The BioLighthouse: Reusable Software Design for Bioinformatics

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Tanner Dowhy

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Or

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada

Abstract

Advances in next-generation sequencing have accelerated the field of microbiology by making accessible a wealth of information about microbiomes. Unfortunately, microbiome experiments are among the least reproducible in terms of bioinformatics. Software tools are often poorly documented, under-maintained, and commonly have arcane dependencies requiring significant time investment to configure them correctly. Microbiome studies are multidisciplinary efforts but communication and knowledge discrepancies make accessibility, reproducibility, and transparency of computational workflows difficult. *The BioLighthouse* uses Ansible roles, playbooks, and modules to automate configuration and execution of bioinformatics workflows. The roles and playbooks act as virtual laboratory notebooks by documenting the provenance of a bioinformatics workflow. *The BioLighthouse* was tested for platform dependence and data-scale dependence with a microbial profiling pipeline. The microbial profiling pipeline consisted of Cutadapt [1], FLASH2 [2], and DADA2 [3]. The pipeline was tested on 3 canola root and soil microbiome datasets with differing orders of magnitude of data: 1 sample, 10 samples, and 100 samples. Each dataset was processed by *The BioLighthouse* with 10 unique parameter sets and outputs were compared across 8 computing environments for a total of 240 pipeline runs. Outputs after each step in the pipeline were tested for identity using the Linux *diff* command [4] to ensure reproducible results. Testing of *The BioLighthouse* suggested no platform or data-scale dependence. To provide an easy way of maintaining environment reproducibility in user-space, Conda [5] and the channel Bioconda [6] were used for virtual environments and software dependencies for configuring bioinformatics tools. *The BioLighthouse* provides a framework for developers to make their tools accessible to the research community, for bioinformaticians to build bioinformatics workflows, and for the broader research community to consume these tools at a high level while knowing the tools will execute as intended.

Acknowledgements

For the support throughout this journey, I would like to first thank my supervisor Dr. Matthew Links. The breadth and depth of his knowledge not only in the field but with respect to the personal aspects of graduate school made this experience extremely positive. Whether I wanted to talk about life, science, or video games, his door was always open and his reassurance throughout the process is something I will be forever grateful for. I would also like to thank my committee: Dr. Tim Dumonceaux, Dr. Michael Horsch, and Dr. Tony Kusalik. Their support and patience through this process, and the time taken to review my thesis is greatly appreciated. In addition, I would like to thank my parents. Without their care and support, none of this would have been possible. Funding for this thesis was provided through the Plant Phenotyping and Imaging Research Centre (P2IRC) and the Pig Gut Microbiome Project (PGmp). The Global Institute for Food Security served as the lead for the Canada First Research Excellence Fund award responsible for P2IRC. PGmp is funded by Swine Innovation Porc.

To the future, for which I am hopeful.

Contents

List of Tables

List of Figures

List of Abbreviations

ACD	Ajax Command Definition
ARM	Advanced RISC Machine
ASV	Amplicon Sequence Variant
BASH	Bourne Again SHell
bp	base pairs
BSD	Berkley Software Distribution
BWA	Burrows-Wheeler Aligner
DADA	Divisive Amplicon Denoising Algorithm
DAG	Directed Acyclic Graph
DBG	de Bruijn Graph
DNA	DeoxyriboNucleic Acid
EA	Electronic Arts
EMBOSS	European Molecular Biology Open Software Suite
FLASH	Fast Length Adjustment of SHort reads
GB	GigaByte
GDL	GNU Data Language
GNU	GNU's Not Unix
GPU	Graphics Processing Unit
GUI	Graphical User Interface
GWMS	Graphical Workflow Management System
HPC	High-Performance Computer
HTML	HyperText Markup Language
IBD	Inflammatory Bowel Disease
IgE	Immunoglobulin E
ILP	Interactive Literate Programming
IT	Information Technology
ITS	Internal Transcribed Spacer
JSON	JavaScript Object Notation
mPUMA	microbial Profiling Using Metagenomic Assembly
NGS	Next-Generation Sequencing
OLC	Overlap-Layout Consensus
OS	Operating System
OTU	Operational Taxonomic Unit
PCR	Polymerase Chain Reaction
PDF	Portable Document Format
PEAR	Paired-End reAd mergeR
rDNA	ribosomal DeoxyriboNucleic Acid
RDP	Ribosomal Database Project
rRNA	ribosomal RiboNucleic Acid
SADI	Semantic Automated Discovery and Integration
SLURM	Simple Linux Utility for Resource Management
SSD	Solid-State Drive
SSH	Secure SHell
USD	United States Dollars
UT	Universal Target
VCS	Version Control System
YAML	Yet Another Markup Language

1 Introduction

Next-Generation Sequencing (NGS) has enabled the production of large *omics* datasets containing billions of biological sequences. *Omics* refers to biological fields that end with the -omics suffix such as genomics, phenomics, proteomics, metabolomics, and transcriptomics. A sub-discipline of genomics, metagenomics, is concerned with understanding and characterizing the genomes of microbes. Metagenomics provides indispensable insight into the diversity of previously uncultured microorganisms (microorganisms which cannot be cultivated in a laboratory setting) [8]. Metagenomic datasets represent a microbiota containing taxa from the 3 domains of life: Archaea, Bacteria, and Euakaryota. The term microbiota is commonly used to connote the list of all organisms from a specific environment. By contrast the term microbiome was defined by Lederberg and McCray [9] as, "the ecological community of commensal, symbiotic, and pathogenic microorganisms." More clearly, a microbiome is a catalog of microorganisms and their genes within an ecological niche. Ecological niches for microbiome studies can include the gastrointestinal microbiome to study organism health and soil to understand how microbial diversity relates to crop performance.

Microbial profiling is a multidisciplinary effort that encapsulates elements of molecular and computational biology, microbiology, statistics, and computer science. The multidisciplinary nature of microbial profiling significantly affects productivity of biological and non-biological scientists. Differences in vocabularies and foundational knowledge make communication of tasks and goals difficult. Additionally, microbial profiling productivity is influenced by dependence on work by others. Computational (e.g. bioinformatics) researchers are dependent on data generation and biologists are currently dependent on processed data and computational tool development.

The recent acceleration of microbial profiling technologies requires researchers to constantly update their skills. Having current knowledge when working with NGS data is necessary. There are many tools that perform similar tasks, a large number of file formats, and complex software dependencies which makes choosing appropriate tools difficult. NGS data can be on the order of petabytes in size and thus manual methods and computational methods such as Microsoft Excel are insufficient. As a result, microbiome datasets require high-throughput computational methods for multidisciplinary users to conduct analyses [10]. Researchers lacking up-to-date computational skills often have difficulty choosing the correct tools for analyses, validating methods, and sharing results [11].

The overwhelming breadth of knowledge required to make informed decisions relating to bioinformatics results in researchers taking *heuristic* approaches to choosing methods. Researchers may adopt methods solely based on the journal they were published in or whether they can easily get the software to work, regardless of

whether the results are correct. Factors such as favourable results, ease of tool configuration, and intuitiveness of workflow execution influence the choice of tool used. New tools must be broadly approachable by the research community but also reusable so they can be formally assessed by the community for reproducibility and accuracy.

Reproducibility forms the foundation of scientific research, as it allows scientists to build on each others' work with confidence [12]. By consistently reproducing previous results, one can rest assured they were not obtained by chance. Additionally, improved results can be compared to the literature as positive controls for *in silico* experiments. The findings of a scientific analysis are reproducible if the steps undertaken to obtain them are comprehensively documented so others can achieve semantically consistent results [12]. Full reproducibility is difficult to implement for a number of reasons [13]. Complex tools with confusing interfaces are pervasive in NGS experiments so simply getting a tool configured to run on sample data can take hours. Coupled with poor documentation, large datasets, and no reproducible standard in bioinformatics, it can be difficult to know exactly how to reproduce an analysis. Reproducibility is essential for not only experimental studies, but for the development of clinical trials using NGS for diagnosis and treatment of diseases.

Bioinformatics is becoming increasingly prevalent in life sciences, and many scientists outside of computer science are finding themselves running complex computational workflows. Non-computational researchers can easily struggle to articulate a fully reproducible environment since there exists no standard approach to making a computational workflow reproducible. Similarly, something a computational researcher may deem trivial could prove vital in reproducing a workflow for others.

Consider sequence alignments as they are common in bioinformatics and allow for identification of organisms and approximate comparisons of evolution. The Burrows-Wheeler Aligner (BWA) [7] is an impactful tool used in many pipelines and has been cited over 16,000 times. Nekrutenko and Taylor [11] investigated fifty published papers using BWA and observed thirty-one of them documented neither software version nor parameters used [7]. Descriptive information about datasets, tools, and their invocations to perform the analysis must be documented for a computational workflow to be reproducible [13].

Scientific workflow systems have been developed to assist researchers in their analyses. These workflow systems often abstract away configuration and execution of tools so users only need to specify parameters and higher-level instructions for use. The simplifications provided by workflow systems enhance reproducibility by programmatically handling certain abstractions to minimize errors made by users. Accessibility and transparency have been proposed, in addition to reproducibility, for the development of a robust workflow system [13]. An accessible platform allows users to run an analysis, regardless of their backgrounds. Transparency enables users to share their results and workflow methods. Transparency allows reproducibility and inspection of the workflows [13]. Data and workflow provenance are often recorded to maintain transparency.

Scientific workflow systems chain tasks together into an analysis pipeline where outputs from a task can be used as inputs for a subsequent task [14]. Such analysis pipelines are common in microbial profiling and yet many require researchers to conduct each step independently without automation and lack the provenance

to enable other researchers to reproduce the analysis. Scientific workflow systems can integrate well with provenance tracking systems if all environment and methodology information is readily available [14]. A scientific workflow system must also enable sharing reproducible workflows and be agile, meaning supporting various computing environments so issues with differing hardware and operating systems are abstracted away from the analytical processing itself.

The main objective of this thesis was to create an accessible, reproducible, and transparent workflow management system to configure and execute scientific workflows. To satisfy the main objective, I developed *The BioLighthouse* which extends Ansible [15]. To test for reproducibility, I implemented a microbial profiling pipeline into *The BioLighthouse*. The pipeline consisted of Cutadapt [1], FLASH2 [2], and DADA2 [3]. Reproducibility of the pipeline was tested from the perspective of platform dependence and data-scale dependence. To test for platform dependence, I executed the pipeline on 9 different machines including cloud machines, to shared machines, and HPC clusters. Outputs from each step in the pipeline were compared and deemed platform independent if they were identical. To test for data-scale dependence, I executed the pipeline on 3 microbial profiling datasets of differing orders of magnitude: 1 sample, 10 samples, and 100 samples. The tests were conducted across the 8 different machines to determine if *The BioLighthouse* results were dependent on the scale of the data.

The organization of this thesis is as follows. Chapter 2 and 3 cover the background work and concepts this thesis was built upon. Chapter 4 outlines the goals and objectives for this research. Chapters 5 and 6 describe the methods and results respectively and Chapter 7 discusses the results and implications of the results. Finally, Chapter 8 concludes the thesis and outlines possible future work.

2 Background

This chapter builds on the microbiology and bioinformatics development concepts introduced in Chapter 1. It begins by discussing the benefits and challenges in microbial profiling, and leads into the impacts on computational researchers. This chapter continues with an an introduction to provenance in Next-Generation Sequencing (NGS) computational pipelines and ends with an introduction to Ansible [15], the technology this thesis builds upon.

Culturing of microorganisms is a laboratory approach used to study isolated colonies of microbial species. Organisms are grown (cultured) in a controlled environment in order to accurately describe each individual microorganism. Pure cultures are cultures in which only organisms of a single species is present. For an organism to be recognized as a species, isolation of the organism into a pure culture is required [16].

The tree of life attempts to encompass the total diversity of sequenced genomes but most methods rely on culturing for classification of organisms [17]. The true scale of the tree of life remains unknown [17] since most microorganisms are uncultured and cannot currently be grown in a laboratory setting.

Uncultured organisms are thought to comprise over 99% of all microbes [8], and have large impacts on the ecological niches they occupy. Uncultured organisms are not necessarily impossible to culture. Instead, uncultured microorganisms present an opportunity to uncover critical biological information and gain access to currently hidden biological diversity [18]. At present, there exist many variables that must be optimized in order to culture most microorganisms: nutrients, pH, and temperature [18]. Microbiome studies observe microbes naturally as they occur and thus the variables for the microbes' growth are implicitly accounted for.

Microbiome studies obtain samples directly from an ecological niche without the need to first culture any microbe, revolutionizing the study of uncultured microbes. By studying microbes as they occur naturally, a profile of the microbes in an ecological niche can be generated. This profile provides a more thorough insight into the biological diversity within an ecological niche since most microbes are thought to be uncultured.

Microbial profiling is an approach to study the microbiome that uses DNA sequences as barcodes to catalog microbes and their total DNA. DNA barcodes are specific DNA sequences used to identify organisms [19]. Barcodes in unknown samples are compared to a collection of sequences from known reference samples for classification [20]. DNA barcodes common to all microbes intended for study are sequenced to catalog the microbes. Common DNA barcodes for microbial profiling are 16S for bacteria, 18S for eukaryotes, Internal Transcribed Spacers (ITS) for fungi, and cpn60 for multiple domains of life.

The accuracy of a DNA barcode for identification of microbes is dependent on the barcoding gap, or

the distance between the intraspecific variation and interspecific divergence of the barcode [19]. Barcoding gaps are the genetic distance between the intraspecific sequences for the same gene and the divergence separating *sister* species [19] used to identify species. A larger barcoding gap means more genetic diversity can be expressed by the gap. By expressing more genetic diversity, larger barcoding gaps provide better identification of closely related organisms that may differ by only a single nucleotide.

DNA barcodes for microbial profiling have been successfully used in a variety of ways. Morgan et al. [21] analyzed intestinal biopsies and stool samples from 231 individuals with Inflammatory Bowel Disease (IBD) and healthy individuals with 16S rRNA encoding gene sequencing. The authors found that IBD is associated with major changes in 2 bacterial phyla when compared to healthy controls: Firmicutes and Proteobacteria. The authors also found significant shifts in the gastrointestinal microbiome for subjects who received treatment for IBD showing the potential for disease diagnosis with microbial profiling. Turnbaugh et al. [22] studied samples from lean and obese mice to determine how the microbiome affects host physiology. The authors found obesity to be associated with shifts in two bacterial phyla: Bacteroidetes and Firmicutes. Further, Turnbaugh et al. demonstrated that changes in the microbiome affect the metabolic potential of the mouse gut microbiome suggesting the gut microbiome should be considered in the genetic factors contributing to obesity [22]. West et al. [23] analyzed 4 samples from each of 10 healthy infants and 10 infants who developed an IgE-associated eczema. IgE (immunoglobulin E) is an antibody for which elevated levels are associated with allergy related inflammation [24]. IgE is useful as a biomarker as its quantities can be studied in relation to the host microbiome for possible treatment of IgE-associated diseases. The authors confirmed previous findings that bacteria from the Phylum Proteobacteria contribute to IgE-associated eczema in infants [23]. Turnbaugh et al. and West et al. are both examples of using microbial profiling to study host-microbiome interactions. The successful applications of DNA barcodes described required teams and modern resources to satisfy their objectives. The resources included high-throughput DNA sequencing technologies as well as advanced computing and statistical methods.

The sequencing of environmental microbiome samples is a compounding effort that introduces randomness at multiple points. Researchers obtain a random biological sample from an environment and extract the DNA. The DNA is amplified using Polymerase Chain Reaction (PCR), introducing variation in 2 ways. The sequences may not amplify in the same ratio as their true richness and the number technical artifacts in the PCR process can become inflated. PCR doubles the sequence count every cycle and thus sequence count grows exponentially. PCR is imperfect, allowing for single nucleotide errors to occur during amplification (i.e. polymerase base substitution). Early errors in PCR become the most amplified because they are copied more times in subsequent rounds of PCR. A microbial profiling dataset will contain technical artifacts from both sequencing and PCR, and accounting for these artifacts is difficult.

Questions of how to treat the data arises with the stochastic noise introduced before the data is processed. Many sequences have low abundance making it difficult to know if the sequences are truly rare or just underrepresented in the sample. A common method to address this is rarifying the samples to remove low

abundance reads; however, important information is lost as rare organisms may be vital to the microbiome. There is also dispute on aggregating microbiome data. Raw microbial profiling data is typically in the form of DNA sequences that cannot be analyzed directly and must first be processed. Aggregation of microbial profiling data has been commonly based on a 97% sequence identity threshold to account for random sequencing errors and amplification artifacts.

Approaches such as mPUMA [25] and DADA2 [3] were developed to obtain single-basepair-level resolution of aggregated microbiome data. The mPUMA pipeline uses *de novo* assembly to assemble sequences without use of a reference database, then clusters sequences at 100% identity to form the dataset. The DADA2 algorithm attempts to learn the sequencing errors in order to denoise the raw data and produce the processed dataset.

Many researchers do not take a data-centric approach to microbiome data. Particularly, biological scientists tend to be more comfortable analyzing microbiome data based on taxonomic assignments as opposed to raw sequences. Bergey's Manual of Determinative Bacteriology [26] is the standard system for taxonomic identification of prokaryotes. The goal of Bergey's Manual was to create a classification system to model the phylogeny of prokaryotes [27]. The microbial taxonomy system as described in Bergey's Manual originally classified organisms based on "morphological, cultural, physiological, and pathogenic" [26] characteristics. Missing from this list was sequence information as the system was built without it, but in recent versions of the manual 16S rRNA gene sequences have been added to the system. The 16S gene is present in all prokaryotes but has some limitations when defining close taxonomic relationships. 16S sequences are often limited to genus-level resolution when distinguishing microbes [20]. Additionally, there exists a lack of consistency when defining boundaries of genera and limitations for defining higher taxonomic structure using the 16S gene [27]. 16S sequences have not been identified for all validly named species currently in the manual so full characterization of those microbes is incomplete. Disagreements when assigning taxonomy based on sequences are expected since incomplete sequence information was used in defining taxa in Bergey's Manual.

Assessing microbial taxonomy can introduce uncertainty in two ways. Firstly, incomplete sequence information in Bergey's system for microbial taxonomy leads to uncertainty in assigning taxonomy based on the organisms' sequences, especially for organisms identified before the introduction of sequence information to the system. Secondly, taxonomy is assigned based on a database of known sequence-to-taxon relationships. Some organisms may not be classified based on Bergey's system and thus cause a loss of taxonomic resolution. For sequences not in the database, the closest taxonomic match is assigned to that sequence and so the taxonomic system is blind to novel organisms.

It is clear currently that NGS and microbial profiling methods are imperfect. As computational and laboratory methods evolve to address the imperfections, it is imperative that researchers understand where these imperfections exist and tailor their methods to mitigate the impacts of them. Laboratory methods are outside the scope of this thesis; however, on the computational side there are two major cases allowing for mitigating steps in microbial profiling. Aggregating sequences at 97% identity to account for random

sequencing errors is outdated as there are assembly algorithms and denoising algorithms that attempt to solve sequencing errors. Additionally, assigning taxonomy should be done at the last possible moment. In biological studies, it is obvious that biological meaning needs to be extracted from the data. However imperfections in the classification systems and absence of many organisms in databases results in a loss of resolution. Instead, all statistics should be performed on the sequences themselves and taxonomy should be assigned after to suggest biological meaning based on the classification system used.

Microbial profiling is a relatively new field that started to pick up pace in the early 2000s. Before the introduction of NGS platforms around 2008, the largest limitation to microbial profiling was sequencing scale. Microbial profiling data generation was often done with Sanger sequencing prior to NGS. Sanger sequencing is a labour-intensive method that often limits the scale of sequencing experiments. In 2002, Hill et al. [28] published the largest microbial profiling study at the time using Sanger sequencing. The dataset consisted of 1,125 cloned sequences across 4 libraries (samples of DNA cloned into *Escherichia coli*) extracted from 2g of feces, which identified 398 unique nucleotide sequences. The study considered microbes in pig feces, which has an estimated 10^{11} organisms per gram of feces [28]. The 1,125 cloned sequences were only a small sample of the estimated bacterial richness in pig feces (10^{11}/gram) resulting in microbial diversity that was presumed un-captured. NGS has enabled sequencing scale many orders of magnitude higher than Sanger sequencing. Taye et al. studied the rhizosphere microbiome of *Brassica napus* using NGS and generated on the order of 10^9 sequences across 477 samples [29]. NSG has greatly increased the velocity and volume at which sequencing data is generated making it much more efficient than Sanger sequencing for studies at scale.

The acceleration of microbial profiling technologies requires researchers to constantly update their skills. Having current knowledge when working with bioinformatics data is necessary because there are many tools that do similar jobs, a large number of file formats, and complex software dependencies that makes choosing appropriate tools difficult. Microbiome datasets require high-throughput computational methods for multi-disciplinary users to conduct analyses [10]. Researchers lacking up-to-date computational skills often have difficulty choosing the correct tools for analyses, validating methods, and sharing results [11].

2.1 Bioinformatics Development

For consistency throughout this thesis, the actors in bioinformatic processes can be categorized into 4 groups: life scientists, developers, bioinformaticians, and tool users. A simplified multidisciplinary researcher interaction is outlined in Figure 2.1. Life scientists are the actors involved in laboratory and field work. Life scientists include those who perform sample collection, sample processing, DNA sequencing, and any other action generating a data product from biological samples. Developers are the actors who develop data processing and analysis tools for the scientific community. Bioinformaticians are the actors who consume the products from the developers and life-scientists in order to generate a pipeline and organize the data. Tool

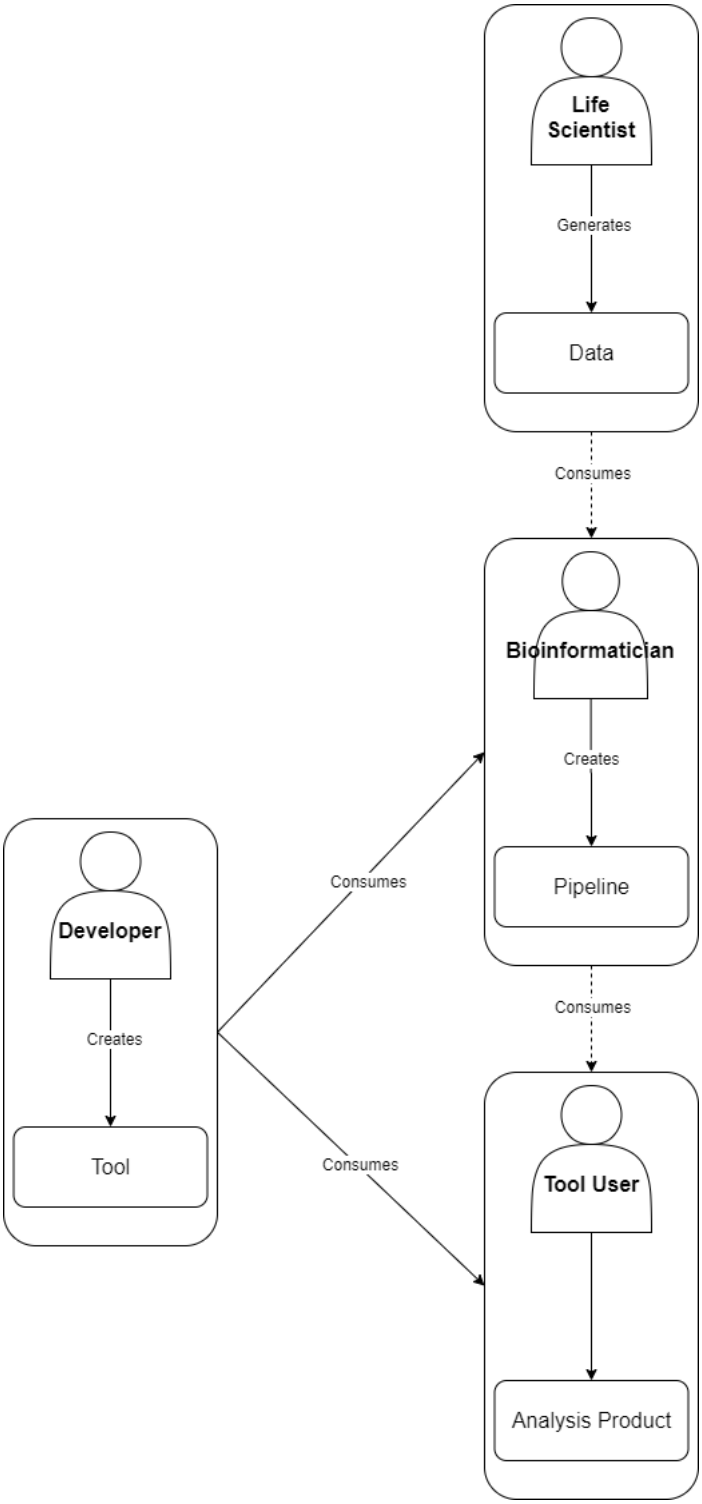


Figure 2.1: The actors involved in a bioinformatics project. Life scientists are involved with generating data from biological samples and developers create computational tools to perform a task. In this model, the bioinformatician consumes data from the life scientist and tools from the developer in order to produce an analysis pipeline that is subsequently consumed by the tool user to generate the analysis product. Often a single actor will assume multiple roles in this model which is expected but the distinction is made in order to know what role an actor occupies at each stage in the process.

users are the actors that consume data products and tools to conduct analyses. Interpretation of the results occurs outside of this model. The software and data products may be received from the bioinformatician, or from the developer and life scientist directly. In practice, the definitions of the actors described here will differ; however, for the purpose of this thesis a clear distinction must be made between these actors for clarity. Further, a researcher may occupy more than one role. For example, a developer may also be a bioinformatician to generate tools, create pipelines, and manage data. Similarly, the bioinformatician who generates the pipeline may also assume the role of tool user when executing the workflow. The purpose of the categorization is to identify what role an actor is occupying at different points in an analysis.

The volume of bioinformatics datasets is often hundreds of gigabytes in size and can be upwards of petabytes in size, making manual interaction with the datasets unreasonable. Bioinformatics datasets often consist of hundreds or thousands of files that are compressed to save space and are thus not human-readable. As a result, code is required for interacting with the data, analysis, and validation of analyses. There are many different types of bioinformatics datasets and variety of formats, further making programmatic interaction necessary.

Recent advances in NGS have accelerated the field of bioinformatics by tapping into the wealth of data available in human, animal, and environmental microbiomes [30]. Until 2008, DNA sequencing was largely centralized to DNA sequencing centers because of its high cost. As shown in Figure 2.2, the cost per megabase of sequencing in 2001 was almost $10,000 USD and still costed almost $1,000 USD per megabase in 2007 [31], a cost far too high for many research labs. The cost associated with sequencing isn't only the sequencing machine but also library preparation, human lab work, and even power for the sequencers. Offloading sequencing to a center mitigated cost and ensured sequencing was done by someone with experience. By 2011, the cost of sequencing had dropped to around $0.10 USD per megabase [31], allowing university laboratories to invest in sequencing machines and training. As the cost of sequencing dropped, platforms like the Illumina MiSeq system emerged so university labs could conduct their own sequencing.

Privacy concerns can arise when using a sequencing center. For instance, Agriculture and Agri-Food Canada has a collection of elite canola lines. Even if the metadata is anonymized properly, the genome sequences will be known to the sequencing center and they will have the biological samples so decentralized sequencing prevents additional sharing of sensitive data.

Since the emergence of decentralized sequencing, scientists have recognized the potential of ubiquitous DNA sequencing. Ubiquitous sequencing is the idea that DNA sequencing can be conducted everywhere. For example, with the 2015 ebola outbreak in the Congo, scientists were working to better understand the virus. Ubiquitous sequencing enabled study of the virus in the field without an extensive laboratory and mitigated risk in transporting samples since genomes were sequenced in the field [32]. As a result, only the digital sequence information needed to be transported which cannot transmit the biological disease.

Oxford Nanopore Technologies [33] specifically aims to make high-performance DNA sequencing accessible and easy to use. Though Oxford Nanopore Technologies offers large parallel sequencing platforms, the

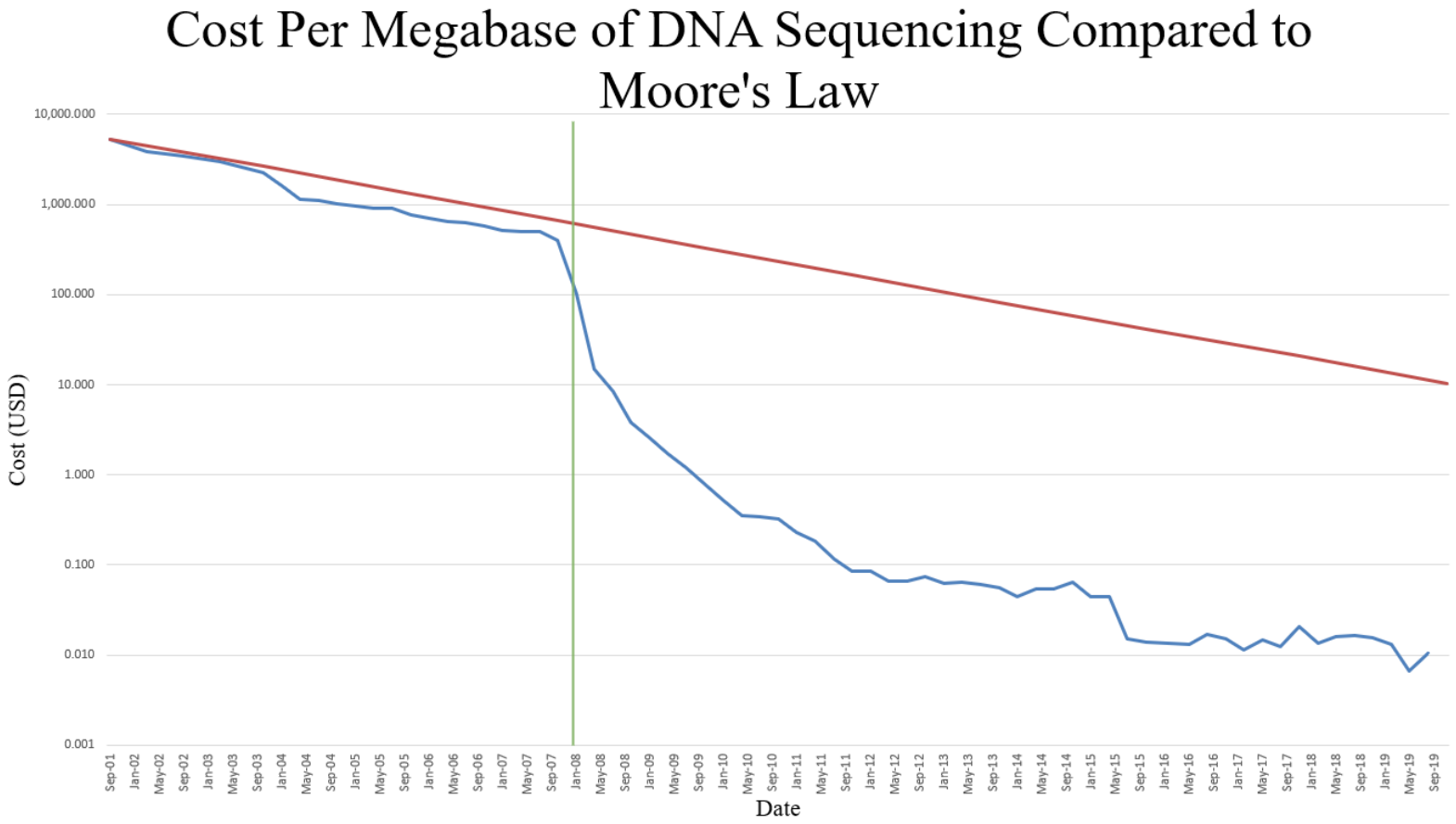


Figure 2.2: Sequencing cost in USD per megabase, compared to Moore's Law. The blue line shows the cost per megabase of sequencing since September 2001 to September 2019. The red line denotes Moore's Law. Data before January 2008, to the left of the green line is the sequencing cost of Sanger sequencing technologies. Data after January 2008, to the right of the green line is the sequencing cost of Next-Generation Sequencing (NGS) technologies. Data was obtained from the National Human Genome Research Institute [31].

MinION and SmidgION devices enable DNA sequencing on personal laptops and mobile phones allowing anyone to sequence anything, anywhere. These devices enable DNA sequencing in the field where network and other lab necessities are not available. This can include sequencing microbes in rain forests, Antarctic ice, and even on the International Space Station. Further, Oxford Nanopore Technologies are working to enable automatic library preparation with the portable microfluidic device, VolTRAX, and analysis with the MinIT making DNA isolation, sequencing and analysis functionally portable. The MinIT is a pre-configured Linux machine about the size of a brick and eliminates the need for a dedicated sequencing laptop. It has a low-cost ARM processor, GPU, 512 GB SSD, and allows analyses to be controlled using a laptop, tablet, or mobile device. Introduction of ubiquitous sequencing platforms makes DNA sequencing accessible to non-experts, and thus the computational processing and analyses must be made accessible as well.

The rapid expansion of the amount of data (volume), the speed data is being generated (velocity), and the many data sources and structures (variety) has exposed a bottleneck in our capacity to analyze *omics* data. Figure 2.2 shows a sharp drop in sequencing cost around January 2008 when NGS technologies were beginning to be widely used. The drop in sequencing cost resulted in a software development cycle centered around prototyping and quick project turn-over, causing many bioinformatics software tools to be under-maintained and out-of-date.

There are also often exceptions made in the ideals of reproducibility because of pressure to release work while it is still relevant [34, 35]. Computational reproducibility is further complicated by a number of factors in addition to new tool development [36]. Long-term studies may see multiple rounds of graduate students from different disciplines. Becoming acquainted with the tools and methods of an established project is time consuming and accurately comparing them to newer tools to confirm results from previous studies is required as research progresses [36].

2.2 Provenance

Provenance in scientific workflows is essential when striving for reproducibility. Provenance refers to the record of the derivation of a set of results [37]. This includes the *prospective provenance*, or the steps required to produce a data product, and the *retrospective provenance*, or the steps that were executed and additional metadata about the execution environment used to generate the data product [37]. The terms are distinct from each other because *prospective provenance* is the recipe required for the analysis of the data whereas *retrospective provenance* includes a ledger of all the steps executed including unnecessary and exploratory tasks invoked by the user. Life science researchers are familiar with a similar approach by maintaining a lab book as a record of everything performed in the lab.

2.3 Ansible

Ansible [15] is a large-scale IT automation framework that can accurately describe an IT application infrastructure. It is used for provisioning of computational environments, configuration of computational environments, and application deployment onto computational environments. Ansible is used by large companies, such as NASA, for configuration of their servers and continuous integration of their platforms. Ansible thus allows companies to increase scale of computational power by adding more servers and performing maintenance and testing in an automated fashion. Full specification of the environment, dependencies, and application code requirements are all included within Ansible recipes making it suitable to facilitate bioinformatics work.

Playbooks are Ansible's instruction language for provisioning, configuration, and deployment based on the human and machine-readable format Yet Another Markup Language (YAML[38]). In the Ansible system, playbooks record the *prospective provenance*. Playbooks are comprised of an inventory and roles. The inventory contains the address of the target machines on which the commands will be run and roles are the instructions to be executed. Ansible encapsulates many configuration and deployment tasks into modules and developers can easily modify them or write their own for their specific needs. Applications can be deployed and configured with Ansible to ensure environments are set up exactly how they are intended by the developer. Ansible is also agentless, meaning a central control node is not required for the execution of playbooks. Instead, playbooks are executed remotely from any machine through the SSH protocol [39].

Ansible is used for provisioning machines and configuring environments but this thesis explores its use for execution of scientific workflows. Ansible runs also have the ability to output varying levels of verbosity to accurately capture the *retrospective provenance* of a scientific workflow. Ansible Tower, or the open-source AWX, is used to effectively control and manage Ansible automation. AWX is an optional graphical user interface for the management of Ansible playbooks. AWX is an ideal candidate for a workflow system because it supports many users and has the potential to expose varying levels of input parameters; however, a graphical user interface is outside the scope of this thesis. Ansible's structured format and extensive logging makes it an ideal candidate to record workflow and data provenance for its runs. Finally, Ansible Galaxy[1] is a repository of community-generated playbooks, roles, and modules readily available for usage with little modification. The public availability of roles and easy installation enhances both accessibility and transparency.

[1]WARNING: a name collision exists between the platform used to host Ansible playbooks and the Galaxy Project commonly used in the bioinformatics space. The word "Project" is conventionally used to distinguish these terms.

3 Literature Review

Next-Generation Sequencing (NGS) datasets require special care for reproducible analysis. Setting up the compute environment properly is necessary, along with the tools and their software dependencies. Additionally, High-Performance Computing (HPC) environments may be required for analysis. Many researchers do not have the knowledge to carry out these analyses using the appropriate tools and computational infrastructure in a way that will be understood by others [11]. This chapter reviews the literature that supports this thesis. It begins with outlining strategies for computational reproducibility in analysis of NGS data as well as infrastructure as code platforms and how they can be applied to reproducibility in bioinformatics. It continues with reviewing marker gene analysis, OTU formation, and how scale in these analyses affect reproducibility. Finally, Ansible [15] and its core concepts are outlined.

3.1 Strategies for Computational Reproducibility

Piccolo and Frampton [12] have proposed a 7-category approach for describing strategies of improving computational reproducibility that fit well with the objectives of this thesis outlined in Chapter 4.

Written Narratives Human readable written narratives are the most basic technique for computational reproducibility. Written narratives are detailed ledgers about everything required to reproduce computational results. It is absolutely necessary for every published analysis to contain some written narrative but the narrative should never exist alone as it often lacks vital details that can result in hundreds of hours of effort to reproduce the results of a published work [40]. Written narratives affect non-computational researchers in two main ways. Firstly, when documenting their own analyses, it is unlikely for non-computational scientists to effectively describe their computing environment and for computational-focused scientists to include information they may deem trivial. Secondly, for conducting their own analyses, the layer of abstraction between a written narrative and implementation on a computer is inherently ambiguous for a researcher to recreate.

Scripts The next most obvious way to enhance reproducibility is to automate tasks with scripts. This may be appealing because scripts are easy to write and flexible enough to work for a variety of programming languages and tools. Scripts can be used to configure software and their dependencies as well as to execute workflows with defined parameters. GNU Make [41] is a utility that can be used to configure operating system and software dependencies for a particular analysis to take place. For execution, the researcher can write their analyses in scripting languages such as PERL, Python and R.

Software scripts have no standard structure making their implementations susceptible to errors. The developer must keep track of all operating system and software dependencies, and make decisions about the level of configuration to perform. This is not robust as scripts are environment dependent and the level of configuration detail may force users to debug upstream dependencies. Further, scripts are only reusable in the strict manner in which they were implemented and only accessible to users with supported operating systems.

Software Installation Frameworks Using existing tools that have been tested and validated is ideal for analyses rather than implementing new tools from scratch. Many frameworks have emerged to easily handle software dependencies of published tools [5, 42, 43, 44, 45]. Many developers choose a continuous integration approach for their software because of the aforementioned software development cycle of rapid prototyping in order to release minor changes often 2.1. This is easily facilitated by the use of software installation frameworks to house and configure different software versions. Software installation frameworks can configure a variety of versions of a particular application as well as solve complex dependency chains.

Python pip is one of the most common frameworks for installing software packages. It allows developers to serve their Python-written software tools as well as continually update them. Proper pip installations solve required software dependencies for immediate use but do not solve operating system dependencies. Similarly, Bioconductor [42] is a popular framework containing hundreds of biological software packages written in the R programming language [46] that solves tool-level dependencies but lacks in solving operating system dependencies.

Conda [5] is another common software installation framework for any language. Conda is accessible across Windows, MacOS, and Linux platforms for package, dependency, and virtual environment management. A benefit of Conda is the ability to create virtual environments and configure tools using a single system. Conda installation can be done in user-space and also has access to many operating system dependencies, making it ideal for configuration of tools on shared systems. The Bioconda [6] channel makes available over 7000 bioinformatics packages for installation with Conda.

Software installation frameworks may offer multiple versions of packages. If, however, the developer decides to make previous versions unavailable this could be problematic if an analysis requires an unavailable version. Many scientific analyses do not age well solely because of software version availability. To solve software and operating system dependencies, the use of software installation frameworks can be easily coupled with scripts to execute a workflow but issues with flexibility as mentioned previously remain. Additionally, the use of scripts or another method are required to carry out and analytical process. Another issue with software frameworks is the possibility of platform dependence. For example, a Debian machine with apt-get and a RedHat machine with yum frameworks often contain differing versions of the same software package.

Interactive Literate Programming As mentioned, it is often the case that we want to combine more than one of these techniques to improve reproducibility. One way of combining scripts and narratives has been

through code comments [12]. Code comments and other forms of narratives, however, become outdated as code evolves [47]. Interactive Literate Programming (ILP) is a powerful way of addressing this. Researchers can integrate descriptive narratives with computer code to execute an analytical pipeline. Once the code is executed, an output containing the code, the narrative, as well as tables, figures and other outputs is created. If modularized properly, the reader can see exactly how a result was obtained. This forces the developer to consider the target audience during development since it is a key part of the deliverable [47].

Interactive systems allow rapid algorithmic exploration, data analysis, and visualization. These systems have become incredibly important in daily scientific work [48]. Experimental work is conducted in an interactive environment rather than as an executable application. The four most common ILP languages are the Interactive Data Language and its opensource GNU Data Language (GDL) [49] based on Matlab, Mathematica [50] and its opensource Sage [51] which is based on Maple, IPython [48] based on Python, and knitr [52]. Sage is a numerical language and IPython offers a GDL kernel [53] so IPython and knitr will be reviewed.

IPython notebooks are executed on a kernel specified upon creation by the user and corresponding to one of a variety of supported programming languages. For a current list see the Jupyter Github page [54]. Specifying a kernel gives full access to the underlying language and no other languages. There is access, however, to other features such as environment variables and BASH (Bourne Again SHell) commands. All inputs and outputs are shown in the order they were run and produced. There is also full operating system access for the machine the notebook is running on, which is especially useful in omics science where data must be manipulated either with the command line, a programming language, or other tool encapsulating such. knitr is written in the R programming language and, like IPython, supports multiple programming languages. Additionally, it can be executed interactively or in HPC environments such as the Cedar cluster from Compute Canada [55]. knitr output integrates well with LaTeX, making it an ideal candidate for producing publication-ready figures.

Both IPython and knitr have easy installations and support many languages. They are portable and produce outputs in common formats such as Hypertext Markup Language (HTML) and Portable Document Format (PDF), making the outputs easy to share. However, outputs only capture the workflow (*Retrospective Provenance*) that was run and are thus not generalizable. If the user understands the workflow, they can change variable values but for lay users changing the script can be difficult and introduce errors. Furthermore, to run an instance of the workflow in a complex environment, such as on a cluster, the user is required to run the job manually and interactive elements are irrelevant. Literate programming notebooks are suited for analyses requiring a modest amount of time and code [12]. Since this is not the case for nearly all NGS experiments, literate programming is not suitable.

Virtual Machines Software dependencies are time-consuming to install, even if automated. Additionally, previous versions of packages may no longer be available for download. Virtual machines address this by encapsulating an entire operating system of all software and code required to recreate a scientific analysis

[56]. Virtual machines are a solution to reproducible environments because a working environment can easily be captured. The virtual machines can be interacted with using a virtualization software such as VirtualBox or VMWare. The operating system of the virtual machine is independent of the user's operating system and the user has full access in the virtual machine to install software and can exploit escalated privileges [12]. Hurley et al. [56] proposes every publication with computational results should aim to have a 'virtual reference environment' accompanying it. Issues of curation arise when there are errors in the environment [56]. For example, if a virtual reference environment is published with errors and is downloaded, the users who downloaded it may not realize there are errors and either fix them or download an updated version of the reference environment. To combat this, Hurley et al. suggest virtual environments should only be used to recreate the results of a publication and not be used for general purpose [56]; however, this thesis intends to deliver a general use method. Use of an environment requires acceptance to all the license agreements associated with the entire software stack associated with the image and most users will not likely perform their due diligence [56]. There is a further concern for software that cannot be packaged into a virtual machine without violating the license.

Finally, virtual environments expose issues with scale. Virtual machine images can possibly be gigabytes in size which can cause problems for storage and sharing. Additionally, virtual machine performance is limited to the hardware it's running on. If a more powerful infrastructure is required, the user may need to purchase cloud resources from a service provider to run the virtual machine or redeploy it on another platform. This may prove costly depending on how the machines are managed. There is more cost overhead the longer the instance is up, especially when analyses are not running and the cores are not being used. However, it may be difficult or error-prone to terminate and reinitialize identical instances when data is being analyzed and if the cloud provided uses proprietary software prohibiting images from being freely shared the scale required may not be achieved [12].

Containerization Software containers encapsulate operating system components, software tools, and scripts to offer a lightweight alternative to virtual machines [12]. A container can package an application along with dependencies to provide a configured environment to execute an analysis. In contrast to virtual machines that include an entire operating system, containers are a process of the host machine's operating system. This means containers have less computational overhead than virtual machines at the cost of computational flexibility.

The opensource Docker platform [57] is widely popular and has been applied to several scientific projects. It allows for the creation, use, and sharing of software containers for directly interfacing with the host machine's Linux-based operating systems [12]. If the host machine's operating system is Mac or Windows, Docker implicitly initializes a virtual machine to run the container, thus subjecting it to additional virtualization overhead. Docker containers are relatively simple to build as the instructions are text-based. Sharing and management of Docker containers is also easy since they can be stored, and changes can be tracked

with a Version Control System (VCS). If a change is made to a container, users can simply download the update as opposed to a virtual machine that would require users to download the entire virtual machine [12]. Docker is said to promote an application-centric container model, meaning containers should be implemented to run individual applications rather than have one container to run all the software [57]. Containers do not consume a large amount of resources and their initialization is fast compared to vitual machines. In a research setting, the application-centric design pattern results in a loosely coupled system in which the individual components have no knowledge of each other in the system. As a result, a flexible, lightweight system that allows components to be chained together in a high-level manner like many software pipelines present in bioinformatics is produced.

Belmann et al. [34] proposed a standardized method of containing bioinformatics software, called BioBoxes. In addition to lack of software availability, Belmann et al. also cite lack of standards in software interfaces as hindering computational reproducibility [34]. Non-standard interfaces are prevalent in bioinformatics because of the presence of many data formats and large numbers of parameters involved in simple tasks. Though containers have been shown to successfully support scientific work, non-standard interfaces continue to obstruct scientific advancement [34]. For example, a researcher may spend a considerable amount of time structuring the data into the correct input format just to get a tool to run. If there are many tools chained together into a workflow data structuring may need to be repeated many times increasing the chance for errors. BioBoxes are software containers containing a tool and its dependencies with standardized interfaces for specific tasks. For example, all BioBox genome assemblers accept input FASTQ files and return the assembled contigs in the same format. Since all tools of the same type have the same interface, they can be easily interchanged in an analysis pipeline.

BioBoxes are powerful tools but have some flaws in practice. They require developers to adopt a development pattern by accepting that standardized interfaces are the best way to facilitate reproducibility which isn't necessarily true. Many bioinformatics tools have the flexibility to accept varying input formats and output varying formats. Standardization of inputs and outputs makes putting workflows together easy but as applications become more complex, classifying them using the BioBox method becomes problematic. Alternative methods could be explored; for example, tools can be containerized to automate file format conversions thus eliminating the need for standardized inputs.

BioContainers [58] is a community-driven project containing over 2076 Docker containers for various bioinformatics tools. They have 18 contributors who will add tools by request on Github `https://github.com/BioContainers`. The publication cites the BioBoxes publication however, it does not follow the BioBox proposed format. All BioContainers are available on Github as well as via Docker Hub `https://hub.docker.com/r/biocontainers/biocontainers`. Though Docker has become popular, the Docker daemon runs as the root user and each container is a child process owned by the Docker daemon. This is a design flaw that could be exploited to obtain escalated privileges [59], which makes Docker unsuitable for HPC environments and other shared resource platforms (e.g. Compute Canada).

Singularity [59] offers a secure alternative to Docker for containerizing applications. Additionally, in order for Docker to run on a traditional HPC machine, replication of a virtual machine cloud-like environment is required which is an unreasonable task [59] in user-space. Singularity's implementation integrates with any resource manager and also includes a Simple Lunix Utility for Resource Management (SLURM) plugin that allows SLURM jobs to be run natively from within a Singularity container [59, 60]. This makes Singularity a candidate for future management and portability of analysis environments but is not explored in this thesis.

Graphical Workflow Management Systems Many scientists turn to Graphical Workflow Management Systems (GWMS) as a response to a lack of familiarity with underlying computational environments. GWMS take the burden of tool configuration off of users in addition to making the execution easier, though GWMS require some initial setup. These systems provide a GUI to run command line driven tools and build workflows using the outputs of some tools as inputs to others. The point-and-click interface allows users to run complex workflows while offloading some accountability onto the platform itself.

The Galaxy Project [13] has emerged as the most popular GWMS in the bioinformatics community. The Galaxy Project empowers non-computational researchers to use command line driven bioinformatics tools and build workflows. The Galaxy Project supports cloud-computing services for the processing of large datasets and other computationally expensive tasks [13]. Interactive, web-based documents are created to describe each experiment to communicate what was performed [13]. The Galaxy Project's most important feature is the empowerment of users to run complex workflows without having to learn how to program or interact with the command line, contributing to the project's accessibility. Additionally, developers can add new command line driven tools by simply writing a configuration file and letting the Galaxy Project framework to abstract the instructions for them [13]. The Galaxy Project tracks data provenance and user annotations when running an experiment. For transparency, The Galaxy Project allows sharing of workflows via public repositories allowing others to reproduce past experiments and reuse the workflows on other datasets.

A major limitation of the Galaxy Project platform is that the workflows are run on a computational environment that houses the Galaxy Project workflow and thus does not work in user-space. Since around 2007, the Galaxy platform has enabled non-computation researchers to build and execute bioinformatics workflows. The Galaxy Project framework allows users to have their own instance of the Galaxy Project on their own Unix-based machine and there are also public instances for users who to run workflows on others' hardware. *Omics* data could possibly contain sensitive information so public Galaxy Project instances should not be used in those cases. Further, analyses are constrained by the hosts' hardware as well as the number of people waiting for workflows to run. For example, the main public Galaxy Project server runs approximately 245,000 jobs each month [61]. Public Galaxy Project servers also have resource quotas. The Galaxy Project states, "Some work is simply too large for the public resource" [61]. The Galaxy Project requires an administrator to initialize an instance of the computational environment. Additionally, workflows are only reproducible on the machine they are executed on since computational environment components are

not captured. Furthermore, there is an abstraction layer between the users' input and the commands that are run since the Galaxy Project works with tools through configuration files. In this case, it is easy for users to only view these tools as black boxes.

Another popular GWMS is the VisTrails [62] platform allowing scientists to design and visualize workflows. The runnable units of the VisTrails platform are dataflows, which are a sequence of operations used to create a visualization [63]. The serve both as records of provenance as well as a formula to automate future runs. Users of this platform can explore the parameter space of their data, query visualization history, and comparatively visualize multiple runs [63]. VisTrails does not support HPC environments. It is the intention of VisTrails to produce a large amount of visualizations however, it doesn't support high-throughput data processing.

The EMBOSS suite [64] contains hundreds of command-line tools to support common bioinformatics tasks such as sequence alignment and nucleotide pattern analysis. EMBOSS Explorer [65] is a web interface to the EMBOSS suite of tools and is the first to demonstrate that user interfaces can be dynamically built from Ajax Command Definition (ACD) files. ACD files describe the parameters a program requires as well as contains information about the input and output requirements of each program. Tools using a format such as ACD allow developers to define their own program interface in contrast to the standardized interfaces proposed in the BioBoxes work [34]. This is ideal as it does not require the developer to adopt an interface they may not agree with.

3.2 Infrastructure As Code

Ansible [15] and its alternatives can be categorized as infrastructure as code platforms and loosely fall under the umbrella of software installation middleware. There are three common alternatives to Ansible: Puppet [66], Chef [67], and SaltStack [68]. Infrastructure as code platforms are mainly for automated configuration management of large infrastructures and continuous integration of applications. Since they are not typically used in academic work, parts of this section are based on anecdotal evidence and knowledge about the software.

Infrastructure as code platforms have two types of users: infrastructure managers and application builders. Infrastructure managers automate and manage machine provisioning while application builders manage application environments and dependencies. Though Puppet, Chef, and SaltStack can configure and provision machines, they each have their own strengths. Puppet specializes in provisioning, while Chef and SaltStack are mainly configuration management platforms.

Since Puppet and Chef are intended to support enterprise-level work, performance is a primary concern and they are implemented in Ruby, whereas SaltStack and Ansible are implemented in Python. Python is more accessible than Ruby to developers and those in the scientific community since many tools are written in Python and Python is a common scripting language used by computational and non-computational researchers. Many researchers with basic programming knowledge have some idea of how to use Python.

Thus, SaltStack and Ansible have a lower barrier to entry than Chef and Puppet

Puppet, Chef, and SaltStack are all based on an agent-master architecture meaning a master server controls the configuration information and each agent node requests its own configuration state from the master. The agent-master architecture has a difficult initial configuration in order to be used since a server must be designated as the master. It is commonly accepted that Chef, in particular, has an extremely difficult learning curve. In contrast, Ansible is agentless meaning it does not require a central control node. Instead, Ansible executes playbooks on remote machines through the SSH protocol [39] meaning playbooks can be run from any computer supporting SSH. Additionally, Ansible playbooks can be run locally to configure a personal environment without the use of SSH.

3.3 Marker Gene Analysis

Marker gene analyses employ DNA barcodes to catalogue biodiversity because they can be used to identify different organisms [20]. These methods produce a profile of present organisms within a sample since DNA barcodes are universal to all relevant taxa. Marker gene methods for microbial profiling are preferred when the goal is to obtain high-level overviews of microbial communities [69]. Identification and tracking of bacteria heavily relies on the sequencing of two genes: 16S rRNA and cpn60.

Marker genes often have multiple conserved regions allowing for universal PCR primers to be designed to target variable regions. PCR primers are specified DNA sequences that bind with a specific section of DNA during PCR. By binding with a target section, the target DNA sequence is built and amplified during subsequent rounds of PCR. Sequencing protocols are often designed to sequence DNA fragments shorter than twice the read length, thus producing reads from both ends of a library [2]. As sequencing accuracy is imperfect, sequencing a DNA fragment from both ends raises confidence in base calls.

The 16S rRNA gene is the most commonly used marker gene for bacterial profiling. It contains several highly variable regions that can be treated as DNA barcodes for taxonomic assignment [20]. There are two main issues with using the 16S rRNA gene for microbial profiling. First, the existence of 16S rRNA paralogs complicates use. Multiple copies of the gene can yield an ambiguous classification and abundance estimation. Second, 16S rRNA studies are often limited to genus level resolution [69].

Protein coding genes have been shown to have superior resolution of closely related bacteria compared to 16S rRNA [70]. The cpn60 is one such gene. It has a larger barcode gap than 16S rRNA (552-558 bp [20]). This suggests the cpn60 Universal Target (UT) gene is a more ideal target for species and subspecies-level characterization of marker gene data [71]. Jayaprakash et al. [72] demonstrated this by identifying distinct strains of *Gardnerella vaginalis* using targeted amplification of the cpn60 UT, where 16S rRNA work prior mistakenly identified the multiple sub-types of *Gardnerella vaginalis* as 1 species within the human vaginal microbiome. The mis-classification by 16S rRNA has lead to an incorrect understanding of the complex nature of this key bacterial member, *Gardnerella vaginalis*, of the vaginal microbiome.

Marker gene analyses commonly group sequences into Operational Taxonomic Units (OTU), a concept introduced through the work of Sokal and Sneath that defined numerical taxonomy [73]. Creation of the OTU system was motivated by the necessity of aggregating sequence data to a manageable size since cataloging the characteristics of all organisms individually is unreasonable in the high dimensions present in microbial profiling [73]. By using Sokal and Sneath's taxonomic system, grouping taxa has a higher predictive value than individually because of high sequence constancy [73].

3.4 OTU Formation

The most commonly implemented method of OTU aggregation is by clustering sequences based on a similarity threshold (typically 97%) to reduce the dataset to a manageable size [74]. One sequence per cluster is assigned as an representative and all sequences within each cluster (an OTU) adopt the details of the representative sequence as their own. Threshold-based clustering has limitations including overestimating evolutionary similarity between the sequences being compared since sequence similarity is not necessarily based on evolution. The commonly used similarity thresholds are relatively low, resulting in similar sequences of different organisms being classified as the same OTU; however, the threshold must be low to account for random sequencing errors [75]. This makes it impossible to identify organisms that differ by a small number of nucleotides [75]. OTU clustering is also inconsistent because it relies on a reference database for taxonomy classification but since an arbitrary member of each cluster is chosen as the identifier, choosing a different representative sequence may yield a different taxonomy classification.

Microbial Profiling Using Metagenomic Assembly (*mPUMA*) [25] is a computational approach that allows for *de novo* assembly of OTUs. The analysis of cpn60 UT sequencing data relies on this pipeline to produce an OTU table. This allows for identification of single nucleotide OTU variants. The method of *de novo* assembly does not require a reference database of sequences to work with which empowers users to discover novel OTUs. As a result, organisms not previously documented in existing databases can be identified.

Assembly can be done using a number of algorithms such as Trinity [76] or gsAssembler (`https://www.bioz.com/result/gsassembler%20software/product/Roche`). gsAssembler employs an Overlap-Layout-Consensus (OLC) method which is heavily influenced by coverage depth [25, 77]. Coverage depth of the sequences must result in full-length coverage of the target. A more ideal alternative is Trinity's de Bruijn graph (DBG) method to decompose the sequences into k-mers because it is not affected by coverage depth [25]. Additionally, though performance can vary greatly depending on the dataset, Trinity was observed to consume fewer resources when dealing with extremely diverse datasets [25]. gsAssembler has internal read tracking allowing taxon abundance calculation [78]. In contrast, Trinity requires the use of a post-hoc step for read tracking because the DBG data structure reduces sequences to component k-mers and thereby loses the ability to track individual sequences in the overall assembled results [25].

After assembly, PCR primers are removed and the assembly chimeras are filtered from the OTUs. The

remaining reads are clustered at 100% identity using CD-hit [79] to remove all redundant sub-sequences [25]. When analyzing protein-coding barcode sequences, BLASTX [80] can be used for translation to a peptide sequence [25]. The nucleotide and peptide sequences are then run through a variety of tools to complete the exploratory analysis. In particular, T-Coffee [81] produces multiple sequence alignments and FastTree [82] creates a phylogenetic tree of the OTUs, which can be used for calculation of UniFrac [83] distances.

Identifying Amplicon Sequence Variants (ASV) is a supervised method for identifying sequence diversity while removing random sequencing errors to produce highly refined OTUs [75]. The literature also refers to OTUs produced by ASV methods as either single nucleotide variants, sub-OTUs, and zero-radius OTUs. Two of the most popular ASV algorithms are Deblur [84] and DADA2 [3]. Both algorithms profile sequencing errors to determine real biological sequences present in the samples. Deblur operates on a per-sample basis to denoise the sequences to reduce the computational demand, allowing datasets to be analyzed on personal computers. This is in contrast to the DADA2 algorithm that pools the samples together to learn the error profiles. Nearing et al. [85] found that DADA2 consistently identified more ASVs while Deblur had slightly larger abundances for some ASVs, though they were similar. DADA2 was chosen for the microbial profiling pipeline in order to possibly identify more novel ASVs.

A characteristic microbial profiling computational analysis starts with paired-end FASTQ data. The goal is to process the data to produce an ASV table and taxonomy table for downstream statistical analysis. Prior to statistical analysis, Taye et al. [29] removed PCR primers from the reads, merged the paired-end reads, and produced the ASV and taxonomy tables with the DADA2 [3] algorithm. The Divisive Amplicon Denoising Algorithm (DADA) [86] built into DADA2 is trained on a model of errors in Illumina-sequenced amplicon reads [3]. The algorithm compares the count of sequences consistent with the error model in order to determine if the sequence was produced erroneously [3]. A later step in the pipeline removes chimeric sequences. Chimeras are PCR artifacts caused by two or more biological sequences incorrectly joined together [87]. Chimeras are a common artifact in microbial profiling datasets that must be removed as they do not represent a true microbe. Approximately 5% of sequences within libraries are thought to be chimeric sequences; however, 16S studies have shown to have upwards of 30% chimeric sequences [88]. Another feature in the DADA2 package is that DADA2 contains multiple methods of chimera removal. Taye et al. [29] analyzed a total of 887 biological samples with a DADA2-based process, demonstrating it can scale to handle large, modern datasets.

3.5 Addressing Computational Scale with Contextual Information

Since around 2008, the cost of sequencing has dropped at a rate higher than Moore's Law [89]. Moore's law refers to the number of transistors on an integrated circuit doubling roughly every two years and is applicable to many other trends. Moore's law does not translate to the capacity to analyze NGS data thus exposing a bottleneck in the storage and analysis of sequencing data. As the sequencing cost is dropping at a rate

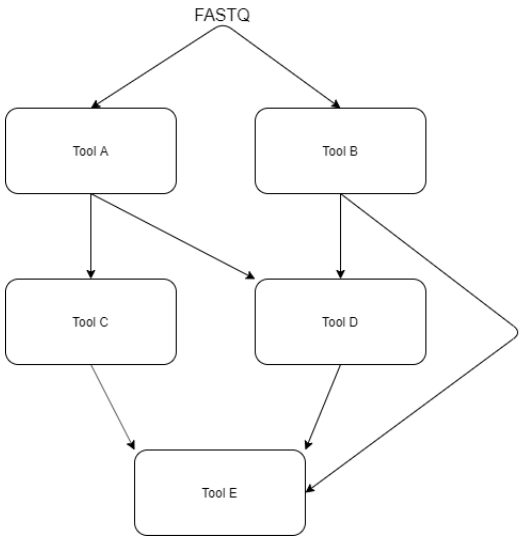


Figure 3.1: A simple ontology example for bioinformatics tools. The nodes represent bioinformatics tools and the edges represent data flow. In this example, Tool A and Tool B accept FASTQ data. The output of Tool A is accepted by both Tool C and Tool D and the output of Tool B is accepted by Tool D and Tool E. Too E also accepts outputs from Tool C and Tool D.

quicker than the computational cost, the bioinformatics cost rises [90]. The cost in this case not only refers to money but also time in data cleaning, processing, analysis, and validation. Different levels of automation can be explored to lower the bioinformatics cost, from small tasks and abstracting to large pipelines. Some tasks are quite easy to automate, such as file conversions and sequence alignments but as workflows become more complex automation implementation is not obvious. One way towards automation in bioinformatics is by using contextual data that provides information about the data [90]. In microbial profiling, contextual data often includes geographic data, temporal data, and sampling procedure [90].

Contextual information can also be utilized to distinguish disparate bioinformatics resources [91]. The bioinformatics space contains an immense set of tools with many sharing some of the same semantics. This has led to the use of the semantic web and ontologies for data integration and interoperability of bioinformatics tools [91]. Generally, the semantic web and ontologies can be thought of as Directed-Acyclic Graphs (DAGs). A DAG is a finite directed graph in which there exists no path from any vertex back to itself. The nodes of an ontology for bioinformatics tools may be bioinformatics tools and the edges are properties shared by the nodes [91]. An example of a simple bioinformatics ontology is shown in Figure 3.1. The edges in the graph represent the dataflow and the nodes are tools. Bioinformatics ontologies can enable workflow building by tracing paths through the graph and discovering tools that are semantically similar. The BioMOBY Project [92] is one example using the semantic web. BioMOBY focused on service description, data discovery, and simple input/output object type definitions [92]. The BioMOBY Project was an integration and interoperability platform that allows searching and retrieval based on hierarchies of tools and objects to explore and retrieve data using existing standards. Building of BioMOBY, the SADI platform also uses the web services paradigm

and semantic web to enable the creation of bioinformatics applications while enabling automated discovery and pipelining of these services [91]. The benefit of a platform like SADI is in the automated discovery of services. Platforms such as the Galaxy Project [13] help in abstracting away difficult parts in building and running tools but the automated discovery in SADI identifies semantically similar pipelines to run given a specific goal. The work undertaken in this thesis has the potential to empower semantic methods for computational reasoning in bioinformatics pipelines. Potential future work could include a semantic structure for bioinformatics tools to chain tools together and build pipelines in an unsupervised fashion.

Semantically similar tools are pervasive in bioinformatics, which may mislead researchers. Tools that are semantically similar may have vastly different algorithms thus resulting in different data products and computational performance. For example, the Deblur and DADA2 algorithms are semantically similar in that they produce an OTU table but their denoising algorithms are different so the OTU tables produced will be different. Whether through lack of knowledge or bias, researchers adopting a semantic web approach may arbitrarily choose one for their analysis assuming the results would be identical. Without understanding the underlying algorithms involved, obtaining a data product may be seen as a goal rather than an experimental result to compare with other paths in the semantic tree. Further, using a web services approach for producing data products limits analyses to the servers hardware. This could result in lengthy analysis time and high infrastructure costs for the individuals housing the tools.

Automated hardware-agnostic data analysis provides a stable way of exploring semantically similar tools. A system of this nature enables researchers to conduct *in silico* experimental replicates of many tools with varying parameter sets. This allows rapid exploration of parameter spaces of a single tool as well as cross-tool comparisons so researchers can make informed analysis decisions.

3.5.1 SLURM

Simple Linux Utility Resource Management (SLURM) [60] is an open-source utility for compute cluster management. SLURM functions in 3 main ways: allocating resources, executing and monitoring work, and queuing jobs when resources are unavailable. User interaction with SLURM is done via command line interface where users can submit jobs, cancel jobs, monitor jobs, and monitoring system state. Because of its simplicity, SLURM is commonly used and is present on the University of Saskatchewan's Plato cluster [93] as well as on Compute Canada's Clusters [55]. SLURM is easily configured by system administrators and its configuration can be updated and modified without affecting running jobs. Further, SLURM is fault tolerant and is able to handle failure of jobs and even nodes without terminating workloads. SLURM has very little overhead when running, only occupying around 2MB of memory[60], and can scale to include thousands of nodes as demonstrated by Compute Canada Cedar with 1,542 nodes.

SLURM users submit jobs using shell scripts that specify compute resource requirements such as threads and memory, and other parameters such as run timeout and run name. For pipeline jobs where outputs of one job are inputs into a subsequent job, SLURM allows users to define dependencies so jobs will execute in the

intended order. For all available job management parameters, refer to Yoo et al. [60] or the documentation of a specific cluster.

By default, SLURM operates on a First-In First-Out priority system. There are however plugins available for priority job scheduling. For example, the Compute Canada Cedar cluster recognizes differing priorities based on accounts. Resource allocation competitions are held ever year in which research groups apply for cluster allocations. Successful applicants will have higher priority than default users and thus will have their jobs execute sooner on the cluster.

SLURM clusters are intended for running jobs in parallel. In microbial profiling, parallel jobs appear in two general forms. First, Cutadapt only runs on one set of paired-end FASTQ files or one single-end read. Microbial profiling datasets often have on the order of tens or hundreds of samples, requiring Cutadapt to run multiple times. In this case, either N jobs can be submitted where N is the number of matched paired-end files, or an array job can be submitted in which the same command is run for each set of paired-end files. Running array jobs are idiomatic but both methods yield the same results. The second form of parallel job is when the tool has parallelism built into it. An example of this is the DADA2 sample inference step in which a boolean value can be set for multicore functionality. In this case, one job can be submitted and many nodes, cores, and other resources can be requested.

3.6 Ansible Concepts

The following section defines Ansible concepts as used throughout this thesis. The relevant definitions were compiled from the Ansible documentation [15] and are used throughout this thesis. The concepts introduced are common to all uses of Ansible so as this thesis extends Ansible, adaptations of the concepts are defined. Many of the concepts have much more information and features outlined in the Ansible documentation [15] that are outside the scope of this thesis.

3.6.1 Nodes

Nodes are computers involved in an Ansible run. There are two types of nodes: control nodes and managed nodes. The control node is the machine from which Ansible is run and in each Ansible run there is only one control node. This machine requires Ansible and Python to be installed. The control node can be a personal machine, shared machine, or server. The control node must also be a Linux or MacOS machine.

Managed nodes are the network devices and servers that are managed with Ansible. They are typically referred to as *hosts*. Each Ansible run is comprised of one or more hosts.

3.6.2 Inventory

An Ansible inventory is a collection of hosts. The two main formats for Ansible inventories are INI format and YAML format. INI format is the most common. An example INI format inventory is shown in Figure 3.2.

```
parent.host.ca

[Database]
database.usask.ca

[Virtual-Machines]
206.167.182.107
206.167.183.104
206.167.181.30

[Shared-Servers]
eightball.usask.ca
beans.usask.ca

[HPC-Clusters]
plato.usask.ca ansible_ssh_user=abc123
cedar.computecanada.ca ansible_ssh_user=fooName

[University-Machines]
plato.usask.ca
eightball.usask.ca
beans.usask.ca

[Compute-Canada]
206.167.182.107
206.167.183.104
206.167.181.30
cedar.computecanada.ca

[Virtual-Machines:vars]
ansible_ssh_private_key_file=path_to_pem_file
input_data=$HOME/in_data

[University-Machines:vars]
ansible_ssh_user=abc123
input_data=$HOME/some/path/data

[Shared-Servers:vars]
ansible_ssh_user=abc123
```

Figure 3.2: Example Ansible inventory.

Each Ansible run is executed against one or more hosts in the inventory depending on the organization of the inventory. Inventories are organized by groups and by default there are 2 groups: all and ungrouped. Figure 3.2 shows a single ungrouped host on line 1. Groups are a powerful way to organize inventories when many different types of nodes are managed. Other than the default groups, Figure 3.2 contains 6 groups: Database, Virtual-Machines, Shared-Servers, HPC-Clusters, University-Machines, and Compute-Canada. The heading in square brackets are the names of the groups and some hosts appear in more than one group.

In configuration management, hundreds or thousands of nodes may be managed so nesting of managed nodes is necessary for organization. For example, a company may organize servers by province, cities within the provinces, and testing and production servers in the cities. Inventory groups can be nested within each other into parent-child groups. Referring to the previous example, the server province group would be the parent of the server city group and the server city group would be the child. In the case of bioinformatics, the number of managed nodes would likely be considerably less (on the order of 10s of servers). As a result, forcing inheritance groups may be unnecessary and can be error-prone. Instead, organizing hosts by groups and defining variables for them is likely sufficient.

To enhance reuse, variables can be defined within inventories. The variable precedence from lowest to highest is as follows: all group, parent group, child group, and host. Lines 20 and 21 in Figure 3.2 show examples of host-level variables. Lines 40 to 49 show group-level variables. Variables that can be included in inventories are Ansible variables such as username, private key location, and connection port, as well as user-defined variables such as the path where the .biolighthouse directory is to be created and the path to the input data. Security issues with inventories exist since potentially sensitive information may be contained within them. For example machine addresses, usernames, and private key locations could be exploited and thus inventories should not be shared.

3.6.3 Modules

Ansible modules are the code that Ansible executes. Ansible has many modules by default that can be executed on hosts in the inventory. For all core Ansible modules view the Ansible documentation [94]. Each module has a defined interface and returns information in JSON format [95]. Modules are written in Python and build a command to be executed usually on hosts but can also be executed on the control node. Modules can be easily written to perform tasks built-in modules cannot.

3.6.4 Tasks and Roles

A task in Ansible is a unit of action. Tasks invoke a module based on user-defined parameters. Ansible roles load variables and tasks based on a known file structure. There are 7 possible sub-directories that make up a role, shown in Figure 3.3, and at least one must be present. Each directory in use must contain a main.yml file. The templates directory are not used for this thesis so the other six will be introduced. Tasks contain one or more files with ordered lists of tasks to be executed on the nodes. Handlers are actions that can be

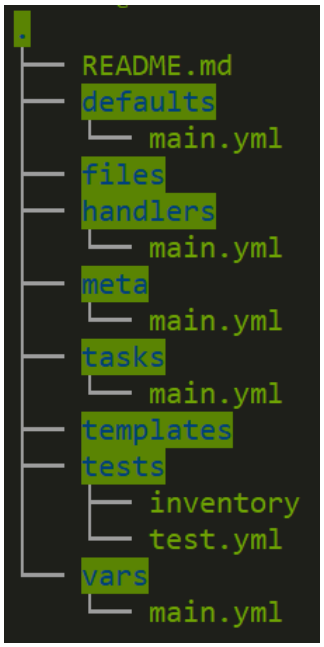


Figure 3.3: File tree showing the structure of Ansible roles.

```yaml
---

- hosts: Shared-Servers
  tasks:
   - name: Transfer data from one host to all shared servers
     synchronize: src="{{ path_to_data }}" dest="{{ path_to_put_data }}"
     delegate_to: Database
  roles:
    - ansible-role-cutadapt
    - ansible-role-flash2

- hosts: HPC-Clusters
  tasks:
   - name: Transfer data from one host to all HPC Clusters
     synchronize: src="{{ path_to_other_data }}" dest="{{ path_to_put_data }}"
     delegate_to: Database
  roles:
    - ansible-role-dada2
```

Figure 3.4: Example of an Ansible playbook.

triggered at the end of a block of tasks and a handler is only triggered once even if requested by multiple tasks. Handlers are usually tasks that clean up an environment after Ansible's execution. For example, a handler may remove an install script after it is run or restart a service after installation. Handlers are referenced by a globally unique name.

Defaults contains YAML files with default values for variables used by an Ansible run.

Vars contains YAML files defining the variables used by the Ansible run. This is different from the Defaults file as the variables in Vars are not initialized with any values. The Files directory contains any files that may be needed by the Ansible run. The Meta directory contains information about the role. The fields in the meta file are: descriptive information, the license, platforms the role was tested on, other roles that are dependencies for the role in question, and flags for Ansible Galaxy.

3.6.5 Playbooks

Playbooks in Ansible are the basis for configuration management on multiple machines. Playbooks define a configuration process in a particular order and can operate synchronously and asynchronously. Playbooks are written in YAML which is human-readable. The intention of using YAML is for playbooks to be a model of a process rather than appear as code. Playbooks consist of one or more plays that map hosts to roles. The Ansible documentation explains plays as a sports analogy. For example, consider the sport of hockey. Loosely, there are 3 types of positions: forward, defence, and goalie. The positions can be thought of as hosts that have different jobs. The forwards, defence, and goalie all have their own plays to execute asynchronously; i.e forwards want to score, defence wants to defend their end of the ice, and the goalie wants to stop the puck and all of the plays can occur at the same time. Further, there are multiple lines of forwards and defence players. Since they all have the same plays to execute, multiple forward lines are analogous to synchronous tasks executing on multiple hosts.

Figure 3.4 is a sample playbook that may be used for bioinformatics tools. The first block is run on all shared servers as specified on line 3. Lines 4 to 7 specify a task to run on all shared servers. The task run transfers files from a database to a specified path on the shared servers. Lines 8 to 10 specify the roles to be run on the shared servers. In this case, the roles run are to configure Cutadapt [1] and Flash2 [2] on the hosts. For this example, the intention is to configure the tools on the hosts that do not need HPC resources. The second block is to be run on HPC clusters as specified on line 12. Lines 13 to 16 define a task that transfers data from the database host to the HPC clusters. In this example, the files transferred would be files merged with Flash2. Lines 17 and 18 specify the role to be run on the HPC clusters to configure DADA2 [3]. DADA2 has multi-core functionality and can take a considerable amount of time to run on datasets.

4 Research Goals

The overarching objective of this thesis was to create an accessible, reproducible, and transparent platform for configuration and execution of bioinformatics workflows. The complex nature of computational workflows in bioinformatics presents several complications to developers, bioinformaticians, and tool users. The hypothesis for this thesis was that Ansible can be extended to create an accessible, reproducible, and transparent platform for configuration and execution of bioinformatics workflows. Ansible was identified as an ideal technology to use for development of a robust research system as it promotes accessibility, reproducibility, and transparency in an IT automation context. Sub-objectives were identified in order to extend Ansible to satisfy the robust principles for bioinformatics workflows.

Building off of Figure 2.1, Figure 4.1 shows a more detailed interaction between developers, bioinformaticians, and tool users. The life scientist actor is omitted from this figure as they are not directly part of the computational interaction. The core functionality of Ansible was intended to be used as much as possible for the configuration management steps, including setting up the environment and file structure for analysis. As mentioned in Section 3.6.3, modules accept parameters in order to build a command to be executed on a command-line interface. Since the bioinformatics tools intended to be supported by *The BioLighthouse* are required to have a command-line interface, developing modules was identified as the best method to extend Ansible's functionality to execute bioinformatics tools. The execution of bioinformatics tools using developed modules is the core functionality of *The BioLighthouse*.

Ansible modules are intended to be concise and have a well-defined functionality, and follows the UNIX philosophy of doing one thing well [15]. Modules attempting to encompass too much functionality forces users to provide too much information in playbooks which may be unclear and error-prone. For example, consider the Burrows-Wheeler Aligner (BWA) top-level interface shown in Figure 4.2. The BWA has 14 use cases implemented, each having their own required and optional parameters. Instead of attempting to encapsulate all 14 use cases into a single module, 14 separate modules should be developed so each module has a well-defined functionality. The reason for developing multiple modules in this case is for the user. If too much is encapsulated into a single module, it convolutes the task invocation by requiring the user to understand the function they require and following that path in the Ansible role. The developer's role as shown in Figure 4.2 is to write a configuration role for their tool and a module for execution of the tool. In the case that tools require multiple modules for execution, still only one configuration role is required assuming proper configuration of the tool enables all features. Continuing with Figure 4.1, the bioinformatician pulls all tools required for an analysis pipeline from Ansible Galaxy. The bioinformatician's task is then to write

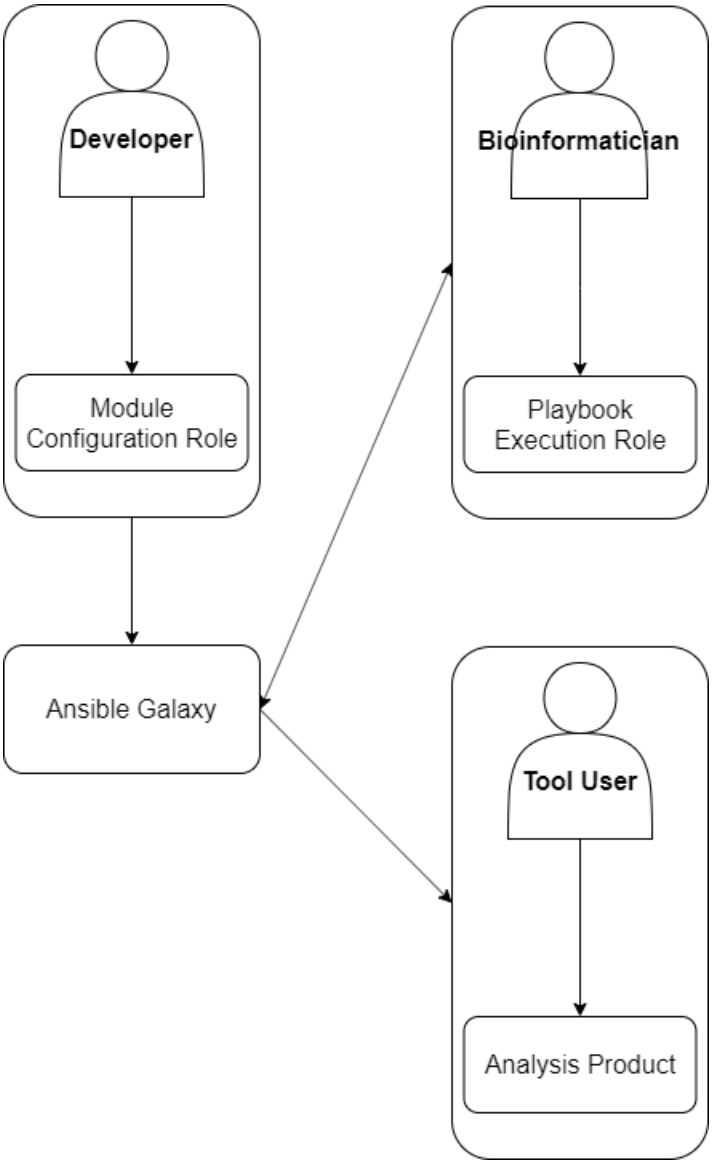


Figure 4.1: The actors involved in a computational analysis as described by this thesis. A developer of a tool writes one or more Ansible modules encapsulating the functionality of their tool. Accompanying the modules is a single Ansible role for configuring that tool. The modules and configuration roles are pushed to Ansible Galaxy where they are pulled by a bioinformatician. The bioinformatician generates an Ansible role for executing the modules in order to conduct an analysis pipeline as well as a playbook for organizing the workflow. The execution role and playbook are pushed back to Ansible Galaxy where they are consumed by the tool user for input of their own parameters and execution of the pipeline.

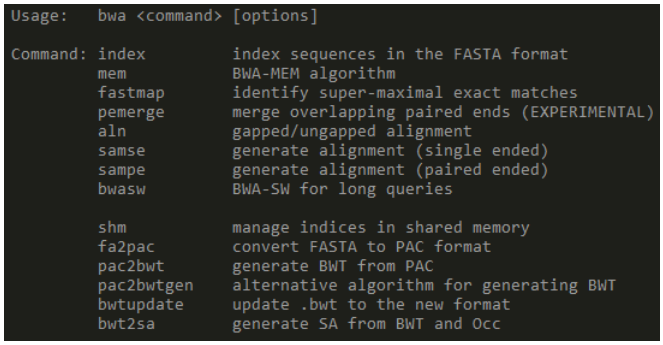


Figure 4.2: Top-level command interface for the Burrows-Wheeler Aligner (BWA) [7]. BWA has multiple use cases and thus multiple different functions. For each command on the top-level interface, there are different required and optional parameters. To ensure well-defined functionality of Ansible modules, tools such as BWA with multiple functions should be broken into multiple Ansible roles.

an additional role that uses the modules to execute an analysis pipeline, as well as a playbook to specify an order for the roles to run. The execution role is intended to be used as a template by the tool user. The tool user's task is to populate the execution role with their parameters, and run it to generate their analysis product.

Accessibility and reproducibility are made difficult by the breadth of computer science concepts required to complete bioinformatics tasks. Differing operating systems and system architectures require time to effectively learn, which in multidisciplinary fields such as bioinformatics is not the main focus for many researchers. Further, it is often the case that analysis machines are shared (e.g. shared lab machines and High-Performance Computing (HPC) clusters). Therefore the first sub-objective for this thesis was to demonstrate the workflow system has no platform dependence. Based on this objective, the hypothesis was that the platform would would return identical outputs for each tool executed across multiple computing environments. Platform independence not only strengthens reproducibility of a system, but also makes the system accessible to more users.

The Linux *diff* command [4] was used to test platform dependence of *The BioLighthouse* and the tools implemented into it. The *diff* command compares two files line-by-line and outputs differences in the files. If the *diff* command yields no output, the contents of two text files are identical.

The scope of testing for platform dependence was the 2 most recent major versions of 3 Linux distributions, a shared university machine, a shared HPC cluster, and Oxford Nanopore Technologies MinIT device. The Linux distributions tested were Ubuntu, Debian, and CentOS. According to W3Techs' survey of the top 10 million websites, 39.3% use Ubuntu, 17.7% use Debian, and 16.6% use CentOS [96] suggesting testing coverage for most Linux users.

SLURM [60] is anecdotally the most common cluster manager used for Canadian Research clusters and is used by the University of Saskatchewan and Compute Canada (the HPC cluster tested in this thesis). Based on the prevalence and accessibility of SLURM clusters specifically in Canada, SLURM was identified as a

reasonable HPC cluster manager to use.

Conda is a common package manager used by scientific researchers. Users have access to operating system dependencies, programming languages, and software tools through Conda's numerous channels, all available for installation into Conda virtual environments. The Bioconda channel contains over 7000 bioinformatics packages [5] available for quick install and is both language- and operating system-agnostic, making it ideal for this thesis for two main reasons. Since Conda is system-agnostic and operates in user-space, it is accessible to users of a variety of machines including shared machines. Additionally, Conda virtual environments have the ability to isolate analysis environments to eliminate collisions of previously installed packages and dependencies. The isolation of analysis environments by Conda provides consistency when reproducing results, motivating the next sub-objective. The platform developed for this thesis must successfully manage and operate within Conda environments.

The third sub-objective of this thesis was to demonstrate the platform has no data-scale dependence. To test this, it was hypothesized that a platform demonstrating data-scale dependence would behave differently on datasets of differing orders of magnitude. The parameters of behaving differently include completion of runs of the platform and identical results of deterministic tools. Microbial profiling datasets can be in a variety of scales ranging from just a few samples to hundreds of samples or more. To test data-scale dependence, 3 datasets at differing orders of magnitude were explored. The 3 datasets consisted of 1 sample, 10 samples, and 100 samples. Timeouts when working with larger datasets and non-deterministic results relating to the data are the most likely cases for data-scale dependence.

Ansible offers many ways in which it is transparent. The developer and bioinformatician record *perspective provenance* in the roles and playbooks they develop. Similarly, the tool user records *retrospective provenance* when they populate the execution role with their parameters. The roles with the parameters run can be shared and used as a ledger for what was done for an analysis. Additionally, the role can be reused to verify results. Sharing of modules and roles was intended to be done with Ansible Galaxy.

5 Materials and Methods

This chapter outlines the methods used in the development and testing of the bioinformatics platform, *The BioLighthouse*. It begins by describing the platform development considerations when making the platform accessible, reproducible, and transparent. Following, the testing of *The BioLighthouse* for data-scale and platform dependence is described.

5.1 Platform Development

This section begins by outlining the method for actors using *The BioLighthouse*. Development of this platform was undertaken as the Developer, Bioinformatician, and Tool User actors described in this thesis and not an omniscient figure. The method for generating modules, roles, and playbooks spawned as results of this thesis are described in Chapter 6. Section 5.1.1 outlines how a developer implements that tool into *The BioLighthouse*. Section 5.1.2 describes the method of a bioinformatician to consume the tool built by the developer in order to build a pipeline, and Section 5.1.3 describes how a tool user executes the pipeline on their own data. Finally, Section 5.1.4 outlines the microbial profiling pipeline tested.

5.1.1 Implementation of a Tool into *The BioLighthouse*

The starting point for developer interaction with *The BioLighthouse* is a published computational tool with a command line interface. The developer must write a configuration role for the tool and a module for execution. For the configuration role, the developer first runs *The BioLighthouse* Conda configuration role (Section 5.1.4) on their computational environment. Use of Conda ensures configuration of a tool is accessible in user-space and reproducible virtual environments. The developer is to then minimally configure the tool within a Conda environment as they normally would. Since bioinformatics tools often have complex chains of dependencies, Conda and Bioconda were intended to be leveraged as much as possible when using *The BioLighthouse*. Using Conda and Bioconda offers consistency in providing a default medium when installing dependencies and eliminates building many dependencies from source. Other methods, such as Git are accessible by Ansible roles and can be used if tools are not available on Conda.

With the tool configured, the developer encapsulates the steps taken for configuration into a role for automated configuration of their tool. The specifics of building the configuration role are outlined in Section 6.1.2. Modules encapsulating the function of a tool are used by the execution roles for invocation and automated execution of the tool. The method of building modules for *The BioLighthouse* is described in

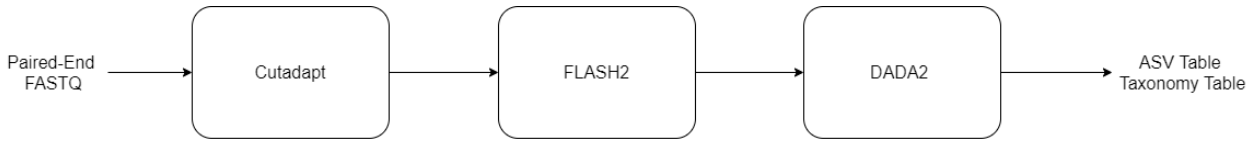


Figure 5.1: The microbial profiling pipeline tested in this thesis. This pipeline accepted paired-end FASTQ data, removed primers with Cutadapt, merged reads with FLASH2, and aggregated the data into ASV and taxonomy tables with DADA2.

Section 6.1.3. To ensure transparency, the module and configuration role is made accessible on Ansible Galaxy.

5.1.2 Pipeline Building

The bioinformatician interacting with *The BioLighthouse* is tasked with building a pipeline for configuration and execution of an analysis. Additionally, the bioinformatician may also stage the data to be analyzed onto the machine. The bioinformatician identifies a collection of *BioLighthouse* tools to run in a pipeline and pulls the configuration roles and modules from Ansible Galaxy. The bioinformatician writes an execution role that uses the modules they pulled to conduct an analysis, as well as a playbook specifying the order in which to run the roles. The method of writing the playbook and execution role are outlined in Section 6.1.5. The bioinformatician pushes the playbook and execution role to Ansible Galaxy.

5.1.3 Execution of Workflows

The tool user identifies a *BioLighthouse* pipeline they would like to run and pulls it from Ansible Galaxy. The user builds their inventory and populates the vars file (Section 3.6.4) with the tool parameter values they intend to run so instead of building a command-line command, the parameter inputs for a tool are defined in a YAML file. The user then executes the playbook and waits for their analysis to complete.

5.1.4 Microbial Profiling Pipeline

The instance of *The BioLighthouse* that was tested with a microbial profiling pipeline is shown in Figure 5.1. The microbial profiling pipeline was modified from the pipeline used in Taye et al. [29]. *The BioLighthouse* pipeline accepted a number of paired-end FASTQ files, removed primers with Cutadapt v1.17 [1], merged the paired-end reads with FLASH2 v2.2.0 [2], then produced an ASV table and taxonomy table with DADA2 v1.8.0 [3]. The tools were configured in virtual environments using Conda v4.7.12 [5], leveraging Conda channels (Bioconda [6]) for dependency installation as often as necessary.

Ansible modules, roles, and playbooks were developed for the microbial profiling pipeline and tested with Ansible v2.8.0 [15]. The outline that was followed was as described above. One or more modules were developed for each tool, as well as a role for configuring the tool. A single execution role and playbook

was written for automated configuration and execution of the pipeline. And finally, the parameters were populated and the pipeline was tested as described in Section 5.2.

5.2 Platform Testing

This section describes testing *The BioLighthouse* for reproducibility in the form of platform independence and data-scale independence. A single run through the microbial profiling pipeline tested is shown in Figure 5.1. The volume of testing was as follows. Three datasets as described in Section 5.2.2 were tested on the pipeline, each with 10 different parameter sets. The testing of each dataset was conducted on 9 machines as described in Section 5.2.1. As a result, the pipeline in Figure 5.1 was executed 270 times (3 datasets x 10 parameter sets x 9 machines). Since the same runs were repeated across all the machines, a total of 30 unique pipelines were run (3 datasets x 10 parameter sets). The data products output at each step of a unique pipeline were compared across all the compute environments using the Linux *diff* command [4]. Any output from the *diff* command would result in a failure of reproducibility.

Differences in tool output can occur at multiple locations in the software stack. In these cases, debugging started at the tool level and worked down to the operating system level. If a difference occurred at the tool level, steps were taken in order to identify what part of the tool gave caused the differing outputs and if there was a method for controlling this. A common source of differences in tool output is random number generation. Some tools with random number generation have ways of setting random seeds or saving models. In the cases in which tools had no method of controlling stochasticity, the output differences were identified as being caused by the tool and not *The BioLighthouse*. The next level of debugging occurred at the software dependency level. Debugging at the software level started with inspecting the Conda environments of the tool for differences in software dependencies and their versions. If the error persisted after this step, the differences were thought to occur at the operating system level. If the operating system dependencies were not available through Conda or any other method for installation in user-space, the tool and *The BioLighthouse* was deemed platform dependent.

The DADA2 sample inference step contained randomness when learning the sequencing errors so one of the parameters tested was the random seed. Setting the random seed ensured deterministic results. In the DADA2 chimera removal and taxonomy classification step, there is also randomness in training a Bayesian classifier. DADA2 had no method of saving models or setting a random seed for this step.

5.2.1 Scope of Platform Testing

Testing for platform dependence was performed in 9 computational environments. The two most recent major versions, as of November 2019, of CentOS, Ubuntu, and Debian were tested and are outlined in Table 5.1. The pipeline was also tested on two shared machines: Compute Canada's Cedar cluster [55], a University of Saskatchewan shared server, and Oxford Nanopore Technologies MinIT machine are outlined in Table 5.2.

5.2.2 Scope of Data-Scale Testing

A characteristic microbial profiling dataset is a collection of paried-end sequencing files in FASTQ format. Each set of paired-end files is the sequencing data from one sample. Three paired-end datasets consisting of 1, 10, and 100 samples were tested. The datasets were generated from the canola root and soil microbiomes in 2017 at three different sites.

The datasets consisting of 1 sample (Small Dataset) and 10 samples (Medium Dataset) were canola root microbiome datasets. The small dataset had 32,163 forward and reverse reads. The medium dataset had 276,101 forward and reverse reads with an average of 27,610 reads per sample. The 100 sample dataset (Large Dataset) was a canola root microbiome dataset that consisted of 2,792,192 forward and reverse reads, with an average of 27,922 reads per sample. The 3 datasets had differing orders of magnitude in sample number and read count in order to test data-scale dependence of *The BioLighthouse*.

The effect of data-scale was also tested on the DADA2 taxonomy assignment. For each dataset, the taxonomy assignment step was run 10 times given the same input. The count of differences in taxonomic assignment for each replicate was generated. The counts were separated by taxonomic level not including species as 16S rRNA does not provide species-level resolution.

Operating System	CPUs	RAM	Image Name
CentOS 6	4	22.5GB	CentOS-6-x64-2019-05
CentOS 7	4	22.5GB	CentOS-7-x64-2019-05
Debian Stretch	4	22.5GB	Debian-9.9.4-Stretch-2019-07
Debian Buster	4	22.5GB	Debian-10.0.1-Buster-2019-07
Ubuntu Xenial	4	22.5GB	Ubuntu-16.04.6-Xenial-x64-2019-07
Ubuntu Bionic	4	22.5GB	Ubuntu-18.04.2-Bionic-x64-2019-07

Table 5.1: Table showing the specifications for the virtual machines tested.

Machine	CPUs	RAM	Operating System
Cedar	101,424	485,826GB	CentOS 7.7
UofS Shared	48	264GB	CentOS 7.6
MinIT	6	8GB	Ubuntu 16.04

Table 5.2: Table showing the specifications for the shared machines tested. Specifications current as of January 2020. Additionally, the specifications for the Oxford Nanopore Technologies MinIT are included.

6 Results

6.1 Resulting Platform

This section describes the results for implementation of *The BioLighthouse*. It presents how the components of the platform were structured and implemented as well as the components that were written as a result. *The BioLighthouse* used Ansible's powerful configuration engine and module framework to describe compute environments and build tool execution commands. Configuration and Execution were separated into their own roles and analyses were stored in a defined directory structure.

6.1.1 Directory Structure

To organize analysis environments and avooid collisions with files already present, I implemented a directory structure for *The BioLighthouse*. The file structure used is shown in Figure 6.1. The entire workflow was written to a hidden directory named ".biolighthouse". On the first level in the ".biolighthouse" directory, a "software" directory was created for the installation of Conda and any other software dependencies. In addition to the "software" directory, a single directory was created for each analysis tool. The sub-directories for each analysis tool included "outputs" for the data products and "reports" for the logging files produced by the tools. The ".biolighthouse" directory was made in the user's home directory for all machines except Compute Canada as the SLURM cluster manager does not have permissions to write to user-owner directories in the home directory. Setting of this directory can be overridden in *The BioLighthouse*, and is automatically set in the roles as described in Sections 6.1.2 and 6.1.5.

In addition to the ".biolighthouse" directory, utilities used by *The BioLighthouse* modules are stored in the "/tmp/biolighthouse" directory . This is described in greater depth in Sections 6.1.2 and 6.1.5. The "/tmp" directory on Linux offers temporary, vulnerable storage of files.

6.1.2 Configuration Roles

Configuration roles implemented into *The BioLighthouse* were built to configure a single tool in a Conda environment. Since Conda environments were being used, *The BioLighthouse* Conda configuration role was required to run prior to all other execution roles. Figure 6.2 shows the layout of a configuration role. Environment setup is necessary for all configuration roles. Setting up the environment ensured the path to the ".biolighthouse" directory was set and transferred the Conda utility file to the "/tmp/biolighthouse".

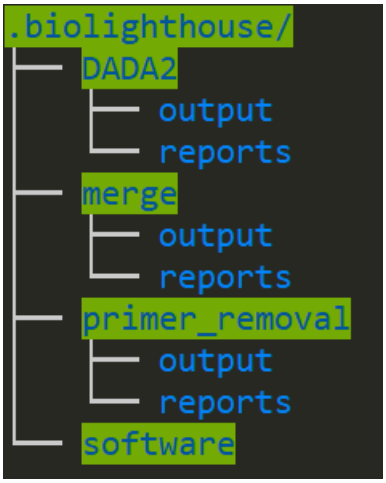


Figure 6.1: The directory structure for configuration and execution of *The BioLighthouse* roles. The first level of the directory contained a software directory along with a directory for each analysis tool. Two sub-directories are within the tool directories, one for storing outputs and one for storing reports generated by the tools. Conda and all other software is configured within the software directory.

A handler (Section 3.6.4) was notified to remove the "/tmp/biolighthouse" directory after execution. The second section shows creation and activation of the Conda environment. A Conda environment was created using the Conda create module written for this thesis. The activation of the environment was put in the ".bashrc" file and a handler was notified to remove the environment activation after completion of the role. Conda environments created by *The BioLighthouse* were named after the tool and were prepended with "bl_" to ensure no *BioLighthouse* environment collisions occurred. The final section shown in Figure 6.2 contained the configuration tasks for the tool and its dependencies. The template role shown in Figure 6.2 was released on Ansible Galaxy for future creation of configuration roles.

The pipeline formally tested in this thesis consisted of 4 configuration roles: Conda [5], Cutadapt [1], FLASH2 [2], and DADA2 [3]. In addition to these roles, I created an additional 9 configuration roles, which are described in Appendix A. The additional roles included PEAR [97] and Deblur [84] as alternatives to FLASH2 and DADA2 respectively, Globus [98] for setting up parallel file transfers, and other common bioinformatics tools for exploring and analyzing Next-Generation Sequencing (NGS) data. Finally, there were 2 additional roles to assist developers and bioinformaticians in generating configuration and execution roles. The roles were templates for configuration and execution roles so they do not need to do the environment and Conda setup.

6.1.3 Tool Modules

Ansible modules encapsulate a tool's functionality. The core functionality of *The BioLighthouse* are Ansible modules built to execute scientific tools. Generally, modules consist of 3 components: argument specification, logic, and return values. The argument specification contains developer-defined parameters as well as

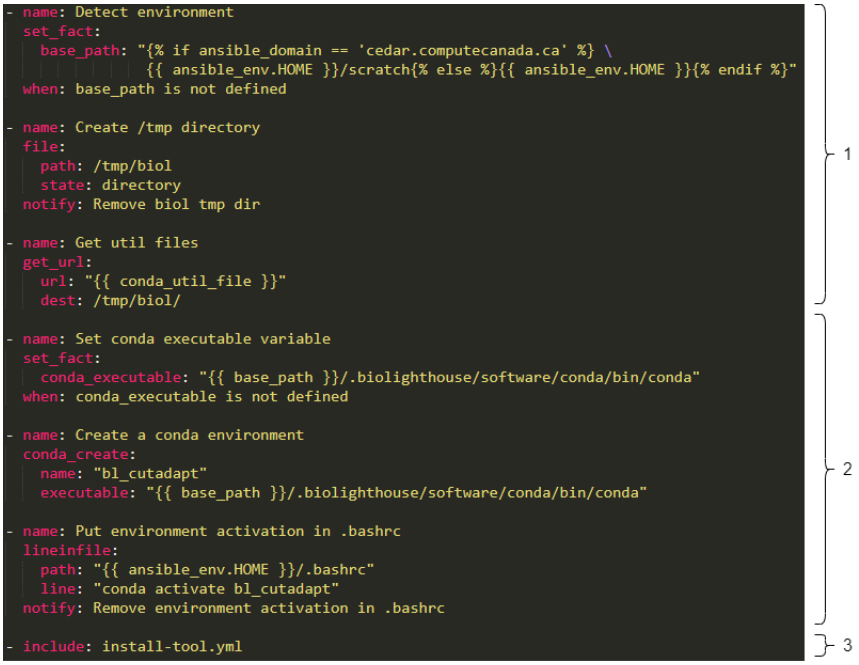


Figure 6.2: The structure of a configuration role. The first section sets up the environment by determining where the .biolighthouse directory is written and transferring utilities to the /tmp/biolighthouse directory. Creation of the /tmp/biolighthouse directory notifies a handler to remove the directory after completion of the role. The second section is the creation and activation of the Conda environment. Activation of the environment notifies a handler to deactivate the environment after completion of the role. The third section contains the installation steps for each tool and dependency required.

all possible parameters for a tool. A condensed version of the FLASH2 module's argument specification developed for this thesis is shown in Figure 6.3. The argument specification is a dictionary that allows the developer to require values for certain parameters and set default values. There are 4 user-defined parameters in Figure 6.3: "base_dir", "hpc", "executable", and "slurm_spec". "base_dir" defined where the ".biolighthouse" directory was written and was set by the role. "hpc" was a boolean variable to specify if the tool was to run on an HPC machine, and "executable" was an override variable if the tool was not in the $PATH[1]. The 'slurm_spec" option specified a dictionary of SLURM cluster parameters for execution of the tool on a SLURM cluster. The SLURM functionality was built as a utility into *The BioLighthouse* (Section 6.1.4).

The second component of modules built the command to be executed based on the argument specification. This component was dependent on the complexity of the tool being run. In the simplest case, the parameters were accepted and placed into a command based on a rigid structure. In more complex cases, the module was built to perform additional steps on the input data, file names, and input parameters in order to properly build the command. The final component of the module was to build the return values. Return values were stored in a dictionary and by default, Ansible suggests returning: if the state has changed, the numerical

[1]$PATH is an environment variable in UNIX-like operating systems that specifies the locations of executable programs.

```
def flash2_arg_spec(slurm, **kwargs):
    spec = dict(
        input_files=dict(type='path', required=True),
        base_dir=dict(type='path', default=expanduser('~')),
        hpc=dict(type='bool', default=False),
        executable=dict(type='path', default=None, required=False),
        """ Definition of all tool parameters """
        slurm_spec=dict(type='dict', default=slurm.slurm_arg_spec(), required=False)
    )
    spec.update(kwargs)
    return spec
```

Figure 6.3: The structure of the FLASH2 argument specification. It contains developer-defined arguments as well as all parameters possible for execution of a tool.

return value, the standard error, and the command that was run. In the case of execution modules, specifying state was ambiguous and so only standard error, the command that was run, and the numerical return value were returned.

The microbial profiling pipeline tested in this thesis required 6 modules to be developed. The modules were for creation of Conda [5] environments, installation of Conda packages, Cutadapt [1] in paried-end mode, FLASH2 [2] for merging, DADA2 [3] sample inference for removing sequencing errors, and DADA2 chimera removal and taxonomy assignment. An additional 7 modules were developed for *The BioLighthouse* but were not formally tested for this thesis. The modules included Cutadapt in single-end mode, and 6 Globus [98] modules for managing endpoints and transferring files.

6.1.4 Utilities

Helper code used by modules are utilities. *The BioLighthouse* uses 3 types of utilities. The first type of utility added functionality to *The BioLighthouse*, for example the Tool and SLURM utilities. The Tool utility was a *Tool* object composed of the executable name and the ".biolighthouse' directory where the analysis was written (6.1.1). The SLURM utility provided the argument specification for interacting with a SLURM cluster as well as logic for building the SLURM submission command. The Tool and SLURM utilities were used by all modules developed. The second type of utility was to encapsulate common arguments and functions of multi-function tools. Multi-function tools required a module for each function but much of the arguments and logic for building the command were shared between these modules. Both the first and second types of utilities reduce redundant code across modules. The third type of utility was R Script templates. When tools required execution of R Scripts, as in the case of DADA2, an R Script of the workflow was built. The R Script accepted arguments for every variable in the script in order to execute an instance of that script.

The utilities I implemented for this thesis included R Scripts for running DADA2 sample inference and taxonomy assignment, helper utilities for Conda, Globus, and Cutadapt, and the Tool and SLURM utilities.

Figure 6.4: The structure of an execution role. The first section sets up the environment in the same method as the configuration role. The second section contained all the steps of the pipeline. For each tool, the Conda environment was activated, the tool was run, and the Conda environment was deactivated.

6.1.5 Execution Roles

The execution role of a workflow used the developer-produced modules to execute a scientific workflow. The structure of an execution role is shown in Figure 6.4. *The BioLighthouse* execution roles first set up the environment then ran a sequence of tool modules. Setting up the environment was similar to the environment setup described in Section 6.1.2. The ".biolighthouse" path was set, the file structure was setup as described in Section 6.1.1, the utility files were placed in the "/tmp/biolighthouse" directory, and the Conda executable was set. After the environment is set up, the tools are executed. Prior to invocation of each module, the Conda environment with the configured tool was activated. The Conda environment was deactivated after each step. No handler was set to dynamically deactivate the environments at each step since handlers run after the role was completed. To run the execution role, a YAML file containing the parameters was included in the role.

A single execution role template to document the *Prospective Provenance* of the pipeline was generated for this thesis. A total of 30 variable files were saved to document the 10 parameter set runs across the 3 datasets to record the *Retrospective Provenance*.

6.2 Testing Results

This section describes the results of the microbial profiling pipeline. The playbook for the pipeline run was executed on the machines described in Table 5.1 to test for platform dependence. The Oxford Nanopore Technology's MinIT machine was unable to support the pipeline so its results and implications for determining platform dependence are discussed in Sections 6.2.4 and 7.2. The playbook was also run on 3 datasets of differing size described in Section 5.2.2 to test for data-scale dependence. Each dataset was tested with 10 unique parameter sets resulting in 30 pipeline runs across 8 compute environments (not including the MinIT) for a total of 240 pipeline runs, or 80 pipeline runs per dataset.

6.2.1 Small Dataset

The testing of the small dataset was done on 1 sample. The input was 2 FASTQ files corresponding to the forward and reverse reads of one sample. A single run of this dataset was done as described in Figure 6.5, and this was repeated with 10 unique parameter sets. Each parameter set was tested in the 8 environments. The outputs of each tool within the pipeline were tested using the Linux *diff* command. Cutadapt output 2 FASTQ files: the forward reads with primers removed, and the reverse reads with primers removed. Flash2 output 1 FASTQ file, the merged forward and reverse reads. DADA2 sample inference output 1 comma separated value file: the ASV table with sequencing errors removed. DADA2 chimera removal and taxonomy assignment output 2 comma separated value files: the final ASV table with chimeras removed, and a table with taxonomy defined. For the DADA2 chimera removal and taxonomy assignment step, only the final ASV

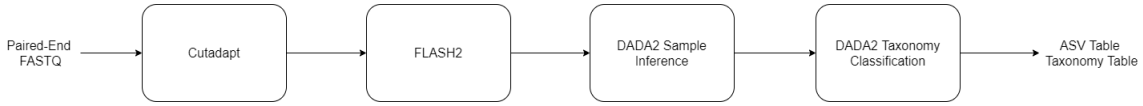


Figure 6.5: The pipeline that was tested. The pipeline contained the same tools as described in Figure 5.1, however two modules were created for the DADA2 package.

tables were compared. The taxonomy tables were not identical because the DADA2 R package required the RDP classifier to be retrained each time it is run.

Each pipeline run resulted in 5 files to be compared. Across the 8 machines, this resulted in 35 file comparisons. Using the Linux *diff* command, I compared all output files for the small dataset. All outputs for Cutadapt, FLASH2, and DADA2 sample inference showed exact identity.

6.2.2 Medium Dataset

The testing for the medium dataset was done on 10 samples. The input for the pipeline was 20 FASTQ files corresponding to the forward and reverse reads of 10 samples. A single run of the medium dataset was done as described in Figure 6.5, and this was repeated with 10 different parameter sets. Each parameter set was tested in the 8 environments. The outputs for each tool within the pipeline was tested with the Linux *diff* command. Cutadapt output 20 FASTQ files: the forward reads with primers removed, and the reverse reads with the primers removed. Flash2 output 10 FASTQ files: the merged forward and reverse reads. DADA2 sample inference output 1 comma separated value file: the ASV table with sequencing errors removed. DADA2 chimera removal and taxonomy assignment output 2 comma separated value files: the final ASV table with chimeras removed, and the table with taxonomy defined. For the DADA2 chimera removal and taxonomy assignment step, only the final ASV tables were compared. The taxonomy tables were not identical because the DADA2 R package required the RDP classifier to be retrained each time it is run.

Each pipeline resulted in 33 files to be compared. Across 8 machines, this resulted in two hundred 231 comparisons for identity. Using the Linux *diff* command, I compared all output files for the small dataset. All outputs for Cutadapt, FLASH2, and DADA2 sample inference showed exact identity.

6.2.3 Large Dataset

The testing for the large dataset was done on 100 samples. The input was 200 FASTQ files corresponding to the forward and reverse reads of 100 samples. A single run of the large dataset was done as described in Figure 6.5, and this was repeated with 10 different parameter sets. Each parameter set was tested in the 8 environments. The outputs for each tool within the pipeline was tested with the Linux *diff* command. Cutadapt output 200 FASTQ files: the forward reads with primers removed, and the reverse reads with primers removed. Flash2 output 100 FASTQ files: the merged forward and reverse reads. DADA2 sample inference output 1 comma separated value file: the ASV table with sequencing errors removed. DADA2

chimera removal and taxonomy assignment output 2 comma separated value files: the final ASV table with chimeras removed, and the table with taxonomy assigned. For the DADA2 chimera removal and taxonomy assignment step, only the final ASV tables were compared. The taxonomy tables were not identical because the DADA2 R package required the RDP classifier to be retrained each time it is run.

Each pipeline resulted in 302 files to be compared. Across 8 machines, this resulted in 2,114 comparisons for identity. Using the Linux *diff* command, I compared all output files for the small dataset. All outputs for Cutadapt, FLASH2, and DADA2 sample inference showed exact identity.

6.2.4 Testing of Taxonomic Assignment

DADA2 taxonomy assignment was ran 10 times on the same input for the small, medium, and large datasets. For each dataset, the differences in taxonomy assignment were counted at differing levels of taxonomy: kingdom, phylum, class, order, family, and genus. The full summary of the taxonomy assignment results are included in Appendix B. Figure 6.6 shows the results of taxonomy for the 3 datasets at phylum, order, and genus level. Each entry in the matrix is the count of differences in taxonomic assignment between two replicates divided by the total number of taxonomic classifications. The *undetermined* classification was treated as the same classification.

The small and medium datasets contained no differences in taxonomic assignment at the kingdom level. All other levels for the 3 datasets produced non-deterministic results. At the genus level, the median proportion of differences in taxonomic assignments was 0.0206 (small dataset), 0.0139 (medium dataset), and 0.0207 (large dataset). The standard deviations for taxonomic assignments at the genus level were 0.00590 (small dataset), 0.00251 (medium dataset), and 0.00085 (large dataset).

6.2.5 MinIT Testing

Installation of Conda as implemented in *The BioLighthouse* did not work on the MinIT machine. Archiconda [99] was required for the 64-bit ARM processor on the MinIT; however, only the conda-forge channel was available and *The BioLighthouse* required the bioconda channel. The tools in the conda-forge channel were available for Archiconda so the tools and thus the pipeline could not be conducted on the MinIT. As a result, testing of *The BioLighthouse* was not conducted on the MinIT.

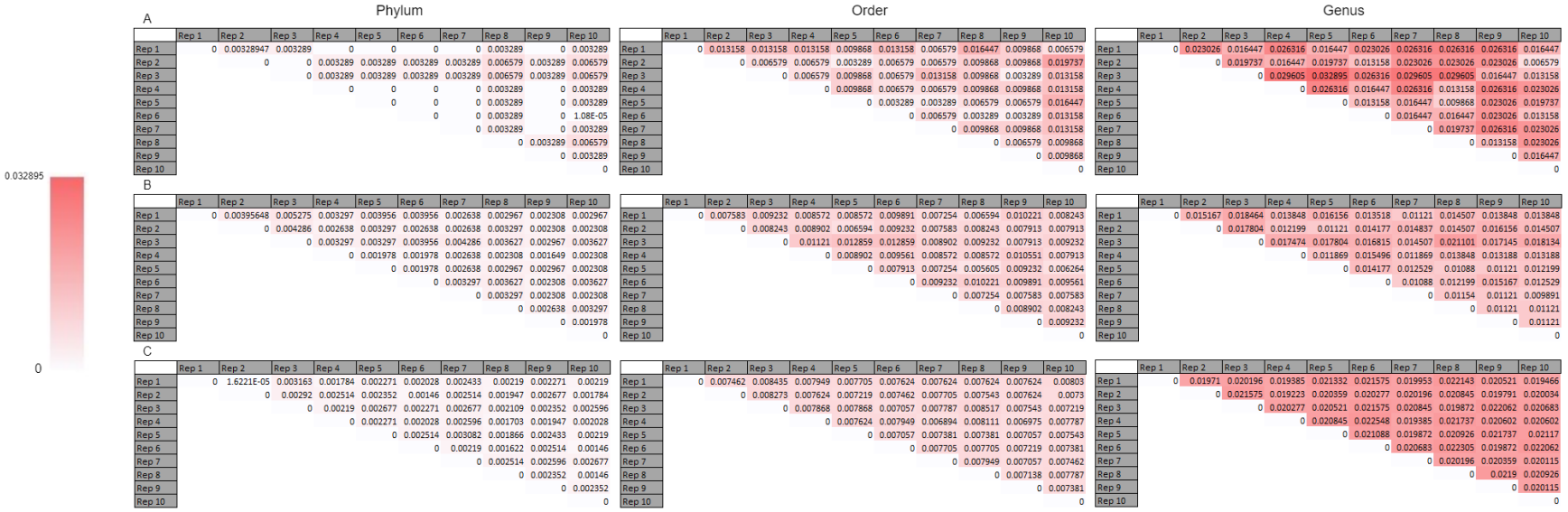


Figure 6.6: The proportion differences in taxonomic assignment of the 10 replicates for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment. A denotes the results for the small dataset, B denotes the results for the medium dataset, and C denotes the results for the large dataset. Increasing proportions of errors were observed with increasing taxonomic rank. The taxonomic levels shown in this figure from left to right are: phylum, order, and genus. The matrices for all taxonomic levels are included in Appendix B

7 Discussion

The BioLighthouse is a novel application of Ansible to scientific workflows. The core functionality of Ansible was extended to execute scientific workflows in an accessible, reproducible, and transparent fashion. An instantiation of *The BioLighthouse* was demonstrated with a microbial profiling pipeline. The microbial profiling pipeline was tested on Linux cloud machines, shared machines, and a High Performance Computing (HPC) cluster to demonstrate platform independence. The pipeline was also tested on a small (1 sample), medium (10 sample), and large (100 sample) dataset to demonstrate data-scale independence. *The BioLighthouse* successfully operated within Conda environments and enables sharing and reuse of workflows using the Ansible Galaxy platform. Roles developed in this thesis were shared on Ansible Galaxy and developers in the future may share their workflows in the same manner. Since Ansible and subsequently *The BioLighthouse* accurately describe compute environments and workflows respectively, precisely the same commands were run with each role execution.

7.1 Scale in Bioinformatics

The development of inexpensive nucleic acid sequencing has enabled the production of large datasets for microbial profiling. Larger datasets with deeper sequencing provides more comprehensive insights into microbiome studies. Cheaper sequencing allows for datasets to consist of more samples, which enables more complex experimental designs and provides more statistical power to machine learning methods, making them tractable. With the increase in data-scale, larger compute infrastructure is often needed to conduct analyses in a timely fashion. Sometimes, machines consisting of few cores such as virtual and personal machines are sufficient for analysis as in the case of the small and medium sized datasets tested in this thesis. However, it is often the case that HPC infrastructure is desired or even required for analyses to complete in a reasonable amount of time. This was observed in the case of the large dataset tested in this thesis where some steps took around 24 hours to complete on the virtual machines. Particularly in the DADA2 sample inference step, upwards of 20 hours was required for testing each parameter set on the virtual machines having only four (4) cores.

WestGrid, a partner within the Compute Canada organization, provides institutional researchers with HPC infrastructure in western Canada. Compute Canada offers sessions at various institutions with high-level overviews of running jobs on their infrastructure to enable researchers to analyze data at scale. Even with the resources enabling non-computational researchers to utilize HPC infrastructure, it is not the focus

of their research to learn how to use HPC machines. Additionally, non-computational researchers often lack basic foundational knowledge to use HPC machines effectively and troubleshoot errors when they inevitably arise.

Testing of the small, medium, and large datasets behaved in the exact same manner and produced identical results on all machines tested for all deterministic steps. Other than the time taken for each step, which was expected, data-scale had no effect on the results of *The BioLighthouse*. Thus, testing of *The BioLighthouse* provided no evidence for the platform's data-scale dependence.

The results of testing the non-deterministic taxonomy assignment, however, behaved differently. The taxonomy classifier in DADA2 required to be retrained each time it was run. There was no method to set a random seed or save the module of the classifier for future use. The median proportion of differences in taxonomic assignment for the small, medium, and large datasets were approximately the same but the standard deviation for each differed. The small dataset had the highest standard deviation (0.00590) and the large dataset had the smallest standard deviation (0.00085), suggesting more stable results for larger-scale data.

Deterministic tools have been shown to run identically on data of varying scales in *The BioLighthouse* but the suggestion of data-scale dependence for the non-deterministic taxonomy classification demonstrates the impact of non-reproducible results. Non-determinism is the case for some tools whether it is an intentional design decision or an oversight made by the developers. Randomness in tools for scientific purposes is not desired as it introduces unnecessary variation into results. When developing tools with random number generation, there should be options to either set random seeds or save models for future use. It is sometimes the case in which randomness is fundamentally necessary, as in the case where the definition of the result is not discrete. Though reproducibility in these cases may be explored in future work, they were not the focus of this thesis.

The availability of the SLURM utility built into *The BioLighthouse* greatly lowers the barrier of use for SLURM HPC clusters by non-computational researchers. As a result, analysis throughput is affected in two ways. Firstly, a large amount of compute resources can be easily utilized for analyses. Secondly, users are not required to invest time in learning how to run jobs on HPC machines, thus indirectly increasing throughput. Researchers using the SLURM utility can trust it will build the command correctly and do not need to worry about troubleshooting errors. Should an error arise, it is the job of the developer to fix it and push an update to their module or utility.

Something to be noted is SSH connections can be closed at any time by shared machines. As data-scale increases, the time taken for steps to complete increases the chance the SSH connection will be closed. If the SSH connection is closed before *The BioLighthouse* has completed its run, the run will be incomplete. Ansible has an environment variable that can be set for a connection timeout but it has no power over the host closing the connection. The SSH connection closing was observed in the case of testing on the UofS shared server, a time-shared machine. A simple solution around the SSH connection forcibly closing was to

execute each analysis step as an asynchronous task and check on its completion periodically. By executing the tasks asynchronously the SSH connection was closed, resetting the timeout set by the administrators of the shared machines. The task continued to run on the machine and was checked on every minute until the task was completed. Additionally, it may be the case that users are vast geographic distances from the hosts and network reliability of persistent connections could be impacted. Since asynchronous tasks do not require persistent connections, the chance network related errors arise is lowered.

There is also a limitation to using *The BioLighthouse* on HPC machines. Whereas running workflows on regular machines waits for the execution task to complete and reports success or error, the execution task for HPC machines only reports success or failure of submission of the job to the SLURM scheduler. This puts emphasis on the developer's testing as the developer must be sure their module will execute properly. For future work, data validation steps will be implemented but was outside the scope of this thesis.

7.2 *In Silico* Experimentation

In silico experiments were conducted using *The BioLighthouse* microbial profiling pipeline in order to ensure reproducible results as reproducibility through experimentation is the basis of the scientific method. 3 microbial profiling datasets were run through the pipeline each with 10 unique parameter sets. The tests were repeated in 8 computational environments for a total of 240 pipeline runs. The measurement of reproducibility used was file comparisons of tool outputs from the unique parameter sets across the computational environments (virtual machines, shared machines, HPC cluster) using the Linux *diff* command [4]. 2,373 file comparisons of tool outputs were created. The file comparisons showed exact reproducibility, suggesting no platform dependence for *The BioLighthouse*.

It was the intention of this research to demonstrate the utility of *The BioLighthouse* on portable analysis machines using Oxford Nanopore Technologies MinIT machine. Though *The BioLighthouse* was not able to work on the MinIT machine, this case alone does not suggest platform dependence. Ansible was able to interact with the MinIT and an alternative version of Conda (Archiconda [99]) was successfully configured with a role. The main issue with this platform was that the tools being tested had not been developed for the MinIT's 64-bit ARM processor.

Portable sequencing devices enable field studies in which transporting biological samples may be difficult, or even dangerous in the case of invasive species and infectious diseases. Use of *The BioLighthouse* on a portable system like the MinIT has a huge potential to further enable these field studies by automating the tedious computational work that would normally require other team members. Future iterations of this work may look to expand the Bioconda channel to build tools on the MinIT to enable portable analysis on low-cost ARM processor machines.

In vitro and *in vivo* experimentation done by life scientists maintains reproducibility generally through use of a laboratory notebook. Documentation within laboratory notebooks allows researchers to track variables,

measurements, and observations but manual documentation alone is not sufficient for *in silico* experimentation [12]. Firstly, there are far too many features to track for computational experimentation including operating system information, operating system dependencies, tool versions, and tool dependencies. Secondly, vocabularies greatly differ in multidisciplinary teams common in bioinformatics. For example, this thesis tested tool outputs with *diff* to conclude if results across platforms were the *same*. Biological experimentation is noisy and thus a life scientist may wrongly conclude a deterministic tool output differing an arbitrarily small threshold to be the *same*.

The BioLighthouse takes on the role of a laboratory notebook for *in silico* experimentation. Playbooks and roles document the exact steps that were run for an analysis starting from configuration of tools and continuing through the tool execution. Included with each task is a written narrative explaining what the task does. One of the reproducibility design decisions of *The BioLighthouse* was to leverage Conda virtual environments. The intention of operating within a Conda environment was to sterilize the analysis environment so every pipeline is executed on a level playing field. The sterile analysis environment is analogous to a biologist conducting experiments under the same conditions as previous replications. Further, *The BioLighthouse* describes the entire analysis environment including tools versions and dependencies. The description is based on the developer's configuration role; however, future utilities could be developed to capture a developer's minimal tool configuration in a Conda environment automatically for role building.

New tools must be broadly approachable by the scientific community to catalyze scientific discussion. Scientific results can be affirmed and disputed in order to build upon the results and advance the field. When exploring a new tool for use in personal work, researchers often test the tool with published data to ensure the tool can at least reproduce the results previously published. Additionally, many tools do similar things and argue why the method used outperforms others. Especially in the bioinformatics space, it is right to be skeptical because of highly variable data, rapid prototyping of tools, and pressure to publish work while it is relevant. Tool performance can be dependent on the data being analyzed so exploratory experimentation must be done to choose an appropriate tool. Further, researchers in the bioinformatics space have constantly evolving toolkits as new tools are released and old tools become obsolete. With *The BioLighthouse*, only a single developer is required to take the time to fully configure and test the tool in different environments allowing others to easily add it to their toolbox. Instead of fighting through getting the tool to work properly and run down all the dependencies, all subsequent users are only required to learn what the tool does on a high-level to use the tool in the manner it was previously published.

The BioLighthouse enables researchers to experiment between and within tools. Since many tools in the bioinformatics space are similar, researchers can easily see for themselves which tool is most appropriate for their analysis if they are built out in *The BioLighthouse* method. For example, consider Figure 7.1. FLASH2 [2] and DADA2 [3], two tools formally tested in this thesis, are semantically similar to PEAR [97] and Deblur [84], supplementary tools with *BioLighthouse* roles. PEAR and FLASH2 can be swapped along with Deblur and DADA2 for alternative insights into the data and to make an informed decision on what to

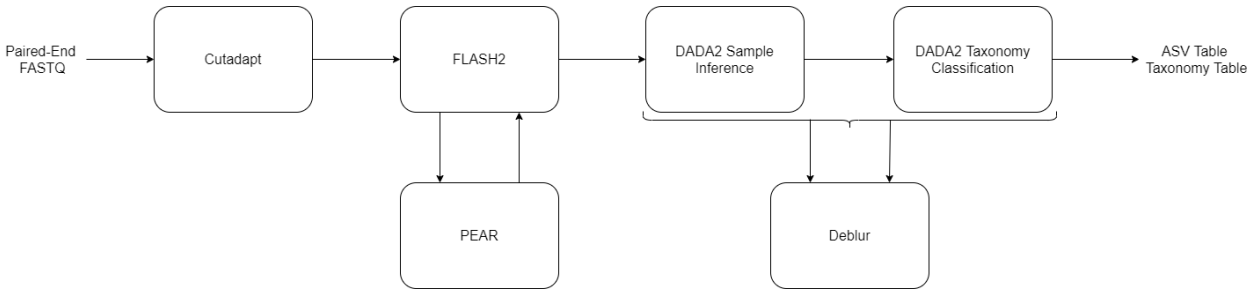


Figure 7.1: The microbial profiling pipeline tested in this thesis presented with alternatives. PEAR and Deblur were also implemented as supplementary tools and are alternatives to the FLASH2 and DADA2 steps. Provided the interfaces are compatible, semantically similar tools implemented in *The BioLighthouse* can be easily swapped.

use. Alternative tools can be implemented in the same method as demonstrated in this thesis using modules, utilities, and roles.

I also built out EMBOSS [64] in *The BioLighthouse* method. Since it is a suite of common bioinformatics tools, *The BioLighthouse* EMBOSS implementation can be extended into the pipeline using Ansible modules that explore the impact of the results when changing the FLASH2 for PEAR and DADA2 for Deblur. When experimenting with within-tool variation, *The BioLighthouse* enables researchers to specify an arbitrary number of parameter combinations to test. Testing within and between tools also relates to scale in testing. *The BioLighthouse's* effects on throughput makes exploring parameter spaces and comparing tools accessible to all researchers regardless of skill and resource requirements. Other variables such as ease of configuration, necessity for HPC machines, and biases are eliminated so a data-driven decision can be made.

7.3 Multidisciplinary Researcher Interaction

Researchers from multiple disciplines are involved in Next Generation Sequencing (NGS) studies from data generation to data analysis. Figure 2.1 introduced the general interaction of the multidisciplinary researchers that *The BioLighthouse* was built for. The actors modelled were the life scientist, developer, bioinformatician, and tool user. Often in laboratory settings, sequencing data is generated by life scientists and is sent to a bioinformatician. Using a toolkit of scripts, possibly from previous projects, the bioinformatician processes the data for downstream statistical analysis. The analysis is often viewed as a black box to life scientists. Something as seemingly trivial as changing a single parameter could require the bioinformatician to execute the entire analysis again. At this point, the bioinformatician either needs to rerun the workflow, which could take a long time or the biologist may ask for the the script or an explanation and try to rerun the processing themselves.

Since these interactions are multidisciplinary, vocabularies differ greatly so it may be difficult to understand what exactly the biologist wants or why they want a different run of the data. Further, in the case

where the biologist asks for the scripts to rerun the data themselves, issues described in Section 3.1 about strategies for computational reproducibility arise. Since bioinformaticians' tool kits are often personal scripts generated from previous work, others will likely have difficulty interpreting the code and making modifications. Additionally, the accompanying written narrative of the script is likely insufficient for a biologist to conduct the analysis as intended. Further, the biologist may have out-of-date versions of the software being used or may not have the tools installed on their analysis machine altogether. As a result, the script may not work, give spurious results, or the biologist may not be able to set up their analysis environment properly.

Developers who publish a tool often have difficulty in making their tool accessible to the research community. Tools are often housed on software repositories and include written narratives and possibly scripts for configuring the tool. The developer may even share the tool on a software installation framework such as Conda, Bioconductor, and Docker. *The BioLighthouse* solution demonstrated it can build from source, execute scripts, and leverage Conda to funnel the many methods of software configuration into one method with roles. Roles greatly improve the accessibility to configuration of tools by reducing the configuration recipe to a single command while maintaining transparency. Additionally, the logic that developers build into modules abstracts away the necessity of the user to identify software use cases by building the use cases directly into the modules and only requiring the user to specify parameters.

The toolkits of bioinformaticians are constantly evolving as new tools are published and old methods become obsolete. Consuming modules from developers allows for higher-level pipeline building since the use cases have been clearly defined. A bioinformatician using *The BioLighthouse* can trust the developer has sufficiently tested the modules and configuration roles so no changes need to be made on the bioinformatician's end for updating tools. Additionally, the formal structure of execution roles provides a method for bioinformaticians to organize different pipelines for reuse and sharing.

Reuse and sharing of roles also directly affects the interaction between the bioinformatician and the tool user. Instead of conducting analyses themselves, the bioinformatician can focus on building out pipelines for others to consume. The transparency of roles still allows the tool user to treat the analysis as a black box while exposing the *Retrospective Provenance* if the user chooses.

The previously described case in which an analysis product given from the bioinformatician to the life scientist required a subsequent run of the pipeline is directly influenced by *The BioLighthouse*. The tool user may run arbitrarily large numbers of parameter sets as described in Section 7.2. With much of the time-consuming work offloaded to roles, the bioinformatician can play an administrative role only getting involved when an error manifests. The error handling in modules built by the developer provides valuable information when a task fails so it can be easily debugged.

Adoption of *The BioLighthouse* is mainly dependent on developers. If developers are deploying their tools in *The BioLighthouse* method, bioinformaticians and tool users may be more inclined to try it. Further refinement of tool development into *The BioLighthouse* is required to lower the barrier to entry for developers. Steps such as the development of template configuration and execution roles was the first step towards

promoting adoption. The template roles abstract away the need for environment setup by the developer and bioinformatician so they can focus on configuration and pipeline building. The main function of *The BioLighthouse* is to encapsulate logic for pipeline building and tool execution. *The BioLighthouse* has pushed the boundaries of Ansible configuration to execute workflows in an accessible, reproducible and transparent fashion. *The BioLighthouse* provides a platform for developers to share their tools, for bioinformaticians to build pipelines for use and reuse, and for users to execute, reuse, and share workflows.

8 Conclusion

The BioLighthouse serves as a proof-of-concept that Ansible can be extended to not only configure scientific workflows, but also execute bioinformatics tools in an accessible, reproducible, and transparent fashion. *The BioLighthouse* is platform independent and allows users to execute scientific workflows on personal, shared, and high-performance machines. Testing of *The BioLighthouse* also showed no data-scale dependence. *The BioLighthouse* successfully operated in user-space using Conda environments to ensure analysis environments are constant and allow for configuration of shared machines.

There are some limitations for use of *The BioLighthouse*. Though *The BioLighthouse* is platform independent, the computing environment must support Conda as well as support all the tools in the pipeline. This was demonstrated in the case of Oxford Nanopore Technologies MinIT machine in which no tool in the microbial profiling pipeline was built for the 64-bit ARM processor used by the MinIT platform. The MinIT has a large potential to be a useful machine for use of *The BioLighthouse* in the field for immediate data processing and *The BioLighthouse* will be extended in the future as tools are compiled for the MinIT platform. Since bioinformatics tools can have large numbers of parameters and behave differently on varying datasets, edge-case errors are expected to arise as more researchers use the tools. Edge-case errors are errors that were not identified through the regular testing process. Developers can generally expect their tools to execute as intended and play an administration role to troubleshoot edge-case errors when they arise. The final major limitation of *The BioLighthouse* is caused by the tools themselves. Certain tools are non-deterministic and produce different results when run with identical inputs. This was observed in the DADA2 taxonomy classification where DADA2 internally uses the RDP classifier and the classifier itself must be retrained for each execution. Non-determinism is the case for some tools and these must be clearly identified by developers when implementing them into *The BioLighthouse*. By identifying these tools, tool users can take the necessary steps for data validation and developers can work to eliminate non-determinism from their tools if randomness is not fundamentally necessary.

In the immediate future, work to promote adoption of *The BioLighthouse* will be done. Firstly, current tools in bioinformatics will be built-out into modules which supports accessibility to contemporary tools for users. Implementation of tools will continue with the supplementary tools written for this thesis (Appendix A). Secondly, the method of implementing tools into modules and configuration roles will be refined to make adoption by developers easier. Examples of refinements could include automated environment capture for building of configuration roles as well as automated argument specification generation for building modules. By automating as much as possible for developers, adopting *The BioLighthouse* method becomes less invasive

and thus it becomes more likely for the developer to contribute. Additionally, *The BioLighthouse* presently generates results based on testing by the developer. For future work, data validation methods could be built into *The BioLighthouse* modules to notify users of data products that may require some attention. In the case of microbial profiling, for example, this could include notifying users of large unassigned FASTQ files which could require resequencing, and notifications about large numbers of reads not being merged. Extending *The BioLighthouse* to include minor data validation will further help non-computational researchers obtain valid, robust results and will also help minimize edge-case errors.

All roles and modules developed for this thesis are available on Ansible Galaxy and *The BioLighthouse* code is available on Ansible Galaxy (https://galaxy.ansible.com/coadunate/thebiolighthouse). *The BioLighthouse* is released under the BSD license.

References

[1] Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet. Journal*, 17(1):10–12, 2011.

[2] Tanja Magoč and Steven L Salzberg. Flash: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, 27(21):2957–2963, 2011. Oxford University Press.

[3] Benjamin J Callahan, Paul J McMurdie, Michael J Rosen, Andrew W Han, Amy Jo A Johnson, and Susan P Holmes. DADA2: high-resolution sample inference from illumina amplicon data. *Nature Methods*, 13(7):581, 2016. Nature Publishing Group.

[4] Michael Kerrisk. diff command man page. `http://man7.org/linux/man-pages/man1/diff.1.html`. Accessed: 2020-02-10.

[5] Anaconda. Anaconda software distribution. `https://anaconda.com`. Computer Software. Vers. 4.7.12. Accessed: 2020-02-17.

[6] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A Chapman, Jillian Rowe, Christopher H Tomkins-Tinch, Renan Valieris, and Johannes Köster. Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, 15(7):475–476, 2018. Nature Publishing Group.

[7] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. Oxford University Press.

[8] Jo Handelsman, Michelle R Rondon, Sean F Brady, Jon Clardy, and Robert M Goodman. Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & Biology*, 5(10):R245–R249, 1998. Elsevier.

[9] Joshua Lederberg and Alexa T McCray. Ome sweetomics–a genealogical treasury of words. *The Scientist*, 15(7):8–8, 2001. Scientist Inc.

[10] Jack A Gilbert and Christopher L Dupont. Microbial metagenomics: beyond the genome. *Annual Review of Marine Science*, 3:347–371, 2011.

[11] Anton Nekrutenko and James Taylor. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. *Nature Reviews Genetics*, 13(9):667, 2012. Nature Publishing Group.

[12] Stephen R Piccolo and Michael B Frampton. Tools and techniques for computational reproducibility. *Gigascience*, 5(1):30, 2016. BioMed Central.

[13] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010. BioMed Central.

[14] Susan B Davidson, Sarah Cohen Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M McPhillips, Shawn Bowers, Manish Kumar Anand, and Juliana Freire. Provenance in scientific workflow systems. *IEEE Data Eng. Bull.*, 30(4):44–50, 2007.

[15] Inc. Red Hat. Ansible documentation. `https://docs.ansible.com/ansible/latest/index.html`. Accessed: 2020-02-17.

[16] Karsten Zengler, Gerardo Toledo, Michael Rappé, James Elkins, Eric J Mathur, Jay M Short, and Martin Keller. Cultivating the uncultured. *Proceedings of the National Academy of Sciences*, 99(24):15681–15686, 2002. National Acad Sciences.

[17] Laura A Hug, Brett J Baker, Karthik Anantharaman, Christopher T Brown, Alexander J Probst, Cindy J Castelle, Cristina N Butterfield, Alex W Hernsdorf, Yuki Amano, Kotaro Ise, et al. A new view of the tree of life. *Nature Microbiology*, 1(5):16048, 2016. Nature Publishing Group.

[18] Eric J Stewart. Growing unculturable bacteria. *Journal of Bacteriology*, 194(16):4151–4160, 2012. Am Soc Microbiol.

[19] Christopher P Meyer and Gustav Paulay. DNA barcoding: error rates based on comprehensive sampling. *PLoS Biology*, 3(12), 2005. Public Library of Science.

[20] Matthew G Links, Tim J Dumonceaux, Sean M Hemmingsen, and Janet E Hill. The chaperonin-60 universal target is a barcode for bacteria that enables de novo assembly of metagenomic sequence data. *PloS One*, 7(11):e49755, 2012. Public Library of Science.

[21] Xochitl C Morgan, Timothy L Tickle, Harry Sokol, Dirk Gevers, Kathryn L Devaney, Doyle V Ward, Joshua A Reyes, Samir A Shah, Neal LeLeiko, Scott B Snapper, et al. Dysfunction of the intestinal microbiome in inflammatory bowel disease and treatment. *Genome Biology*, 13(9):R79, 2012. BioMed Central.

[22] Peter J Turnbaugh, Ruth E Ley, Michael A Mahowald, Vincent Magrini, Elaine R Mardis, and Jeffrey I Gordon. An obesity-associated gut microbiome with increased capacity for energy harvest. *Nature*, 444(7122):1027, 2006. Nature Publishing Group.

[23] Christina E West, Patrik Rydén, Daniel Lundin, Lars Engstrand, Meri K Tulic, and Susan L Prescott. Gut microbiome and innate immune response patterns in IgE-associated eczema. *Clinical & Experimental Allergy*, 45(9):1419–1429, 2015. Wiley Online Library.

[24] Hannah J Gould, Brian J Sutton, Andrew J Beavil, Rebecca L Beavil, Natalie McCloskey, Heather A Coker, David Fear, and Lyn Smurthwaite. The biology of IgE and the basis of allergic disease. *Annual Review of Immunology*, 21(1):579–628, 2003. Annual Reviews 4139 El Camino Way, PO Box 10139, Palo Alto, CA 94303-0139, USA.

[25] Matthew G Links, Bonnie Chaban, Sean M Hemmingsen, Kevin Muirhead, and Janet E Hill. mPUMA: a computational approach to microbiota analysis by de novo assembly of operational taxonomic units based on protein-coding barcode sequences. *Microbiome*, 1(1):23, 2013. BioMed Central.

[26] David Hendricks Bergey, Robert Stanley Breed, Everitt George Dunne Murray, A Parker Hitchens, et al. Manual of determinative bacteriology. *Manual of Determinative Bacteriology. Fifth Edn.*, 1939. London, Bailliere, Tindall & Cox.

[27] George M Garrity, Julia A Bell, and Timothy G Lilburn. Taxonomic outline of the prokaryotes. *Bergey's Manual of Systematic Bacteriology*, 2004. Springer-Verlag New York, NY.

[28] Janet E Hill, Robyn P Seipp, Martin Betts, Lindsay Hawkins, Andrew G Van Kessel, William L Crosby, and Sean M Hemmingsen. Extensive profiling of a complex microbial community by high-throughput sequencing. *Appl. Environ. Microbiol.*, 68(6):3055–3066, 2002. Am Soc Microbiol.

[29] Zelalem M Taye, Bobbi L Helgason, Jennifer K Bell, Charlotte E Norris, Sally Vail, Stephen J Robinson, Isobel AP Parkin, Melissa Arcand, Steven Mamet, Matthew G Links, et al. Core and differentially abundant bacterial taxa in the rhizosphere of field grown brassica napus genotypes: Implications for canola breeding. *Frontiers in Microbiology*, 10:3007, 2020. Frontiers.

[30] Jorge S Reis-Filho. Next-generation sequencing. *Breast Cancer Research*, 11(3):S12, 2009. BioMed Central.

[31] Kris A. Wetterstrand. The cost of sequencing a human genome. `https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost`. Accessed: 2020-02-18.

[32] Joshua Quick, Nicholas J Loman, Sophie Duraffour, Jared T Simpson, Ettore Severi, Lauren Cowley, Joseph Akoi Bore, Raymond Koundouno, Gytis Dudas, Amy Mikhail, et al. Real-time, portable genome sequencing for ebola surveillance. *Nature*, 530(7589):228–232, 2016. Nature Publishing Group.

[33] Oxford Nanopore Technologies. `https://nanoporetech.com`. Accessed: 2020-01-22.

[34] Peter Belmann, Johannes Dröge, Andreas Bremges, Alice C McHardy, Alexander Sczyrba, and Michael D Barton. Bioboxes: standardised containers for interchangeable bioinformatics software. *Gigascience*, 4(1):47, 2015. BioMed Central.

[35] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10):e1003285, 2013. Public Library of Science.

[36] Barbara R Jasny, Gilbert Chin, Lisa Chong, and Sacha Vignieri. Again, and again, and again.... *Science*, 334:1225, 2011. American Association for the Advancement of Science.

[37] Susan B Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1345–1350. ACM, 2008.

[38] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. YAML ain't markup language (YAML^{TM}) version 1.1. *YAML.org, Tech. Rep*, page 23, 2005.

[39] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) protocol architecture. 2006.

[40] Keith A Baggerly and Kevin R Coombes. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *The Annals of Applied Statistics*, pages 1309–1334, 2009. JSTOR.

[41] Richard M Stallman, Roland McGrath, and Paul D Smith. GNU make: A program for directing recompilation, for version 3.81. *Boston: Free Software Foundation*, 2004.

[42] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004. BioMed Central.

[43] The CRAN project [internet]. Available From https://github.com/cran, Accessed June 11, 2018. Cran.

[44] Python pip [internet]. Available From https://github.com/pypa/pip, Accessed June 11, 2018. PyPA.

[45] Shaun Jackman and Inanc Birol. Linuxbrew and homebrew for cross-platform package management. *F1000Research*, 5:1795, 2016.

[46] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[47] Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. Oxford University Press.

[48] Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007. IEEE.

[49] Sylwester Arabas, Marc Schellens, Alain Coulais, Joel Gales, and Peter Messmer. GNU data language (GDL)–a free and open-source implementation of IDL. In *EGU General Assembly Conference Abstracts*, volume 12, page 924, 2010.

[50] Stephen Wolfram. *The Mathematica*. 1999. Cambridge University Press Cambridge.

[51] Burçin Eröcal and William Stein. The Sage project: Unifying free mathematical software to create a viable alternative to Magma, Maple, Mathematica and MATLAB. In *International Congress on Mathematical Software*, pages 12–27. Springer, 2010.

[52] Yihui Xie. *Dynamic Documents with R and knitr*. 2016. Chapman and Hall/CRC.

[53] GDL IPython kernel [internet]. Available From https://github.com/gnudatalanguage/idl_kernel, Accessed June 28, 2018. GNU Data Language.

[54] Jupyter kernels [internet]. Available From https://github.com/jupyter/jupyter/wiki/Jupyter-kernels/, Accessed November 11, 2018. Jupyter.

[55] Compute Canada. Cedar. `https://docs.computecanada.ca/wiki/Cedar`. Accessed: 2020-02-27.

[56] Daniel G Hurley, David M Budden, and Edmund J Crampin. Virtual reference environments: a simple way to make research reproducible. *Briefings in Bioinformatics*, 16(5):901–903, 2014. Oxford University Press.

[57] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014. Belltown Media.

[58] Felipe da Veiga Leprevost, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, Pablo Moreno, Laurent Gatto, Jonas Weber, et al. Biocontainers: an open-source and community-driven framework for software standardization. *Bioinformatics*, 33(16):2580–2582, 2017. Oxford University Press.

[59] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity: Scientific containers for mobility of compute. *PloS One*, 12(5):e0177459, 2017. Public Library of Science.

[60] Andy B Yoo, Morris A Jette, and Mark Grondona. SLURM: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[61] The Galaxy Project. Account quotas. `https://galaxyproject.org/support/account-quotas/`. Accessed: 2020-01-05.

[62] Louis Bavoil, Steven P Callahan, Patricia J Crossno, Juliana Freire, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. VisTrails: Enabling interactive multiple-view visualizations. In *Visualization, 2005. VIS 05. IEEE*, pages 135–142. IEEE, 2005.

[63] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. VisTrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 745–747. ACM, 2006.

[64] Peter Rice, Ian Longden, and Alan Bleasby. EMBOSS: the European molecular biology open software suite. *Trends in Genetics*, 16(6):276–277, 2000. Elsevier.

[65] EMBOSS explorer [internet]. Available From https://sourceforge.net/projects/embossgui/, Accessed November 11, 2018. Luke McCarthy.

[66] Puppet. Powerful infrastructure automation and delivery. `https://puppet.com/`. Accessed: 2020-02-17.

[67] Chef. `https://chef.io`. Computer Software. Accessed: 2020-02-17.

[68] SaltStack. `https://saltstack.com`. Computer Software. Accessed: 2020-02-17.

[69] Rob Knight, Alison Vrbanac, Bryn C Taylor, Alexander Aksenov, Chris Callewaert, Justine Debelius, Antonio Gonzalez, Tomasz Kosciolek, Laura-Isobel McCall, Daniel McDonald, et al. Best practices for analysing microbiomes. *Nature Reviews Microbiology*, 16(7):410–422, 2018. Nature Publishing Group.

[70] Tobin J Verbeke, Richard Sparling, Janet E Hill, Matthew G Links, David Levin, and Tim J Dumonceaux. Predicting relatedness of bacterial genomes using the chaperonin-60 universal target (cpn60 ut): application to thermoanaerobacter species. *Systematic and Applied Microbiology*, 34(3):171–179, 2011. Elsevier.

[71] John Schellenberg, Matthew G Links, Janet E Hill, Tim J Dumonceaux, Geoffrey A Peters, Shaun Tyler, T Blake Ball, Alberto Severini, and Francis A Plummer. Pyrosequencing of the chaperonin-60 universal target as a tool for determining microbial community composition. *Applied and Environmental Microbiology*, 75(9):2889–2898, 2009. Am Soc Microbiol.

[72] Teenus Paramel Jayaprakash, John J Schellenberg, and Janet E Hill. Resolution and characterization of distinct cpn60-based subgroups of gardnerella vaginalis in the vaginal microbiota. *PLoS One*, 7(8):e43009, 2012. Public Library of Science.

[73] Peter HA Sneath and Robert R Sokal. Numerical taxonomy. *Nature*, 193(4818):855–860, 1962. Nature Publishing Group.

[74] Nastassia V Patin, Victor Kunin, Ulrika Lidström, and Matthew N Ashby. Effects of OTU clustering and PCR artifacts on microbial diversity estimates. *Microbial Ecology*, 65(3):709–719, 2013. Springer.

[75] A Murat Eren, Loïs Maignien, Woo Jun Sul, Leslie G Murphy, Sharon L Grim, Hilary G Morrison, and Mitchell L Sogin. Oligotyping: differentiating between closely related microbial taxa using 16S rRNA gene data. *Methods in Ecology and Evolution*, 4(12):1111–1119, 2013. Wiley Online Library.

[76] Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, et al. Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotechnology*, 29(7):644, 2011. Nature Publishing Group.

[77] Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, et al. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-Bruijn-graph. *Briefings in Functional Genomics*, 11(1):25–37, 2012. Oxford University Press.

[78] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357, 2012. Nature Publishing Group.

[79] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012. Oxford University Press.

[80] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997. Oxford University Press.

[81] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000. Elsevier.

[82] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. Fasttree 2–approximately maximum-likelihood trees for large alignments. *PloS One*, 5(3), 2010. Public Library of Science.

[83] Catherine Lozupone and Rob Knight. UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology*, 71(12):8228–8235, 2005. Am Soc Microbiol.

[84] Amnon Amir, Daniel McDonald, Jose A Navas-Molina, Evguenia Kopylova, James T Morton, Zhenjiang Zech Xu, Eric P Kightley, Luke R Thompson, Embriette R Hyde, Antonio Gonzalez, et al. Deblur rapidly resolves single-nucleotide community sequence patterns. *MSystems*, 2(2):e00191–16, 2017. Am Soc Microbiol.

[85] Jacob T Nearing, Gavin M Douglas, André M Comeau, and Morgan GI Langille. Denoising the denoisers: An independent evaluation of microbiome sequence error-correction methods. Technical report, 2018. PeerJ Inc. http://dx.doi.org/10.7717/peerj.5364.

[86] Michael J Rosen, Benjamin J Callahan, Daniel S Fisher, and Susan P Holmes. Denoising PCR-amplified metagenome data. *BMC Bioinformatics*, 13(1):283, 2012. BioMed Central.

[87] NCBI. `https://www.ncbi.nlm.nih.gov/genbank/rrnachimera/`. Accessed: 2020-03-03.

[88] Brian J Haas, Dirk Gevers, Ashlee M Earl, Mike Feldgarden, Doyle V Ward, Georgia Giannoukos, Dawn Ciulla, Diana Tabbaa, Sarah K Highlander, Erica Sodergren, et al. Chimeric 16S rRNA sequence formation and detection in sanger and 454-pyrosequenced PCR amplicons. *Genome Research*, 21(3):494–504, 2011. Cold Spring Harbor Lab.

[89] Andrea Sboner, Xinmeng Jasmine Mu, Dov Greenbaum, Raymond K Auerbach, and Mark B Gerstein. The real cost of sequencing: higher than you think! *Genome Biology*, 12(8):125, 2011. Springer.

[90] Hanno Teeling and Frank Oliver Glöckner. Current opportunities and challenges in microbial metagenome analysis—a bioinformatic perspective. *Briefings in Bioinformatics*, 13(6):728–742, 2012. Oxford University Press.

[91] Mark D Wilkinson, Benjamin Vandervalk, and Luke McCarthy. The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation. *Journal of Biomedical Semantics*, 2(1):8, 2011. Springer.

[92] Mark D Wilkinson and Matthew Links. Biomoby: an open source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, 2002. Henry Stewart Publications.

[93] Olivier Frisette. Plato hpc cluster. `https://wiki.usask.ca/display/ARC/Plato+HPC+cluster`. Accessed: 2020-02-27.

[94] Inc. Red Hat. Module index. `https://docs.ansible.com/ansible/latest/modules/modules_by_category.html`.

[95] Introducing json. `https://json.org`. Accessed: 2020-01-27.

[96] Web Technology Surveys. Usage statistics of linux for websites. `https://w3techs.com/technologies/details/os-linux`. Accessed: 2020-03-01.

[97] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. Pear: a fast and accurate illumina paired-end read merger. *Bioinformatics*, 30(5):614–620, 2014. Oxford University Press.

[98] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. Sage Publications Sage CA: Thousand Oaks, CA.

[99] Archiconda [internet]. `https://github.com/Archiconda/build-tools/releases`, Accessed 2019-12-29. Archiconda.

[100] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. Elsevier.

Appendix A

Tools with Implemented Configuration Roles

Configuration roles for some common bioinformatics tools were generated to assist researchers in easily configuring the tools. Future work will explore building out execution modules for these tools.

BLAST The Basic Local Alignment Search Tool (BLAST) [100] compared nucleotide sequences to a database in order to detect biological similarity.

BWA The Burrows-Wheeler Aligner (BWA) [7] for mapping nucleotide sequences against a large reference genome.

CD-Hit CD-Hit [79] clusters and compares nucleotide and protein sequences based on similarity thresholds. CD-Hit can also identify duplicate reads from Illumina data, overlapping reads, and cluster rRNA tags into OTUs.

Deblur Deblur [84] profiles nucleotide sequencing errors to distinguish closely related microbes in microbial communities into Amplicon Sequence Variants (ASVs). Deblur is semantically similar to DADA2 [3].

EMBOSS The European Molecular Biology Open Software Suite (EMBOSS) [64] is a collection of common tools for sequence analysis. Some tools include local and global alignment algorithms and file format conversions.

FastTree FastTree [82] is a tool for inferring evolutionary relationships of multiple DNA or protein sequences.

Globus Globus [98] enables large parallel file transfer ideal for Next-Generation Sequencing (NGS) data. This role installs the Globus Toolkit as well as Globus Connect Personal to enable setup of a Globus personal endpoint for file transfers.

PEAR PEAR [97] merges paired-end Illumina reads. It is semantically similar to FLASH2 [2].

T-Coffee T-Coffee [81] is a tool for multiple sequence alignments.

Appendix B

Results for Taxonomic Assignment

Taxonomic assignments for each dataset were generated 10 times with the same parameters. The act of training the RDP classifier each time the classifier was run caused non-determinism at this step. As the taxonomic level gets finer, the proportion of deviations rises, however; there were still deviations observed for the large dataset at the kingdom level.

Small Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0	0	0	0	0	0	0	0	0
Rep 2		0	0	0	0	0	0	0	0	0
Rep 3			0	0	0	0	0	0	0	0
Rep 4				0	0	0	0	0	0	0
Rep 5					0	0	0	0	0	0
Rep 6						0	0	0	0	0
Rep 7							0	0	0	0
Rep 8								0	0	0
Rep 9									0	0
Rep 10										0

Medium Dataset

0.032895

0

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0	0	0	0	0	0	0	0	0
Rep 2		0	0	0	0	0	0	0	0	0
Rep 3			0	0	0	0	0	0	0	0
Rep 4				0	0	0	0	0	0	0
Rep 5					0	0	0	0	0	0
Rep 6						0	0	0	0	0
Rep 7							0	0	0	0
Rep 8								0	0	0
Rep 9									0	0
Rep 10										0

Large Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.000324	0.000243	8.11096E-05	0.000243329	0.000243	0.000243329	0.000243329	0.000406	0.000324438
Rep 2		0	0.000406	0.000243329	8.11096E-05	0.000243	8.11096E-05	8.11096E-05	0.000243	0.000162219
Rep 3			0	0.000324438	0.000324438	0.000324	0.000324438	0.000324438	0.000324	0.000405548
Rep 4				0	0.000162219	0.000162	0.000162219	0.000162219	0.000324	0.000243329
Rep 5					0	0.000324	0	0.000162219	0.000162	8.11096E-05
Rep 6						0	0.000405548	0.000162219	0.000406	0.000486657
Rep 7							0	0.000162219	0.000162	8.11096E-05
Rep 8								0	0.000324	0.000243329
Rep 9									0	0.000243329
Rep 10										0

Figure B.1: The proportion differences in the Kingdom level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.

Small Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.00328947	0.003289	0	0	0	0	0.003289	0	0.003289
Rep 2		0	0	0.003289	0.003289	0.003289	0.003289	0.006579	0.003289	0.006579
Rep 3			0	0.003289	0.003289	0.003289	0.003289	0.006579	0.003289	0.006579
Rep 4				0	0	0	0	0.003289	0	0.003289
Rep 5					0	0	0	0.003289	0	0.003289
Rep 6						0	0	0.003289	0	1.08E-05
Rep 7							0	0.003289	0	0.003289
Rep 8								0	0.003289	0.006579
Rep 9									0	0.003289
Rep 10										0

Medium Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.00395648	0.005275	0.003297	0.003956	0.003956	0.002638	0.002967	0.002308	0.002967
Rep 2		0	0.004286	0.002638	0.003297	0.002638	0.002638	0.003297	0.002308	0.002308
Rep 3			0	0.003297	0.003297	0.003956	0.004286	0.003627	0.002967	0.003627
Rep 4				0	0.001978	0.001978	0.002638	0.002308	0.001649	0.002308
Rep 5					0	0.001978	0.002638	0.002967	0.002967	0.002308
Rep 6						0	0.003297	0.003627	0.002308	0.003627
Rep 7							0	0.003297	0.002308	0.002308
Rep 8								0	0.002638	0.003297
Rep 9									0	0.001978
Rep 10										0

Large Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	1.6221E-05	0.003163	0.001784	0.002271	0.002028	0.002433	0.00219	0.002271	0.00219
Rep 2		0	0.00292	0.002514	0.002352	0.00146	0.002514	0.001947	0.002677	0.001784
Rep 3			0	0.00219	0.002677	0.002271	0.002677	0.002109	0.002352	0.002596
Rep 4				0	0.002271	0.002028	0.002596	0.001703	0.001947	0.002028
Rep 5					0	0.002514	0.003082	0.001866	0.002433	0.00219
Rep 6						0	0.00219	0.001622	0.002514	0.00146
Rep 7							0	0.002514	0.002596	0.002677
Rep 8								0	0.002352	0.00146
Rep 9									0	0.002352
Rep 10										0

Figure B.2: The proportion differences in the Phylum level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.

Small Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.003289	0.006579	0.006579	0	0.003289	0.003289	0.006579	0.003289	0.006579
Rep 2		0	0.003289	0.009868	0.003289	0.006579	0.006579	0.009868	0.006579	0.009868
Rep 3			0	0.006579	0.006579	0.003289	0.009868	0.006579	0.003289	0.006579
Rep 4				0	0.006579	0.003289	0.009868	0.006579	0.003289	0.006579
Rep 5					0	0.003289	0.003289	0.006579	0.003289	0.006579
Rep 6						0	0.006579	0.003289	0	0.003289
Rep 7							0	0.009868	0.006579	0.003289
Rep 8								0	0.003289	0.006579
Rep 9									0	0.003289
Rep 10										0

Medium Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.004286	0.004946	0.004286	0.004616	0.005275	0.003297	0.003297	0.004286	0.004616
Rep 2		0	0.004616	0.005275	0.003627	0.004286	0.004286	0.004286	0.002638	0.003627
Rep 3			0	0.005275	0.006264	0.006264	0.006264	0.004946	0.005275	0.005605
Rep 4				0	0.003627	0.004286	0.004286	0.004946	0.004616	0.004286
Rep 5					0	0.002638	0.003297	0.004616	0.003627	0.003956
Rep 6						0	0.003956	0.005275	0.004946	0.004616
Rep 7							0	0.004616	0.002967	0.003956
Rep 8								0	0.003627	0.005275
Rep 9									0	0.004286
Rep 10										0

Large Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.002677	0.003569	0.002758	0.002677	0.003163	0.003082	0.003407	0.003407	0.003082
Rep 2		0	0.003488	0.002839	0.003082	0.002596	0.002839	0.002839	0.003325	0.002839
Rep 3			0	0.003244	0.00365	0.003163	0.003569	0.003082	0.003407	0.003569
Rep 4				0	0.003001	0.003001	0.003244	0.002758	0.00292	0.002433
Rep 5					0	0.003244	0.003325	0.002839	0.003163	0.00365
Rep 6						0	0.003325	0.002514	0.003163	0.003001
Rep 7							0	0.003082	0.003082	0.003731
Rep 8								0	0.003244	0.002758
Rep 9									0	0.003569
Rep 10										0

Figure B.3: The proportion differences in the Class level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.

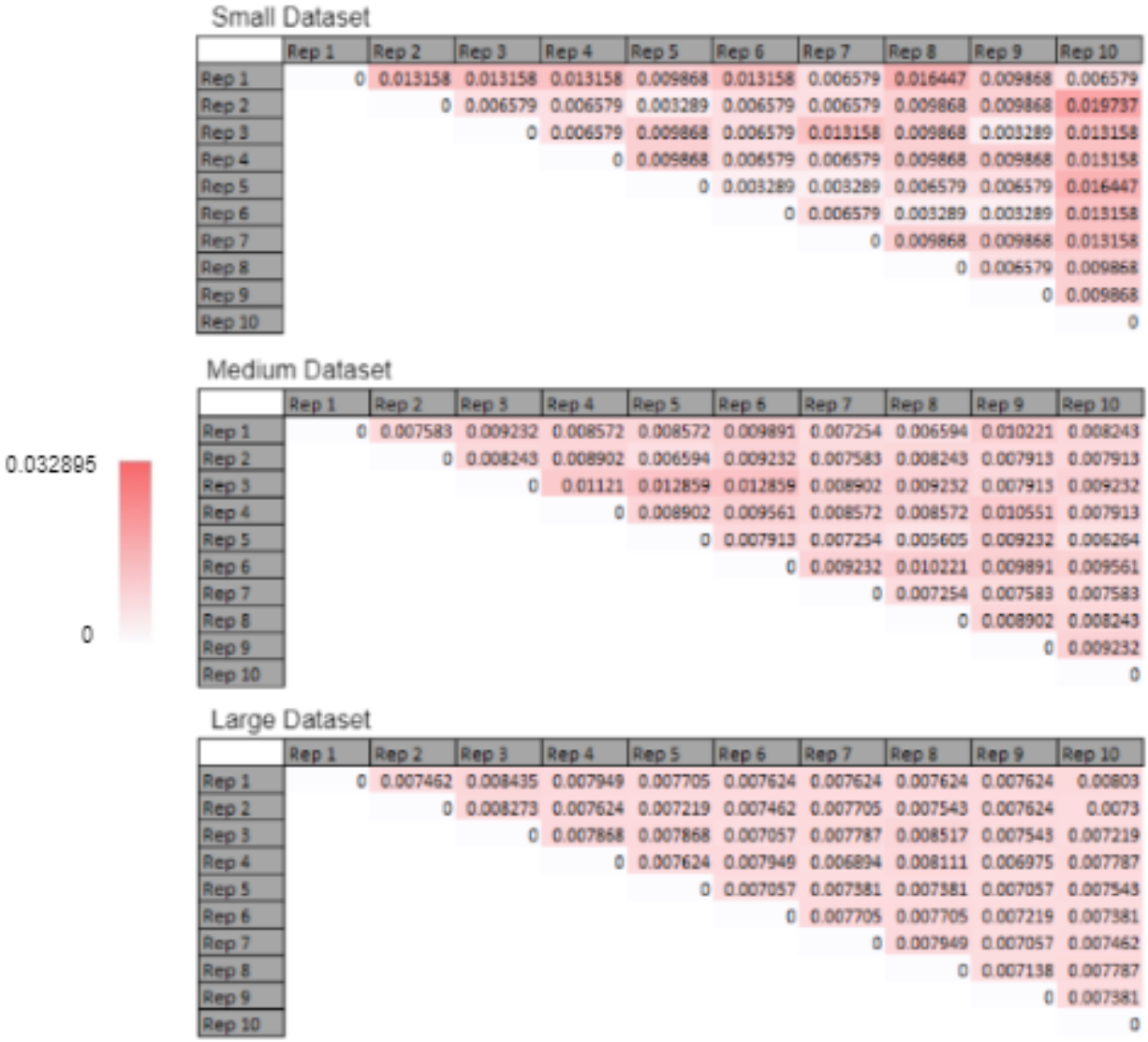


Figure B.4: The proportion differences in the Order level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.

Small Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.013158	0.009868	0.016447	0.006579	0.013158	0.003289	0.013158	0.006579	0.006579
Rep 2		0	0.009868	0.009868	0.006579	0.006579	0.009868	0.013158	0.013158	0.019737
Rep 3			0	0.013158	0.009868	0.009868	0.013158	0.009868	0.003289	0.009868
Rep 4				0	0.016447	0.009868	0.013158	0.016447	0.016447	0.016447
Rep 5					0	0.006579	0.003289	0.006579	0.006579	0.013158
Rep 6						0	0.009868	0.006579	0.006579	0.013158
Rep 7							0	0.009868	0.009868	0.009868
Rep 8								0	0.006579	0.006579
Rep 9									0	0.006579
Rep 10										0

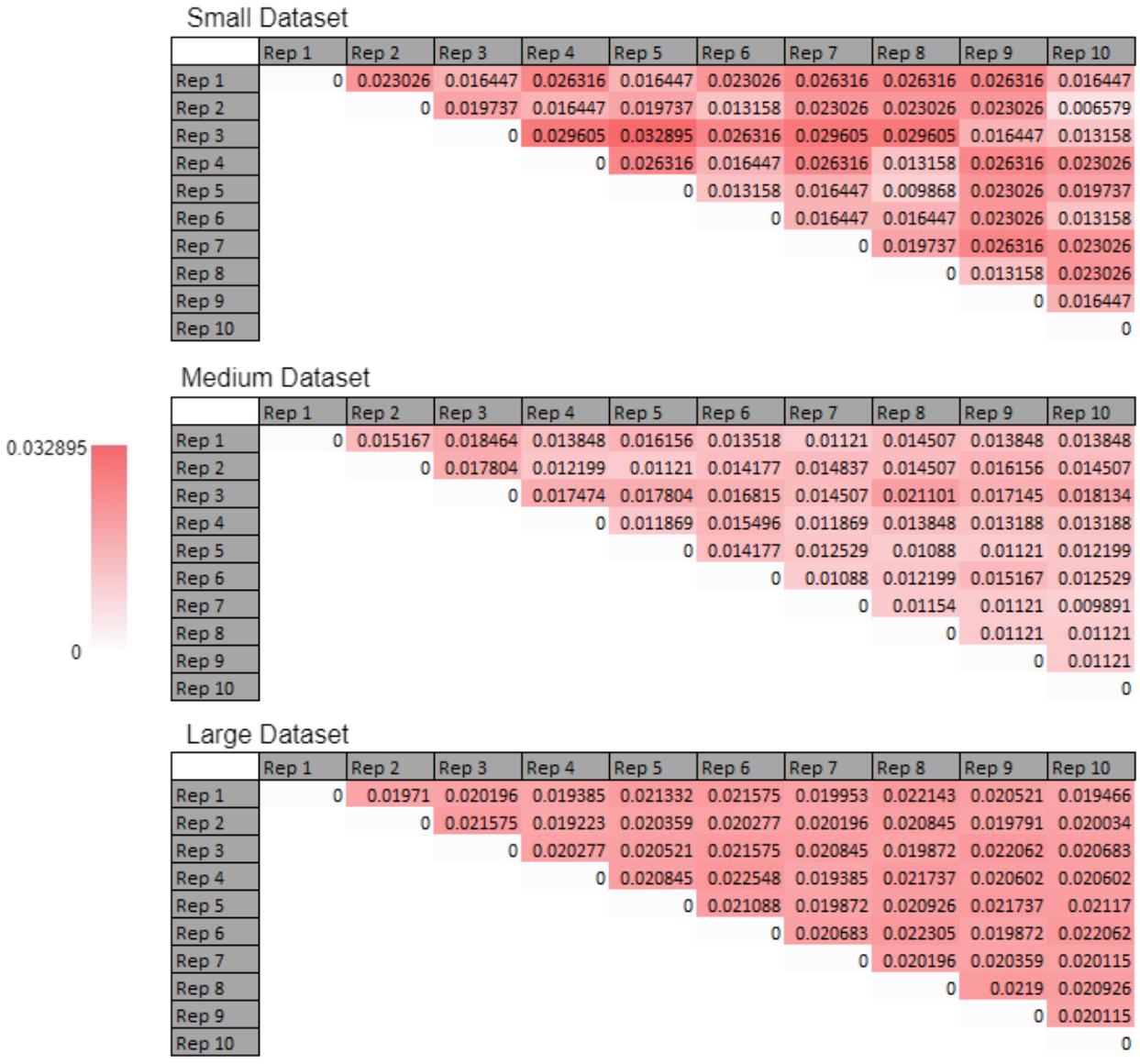
Medium Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.013188	0.013518	0.009891	0.010551	0.012529	0.008572	0.009232	0.01154	0.010551
Rep 2		0	0.013518	0.010221	0.009232	0.010221	0.010551	0.011869	0.012199	0.01154
Rep 3			0	0.013518	0.013518	0.012529	0.013848	0.013518	0.01088	0.012859
Rep 4				0	0.007583	0.011869	0.009232	0.009232	0.009891	0.009232
Rep 5					0	0.009232	0.009561	0.008572	0.010221	0.008572
Rep 6						0	0.01088	0.01154	0.013518	0.01088
Rep 7							0	0.009561	0.010221	0.008902
Rep 8								0	0.010221	0.009232
Rep 9									0	0.011869
Rep 10										0

Large Dataset

	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.008111	0.008192	0.008598	0.007705	0.008192	0.007868	0.008111	0.008192	0.008517
Rep 2		0	0.009003	0.008435	0.008354	0.008841	0.007705	0.009571	0.008598	0.008841
Rep 3			0	0.008517	0.008435	0.008517	0.008192	0.009814	0.008273	0.00876
Rep 4				0	0.009003	0.009246	0.00803	0.009571	0.008517	0.009165
Rep 5					0	0.008111	0.00876	0.008679	0.007949	0.00876
Rep 6						0	0.00876	0.009814	0.008922	0.008679
Rep 7							0	0.008841	0.007624	0.00803
Rep 8								0	0.008598	0.008354
Rep 9									0	0.008517
Rep 10										0

Figure B.5: The proportion differences in the Family level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.



Small Dataset	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.023026	0.016447	0.026316	0.016447	0.023026	0.026316	0.026316	0.026316	0.016447
Rep 2		0	0.019737	0.016447	0.019737	0.013158	0.023026	0.023026	0.023026	0.006579
Rep 3			0	0.029605	0.032895	0.026316	0.029605	0.029605	0.016447	0.013158
Rep 4				0	0.026316	0.016447	0.026316	0.013158	0.026316	0.023026
Rep 5					0	0.013158	0.016447	0.009868	0.023026	0.019737
Rep 6						0	0.016447	0.016447	0.023026	0.013158
Rep 7							0	0.019737	0.026316	0.023026
Rep 8								0	0.013158	0.023026
Rep 9									0	0.016447
Rep 10										0

Medium Dataset	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.015167	0.018464	0.013848	0.016156	0.013518	0.01121	0.014507	0.013848	0.013848
Rep 2		0	0.017804	0.012199	0.01121	0.014177	0.014837	0.014507	0.016156	0.014507
Rep 3			0	0.017474	0.017804	0.016815	0.014507	0.021101	0.017145	0.018134
Rep 4				0	0.011869	0.015496	0.011869	0.013848	0.013188	0.013188
Rep 5					0	0.014177	0.012529	0.01088	0.01121	0.012199
Rep 6						0	0.01088	0.012199	0.015167	0.012529
Rep 7							0	0.01154	0.01121	0.009891
Rep 8								0	0.01121	0.01121
Rep 9									0	0.01121
Rep 10										0

Large Dataset	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10
Rep 1	0	0.01971	0.020196	0.019385	0.021332	0.021575	0.019953	0.022143	0.020521	0.019466
Rep 2		0	0.021575	0.019223	0.020359	0.020277	0.020196	0.020845	0.019791	0.020034
Rep 3			0	0.020277	0.020521	0.021575	0.020845	0.019872	0.022062	0.020683
Rep 4				0	0.020845	0.022548	0.019385	0.021737	0.020602	0.020602
Rep 5					0	0.021088	0.019872	0.020926	0.021737	0.02117
Rep 6						0	0.020683	0.022305	0.019872	0.022062
Rep 7							0	0.020196	0.020359	0.020115
Rep 8								0	0.0219	0.020926
Rep 9									0	0.020115
Rep 10										0

Figure B.6: The proportion differences in the Genus level of taxonomic assignments for the 3 datasets. The opacity of the colour in each cell visually represents the the proportions of differences in taxonomic assignment. The darker the colour, the higher the proportion of differences in taxonomic assignment.