-axis separates multiple children. (A) Patient 13: Shared recombination between the green and yellow parents for two phages at t_7 and t_{90}. (B) Patient 21: Shared recombination between the yellow and orange parents for three phages, two at t_7 and one at t_{90}. (C) Distribution of the number of recombinations for each patient's metagenome. Each facet shows the result when the parent population is set to a different time point. The x-axis represents the treatment (Control or Exposed), and is further subdivided into each remaining time point.

Then, we investigate whether the antibiotic treatment has an effect on recombination frequency. Similarly, we run module-painter on each patient separately but this time we rotate the parent population between the three timepoints to identify recombinations. Overall, module-painter is able to detect a total of 850 recombinations in all 15,034 phages. The distribution of the number of recombination is displayed on figure 4.8C. The number of recombination follows an inverse trend when we compare the patients who received the antibiotic and the ones who did not: For example, when the reference is t_0, the median

number of recombinations decreases for the controls (8.5 to 5) and increases for the exposed patients (3 to 5), while when the reference is t_{90}, the controls increase in recombination (from 6 to 7.5) and decrease for the exposed (from 6 to 3). Thus, through the lens of recombination, module-painter can provide insights on the evolution of communities.

4.4 Discussion and conclusion

Module-painter provides a fresh perspective on phage population analysis by identifying exchanges of functional modules within a population. To our knowledge, this is the first phage clustering method based on recombinations. We show that our approach is able to leverage this information both on simulated data and real data to unravel the kinship between phages. We simulated 600 datasets with varying number of clusters and generations to evaluate module-painter's ability to recover the underlying population structure. Module-painter was able to identify many links between phages and to untangle the simulated population structure to some degree, with little to no false positives. We included two real-world datasets to validate our approach. The first is a high-quality dataset made of phages with known population structure. We showed that module-painter recovered more robust clusters than the original authors, who relied solely on genome organization features. Furthermore, the power of our approach also lies in its resolution, since we were able to recover individual subpopulations from the population in each factory. Finally, we used a metagenomic dataset from a gut microbiome experiment to showcase module-painter's ability to detect recombinations between phages over the course of multiple weeks. A majority of population structure analysis methods rely on multiple sequence alignments to build a recombination model, which is computationally intensive. Instead, through the use of fast alignment tools (e.g. minimap2) and by making parsimony assumptions, module-painter is able to identify recombinations in hundreds of contigs while remaining computationally tractable. However, its speed comes at the cost of a lower sensitivity compared to other tools. Thus, we recommend module-painter as an exploratory tool for

detecting recombinations in metagenomic datasets. The process of viral recombination is still poorly understood, and we hope that module-painter can provide more real-life examples of those processes in metagenomic data.

References

[1] A. G. Cobián Güemes, M. Youle, V. A. Cantú, B. Felts, J. Nulton, and F. Rohwer, "Viruses as winners in the game of life," *Annual Review of Virology*, vol. 3, pp. 197–214, 2016.

[2] F. J. Mojica, C. Díez-Villaseñor, J. García-Martínez, E. Soria, *et al.*, "Intervening sequences of regularly spaced prokaryotic repeats derive from foreign genetic elements," *Journal of molecular evolution*, vol. 60, no. 2, pp. 174–182, 2005.

[3] A. F. Andersson and J. F. Banfield, "Virus population dynamics and acquired virus resistance in natural microbial communities," *Science*, vol. 320, no. 5879, pp. 1047–1050, 2008.

[4] D. Botstein, "A theory of modular evolution for bacteriophages," *Annals of the New York Academy of Sciences*, vol. 354, no. 1, pp. 484–491, 1980.

[5] A. A. Hossain, J. McGinn, A. J. Meeske, J. W. Modell, and L. A. Marraffini, "Viral recombination systems limit crispr-cas targeting through the generation of escape mutations," *Cell Host & Microbe*, 2021.

[6] D. J. Lawson, G. Hellenthal, S. Myers, and D. Falush, "Inference of population structure using dense haplotype data," *PLoS genetics*, vol. 8, no. 1, p. e1002453, 2012.

[7] K. Yahara, P. Lehours, and F. F. Vale, "Analysis of genetic recombination and the pan-genome of a highly recombinogenic bacteriophage species," *Microbial genomics*, vol. 5, no. 8, 2019.

[8] M. L. Pedulla, M. E. Ford, J. M. Houtz, T. Karthikeyan, C. Wadsworth, J. A. Lewis, D. Jacobs-Sera, J. Falbo, J. Gross, N. R. Pannunzio, *et al.*, "Origins of highly mosaic mycobacteriophage genomes," *Cell*, vol. 113, no. 2, pp. 171–182, 2003.

[9] J. Van Etten and D. Bhattacharya, "Horizontal gene transfer in eukaryotes: not if, but how much?," *Trends in Genetics*, vol. 36, no. 12, pp. 915–925, 2020.

[10] H. Ochman, J. G. Lawrence, and E. A. Groisman, "Lateral gene transfer and the nature of bacterial innovation," *nature*, vol. 405, no. 6784, pp. 299–304, 2000.

[11] K. M. Swenson, P. Guertin, H. Deschênes, and A. Bergeron, "Reconstructing the modular recombination history of staphylococcus aureus phages," in *BMC bioinformatics*, vol. 14, pp. 1–9, Springer, 2013.

[12] S. Bérard, A. Chateau, N. Pompidor, P. Guertin, A. Bergeron, and K. M. Swenson, "Aligning the unalignable: bacteriophage whole genome alignments," *BMC bioinformatics*, vol. 17, no. 1, pp. 1–13, 2016.

[13] A. Bergeron, M.-J. Meurs, R. Valiquette-Labonté, and K. M. Swenson, "On the comparison of bacteriophage populations," in *RECOMB International Workshop on Comparative Genomics*, pp. 3–20, Springer, 2022.

[14] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[15] C. Lee and D. Lee, "On a circle-cover minimization problem," *Information Processing Letters*, vol. 18, no. 2, pp. 109–115, 1984.

[16] V. A. Traag, L. Waltman, and N. J. Van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[17] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[18] J. Murphy, F. Bottacini, J. Mahony, P. Kelleher, H. Neve, A. Zomer, A. Nauta, and D. van Sinderen, "Comparative genomics and functional analysis of the 936 group of lactococcal siphoviridae phages," *Scientific reports*, vol. 6, no. 1, pp. 1–13, 2016.

[19] F. Raymond, A. A. Ouameur, M. Déraspe, N. Iqbal, H. Gingras, B. Dridi, P. Leprohon, P.-L. Plante, R. Giroux, È. Bérubé, *et al.*, "The initial state of the human gut microbiome determines its reshaping by antibiotics," *The ISME journal*, vol. 10, no. 3, pp. 707–720, 2016.

[20] J. Ren, K. Song, C. Deng, N. A. Ahlgren, J. A. Fuhrman, Y. Li, X. Xie, R. Poplin, and F. Sun, "Identifying viruses from metagenomic data using deep learning," *Quantitative Biology*, vol. 8, no. 1, pp. 64–77, 2020.

Chapter 5

Conclusion

5.1 Scientific contributions

Our research projects address challenges resulting from the uneven distribution of genetic material by proposing three methods to account for it.

In the first project, we developed DeepImpute, a deep learning approach for the imputation of missing values in single-cell RNA-seq experiments. We show that our approach can successfully recover many of the missing values and improve the downstream analyses.

In the second project, we developed CoCoNet, a deep learning based approach to address the issue of fragmentation in viral metagenomic assemblies. To our knowledge, this is the first method that was specifically developed for viruses. We show CoCoNet outperforms existing methods that were developed for bacteria and can successfully cluster contigs in homogeneous bins.

Our third project explores the genomic structure of viral populations in order to split species-level bins into subpopulations. We show that our tool can identify shared recombinations between phages in the same population and help unravel some of the combinatorial complexity of phage population structure.

5.2 Future work

All three projects in this dissertation showed promising results, and we believe they could be further improved or expanded to other related fields.

Imputation of microbiome data: In the first project, we developed a tool for imputing missing values in single-cell RNA-seq data. Other types of experiments, such as marker-gene sequencing of metagenomic data, handle similar feature and face the same challenges. Thus, expanding DeepImpute to the specifics of those fields could be a natural direction for this project.

Improvement of binning accuracy: In the second project, we showed that we could learn a similarity function to bin contigs together. Although the accuracy of our neural network was quite high (>90-95%), there was a significant amount of false positive predictions in our experiments. This is due to the nature of the comparisons being performed: there are naturally more contig pairs that do not belong to the same genome. One solution would be to integrate the imbalance of classes during the training of the network, by either adjusting the loss function to be biased towards negative examples, or by constructing a training dataset with the same imbalance. Another solution would be to use additional viral features to guide the network in order to do fewer comparisons. An example would be to functionally annotate the contigs, group the contigs with similar functions, and use CoCoNet to only compare contigs with complementary functions.

5.3 Other contributions

- Zhu, X., Wolfgruber, T. K., Tasato, A., Arisdakessian, C., Garmire, D. G., & Garmire, L. X. (2017). Granatum: a graphical single-cell RNA-Seq analysis pipeline for genomics scientists. Genome medicine, 9(1), 1-12.

- Du, Y., Huang, Q., Arisdakessian, C., & Garmire, L. X. (2020). Evaluation of STAR and Kallisto on single cell RNA-Seq data alignment. G3: Genes, Genomes, Genetics, 10(5), 1775-1783.

- Arisdakessian, C., Cleveland, S.B. and Belcaid, M. (2020). MetaFlow— mics: Scalable and Reproducible Nextflow Pipelines for the Analysis of Microbiome Marker Data. In Practice and Experience in Advanced Research Computing (pp. 120-124).
- Garmire DG, Zhu X, Mantravadi A, Huang Q, Yunits B, Liu Y, Wolfgruber T, Poirion O, Zhao T, Arisdakessian C, Stanojevic S. (2021) GranatumX: A community-engaging, modularized, and flexible webtool for single-cell data analysis. Genomics, Proteomics & Bioinformatics;19(3):452-60.
- Belcaid, M., Arisdakessian, C., & Kravchenko, Y. (2021). Efficient DNA sequence partitioning using probabilistic subsets and hypergraphs. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (pp. 4-9).
- Jani, A.J., Bushell, J., Arisdakessian, C.G. et al. (2021). The amphibian microbiome exhibits poor resilience following pathogen-induced disturbance. ISME J 15, 1628–1640. https://doi.org/10.1038/s41396-020-00875-w
- Spotkaeff, C., Arisdakessian, C., Jungbluth, S., Rappe, M., Steward, G., and Nigro, O. (2021). Across the Basement: A Comparison of Viruses in the Crustal Aquifer at North Pond and Juan de Fuca Ridge, vol. 2021
- Belcaid, M., Arisdakessian, C., & Kravchenko, Y. (2022). Taming DNA clustering in massive datasets with SLYMFAST. ACM SIGAPP Applied Computing Review, 22(1), 15-23.
- Nigro, O., Spotkaeff, C. R., Arisdakessian, C., Jungbluth, S., Rappe, M. S., & Steward, G. Phylogenomic comparison of viruses across basalt-hosted oceanic basement systems. In 2022 Astrobiology Science Conference. AGU.
- Cleveland, S., Arisdakessian, C., Nelson, C., Belcaid, M., Frank, K., & Jacobs, G. (2022). The C-MĀIKI Gateway: A Modern Science Platform for Analyzing Microbiome Data. In Practice and Experience in Advanced Research Computing (pp. 1-7).