Tool to remove specific organisms from microbiome sequencing data

Host Contamination Removal Tool (HoCoRT)

Ignas Rumbavicius

Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Tool to remove specific organisms from microbiome sequencing data

Host Contamination Removal Tool (HoCoRT)

Ignas Rumbavicius

Abstract

High Throughput Sequencing (HTS) technologies increased the amount of sequencing data available, and created the demand for fast tools to process it. Host contamination is especially problematic in human microbiome research. A standardized tool for removal of host contamination in such studies is especially important to ensure reproducibility and comparability of the results.

An open-source Python tool for the removal of specific organisms from sequencing reads is presented. It was named HoCoRT (Host Contamination Removal Tool), and is a modular and extendable command line tool which combines existing tools in pipelines to achieve it's goal of host contamination removal. It is distributed via Bioconda to ensure ease of installation. HoCoRT implements a positive security model for input argument validation, and utilizes piping and efficient parallel tools to ensure fast runtime. The pipelines available in HoCoRT were tested in a controlled environment on raw simulated unprocessed sequencing reads. HoCoRT was also tested on real public datasets, and was compared to DeconSeq.

HoCoRT is available at https://github.com/ignasrum/hocort, and supports both Illumina and Nanopore reads. It was found that most HoCoRT pipelines performed well in the simulated dataset tests. HoCoRT was able to find human host contamination in public datasets. It was also shown that the HoCoRT Bowtie2 pipeline was much faster than DeconSeq at aligning both Illumina HiSeq and MiSeq reads, while having higher accuracy and precision, but slightly lower sensitivity (lower by at most 0.0003, where 1.0 is the maximum). Bowtie2 in end-to-end mode is recommended for Illumina reads (both HiSeq and MiSeq) as it maintained fast runtime, high accuracy, high precision and high sensitivity when compared to HISAT2, Kraken2, BBMap, BWA-MEM2, and Minimap2. Minimap2 is recommended for Nanopore reads because it had high sensitivity, precision and accuracy across all Nanopore tests.

Contents

Preface

This masters thesis was written to fulfill the requirements of the Programming and System Architecture masters degree at UiO.

This thesis was a learning experience throughout the whole process as it was the most scientific document I have ever written. It was also the first time I published a production-ready software package.

I would like to thank my supervisors Torbjørn Rognes and Trine B. Rounge for their excellent guidance, and my family for their support.

Ignas Rumbavicius
Oslo, 10 May 2022

Definitions

Microbiome

The different microbial communities coexisting in a specific environment [57].

Sequence Contamination

Contaminated sequences are sequences which contain information from foreign sources [29]. The foreign sources range from sequencing adapters to plasmids and host contamination [29].

Next generation sequencing

Next generation sequencing (NGS), are the massively parallel sequencing technologies able to sequence the entire human genome within a single day [8]. Prior to NGS, the Sanger sequencing technology required a decade to deliver the first draft of the human genome [8].

Sequencing reads

A raw sequence of base pairs (corresponding to a DNA fragment) which comes from a sequencing machine [39].

Unpaired/Single-end sequencing reads

Sequencing reads where each read corresponds to one end of a DNA fragment [1].

Paired-end sequencing reads

Two sequencing reads 1 and 2 where; read 1 corresponds to one end of a DNA fragment, and read 2 corresponds to the other end of the same DNA fragment [54].

True Positive (TP)

An actual contaminant sequence correctly classified as a contaminant sequence [65].

True Negative (TN)

An actual microbial sequence correctly classified as a microbial sequence [65].

False Positive (FP)

An actual microbial sequence incorrectly classified as a contaminant sequence [65].

False Negative (FN)

An actual contaminant sequence incorrectly classified as a microbial sequence [65].

Sensitivity (recall)

Ratio between the retrieved relevant instances to the total number of all relevant instances [65].

$$\frac{TP}{TP + FN}$$

Precision

Ratio between the retrieved relevant instances to the total number of retrieved instances [65].

$$\frac{TP}{TP + FP}$$

Accuracy

Ratio between the correctly classified instances to the total number of instances [65].

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Chapter 1

Introduction

1.1 Background and motivation

As DNA sequencing technologies advanced and High Throughput Sequencing (HTS) became available, the amount of sequencing data increased greatly. These developments enabled us to sequence whole genomes of different organisms in a fast and highly parallel way. They also greatly advanced the field of microbiome studies, which are a very important part of being able to understand the interactions between the host organism and the different microbial communities within it.

The microbiome modulates the host immune responses, is important in the metabolic processes, production of vitamins, and many other processes [9]. It is an important factor for the health and well-being of the host organism.

There are a multitude of reasons for why we need to remove host contamination when conducting research. These range from ethical to practical. Existing tools are often not user friendly, are not always optimally set-up by default and may not be sufficiently extendable to remove host contamination in an optimal way.

Many countries have laws and policies regarding genomic privacy, such as the General Data Protection Regulation (GDPR) in EU, which may limit what researchers are able to study and publish. Monetary fines and other measures may be taken if the entities working with and storing such information do not respect these policies [2].

Sequencing reads that come out of a sequencing machine may have host contamination. To find the host-related sequences, the sequences are mapped to a reference genome. The sequences that map well to reference genomes of suspected hosts, are removed.

Many factors may affect the results of microbiome studies. Erraneous sample handling, choice of DNA extraction methods and other factors may lead to poor and non-reproducible results [9]. Reproducibility and comparability of microbiome research is important to gain a broad insight and knowledge. As pointed out in a review article published in 2021, open-source availability, ease of installation, and a proper user interface of the tools would help with reproducibility and comparability of microbiome studies [9].

Currently there are few tools specifically made for host contamination removal. The main ones are DeconSeq [59] and GenCoF [30]. There are ad-hoc ways of removing host contamination, but this often compromises reproducibility and comparability of the results.

1.2 Sequencing data and reference databases

Sequencing reads come from sequencing machines after sequencing a DNA sample [28]. This sample may contain the host's DNA which then gets sequenced and contaminates the sequencing reads. These host sequences need to be removed when conducting microbiome research. Sequence data may be presented in FASTQ, FASTA, SAM/BAM, GFF/GTF or other formats [72]. File formats such as FASTQ also include nucleotide quality scores along with the actual sequence [72]. These sequences are then aligned with sequences in a reference database to find matching sequences. Being able to do this in a fast and accurate way is very important when doing microbiome research.

1.3 Microbiome

The different microbial communities coexisting in a specific environment constitute a microbiome [57]. These microbes include bacteria, fungi, parasites and viruses [57]. Trillions of microorganisms of thousands of different species may coexist peacefully in a microbiome [67]. If the microbiome is disturbed by bad diets, illnesses or medications, it may lead to diseases in the body of the host [57].

The composition of the gut microbiome has been suggested to greatly affect our health [42]. It may play a role in the development of various allergies and diseases [42]. It may even affect our mental health and personality traits, and may influence our level of stress, anxiety and depressive symptoms [42].

Microbiome samples often contain host contamination. This presents great ethical and privacy issues as human DNA may get sequenced and published in public archives. This DNA may then be used to identify the study subjects, and gain knowledge about them. This may include their ethnicity, their sex and looks (e.g. eye and hair color), and even illnesses and diseases they are genetically predisposed to.

1.4 Proposed Tool

A tool to remove host contamination is needed. Many of the existing tools which are being used today are hard to use and require technical knowledge. Many researchers may leave the default settings or may incorrectly setup the tools, which may result in inefficient or inaccurate removal of host contamination. This may both unnecessarily remove too many non-host sequences, and/or leave too much host contamination.

The problem of fast and accurate host contamination removal is hard to solve. There is a balance between removing too many false positives and too few true positives which has to be met. In addition, certain types of sequences are more prevalent in some genomes than others. For example, repetitive sequences may constitute over 2/3 of the human genome [33]. The host contamination tool may have to be configured for the specific problem and genome for it to be performant and accurate. It also has to be user friendly and extendable, so that most researchers are able to actually use and extend it when needed.

A program which solves the challenges mentioned above regarding host contamination is proposed in this thesis. This tool will combine different existing algorithms and tools to efficiently and accurately remove host contamination. Existing tools will be tested in different configurations and the most performant and accurate of these will be selected and provided as presets. Customization will be available to the user. The tool will published under an open source license.

1.5 Existing Tools

1.5.1 Mapping/Alignment tools

To identify host contamination, sequencing reads have to be aligned to a reference index containing the host genome. The generic tools below are tools that align sequencing reads to reference genomes. These tools are often made to fit into custom pipelines, and a selection of these will be used in the proposed host contamination removal program.

Bowtie2

Bowtie2 [48] is a popular and widely used sequence alignment tool. It is especially good at aligning sequencing reads from around 50 to 1000s of bases to relatively long genomes [18]. In other words, it is especially good at aligning short sequencing reads to relatively long genomes such as the human genome. For the human genome, Bowtie2 [48] uses around 3.2 GB of RAM [18].

It supports gapped, local, and paired-end alignments [18]. It is memory-efficient and relatively fast. It is a command line based tool and supports Windows, Mac OS X, and Linux [18].

It provides many different command line parameters, and, thereby, many different combinations of them. It may be unclear what the optimal command line parameters are, and the tool may be complicated to configure and use in an optimal manner.

Bowtie2 [48] allows installation through Conda and manual installation by downloading binaries from the internet [18].

Scalable Nucleotide Alignment Program (SNAP)

SNAP [71] (Scalable Nucleotide Alignment Program) is a sequence alignment tool [62]. It is optimized for aligning sequencing reads from around 100 bases and longer [62]. SNAP [71] is around 2–5x faster than tools such as Bowtie2 [48] [63]. It takes unprocessed FASTQ files as input and outputs fully sorted and indexed SAM alignment files [63]. Similarly to Bowtie2 [48], it is also good at aligning shorter sequencing reads from Illumina machines to reference genomes.

It is primarily made for HPC and requires 10s of gigabytes of RAM. Aligning to the human genome requires at least 48 GB of RAM, and preferably around 64 GB [63]. It leverages these large memory capacity requirements to achieve greater accuracy and faster speed (10x-100x) than tools such as BWA [49] [71].

SNAP [71] can be used across a large range of read lengths from 100 to 10000 base pairs, thereby supporting next generation sequencing reads [71].

Similarly to Bowtie2 [48], SNAP [71] provides many different options, which makes it hard to set up in an optimal way. It allows installation via Conda or by manually downloading binaries [61, 62]. SNAP [71] runs on the command line and supports Linux, Mac OS X and Windows [62].

HISAT2

HISAT2 [43] is an alignment tool for mapping next-generation sequencing reads to either a population of human genomes or a single genome [37]. It outputs alignments in SAM format, and is distributed under GPLv3 [38]. HISAT2 [43] runs on the command line under Linux, Mac OS X and Windows [38]. HISAT2 [43] uses about 6.7 GB of RAM for the human reference genome [37].

It is based on the implementations of Bowtie2 [48] and HISAT, and uses the original idea of building graph FM indexes [37].

HISAT2 [43] is also a complicated tool offering many features and even more different possible configurations. Just like Bowtie2 [48] and SNAP [71], it can be installed either using Conda or manually by downloading binaries [36, 38].

Basic Local Alignment Search Tool (BLAST)

BLAST [4] is a heuristic algorithm and program for aligning sequences of nucleotides and amino acids [14]. The user can align a sequence to a database of sequences, and find the sequences in the database which match the query sequence above a certain threshold.

BLAST [4] can output to HTML, plain text, XML, and other formats.

BLAST [4] is available online on the NCBI website [14]. It is also available locally through Conda, or as binaries on the NCBI websites [11, 12]. BLAST [4] supports Windows, Linux, and Mac OS [13].

Minimap2

Minimap2 [52] is a sequence aligner which is available as a command line tool [56]. Minimap2 [52] works with sequence lengths of up to hundreds of megabases. It is tens of times faster than mainstream aligners (such as BLASR [27] and BWA-MEM [51]) at aligning long next-generation sequences (noisy >10kb) [56]. It is about three times faster than Bowtie2 [48] for >100bp Illumina reads [56]. Minimap2 [52] provides a few command-line presets for short and long sequencing reads which makes it easier to use. Minimap2 [52] uses around 6.8 GB of memory for human reads.

Minimap2 [52] can output to SAM format, and is available in the form of downloadable binaries or through Conda [55, 56]. Minimap2 [52] officially supports only Linux [56].

BWA-MEM2

BWA-MEM2 [68] is a sequence aligner, and it is the successor to BWA-MEM [51] [22]. It gives the same alignments as BWA-MEM, and is 1.3–3.1x faster [22]. Default BWA-MEM2 [68] uses about 10 GB of memory, but the git repository also provides a separate branch providing another 10%–30% speedup by using enumerated radix trees [22]. The downside is that this version uses about 60 GB of memory [22].

The output is in the SAM format [22].

BWA-MEM2 [68] supports Linux, and can be installed either by downloading binaries or through Conda [21, 22].

BBMap

BBMap [20] is a sequencing read aligner for DNA and RNA sequencing reads [7]. It supports Illumina, PacBio, 454, Sanger, Ion Torrent, and Nanopore platforms [7]. The only dependency it has is Java, and it can therefore run on any platform [6]. BBMap [20] has no upper limit to genome size or a limit on number of contigs [7]. BBMap [20] uses about 3–6 bytes of memory per reference base [7]. The human reference genome assembly GRCh37.p13 is 3,234,834,689 bases long [40].

$3,234,834,689 * 3 = 9,704,504,067B \approx 9.7GB$

$3,234,834,689 * 6 = 19,409,008,134B \approx 19.4GB$

BBMap [20] uses about 9.7–19.4 GB of memory.

BBMap [20] can be installed either by downloading binaries or through Conda [5, 6].

1.5.2 Taxonomic classification tools

Kraken2

Kraken2 [69] is a taxonomic classifier. It was made partly as a solution to Kraken 1's high memory requirements (can easily exceed 100GBs of RAM) [69]. Kraken2's [69] default database memory requirements have been reduced to around 29 GB [47]. The default database contains different reference genomes. The memory requirements can be reduced by building a custom database which only contains the necessary genomes (human genome when removing human host contamination). Aside from memory requirements, Kraken2 [69] provides other improvements, such as faster database build times, smaller database sizes, and faster classification maintaining the same accuracy as Kraken 1.

Kraken2 [69] can be installed either by downloading the source code and installing manually, or through Conda [46]. Kraken2 [69] officially supports only Linux [47].

Centrifuge

Centrifuge [44] is a taxonomic classifier for metagenomic sequences [25]. It allows fast and accurate labelling and quantification of microbial species [25]. Centrifuge [44] requires a relatively low amount of memory, 4.2 GB for 4078 bacterial and 200 archeal genomes.

It supports Linux and Mac OS, and can be installed by downloading binaries or through Conda [24, 26].

1.5.3 Comparison

Memory usage depends on many factors such as the reference genome size and speed that is wished to be achieved. Therefore, only a suggestion of memory usage will be provided in the table below.

The sequence lengths are also a suggestion as it was hard to find clearly defined limits for what the programs can handle.

Table 1.1: Comparison of generic tools

Tool	Sequence Length	Memory Usage	Operating System
Minimap2 [52]	>100Mbp	6.8 GB [52]	Linux
BWA-MEM2 [68]	70bp–1000bp+	10–60 GB [22]	Linux
SNAP [71]	100bp+	48–64 GB [63]	Windows & Linux & Mac OS
BBMap [20]	50bp–1000bp+	9.7–19.4 GB [7, 40]	Windows & Linux & Mac OS
Kraken2 [69]	—	29 GB [47]	Linux
Bowtie2 [48]	50bp–1000bp+	3.2 GB [18]	Windows & Linux & Mac OS
HISAT2 [43]	Next gen.	6.7 GB [37]	Windows & Linux & Mac OS
BLAST [4]	—	—	Windows & Linux & Mac OS
Centrifuge [44]	—	4.2 GB [44]	Linux & Mac OS

1.5.4 Host Contamination Removal Tools

These tools specifically tackle the problem of host contamination removal.

GenCoF

GenCof [30] is a GUI for filtering out human reads from metagenomic files [34]. It uses the programs Sickle or Prinseq, Bowtie2 [48], and Fastq or Fasta Splitter [30, 34]. This limited number of tools cannot be easily expanded as the tools directly integrated into the source code of GenCof [30].

GenCof [30] supports Linux and Mac OS, and can be installed by downloading binaries from the internet [34].

DeconSeq

DeconSeq [59] provides a user-friendly command-line interface for host contamination removal from sequence datasets [31]. Again, this tool suffers from the same issues that GenCof [30] suffers from as it cannot be easily extended as the usage of BWA-SW [53] is directly integrated into the source code [31].

DeconSeq [59] can be downloaded as a binary [31].

1.5.5 Samtools

Samtools [50] is used to manipulate files where sequence alignment data is stored. Two examples of such file types are; SAM files and BAM files.

The alignment/mapping tools mentioned above often output SAM or BAM files. Samtools [50] will be used to extract specific reads from the alignment files.

Samtools [50] is available on Bioconda [66] and as a binary. Samtools [50] supports Linux and Mac OS.

Chapter 2

Goals

2.1 General Requirements

First of all, the proposed tool has to be able to remove host contamination. It needs to be performant, accurate, easy to use, maintainable and extensible. The tool will combine existing algorithms and solutions to achieve these goals.

2.1.1 Performance and Accuracy

There is a major tradeoff between performance and accuracy. The tool can be extremely fast and inaccurate, or it can be extremely accurate, but slow. The goal is to provide configurations with optimal tradeoffs, so the tool is both performant and accurate in different contexts and problems.

2.1.2 Usability

The tool has to work with different model organisms and be able to be applied to different problems. The ability to provide custom configurations is important here. Existing tools can be combined and setup with different parameters to solve most problems. The proposed tool will be provided at different abstraction levels. A code library will be provided for use in custom pipelines and scripts. A terminal interface will be provided for use on headless Unix based machines. A graphical interface may optionally be provided for users who are unable to use the terminal interface. Lastly, there will be focus on documentation, and clear and understandable error messages.

2.1.3 Maintainability and Extensibility

An important aspect of this tool will be modularity. To make it maintainable, it must be easy to remove and add the existing tools. This will be achieved by having a modular architecture as shown in figure 3.1.

Extensibility will be achieved by both having a modular architecture and enabling the users to provide custom configurations.

2.2 Goals

It is assumed that all quality control is done prior to providing the sequence data as input to the proposed tool. Therefore, data quality control will not be part of the goals.

2.2.1 Primary Goals

- Improved removal of host contamination
  - Provide access to a code library
  - Provide a terminal interface
  - The tool has to be simple to use (simple parameters, optimized presets)
  - The tool has to be simple to maintain (modular, high cohesion, low coupling)
  - Target OS: Linux x86

2.2.2 Secondary Goals

- Modular architecture (high cohesion, low coupling)
- Provide optimal configuration presets
- Accept custom configurations (provided by user)
- Abstract away as many parameters as possible, but leave the freedom to change them when needed
- Provide good documentation
- Output clear and understandable error messages

2.2.3 Optional Goals

- Provide a graphical interface (GUI)

Chapter 3

Methods

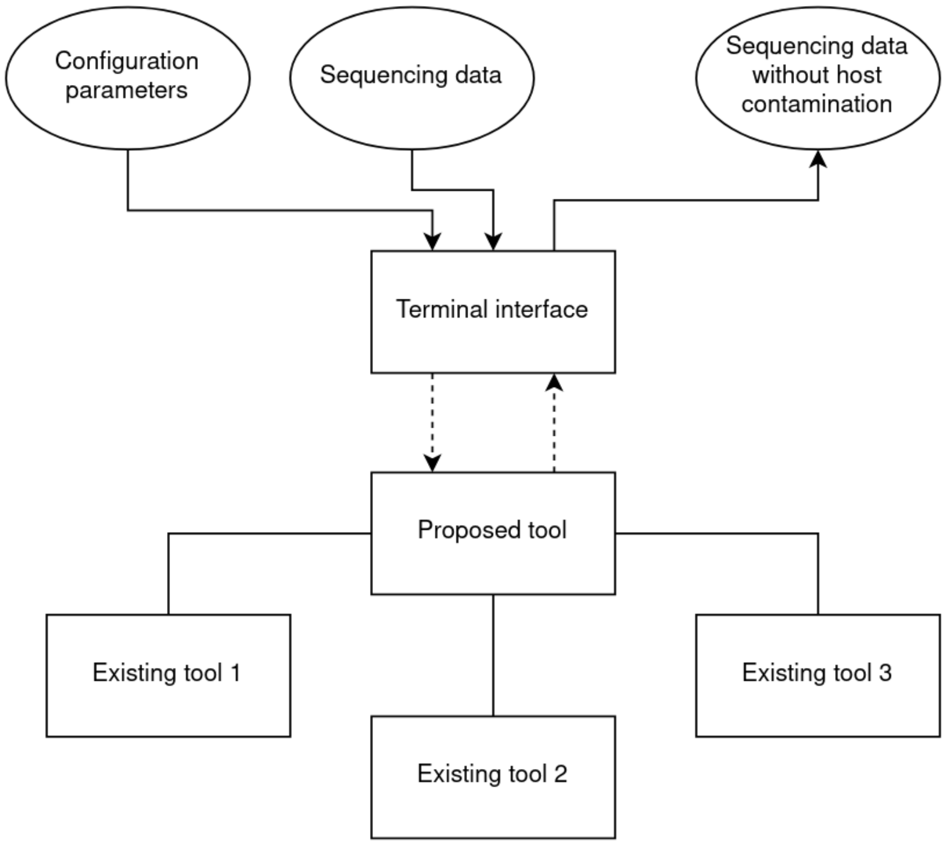

Figure 3.1: Proposed tool

3.1 Code library

The code library will be the main program, and, thereby, the focus of this project. It will accept sequencing reads as input, and will remove host contamination from them. It will combine and configure different existing tools to remove the host DNA sequences. The code library will also allow the user to supply custom configuration parameters.

The terminal and, optionally, graphics interfaces will be built on top of the code library. The expectation is that most users will prefer to use either the code library in custom pipelines or the terminal interface, rather than the graphical interface. Therefore, the code library and the terminal interface will have the highest priority.

3.1.1 Modularity and APIs

The modularity aspect of the proposed tool is visible in figure 3.1. The code library will provide an API for easy access in custom pipelines or other applications. This way, different terminal and graphics interfaces may also be built on top of this API. Documentation will be provided for this API for ease of use and simple maintenance.

The supported existing tools will have APIs implemented as custom classes which expose their functionality in a simple and secure way. The existing tools are often terminal based, and it is important to not directly run user input in the terminal to avoid malicious code being run.

3.2 Command line interface (CLI)

As mentioned earlier, a command line interface will be built on top of the code library. This interface will be quite simple. The only things the user will provide will be the configuration parameters and the sequencing reads. These will be passed on to the code library and the actual work of removing host contamination will be done in the code library.

```
proposed-tool config sequences
```

3.3 Development methodology

The development methodology chosen is Kanban. A Kanban board will be implemented using tools such as Trello. Kanban was chosen mainly because of its efficiency as it has been suggested that using Kanban over Scrum reduces the number of bugs and increases productivity [60].

3.4 Main tools

The main tools which will be used for the development of the proposed tool are Python and Git. Github will be used to publish the source code and documentation, and also to run continuous integration tests. Pytest will be used for development of tests.

3.5 Documentation

Good documentation allows the users to easily understand how to use a tool, and the maintainers and developers to easily understand how a tool works.

The programming language used will be Python, and pdoc3 will be used for the technical part of the documentation. Some technical documentation will also be provided in the form of comments in the source code.

The documentation related to the usage of the tool will be on the Github pages and also in the tool itself as part of a help command. These choices are based on the availability and accessibility of the resulting documentation. It must be easy and simple to find and access the documentation for it to have value.

3.6 Distribution

Because the proposed tool will heavily depend on different existing tools, there will be a need to manage dependencies in an effective manner. Conda is a popular tool to do this in many situations where software needs to be distributed. Bioconda [66] is a channel for the conda package manager which specializes in distribution of bioinformatics software [10]. Because most bioinformatics tools are already available on Bioconda [66], this will be a great solution for managing dependencies. Alternatively, a Docker [15] container may also be provided. The main requirement here is that the tool must be very simple to install. The user should not have to fiddle with setup of the dependencies and their versioning.

A potential problem is redistribution of existing tools if a tool such as Docker [15] is chosen for distribution. The licensing may not allow the tools to be redistributed. This is uncommon as most of the bioinformatics tools are open source. In this case, even if we pick a tool distributed under an invasive license such as GPLv3, the software may be redistributed as long as the receiving party receives the software with the same unchanged license.

3.7 Development Process

3.7.1 Methodology

During the development of HoCoRT, Kanban was utilized as the development methodology. A Trello board was utilized with "To Do", "Doing", and "Done" columns. New ideas and requirements were continuously being placed in the "To Do" column. All the items in all the columns were continuously being reprioritized. During the development, items were being moved from the "To Do" column over to the "Doing" column when they were being actively worked on. Once the items/tasks were completed, they were moved to the "Done" column.

Kanban was chosen because of its efficiency as it has been suggested that using Kanban over Scrum reduces the number of bugs and increases productivity [60].

3.7.2 Tools

The programming language chosen was Python. Python was chosen because it is simple to use and, therefore, fast to develop and iterate with. While the Python interpreter is often slow, it is possible to write fast and efficient Python code by utilizing code libraries (often implemented in other languages), and utilizing efficient implementations of array-related operations [23]. Python does have some issues, especially in larger projects. Python does not have interfaces (such as in Java), and it is, therefore, harder to enforce stable interfaces to classes and other structures. This is because Python is an interpreted language where the code is being interpreted at runtime. Java, on the other hand, is a compiled statically-typed language where the code is checked during compilation (before running it). This makes it easier to enforce interfaces.

Vim was the editor of choice. This choice was made because HoCoRT was developed in the command line on a remote server, which made graphical editors hard to use. The version control system utilized was git, more specifically GitHub. Trello was used to implement a Kanban board. All the development was done on machines running the Linux operating system.

3.7.3 Version 0.1 to 0.2 refactor

The alignments/mappings were distinguished based on mapping quality in version 0.1 of HoCoRT. During some ad-hoc benchmarking experiments it was proved to be a bad way to differentiate alignments. In the move from version 0.1 to 0.2 of HoCoRT, a new system was implemented. Since version 0.2, SAM flags are used to detect un-/mapped reads. This improved the performance of many HoCoRT pipelines from practically unusable to good.

Another improvement was implemented in the move from version 0.1 to 0.2 of HoCoRT. In version 0.1, intermediary files were produced between different processes/tasks. Since version 0.2, there are no intermediary files and the stdout from one process is passed as input to the next. This improved the I/O access, storage use, and the speed of the pipelines. Still, the combination pipelines write to disk when many of them could utilize piping.

Chapter 4

Results

4.1 HoCoRT

A python package named HoCoRT (Host Contamination Removal Tool) was developed. It simplifies the process of removing specific sequences from sequencing reads. HoCoRT can be used for both host contamination removal, but also for removing or extracting any other sequences.

HoCoRT consists of a collection of pipelines. These pipelines were evaluated along with two ad-hoc ways of using Bowtie2 to remove host contamination. During evaluation testing, human host contamination was removed from simulated sequencing reads.

4.1.1 Software library

HoCoRT was written in Python, and can be imported and used in Python scripts and programs. It exposes different aligners to Python code, and may be used to create indexes and align using the different supported tools.

Architecture

The architecture consists of the following components:

- CLI interface
- Aligner and classifier classes
- Pipeline classes
- File parsing classes
- Command execution functions

First the CLI interface calls the selected pipeline. The pipelines then call aligner and classifier classes and the file parsing classes. Aligner and classifier classes and the file parsing classes build commands which, upon execution, utilize external tools to map reads and parse files. These commands are executed by passing them to the command execution functions in HoCoRT. One of the execution functions is able to pipe the output from one command to another to ensure low storage use and fewer I/O accesses to increase speed.

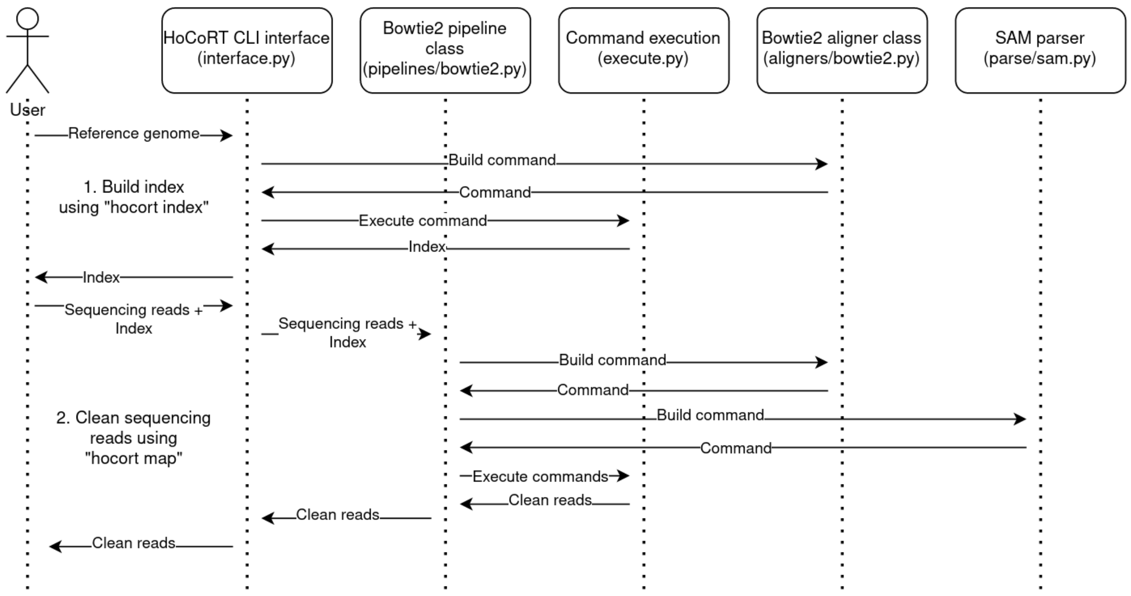
User
HoCoRT CLI interface (interface.py)
Bowtie2 pipeline class (pipelines/bowtie2.py)
Command execution (execute.py)
Bowtie2 aligner class (aligners/bowtie2.py)
SAM parser (parse/sam.py)
Reference genome
1. Build index using "hocort index"
Build command
Command
Execute command
Index
Index
Sequencing reads + Index
Sequencing reads + Index
Build command
Command
2. Clean sequencing reads using "hocort map"
Build command
Command
Execute commands
Clean reads
Clean reads
Clean reads

Figure 4.1: HoCoRT architecture using HoCoRT Bowtie2 [48] pipeline as example. First an index is built using the aligner and classifier classes. Second, during the cleaning of the reads, the pipeline builds commands using the aligner/classifier classes and SAM parser (samtools [50]). These commands are passed to execute.py where they are executed. The combination pipelines use multiple aligner/classifier classes to build the commands, and map reads.

4.1.2 Logging

Logging in HoCoRT is done using the default Python "logging" module. The execution function in execute.py catches the stdout and stderr output and logs it using the logging module in Python. The logging module is configured by configure_logger function in logging.py. It may be configured to log different levels of output messages depending on chosen verbosity. The output may be logged to a log file. There may not be any log output at all if HoCoRT is ran using the "–quiet" flag.

Four different logging levels are used; INFO, DEBUG, WARNING, and ERROR. If the "–quiet" flag is used, only WARNING and ERROR messages are shown.

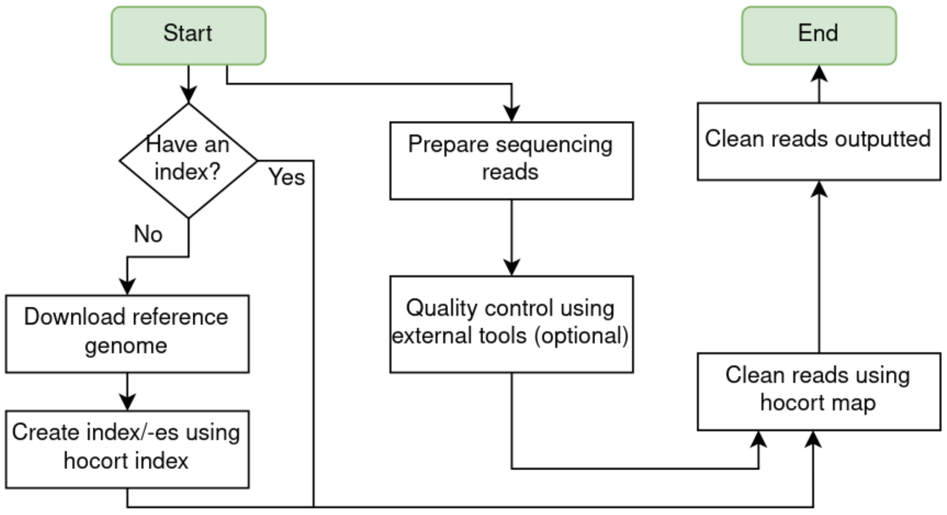
Start
Have an index?
Yes
No
Download reference genome
Create index/-es using hocort index
Prepare sequencing reads
Quality control using external tools (optional)
End
Clean reads outputted
Clean reads using hocort map

Figure 4.2: HoCoRT flow diagram. A simple flow through HoCoRT is shown. A reference genome is downloaded and an index is built from it. Meanwhile sequencing reads are prepared and processed. The reads are then mapped to the index, and the clean reads are outputted.

4.1.3 Dependencies

External programs (may be installed using Conda):

- Bowtie2 [48] (Tested with version 2.4.5)
- HISAT2 [43] (Tested with version 2.2.1)
- Kraken2 [69] (Tested with version 2.1.2)
- BWA-MEM2 [68] (Tested with version 2.2.1)
- BBMap [20] (Tested with version 38.96)
- Minimap2 [52] (Tested with version 2.24)
- samtools [50] (Tested with version 1.15)

Some problems regarding dependency versioning were encountered during the development of HoCoRT.

One problem was with the tbb [3] (Intel Threading Building Blocks) package. The tbb package is a parallel computing API developed by Intel. Bioconda [66] repository hosts packages, such as Bowtie2 [48], which are all compiled for tbb version 2020 (as of March 19 2022). Bioconda [66] does not host the tbb package, it is hosted on conda forge instead. Newer versions of tbb are available on conda forge, such as version 2021, and conda forge does not have any global pinning restricting access to newer versions. This means that Bowtie2 [48] compiled for tbb version 2020, and tbb version 2021 may be installed at the same time. When tbb version 2021 is installed, and the Bowtie2 [48] package installed was compiled for tbb version 2020, Bowtie2 [48] simply crashes. This was fixed by pinning tbb to 2020.3 in the HoCoRT Bioconda [66] recipe.

There was also a problem where samtools [50] version 1.7 kept crashing. This was fixed by requiring samtools [50] version 1.8 or later in the HoCoRT Bioconda [66] recipe.

Another dependency versioning problem was with Kraken2 [69] versions. An outdated version of Kraken2 [69] was installed by Bioconda [66], version 2.0.7 beta to be specific (2.1.2 being latest on March 19 2022). The NCBI server URLs had changed between these versions, and Kraken2 [69] was no longer able to download the necessary files to build an index. This was resolved by forcing the installation of an up-to-date version of Kraken2 [69] (2.1.2).

4.1.4 Abstract classes

The Python default module/package called "abc" (Abstract Base Classes) was utilized in an attempt to give e.g. aligners a consistent interface. Since Python is an interpreted language, there is no compiler to enforce consistent interfaces (e.g. interfaces in Java). Simply writing consistent function/class declaration would have been good enough, but ABCs give just a little bit of extra enforcement from the programming language itself. This helps the developers when adding support for additional tools because more mistakes are able to be caught by the Python interpreter.

Aligner abstract class

An aligner ABC was written to guide the development of aligner-exposing class implementations. The aligner ABC defines 3 methods, an index building method, a mapping method, and an index building CLI interface method. The index building and mapping methods are meant to build commands which are later executed with the command execution functions in execute.py. The index building interface is used by "hocort index" to provide a simple easy to use CLI interface to generate an index.

Classifier abstract class

A classifier ABC was written to guide the development of classifier-exposing class implementations. The classifier ABC defines the same methods as the aligner ABC as described above. The only difference is what the expected output is for the mapping method. The classifier tools often return classified/unclassified files with the actual reads instead of SAM or BAM alignment files (e.g. Kraken2 [69] returns FastQ files). The aligner tools, on the other hand, return SAM or BAM files with information about whether the reads mapped or not.

Pipeline abstract class

A pipeline ABC was written to guide the development of additional pipeline class implementations. The pipeline ABC implements some utility methods for logging. It also defines a run method to execute the pipeline, and an interface method which provides a CLI interface to use the pipeline. This way, the pipeline may be imported and used directly in Python scripts and programs, or it can be used from the HoCoRT command line interface.

4.1.5 Communication with existing tools

HoCoRT is a tool which utilizes existing tools to remove host contamination from sequencing reads. There was, therefore, a need to communicate with these tools. Different solutions to this problem were explored.

The focus was on Bash scripting and the Python programming language. Bash scripting is natural to go to when trying to utilize shell commands to build pipelines and other tools. An example of this is BBMap [20]. BBMap [20] utilizes Bash scripting for its command line interface where it calls the BBMap [20] Java program from the shell.

In the end, Python code proved to be more portable, extendable, maintainable, and more secure than Bash scripting. For example, the command line interface of BBMap [20], which is written in Bash, has an injection attack vulnerability. Assume there is a file named "test". The following command will delete the file:

```
bbmap.sh "test; rm test"
```

This is because the arguments are directly expanded and executed in the shell. This injection attack problem also applies to the Python function subprocess.Popen(shell=True).

If the SUID bit was enabled for the BBMap [20] binary or BBMap [20] otherwise got root privileges, the commands injected would also have root privileges. Therefore, it is possible to exploit this vulnerability to gain root privileges.

This does not happen when using HoCoRT as this was solved by using the Python function subprocess.Popen(shell=False). This method of the subprocess package injects the arguments directly into the process and does not execute them in the shell. In addition, HoCoRT validates all input arguments using a positive security model. The positive security model only allows valid characters in the arguments. This way, HoCoRT is even more secure than the underlying tools.

Validating arguments this way has it's problems as different operating and file systems have different rules for what is considered a valid path. "test; rm test" is a valid filename on most Unix systems, but HoCoRT would not consider it as valid. Ideally, such injection attack problems should be solved by the underlying tools such as BBMap [20] by not allowing the execution of the arguments. Validating the input is the best solution as there is always a possibility of the underlying tools containing such attack vectors.

The current solution utilizing Python is portable and allows future Windows support. Supporting Windows would be hard or impossible if HoCoRT was implemented in the form of Bash scripts. While pure Python code is slower than many other languages, it is plenty fast for a CLI tool, such as HoCoRT, which utilizes existing tools written in performant languages such as C and C++.

Aside from the shell injection attacks, another potential problem is trojan attacks. A trojan program is a malicious program disguised as a legitimate program. In the case of HoCoRT, a trojan attack can happen by installing a program which is named e.g. "bowtie2", but is not actually a legitimate Bowtie2 [48] copy. HoCoRT would call this program when running the Bowtie2 [48] pipeline. This would happen because HoCoRT does not specify the specific location of the programs it calls. A solution could be to specify the whole path of the program e.g.:

```
/home/user/miniconda3/envs/hocort/bin/bowtie2
```

This is hard to implement while remaining portable and simple to use because HoCoRT does not require the user to install the dependencies in a specific directory or way e.g. via conda. Therefore, considering that HoCoRT should never be run with root privileges, it was chosen to keep it simple, regardless of this potential vulnerability. The implemented simple solution is to check whether HoCoRT is ran with root privileges, and exit if it is. This way an attacker can not gain root access via HoCoRT, and HoCoRT remains portable and simple. The downside is that it is still possible for the attacker to e.g. delete the home directory. It is worth noting that HoCoRT does not make exploiting trojan attacks easier, as using Bowtie2 [48] by itself allows the same exploits if it is compromised.

4.1.6 HoCoRT pipelines

The HoCoRT pipelines were named after the tools they utilize. For example, the Bowtie2 [48] pipeline uses Bowtie2 [48] to map the reads, and Kraken2Bowtie2 first maps using Kraken2 [69] then maps using Bowtie2 [48]. Combining different mappers in a single pipeline may give advantages such as better accuracy or precision, or faster runtime.

It was found that Kraken2 [69] is really fast at classifying the sequencing reads with approximate accuracy. This inspired the creation of the

Kraken2Bowtie2 pipeline. The reads are first approximately classified using Kraken2 [69], then the outputted reads are mapped using Bowtie2 [48]. This makes this pipeline extremely fast when the percentage of host reads is high. The downside is that it is less precise when compared to some of the other pipelines.

Minimap2 [52] can map Nanopore reads as well as Illumina reads.

The following pipelines are available in HoCoRT version 1.0.0:

- HoCoRT BBMap
  First maps with BBMap [20] with default parameters, then uses "samtools fastq" to output FastQ. Offers a preset for Illumina reads, and another preset for Nanopore reads. Nanopore preset splits the reads into shorter ones before mapping.

- HoCoRT Bowtie2
  First maps with Bowtie2 [48], then uses "samtools fastq" to output FastQ. Offers a preset to run Bowtie2 [48] in end-to-end mode, and another preset to run Bowtie2 [48] in local mode.

- HoCoRT Bowtie2Bowtie2
  Utilizes the other HoCoRT pipelines to combine different tools. First maps with Bowtie2 [48] in end-to-end mode, then with Bowtie2 [48] in local mode. The output from the first run is passed as input to the second run.

- HoCoRT Bowtie2HISAT2
  Utilizes the other HoCoRT pipelines to combine different tools. First maps with Bowtie2 [48] in end-to-end mode, then with HISAT2 [43] with default parameters. The output from the first run is passed as input to the second run.

- HoCoRT BWA_MEM2
  First maps with BWA-MEM2 [68] with default parameters, then uses "samtools fastq" to output FastQ.

- HoCoRT HISAT2
  First maps with HISAT2 [43] with default parameters, then uses "samtools fastq" to output FastQ.

- HoCoRT Kraken2
  Maps with Kraken2 [69] with default parameters.

- HoCoRT Kraken2Bowtie2
  Utilizes the other HoCoRT pipelines to combine different tools. First maps with Kraken2 [69] with default parameters, then with Bowtie2 [48] in end-to-end mode. The output from the first run is

passed as input to the second run.

- HoCoRT Kraken2HISAT2
  Utilizes the other HoCoRT pipelines to combine different tools. First maps with Kraken2 [69] with default parameters, then with HISAT2 [43] with default parameters. The output from the first run is passed as input to the second run.

- HoCoRT Kraken2Minimap2
  Utilizes the other HoCoRT pipelines to combine different tools. First maps with Kraken2 [69] with default parameters, then with Minimap2 [52] with default parameters. The output from the first run is passed as input to the second run.

- HoCoRT Minimap2
  First maps with Minimap2 [52] with default parameters, then uses "samtools fastq" to output FastQ. Offers a preset for Illumina reads, and another preset for Nanopore reads.

4.1.7 HoCoRT pipeline compressed input/output support

Table 4.1: HoCoRT Pipeline compressed input/output support

Pipeline	.gz input	.gz output
Bowtie2	yes	yes
BWA-MEM2	yes	yes
HISAT2	yes	yes
BBMap	yes	yes
Kraken2	yes	no
Minimap2	yes	yes
Kraken2Bowtie2	yes	yes
Kraken2Minimap2	yes	yes
Kraken2HISAT2	yes	yes

Most of the HoCoRT pipelines utilize samtools [50] as the last step to convert from SAM/BAM to FastQ, with the exception being the Kraken2 [69] pipeline. Because samtools is able to compress datasets just by adding ".gz" to the extension, almost all pipelines support gzip compression. This reduces the size of the datasets to about a third of the uncompressed file size. Kraken2 [69] outputs FastQ directly instead of outputting an alignment file, and compressed output using samtools [50] is therefore unsupported.

4.1.8 Command line interfaces

A user friendly command line interface was developed for this project. This interface consists of two subinterfaces. These are the "index" and "map" subinterfaces. They expose the underlying system and provide a simple interface to the user.

Command line host contamination removal interface (hocort map)

The first subinterface is the host contamination removal interface. It is implemented as a subinterface as a part of the main hocort command line interface. This tool exposes the pipelines to the user and makes removal of host contamination a rather simple process. It may be ran by executing the following command in the terminal:

```
hocort map
```

Command line index generation interface (hocort index)

The different aligners/mappers require an index to be generated prior to using them. This presents an issue as the interfaces to generate the indices are different for the different tools. This makes the process difficult for many users. To solve this issue, a subinterface "hocort index" was developed and implemented as part of the main hocort command line interface. It can be ran by executing the following command in a terminal:

```
hocort index
```

This tool provides a, mostly, consistent interface for generating an index. Two different potential solutions were evaluated.

1. Simple solution
   This proposal was to implement a simple solution where a tool is developed where a mapping tool is chosen, and the exact input and output paths are given as arguments by the user. This way, the solution would be as simple as possible, and therefore easy to maintain and extend. This is the solution which was chosen.

2. Advanced solution
   The advanced proposal proposed implementing automatic index building and detection. For the combination pipelines, the indexes would be linked. All the indexes (and combinations of indexes) would have some way of identifying them, and the user would choose which one to use when mapping. For example, an index combination could be named "Human Host".

   The indexes would be built by choosing a HoCoRT pipeline, and providing an index name and reference genome. A hash of the reference genome would be linked to the index, and if one already exists, the user would be asked whether a second should be built. The available indices would be listed when running the help command of a HoCoRT pipeline.

4.1.9 Data quality control

Data quality control refers to the process of trimming adapters, trimming low quality nucleotides, filtering out unwanted sequences, and other sequencing read data quality control.

Input data quality control was considered to be added to HoCoRT, but in the end it was chosen to not do so. It would have been added as either a selection or flag directly in the HoCoRT pipelines, or it would have a separate tool in addition to "hocort map" and "hocort index" (e.g. "hocort qc").

Different datasets from different sources often require different preprocessing of the reads. Some measures such as removing low quality nucleotides/reads, are universal, but other measures may depend on the sequencing machine used. For example, the Nanopore MinION and Illumina HiSeq technologies are very different in terms of sequence length and error rate. These technologies may require different preprocessing.

Other arguments were modularity and cohesion. HoCoRT is a tool to map reads and remove host contamination, it's job is not necessarily to do data quality control. Regardless, the internal architecture of HoCoRT is highly modular with high cohesion and low coupling, so it is possible to add data quality control in the future if there is demand.

4.1.10 Distribution

It was chosen to distribute HoCoRT through Bioconda [66] and via manual installation. Bioconda [66] is a channel for the conda package manager, and it is commonly used to distribute bioinformatics tools. Bioconda [66] hosts over 3000 bioinformatics packages. All packages added to Bioconda also have a corresponding automatically created Docker [15] BioContainer [10].

Included with the source code of HoCoRT, are some manual installation helper scripts. An install.sh, an uninstall.sh, and the environment.yml files allow simple manual installation of HoCoRT. The install.sh script creates a conda environment, called "hocort", with all the necessary dependencies defined in environment.yml. It then installs HoCoRT from the GitHub repository using pip. The uninstall.sh script just deletes the "hocort" conda environment.

This solution allows the users to install either manually, through Bioconda [66], or install the BioContainer Docker [15] image.

Creating a Bioconda recipe

The creation of the HoCoRT Bioconda [66] recipe was a mostly simple process. By reading the existing recipes for different packages and reading the Bioconda [66] documentation, HoCoRT was published on Bioconda [66] in the span of a few days. There were problems encountered with dependency versioning. These problems were described in detail in section 4.1.3.

The tool bioconda-utils was used to test the HoCoRT package locally before submitting a pull request. When you submit a pull request (to add or update a recipe) to Bioconda [66], automatic tests are run to test the recipe/package.

There was an issue encountered where the automatic tests tested both the package installed locally, but also did a mulled build of the Docker [15] container. This build did not have access to the HoCoRT source test data and tests, and some necessary testing packages such as pytest. The testing could, therefore, not be completed. The solution was to just run "hocort -h" to check if hocort runs at all. This should be enough as HoCoRT tests were already ran during the Bioconda [66] local installation test.

4.2 Functional testing and evaluation testing

There is a distinction made between functional testing and evaluation testing. "Functional testing" will be used to describe the process of testing whether the software works, not how well it works. "Evaluation testing" will be used to describe the process of testing and evaluating how well the software works. Synthetic sequencing reads and real data was used to evaluate HoCoRT pipelines and the other solutions. A reproducible and automated process for generating the synthetic sequencing reads and gathering the evaluation data is presented.

4.2.1 Functional testing methodology

pytest

Pytest was used as the testing framework for testing HoCoRT. The tests are simple and can be run by navigating to where HoCoRT is located and executing the following command:

```
pytest tests
```

Test data

Test data refers to data which was used to test the code. Here the emphasis was on whether things are working, not on how well they are working. The test dataset was manually written and is small and simple. An important restricting factor were the GitHub file size limits. GitHub has a file size limit of 100MB which is limiting when considering that FastQ files are often large. To upload the project to GitHub, the limits have to adhered to, and the test dataset was, therefore, small (less than 20MB).

Another restricting factor is runtime. The tests would take a long time to run with large input files. The advantage larger test datasets would give is an evaluation of quality of the run (accuracy, precision, sensitivity). To limit the runtime of the tests, it was chosen to make the test data as simple and small as possible. Therefore, the focus with the test datasets is on whether things work or not.

4.2.2 Evaluation testing methodology

During the evaluation process, the performance of all HoCoRT pipelines was measured.

Evaluation datasets included both simulated and real datasets. Refer to section 4.3 for results of simulated datasets, and section 4.4 for results of real datasets.

All of the evaluation testing source code is available at https://github.com/ignasrum/hocort-eval.

Evaluation testing hardware

All of the evaluation tests were ran on the following hardware:

- CPU: AMD Ryzen 7 1700X 8 cores/16 threads @ 3.4 GHz
- RAM: Kingston FURY Beast DDR4 CL16 Dual Channel 64 GB @ 3200 MHz
- HDD: 1 Seagate IronWolf 4TB 3.5" NAS HDD 5900RPM

Generating indexes

The human reference genome GRCh38.p13 was downloaded from https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39/. The required indexes were built using "hocort index" from this reference genome.

Example of building index for Bowtie2 [48] using "hocort index":

```
hocort index bowtie2 -i grch38_p13.fasta -o bt2_idx/grch38_p13
```

Generation of synthetic evaluation datasets

Illumina (HiSeq and MiSeq) and Nanopore evaluation datasets were generated using InSilicoSeq [35] (Illumina) and NanoSim [70] (Nanopore).

Generated read lengths:

- HiSeq = 125 bp

- MiSeq = 300 bp
- Nanopore = 54 bp to 98320 bp with an average of 2159 bp (measured using readlength.sh from BBMap [20])

There were two types of datasets generated; gut microbiome and oral microbiome datasets.

7 datasets of each type (gut microbiome, and oral microbiome) were generated for each of the type of reads (HiSeq, MiSeq, Nanopore).

```
7 datasets x 2 dataset types x 3 types of reads = 42 datasets
```

Example dataset sizes:

Table 4.2: All datasets were generated using seed "123". All sizes are compressed using gzip. Nanopore datasets are unpaired, while HiSeq and MiSeq are paired.

Dataset	Read count	Size	% size
HiSeq gut microbiome R1	5 million	334M	100%
HiSeq gut microbiome R2	5 million	342M	102%
MiSeq gut microbiome R1	5 million	1.2G	359%
MiSeq gut microbiome R2	5 million	1.3G	389%
Nanopore gut microbiome	5 million	11G	3293%
HiSeq oral microbiome R1	5 million	344M	103%
HiSeq oral microbiome R2	5 million	352M	105%
MiSeq oral microbiome R1	5 million	1.2G	359%
MiSeq oral microbiome R2	5 million	1.3G	389%
Nanopore oral microbiome	5 million	11G	3293%

One part of the datasets were human reads, while the rest was "other" reads. Various bacterial, viral, and fungi genomes were selected pseudo-randomly to compose the "other" part of the dataset. These genomes were downloaded from NCBI and synthetic reads were generated from them. The synthetic reads were then mixed and shuffled to achieve the desired abundance and number of reads.

The abundance ratios chosen were:

- 1% human, 99% other for gut microbiome datasets
- 50% human, 50% other for oral microbiome datasets

Each one of the 7 datasets consisted of 5 million reads. The datasets were generated with seeds: 123, 124, 125, 126, 127, 128, 129.

The process was automated by building a reproducible Snakemake [45] pipeline. This pipeline downloads the required data, generates the datasets, and maps the reads with the different tools. The tools evaluated, by this pipeline, were the HoCoRT pipelines and two ad hoc ways of running Bowtie2 [48].

As mentioned at the beginning, this Snakemake [45] pipeline is available at https://github.com/ignasrum/hocort-eval. Several FastQ manipulation tools were written to aid the generation of the datasets. The tools are available at https://github.com/ignasrum/fastq-tools.

The Illumina HiSeq and MiSeq datasets were generated using the pre-built InSilicoSeq [35] HiSeq and MiSeq error models. The so called "read profile" used by NanoSim [70] was generated from the NCBI SRA dataset with accession id ERR3279199. This dataset consists of unpaired human Nanopore MinION sequencing reads, more specifically, the NA12878 sample and one more individual with ataxia-pancytopenia syndrome. The Nanopore basecaller chosen was guppy.

The generated Illumina reads were paired, whereas the generated Nanopore reads were unpaired.

Chimeric reads were not taken into consideration, and no data quality control or preprocessing was done on the datasets. This was done to ensure diversity of reads within the dataset to see how well the different tools map e.g. low quality sequences.

Conda environments and seeds were used to ensure reproducibility.

The first output file was compared to the first input file. The sequences were named depending on where they came from (either "human" or "other"). The number of sequences in the two files was counted and compared. This way, the number of removed and remaining human and other sequences was measured.

The following tools/pipelines were tested:

- Bowtie2_end-to-end (HoCoRT pipeline)
  Bowtie2 [48] in default "end-to-end" mode combined with "samtools fastq" to convert SAM to FastQ.

- Bowtie2_local (HoCoRT pipeline)
  Bowtie2 [48] in default "local" mode combined with "samtools fastq" to convert SAM to FastQ.

- Bowtie2_end_to_end_un_conc (not a HoCoRT pipeline)
  Bowtie2 [48] in default "end-to-end" mode with –un-conc for FastQ output. "–un-conc" writes pairs that don't align concordantly to a

specified path. The following command was used in this test:

```
bowtie2 -x index/prefix --end-to-end -p 16 -1 input_R1.fq
-2 input_R2.fq --un-conc-gz bowtie2_output > /dev/null
```

- Bowtie2_local_un_conc (not a HoCoRT pipeline)
  Bowtie2 [48] in default "local" mode with –un-conc for FastQ output. "–un-conc" writes pairs that don't align concordantly to a specified path. The following command was used in this test:

  ```
  bowtie2 -x index/prefix --local -p 16 -1 input_R1.fq
  -2 input_R2.fq --un-conc-gz bowtie2_output > /dev/null
  ```

- HISAT2 (HoCoRT pipeline)
  HISAT2 [43] with default configuration combined with "samtools fastq" to convert SAM to FastQ.

- Kraken2 (HoCoRT pipeline)
  Kraken2 [69] with default configuration.

- BBMap_default (HoCoRT pipeline)
  BBMap [20] with default configuration combined with "samtools fastq" to convert SAM to FastQ.

- BBMap_fast (HoCoRT pipeline)
  BBMap [20] with parameter "fast=t" combined with "samtools fastq" to convert SAM to FastQ.

- BWA_MEM2 (HoCoRT pipeline)
  BWA-MEM2 [68] with default configuration combined with "samtools fastq" to convert SAM to FastQ.

- Kraken2Bowtie2 (HoCoRT pipeline)
  Kraken2 [69] with default configuration followed by Bowtie2 [48] in default "end-to-end" mode combined with "samtools fastq" to convert SAM to FastQ.

- Kraken2HISAT2 (HoCoRT pipeline)
  Kraken2 [69] with default configuration followed by HISAT2 [43] with default configuration combined with "samtools fastq" to convert SAM to FastQ.

- Bowtie2Bowtie2 (HoCoRT pipeline)
  Bowtie2 [48] in default "end-to-end" mode followed by Bowtie2 [48] in default "local" mode combined with "samtools fastq" to convert SAM to FastQ.

- Bowtie2HISAT2 (HoCoRT pipeline)
  Bowtie2 [48] in default "end-to-end" mode followed by HISAT2 [43] with default configuration combined with "samtools fastq" to convert SAM to FastQ.

- Minimap2_illumina (HoCoRT pipeline)
  Minimap2 [52] with default "genomic short-read mapping" preset (sr) combined with "samtools fastq" to convert SAM to FastQ.

- Minimap2_nanopore (HoCoRT pipeline)
  Minimap2 [52] with default "Nanopore vs reference mapping" preset (map-ont) combined with "samtools fastq" to convert SAM to FastQ.

- Kraken2Minimap2_nanopore (HoCoRT pipeline)
  Kraken2 [69] with default configuration followed by Minimap2 [52] with default "Nanopore vs reference mapping" preset (map-ont) combined with "samtools fastq" to convert SAM to FastQ.

- Kraken2Minimap2_illumina (HoCoRT pipeline)
  Kraken2 [69] with default configuration followed by Minimap2 [52] with default "genomic short-read mapping" preset (sr) combined with "samtools fastq" to convert SAM to FastQ.

4.3 Evaluation results for simulated datasets

4.3.1 Simulated gut microbiome

As shown in figure 4.3 and tables 4.3, 4.4, and 4.5, Kraken2 was the fastest with HiSeq, MiSeq, and Nanopore reads. BWA-MEM2 [68] and BBMap [20] had the slowest runtimes in the HiSeq and MiSeq tests. BWA-MEM2 [68] did not scale well going from HiSeq to MiSeq reads as the runtime was about 9 times slower (2760 sec/315 sec=8.76), while the total MiSeq dataset size was three to four times as large as HiSeq as shown in table 4.2.

Minimap2 [52] scaled well with larger amounts of data as it ran twice as slow going from HiSeq to MiSeq datasets (figure 4.3 and tables 4.3, and 4.4), while the datasets were three to four times larger as shown in table 4.2. Minimap2 [52] also ran as fast as Kraken2 [69] in the Nanopore test (figure 4.3 and tables 4.5). It is worth remembering that the simulated gut microbiome datasets contained low percentage of host contamination (1%). Minimap2 [52] did not scale as well in the Nanopore test with a larger percentage of host contamination (50%). This is shown in table 4.17.

Combination pipelines (such as Bowtie2HISAT2 and Kraken2Bowtie2) were faster than combining the runtimes of the underlying tools when ran by themselves (figure 4.3 and tables 4.3, and 4.4). This is because in the combination pipelines, the second run takes the output from the first run as input, where most reads are already removed. It should be noted that the combination pipelines in HoCoRT version 1.0.0, which was used in this testing, did not utilize piping and instead wrote and read to and from the disk. The runtimes of the combination pipelines could be improved further.

As defined in the definitions chapter, accuracy is the ratio between the correctly classified instances to the total number of instances. The accuracy was very similar for all of the tools. As shown in figure 4.4 and tables 4.6, 4.7 and 4.8, all of the tools had an accuracy of approximately 0.99 except BWA-MEM2 [68] which had an accuracy of 0.97 in the HiSeq test and 0.91 in the MiSeq test.

As defined in the definitions chapter, precision is the ratio between the retrieved relevant instances to the total number of retrieved instances. All tools had a precision of 0.76 or higher (figure 4.5 and tables 4.9, 4.10, and 4.11). There was significant variance between tools. In the MiSeq test (table 4.10), HISAT2 [43] had a precision of 0.9227 while Minimap2_illumina had a precision of 0.7699. BWA-MEM2 [68] had consistent low precision of 0.26 with HiSeq reads (table 4.9), and 0.10 with MiSeq reads (table 4.10). As shown in tables 4.12 and 4.13, BWA-MEM2 [68] had a sensitivity of 1.0, this means that BWA-MEM2's [68] output contained around 80% false positives. It is not clear what caused this.

As defined in the definitions chapter, sensitivity is the ratio between the retrieved relevant instances to the total number of all relevant instances. In this case, sensitivity tells us how much host contamination/human reads were removed. As shown in figure 4.6 and tables 4.12, 4.13, and 4.14, most tools had a sensitivity of 1.0 or close to it. The exception was Bowtie2_end_to_end_un_conc which had a sensitivity of 0.36 with HiSeq reads (table 4.12), and 0.04 with MiSeq reads (table 4.13). Bowtie2_local_un_conc had similar results with 0.46 in the HiSeq test (table 4.12), and 0.22 in the MiSeq test (table 4.13). This tells us that Bowtie2_end_to_end_un_conc and Bowtie2_local_un_conc removed few human reads. It is worth noting that there were many sequences which could not be identified by any tools in the Nanopore datasets. This limited the maximum possible sensitivity, which was 0.95 in the Nanopore test (table 4.14).

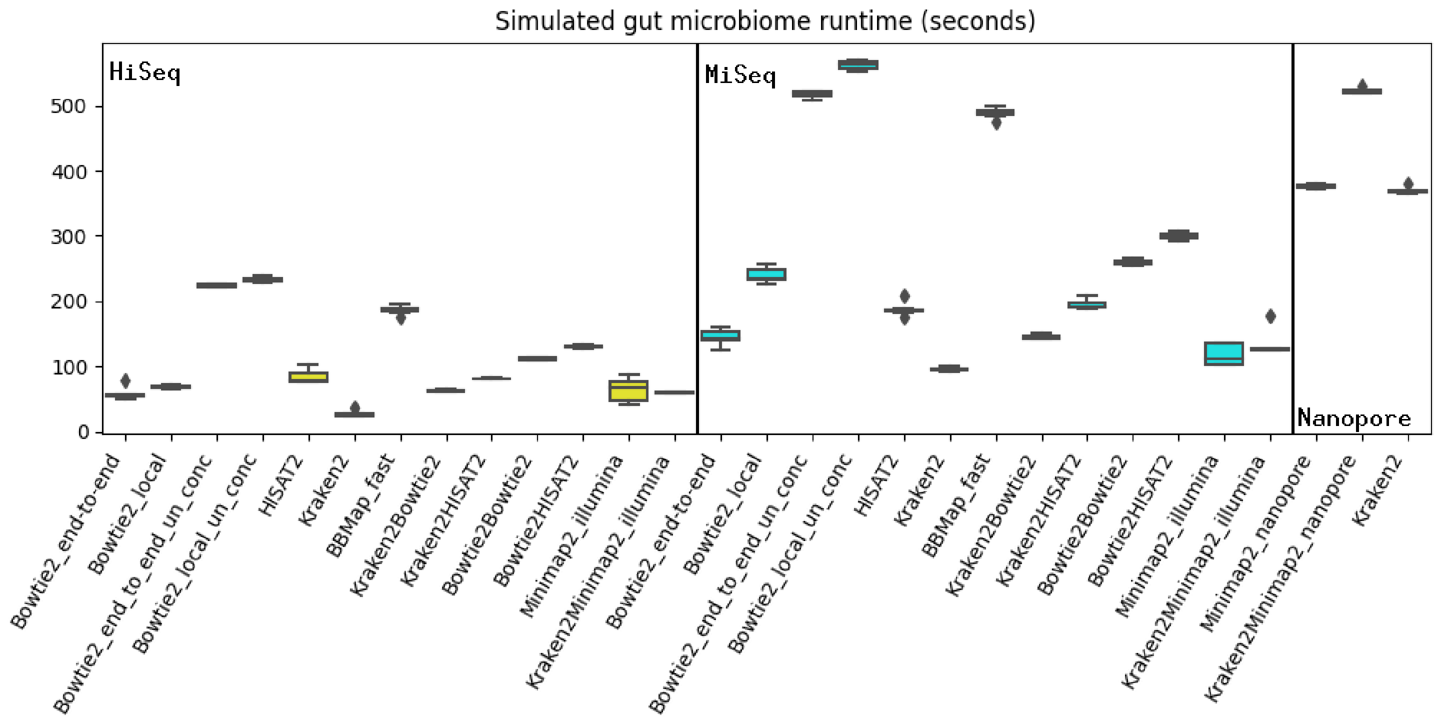

Figure 4.3: Simulated gut microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.3, 4.4, and 4.5 for complete results.

Table 4.3: Simulated HiSeq gut microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were paired-end. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	57.6	8.9	54.8	49.9	78.8
Bowtie2_local	68.5	1.8	67.8	66.0	71.0
Bowtie2_end_to_end_un_conc	223.6	2.3	224.3	220.6	226.8
Bowtie2_local_un_conc	232.4	3.2	231.4	227.5	238.3
HISAT2	84.1	10.7	78.5	75.8	103.3
Kraken2	26.9	4.1	25.2	23.1	36.0
BBMap_default	1053.2	11.8	1051.4	1038.7	1077.4
BBMap_fast	186.3	5.5	188.0	175.5	194.6
BWA_MEM2	315.8	33.0	312.6	267.3	366.2
Kraken2Bowtie2	62.1	1.3	62.0	60.6	64.1
Kraken2HISAT2	81.1	0.6	81.2	80.0	82.0
Bowtie2Bowtie2	111.0	2.1	111.1	108.5	114.3
Bowtie2HISAT2	130.0	2.0	129.3	127.4	133.3
Minimap2_illumina	62.6	16.7	67.4	39.7	86.1
Kraken2Minimap2_illumina	59.4	0.5	59.4	58.7	60.5

Table 4.4: Simulated MiSeq gut microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were paired-end. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	145.1	11.3	142.5	125.1	160.5
Bowtie2_local	239.7	10.8	234.5	225.5	257.2
Bowtie2_end_to_end_un_conc	517.0	4.7	519.2	508.2	521.8
Bowtie2_local_un_conc	562.3	6.1	563.4	553.2	570.2
HISAT2	187.1	9.8	185.3	174.4	209.0
Kraken2	95.0	2.8	95.5	90.6	99.5
BBMap_default	2338.7	226.7	2300.2	2056.0	2749.4
BBMap_fast	488.3	7.2	488.8	474.4	499.1
BWA_MEM2	2760.0	448.4	3023.1	1795.2	3049.8
Kraken2Bowtie2	144.9	3.0	143.3	142.1	150.4
Kraken2HISAT2	195.2	7.8	190.6	189.3	209.2
Bowtie2Bowtie2	259.0	3.8	257.7	255.2	266.8
Bowtie2HISAT2	299.9	4.8	299.7	292.7	308.3
Minimap2_illumina	118.2	15.4	110.4	102.5	136.1
Kraken2Minimap2_illumina	133.0	17.9	126.1	123.8	176.8

Table 4.5: Simulated Nanopore gut microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were unpaired. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	375.7	3.3	375.4	370.6	380.9
Kraken2Minimap2_nanopore	522.1	3.5	521.9	518.6	529.4
Kraken2	369.0	4.9	367.1	365.6	380.9

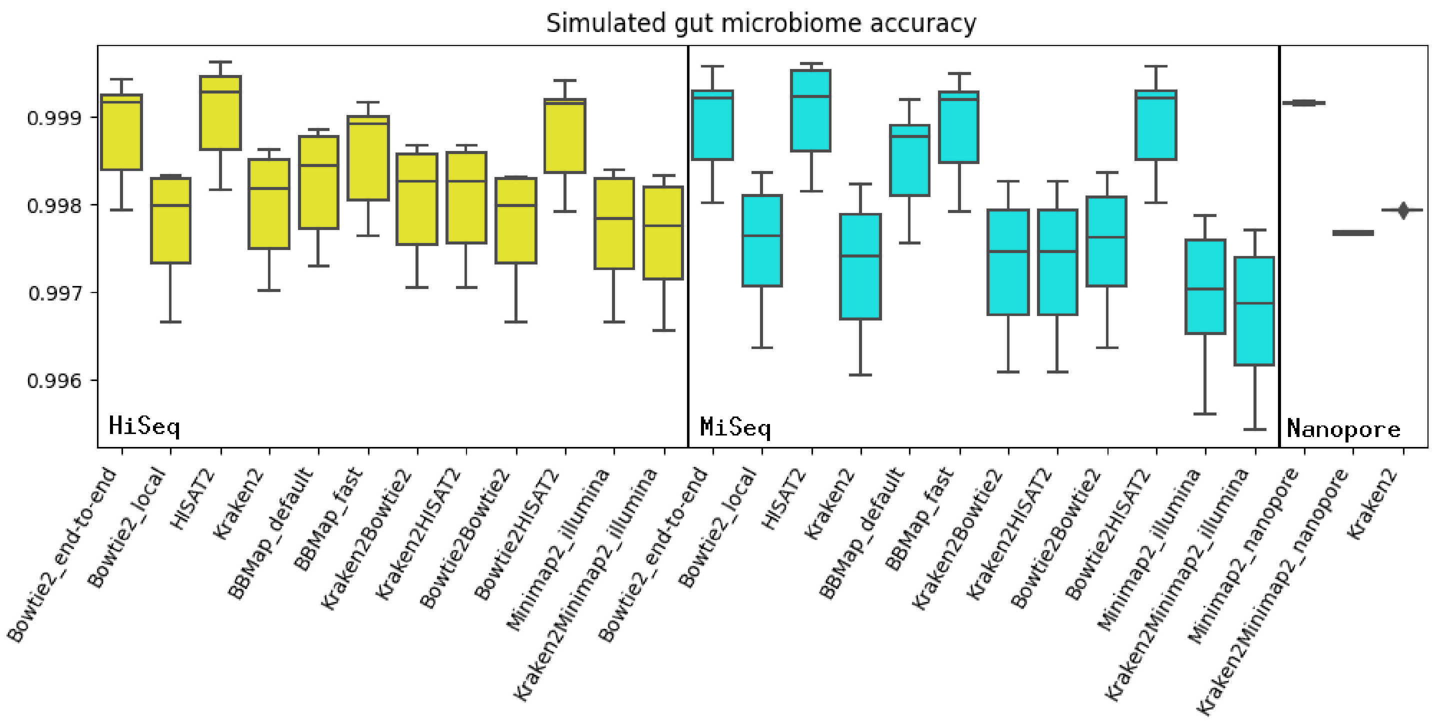

Figure 4.4: Simulated gut microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.6, 4.7, and 4.8 for complete results.

Table 4.6: Simulated HiSeq gut microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were paired-end. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9988	0.0005	0.9992	0.9979	0.9994
Bowtie2_local	0.9978	0.0006	0.9980	0.9967	0.9983
Bowtie2_end_to_end_un_conc	0.9934	0.0002	0.9935	0.9931	0.9935
Bowtie2_local_un_conc	0.9941	0.0002	0.9940	0.9938	0.9944
HISAT2	0.9990	0.0005	0.9993	0.9982	0.9996
Kraken2	0.9980	0.0006	0.9982	0.9970	0.9986
BBMap_default	0.9982	0.0006	0.9985	0.9973	0.9989
BBMap_fast	0.9986	0.0006	0.9989	0.9976	0.9992
BWA_MEM2	0.9720	0.0013	0.9724	0.9695	0.9736
Kraken2Bowtie2	0.9980	0.0006	0.9983	0.9970	0.9987
Kraken2HISAT2	0.9980	0.0006	0.9983	0.9971	0.9987
Bowtie2Bowtie2	0.9977	0.0006	0.9980	0.9967	0.9983
Bowtie2HISAT2	0.9988	0.0005	0.9992	0.9979	0.9994
Minimap2_illumina	0.9977	0.0007	0.9978	0.9967	0.9984
Kraken2Minimap2_illumina	0.9976	0.0007	0.9978	0.9966	0.9983

Table 4.7: Simulated MiSeq gut microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were paired-end. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9989	0.0005	0.9992	0.9980	0.9996
Bowtie2_local	0.9975	0.0007	0.9976	0.9964	0.9984
Bowtie2_end_to_end_un_conc	0.9904	0.0000	0.9904	0.9904	0.9905
Bowtie2_local_un_conc	0.9919	0.0002	0.9919	0.9917	0.9923
HISAT2	0.9990	0.0005	0.9992	0.9982	0.9996
Kraken2	0.9973	0.0008	0.9974	0.9961	0.9982
BBMap_default	0.9985	0.0006	0.9988	0.9976	0.9992
BBMap_fast	0.9989	0.0005	0.9992	0.9979	0.9995
BWA_MEM2	0.9128	0.0055	0.9135	0.9044	0.9208
Kraken2Bowtie2	0.9973	0.0008	0.9975	0.9961	0.9983
Kraken2HISAT2	0.9973	0.0008	0.9975	0.9961	0.9983
Bowtie2Bowtie2	0.9975	0.0007	0.9976	0.9964	0.9984
Bowtie2HISAT2	0.9989	0.0005	0.9992	0.9980	0.9996
Minimap2_illumina	0.9970	0.0008	0.9970	0.9956	0.9979
Kraken2Minimap2_illumina	0.9967	0.0008	0.9969	0.9954	0.9977

Table 4.8: Simulated Nanopore gut microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using NanoSim. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). The datasets were unpaired. Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9992	0.0000	0.9992	0.9991	0.9992
Kraken2Minimap2_nanopore	0.9977	0.0000	0.9977	0.9977	0.9977
Kraken2	0.9979	0.0000	0.9979	0.9979	0.9980

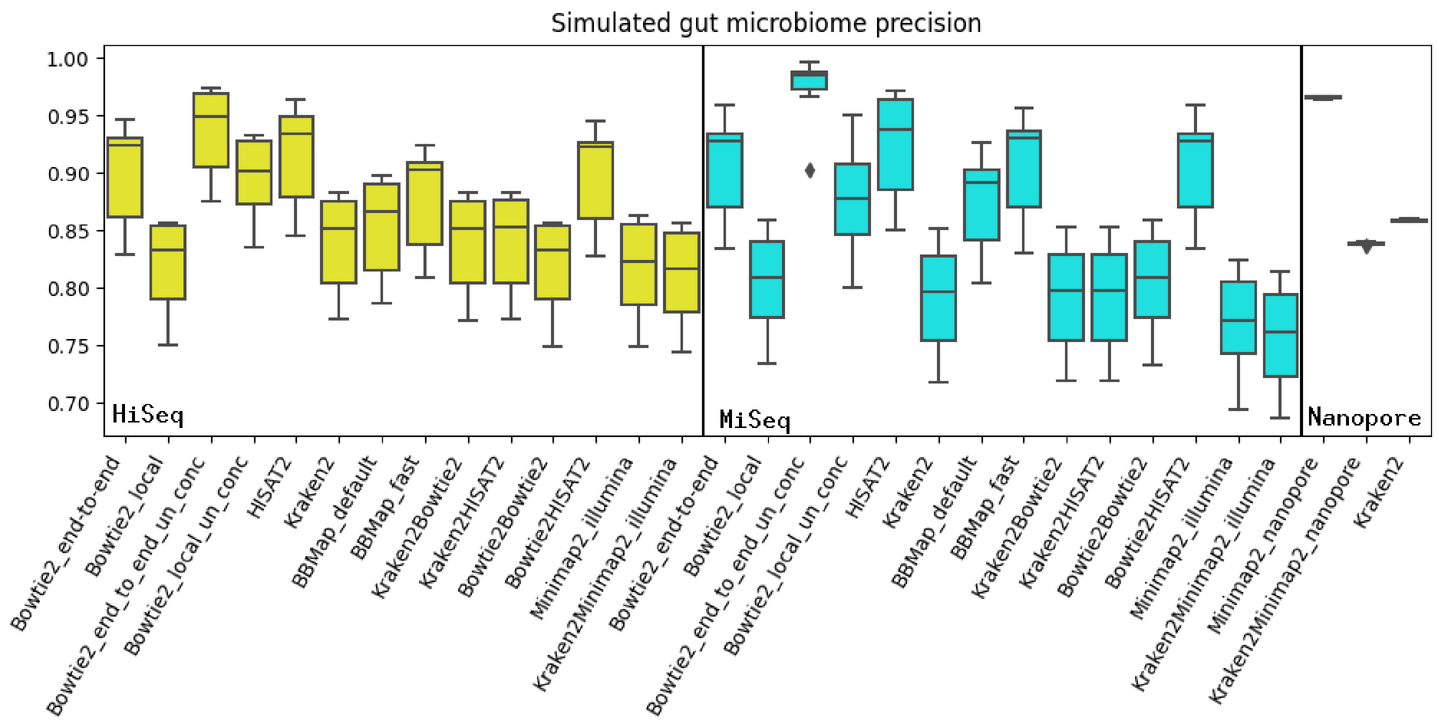

Figure 4.5: Simulated gut microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.9, 4.10, and 4.11 for complete results.

Table 4.9: Simulated HiSeq gut microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.8977	0.0427	0.9241	0.8294	0.9465
Bowtie2_local	0.8184	0.0403	0.8326	0.7498	0.8566
Bowtie2_end_to_end_un_conc	0.9357	0.0368	0.9498	0.8758	0.9741
Bowtie2_local_un_conc	0.8959	0.0351	0.9017	0.8358	0.9330
HISAT2	0.9145	0.0427	0.9337	0.8449	0.9642
Kraken2	0.8385	0.0429	0.8517	0.7734	0.8834
BBMap_default	0.8520	0.0440	0.8663	0.7872	0.8977
BBMap_fast	0.8759	0.0441	0.9034	0.8093	0.9236
BWA_MEM2	0.2634	0.0088	0.2657	0.2468	0.2746
Kraken2Bowtie2	0.8383	0.0430	0.8520	0.7721	0.8832
Kraken2HISAT2	0.8385	0.0431	0.8524	0.7726	0.8834
Bowtie2Bowtie2	0.8182	0.0403	0.8324	0.7496	0.8565
Bowtie2HISAT2	0.8956	0.0423	0.9226	0.8278	0.9456
Minimap2_illumina	0.8168	0.0427	0.8230	0.7495	0.8624
Kraken2Minimap2_illumina	0.8104	0.0425	0.8167	0.7437	0.8568

Table 4.10: Simulated MiSeq gut microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9045	0.0427	0.9282	0.8344	0.9592
Bowtie2_local	0.8045	0.0442	0.8090	0.7335	0.8595
Bowtie2_end_to_end_un_conc	0.9725	0.0297	0.9856	0.9027	0.9969
Bowtie2_local_un_conc	0.8768	0.0489	0.8777	0.7999	0.9509
HISAT2	0.9227	0.0454	0.9374	0.8499	0.9716
Kraken2	0.7903	0.0465	0.7971	0.7182	0.8523
BBMap_default	0.8730	0.0416	0.8911	0.8048	0.9267
BBMap_fast	0.9046	0.0440	0.9303	0.8303	0.9569
BWA_MEM2	0.1032	0.0058	0.1036	0.0947	0.1121
Kraken2Bowtie2	0.7909	0.0464	0.7978	0.7189	0.8527
Kraken2HISAT2	0.7909	0.0464	0.7978	0.7188	0.8527
Bowtie2Bowtie2	0.8043	0.0441	0.8088	0.7335	0.8593
Bowtie2HISAT2	0.9045	0.0427	0.9282	0.8344	0.9593
Minimap2_illumina	0.7699	0.0447	0.7717	0.6945	0.8244
Kraken2Minimap2_illumina	0.7567	0.0444	0.7618	0.6866	0.8136

Table 4.11: Simulated Nanopore gut microbiome precision results. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9660	0.0008	0.9663	0.9644	0.9671
Kraken2Minimap2_nanopore	0.8386	0.0012	0.8389	0.8362	0.8403
Kraken2	0.8591	0.0009	0.8594	0.8576	0.8605

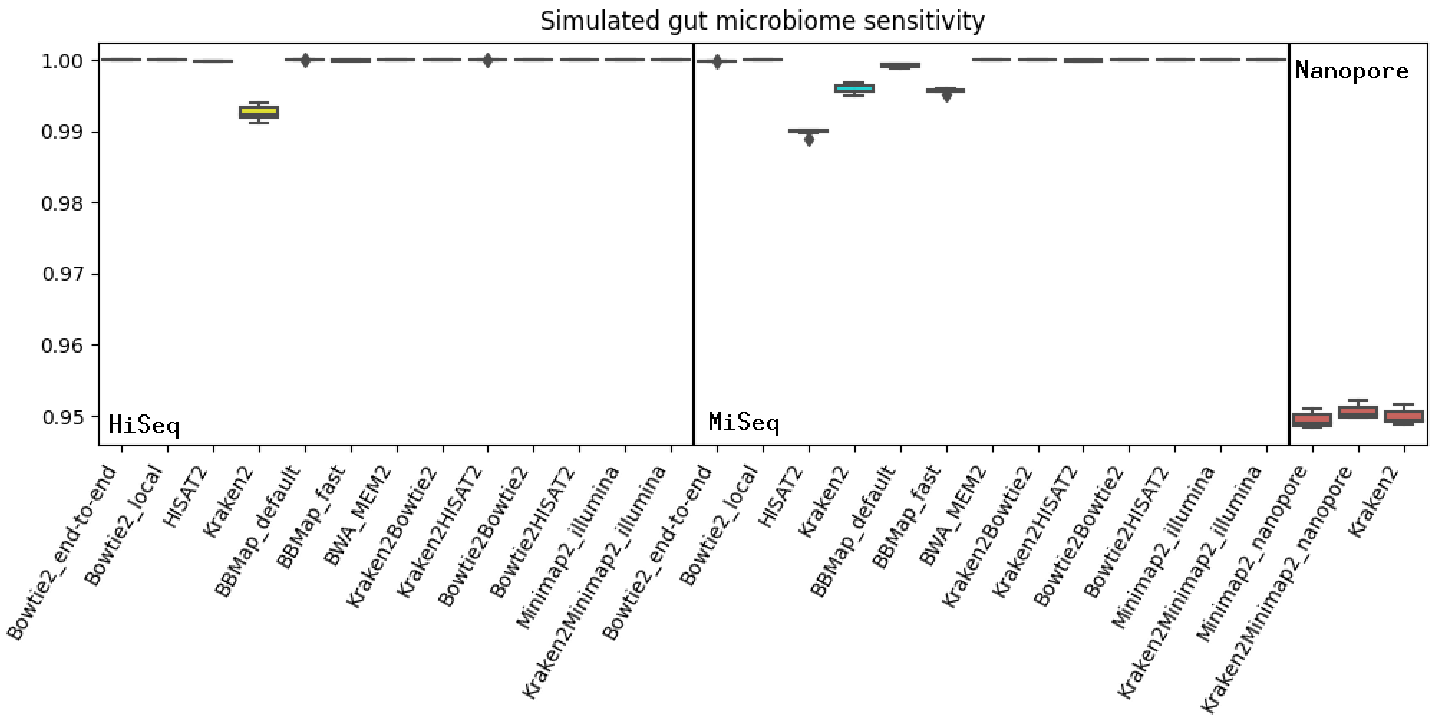

Figure 4.6: Simulated gut microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.12, 4.13, and 4.14 for complete results.

Table 4.12: Simulated HiSeq gut microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_local	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_end_to_end_un_conc	0.3633	0.0044	0.3615	0.3575	0.3689
Bowtie2_local_un_conc	0.4616	0.0078	0.4582	0.4539	0.4739
HISAT2	0.9998	0.0000	0.9998	0.9998	0.9999
Kraken2	0.9926	0.0010	0.9925	0.9912	0.9941
BBMap_default	1.0000	0.0000	1.0000	1.0000	1.0000
BBMap_fast	1.0000	0.0000	1.0000	0.9999	1.0000
BWA_MEM2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000

Table 4.13: Simulated MiSeq gut microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9999	0.0000	0.9999	0.9999	0.9999
Bowtie2_local	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_end_to_end_un_conc	0.0461	0.0041	0.0479	0.0375	0.0504
Bowtie2_local_un_conc	0.2245	0.0102	0.2211	0.2120	0.2400
HISAT2	0.9899	0.0004	0.9899	0.9890	0.9903
Kraken2	0.9959	0.0006	0.9957	0.9950	0.9968
BBMap_default	0.9993	0.0002	0.9994	0.9989	0.9995
BBMap_fast	0.9957	0.0002	0.9957	0.9953	0.9960
BWA_MEM2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2HISAT2	0.9999	0.0000	0.9999	0.9999	1.0000
Bowtie2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000

Table 4.14: Simulated Nanopore gut microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9495	0.0010	0.9490	0.9485	0.9511
Kraken2Minimap2_nanopore	0.9506	0.0010	0.9501	0.9497	0.9523
Kraken2	0.9499	0.0010	0.9493	0.9489	0.9516

4.3.2 Simulated oral microbiome

As shown in figure 4.7 and tables 4.15 and 4.16, BBMap [20] with default settings (BBMap_default) was the slowest tool in the HiSeq and MiSeq tests with the simulated oral microbiome datasets (50% host contamination). BBMap_default was approximately 10x slower with MiSeq reads than with HiSeq reads even though the MiSeq datasets were 3 to 4 times larger. It is worth noting that BBMap_fast was 10 to 30 times faster than BBMap_default (tables 4.15 and 4.16). As shown in figure 4.7 and tables 4.15, 4.16, and 4.17, Kraken2 [69] had the fastest runtime of all the tools measured in the HiSeq, MiSeq, and Nanopore tests.

The Kraken2 [69] + another tool combination pipelines had a fast runtime in these tests with high percentage of host contamination, which was not the case in the previous tests with simulated gut microbiome datasets with lower percentage of host contamination. This can be seen in figure 4.7 and tables 4.15, 4.16, and 4.14. In the HiSeq test, Kraken2Bowtie2 had a mean runtime of 46.8 seconds, while Bowtie2 [48] alone in end-to-end mode had a mean runtime of 471.5 seconds.

Even though BBMap_default was the slowest tool, it did not have the best accuracy. As shown in figure 4.8 and tables 4.18 and 4.19, BBMap_default's accuracy was similar to BBMap_fast's (BBMap [20] ran with the "–fast" flag) where both had an accuracy higher than 0.99.

Bowtie2_end_to_end_un_conc and Bowtie2_local_un_conc had low accuracy (0.73 or lower) in the HiSeq and MiSeq tests with simulated oral microbiome datasets (figure 4.8 and tables 4.18, and 4.19). This was not the case previously in the simulated gut microbiome tests because even though the tool had low sensitivity (tables 4.12 and 4.13) there were only 1% positive samples. In the current tests, these ways of mapping sequences had low sensitivity (0.46 or lower, see tables 4.24 and 4.25) and because there were 50% positive samples, it resulted in low accuracy. As shown in figure 4.8 and tables 4.18, 4.19, and 4.20, all the other tools had an accuracy higher than 0.97.

All of the tools had a precision higher than 0.99, with the exception being BWA-MEM2 [68] (figure 4.9 and tables 4.21, 4.25, and 4.23). BWA-MEM2 [68] had a precision of 0.97 in the HiSeq test, and 0.94 in the MiSeq test. This means that BWA-MEM2 [68] had more false positives than the other tools.

Kraken2 [69] did not have a sensitivity of 1.0 as most other tools did (figure 4.10 and tables 4.24, 4.25, and 4.26). This means that Kraken2 [69] did not remove all the human reads which most other tools did. As with the gut microbiome tests, Nanopore datasets contained many unidentifiable reads which limited the maximum sensitivity to 0.95 (table 4.26).

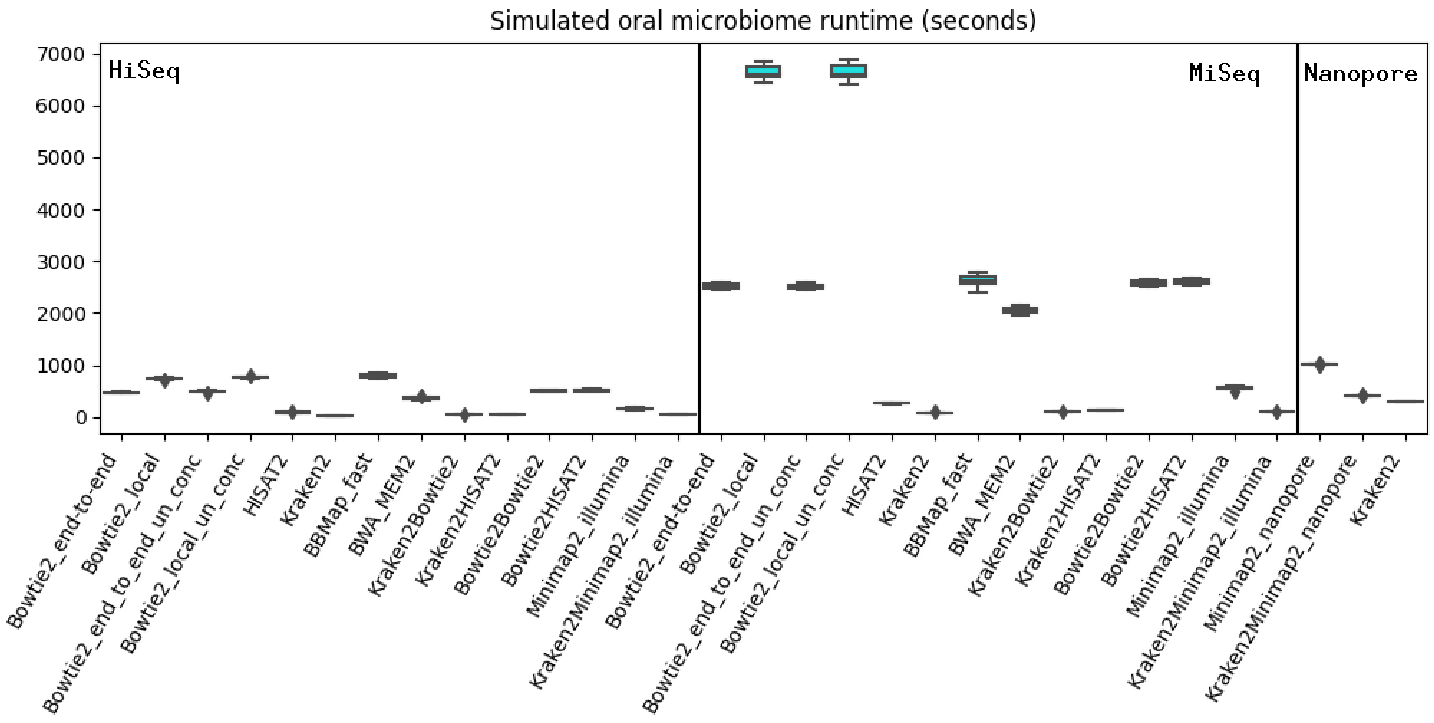

Figure 4.7: Simulated oral microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.15, 4.16, and 4.17 for complete results.

Table 4.15: Simulated HiSeq oral microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	471.5	11.1	473.8	455.2	491.0
Bowtie2_local	735.6	14.6	737.4	708.5	757.6
Bowtie2_end_to_end_un_conc	498.7	15.3	503.1	465.0	515.9
Bowtie2_local_un_conc	769.0	13.3	767.5	752.6	797.7
HISAT2	93.9	7.2	90.2	87.6	108.9
Kraken2	27.3	2.6	28.1	22.6	31.3
BBMap_default	9351.0	1133.5	9402.5	7979.4	11209.5
BBMap_fast	803.0	36.4	811.1	750.2	846.1
BWA_MEM2	363.7	18.3	361.8	341.1	399.2
Kraken2Bowtie2	46.8	6.5	44.7	42.9	62.7
Kraken2HISAT2	53.5	0.9	54.0	51.8	54.8
Bowtie2Bowtie2	503.0	15.6	494.2	485.6	533.0
Bowtie2HISAT2	514.2	15.8	519.1	495.8	540.3
Minimap2_illumina	165.5	13.6	169.9	146.5	188.3
Kraken2Minimap2_illumina	50.2	0.2	50.3	49.8	50.4

Table 4.16: Simulated MiSeq oral microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	2518.9	50.7	2522.0	2454.3	2601.1
Bowtie2_local	6633.8	139.3	6611.4	6445.6	6846.5
Bowtie2_end_to_end_un_conc	2514.8	45.1	2513.1	2461.0	2587.6
Bowtie2_local_un_conc	6641.6	157.6	6613.0	6403.5	6878.4
HISAT2	267.3	11.3	261.8	253.3	284.9
Kraken2	80.2	5.9	78.0	74.4	94.1
BBMap_default	84978.6	5991.4	84476.1	72676.3	91838.4
BBMap_fast	2617.7	126.2	2627.8	2409.2	2788.9
BWA_MEM2	2053.9	55.6	2063.6	1964.7	2139.8
Kraken2Bowtie2	101.9	2.3	101.3	99.5	107.2
Kraken2HISAT2	128.5	0.7	128.3	127.2	129.6
Bowtie2Bowtie2	2582.3	49.6	2577.9	2521.3	2655.9
Bowtie2HISAT2	2601.9	47.0	2605.5	2530.0	2668.1
Minimap2_illumina	554.4	32.5	548.4	489.6	601.8
Kraken2Minimap2_illumina	104.2	0.3	104.3	103.5	104.6

Table 4.17: Simulated Nanopore oral microbiome runtime results. Runtimes were measured in seconds. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	1015.0	10.1	1013.7	998.3	1034.1
Kraken2Minimap2_nanopore	409.4	8.8	408.4	397.2	428.5
Kraken2	303.9	1.7	303.5	301.9	307.2

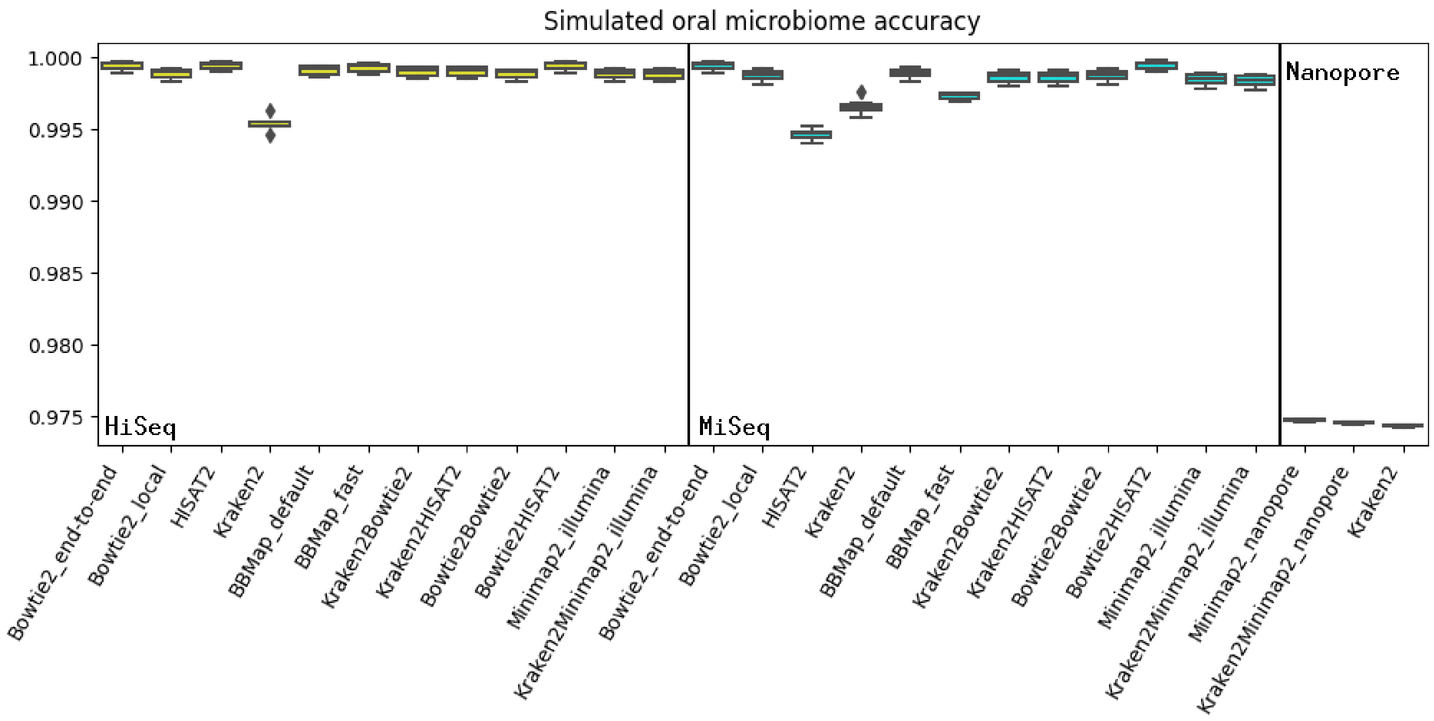

Figure 4.8: Simulated oral microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.18, 4.19, and 4.20 for complete results.

Table 4.18: Simulated HiSeq oral microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9994	0.0003	0.9996	0.9990	0.9997
Bowtie2_local	0.9989	0.0003	0.9990	0.9983	0.9992
Bowtie2_end_to_end_un_conc	0.6809	0.0021	0.6812	0.6777	0.6836
Bowtie2_local_un_conc	0.7300	0.0033	0.7295	0.7264	0.7351
HISAT2	0.9994	0.0003	0.9995	0.9990	0.9997
Kraken2	0.9954	0.0005	0.9952	0.9946	0.9964
BBMap_default	0.9991	0.0003	0.9992	0.9986	0.9994
BBMap_fast	0.9992	0.0003	0.9994	0.9988	0.9996
BWA_MEM2	0.9859	0.0007	0.9860	0.9846	0.9867
Kraken2Bowtie2	0.9990	0.0003	0.9991	0.9985	0.9993
Kraken2HISAT2	0.9990	0.0003	0.9991	0.9985	0.9993
Bowtie2Bowtie2	0.9989	0.0003	0.9990	0.9983	0.9992
Bowtie2HISAT2	0.9994	0.0003	0.9996	0.9989	0.9997
Minimap2_illumina	0.9988	0.0003	0.9989	0.9983	0.9992
Kraken2Minimap2_illumina	0.9988	0.0003	0.9989	0.9983	0.9992

Table 4.19: Simulated MiSeq oral microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9994	0.0003	0.9995	0.9989	0.9997
Bowtie2_local	0.9987	0.0004	0.9988	0.9981	0.9992
Bowtie2_end_to_end_un_conc	0.5230	0.0017	0.5233	0.5195	0.5249
Bowtie2_local_un_conc	0.6122	0.0046	0.6112	0.6063	0.6191
HISAT2	0.9946	0.0004	0.9947	0.9940	0.9952
Kraken2	0.9966	0.0005	0.9965	0.9958	0.9976
BBMap_default	0.9989	0.0003	0.9989	0.9983	0.9993
BBMap_fast	0.9973	0.0002	0.9974	0.9969	0.9975
BWA_MEM2	0.9726	0.0130	0.9753	0.9562	0.9958
Kraken2Bowtie2	0.9986	0.0004	0.9987	0.9980	0.9991
Kraken2HISAT2	0.9986	0.0004	0.9987	0.9980	0.9991
Bowtie2Bowtie2	0.9987	0.0004	0.9988	0.9982	0.9992
Bowtie2HISAT2	0.9995	0.0003	0.9996	0.9990	0.9998
Minimap2_illumina	0.9985	0.0004	0.9985	0.9978	0.9989
Kraken2Minimap2_illumina	0.9983	0.0004	0.9984	0.9977	0.9989

Table 4.20: Simulated Nanopore oral microbiome accuracy results. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9747	0.0001	0.9747	0.9746	0.9748
Kraken2Minimap2_nanopore	0.9746	0.0001	0.9746	0.9745	0.9746
Kraken2	0.9744	0.0001	0.9744	0.9743	0.9744

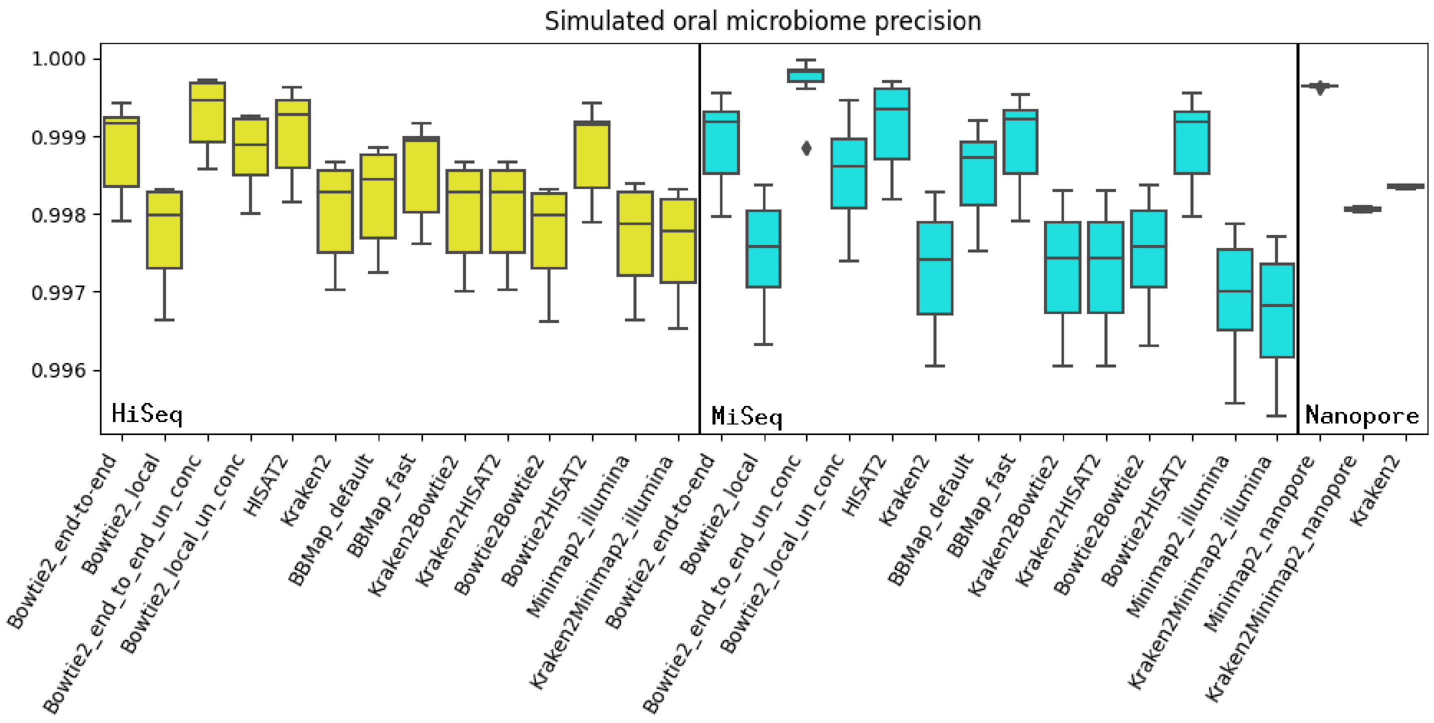

Figure 4.9: Simulated oral microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.21, 4.22, and 4.23 for complete results.

Table 4.21: Simulated HiSeq oral microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9988	0.0006	0.9992	0.9979	0.9994
Bowtie2_local	0.9977	0.0006	0.9980	0.9966	0.9983
Bowtie2_end_to_end_un_conc	0.9993	0.0004	0.9995	0.9986	0.9997
Bowtie2_local_un_conc	0.9988	0.0005	0.9989	0.9980	0.9993
HISAT2	0.9990	0.0005	0.9993	0.9981	0.9996
Kraken2	0.9980	0.0006	0.9983	0.9970	0.9987
BBMap_default	0.9982	0.0006	0.9985	0.9973	0.9988
BBMap_fast	0.9985	0.0006	0.9989	0.9976	0.9992
BWA_MEM2	0.9725	0.0012	0.9728	0.9701	0.9740
Kraken2Bowtie2	0.9980	0.0006	0.9983	0.9970	0.9987
Kraken2HISAT2	0.9980	0.0006	0.9983	0.9970	0.9987
Bowtie2Bowtie2	0.9977	0.0006	0.9980	0.9966	0.9983
Bowtie2HISAT2	0.9988	0.0005	0.9991	0.9979	0.9994
Minimap2_illumina	0.9977	0.0007	0.9979	0.9966	0.9984
Kraken2Minimap2_illumina	0.9976	0.0007	0.9978	0.9965	0.9983

Table 4.22: Simulated MiSeq oral microbiome precision results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9989	0.0005	0.9992	0.9980	0.9996
Bowtie2_local	0.9975	0.0007	0.9976	0.9963	0.9984
Bowtie2_end_to_end_un_conc	0.9997	0.0004	0.9998	0.9989	1.0000
Bowtie2_local_un_conc	0.9985	0.0007	0.9986	0.9974	0.9995
HISAT2	0.9991	0.0006	0.9994	0.9982	0.9997
Kraken2	0.9973	0.0008	0.9974	0.9960	0.9983
BBMap_default	0.9985	0.0006	0.9987	0.9975	0.9992
BBMap_fast	0.9989	0.0006	0.9992	0.9979	0.9995
BWA_MEM2	0.9487	0.0236	0.9529	0.9195	0.9916
Kraken2Bowtie2	0.9973	0.0008	0.9974	0.9961	0.9983
Kraken2HISAT2	0.9973	0.0008	0.9974	0.9961	0.9983
Bowtie2Bowtie2	0.9975	0.0007	0.9976	0.9963	0.9984
Bowtie2HISAT2	0.9989	0.0005	0.9992	0.9980	0.9996
Minimap2_illumina	0.9969	0.0008	0.9970	0.9956	0.9979
Kraken2Minimap2_illumina	0.9967	0.0008	0.9968	0.9954	0.9977

Table 4.23: Simulated Nanopore oral microbiome precision results. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9996	0.0000	0.9996	0.9996	0.9997
Kraken2Minimap2_nanopore	0.9981	0.0000	0.9981	0.9980	0.9981
Kraken2	0.9983	0.0000	0.9984	0.9983	0.9984

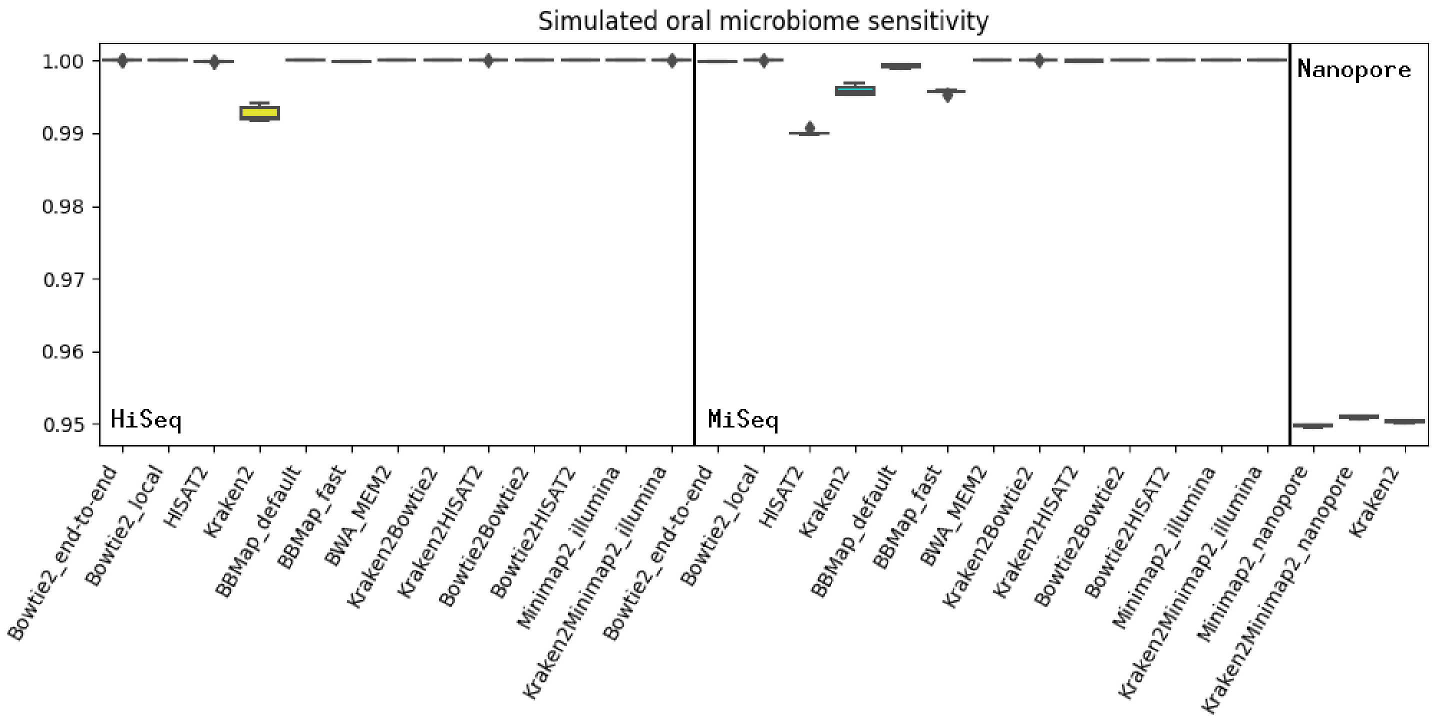

Figure 4.10: Simulated oral microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq and MiSeq pre-built error models) and NanoSim [70] (Nanopore). The HiSeq and MiSeq datasets were paired-end, while the Nanopore datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each boxplot represents 7 runs (1 run x 7 datasets). Yellow boxplots (1st partition from left) represent HiSeq. Cyan boxplots (2nd partition from left) represent MiSeq. Red boxplots (3rd partition from left) represent Nanopore. Outlying tools have been removed as they skew the chart, look at the tables 4.24, 4.25, and 4.26 for complete results.

Table 4.24: Simulated HiSeq oral microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (HiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_local	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_end_to_end_un_conc	0.3620	0.0042	0.3627	0.3554	0.3674
Bowtie2_local_un_conc	0.4606	0.0067	0.4593	0.4537	0.4710
HISAT2	0.9998	0.0000	0.9998	0.9998	0.9999
Kraken2	0.9927	0.0009	0.9922	0.9918	0.9940
BBMap_default	1.0000	0.0000	1.0000	1.0000	1.0000
BBMap_fast	0.9999	0.0000	0.9999	0.9999	0.9999
BWA_MEM2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000

Table 4.25: Simulated MiSeq oral microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using InSilicoSeq [35] (MiSeq pre-built error model). The datasets were paired-end. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Bowtie2_end-to-end	0.9999	0.0000	0.9999	0.9999	0.9999
Bowtie2_local	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2_end_to_end_un_conc	0.0460	0.0034	0.0466	0.0391	0.0499
Bowtie2_local_un_conc	0.2247	0.0093	0.2226	0.2129	0.2388
HISAT2	0.9901	0.0003	0.9900	0.9898	0.9909
Kraken2	0.9959	0.0007	0.9957	0.9953	0.9969
BBMap_default	0.9992	0.0002	0.9993	0.9988	0.9995
BBMap_fast	0.9957	0.0002	0.9958	0.9953	0.9958
BWA_MEM2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2HISAT2	0.9999	0.0000	0.9999	0.9999	1.0000
Bowtie2Bowtie2	1.0000	0.0000	1.0000	1.0000	1.0000
Bowtie2HISAT2	1.0000	0.0000	1.0000	1.0000	1.0000
Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000
Kraken2Minimap2_illumina	1.0000	0.0000	1.0000	1.0000	1.0000

Table 4.26: Simulated Nanopore oral microbiome sensitivity results. Datasets consisted of 5 million reads each, and were generated using NanoSim [70]. The datasets were unpaired. Each dataset contained 50% human reads, and 50% "other" reads (bacteria, viral, fungi). Each row represents 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. All numbers are rounded to 4 decimals.

Pipeline	mean	std	median	min	max
Minimap2_nanopore	0.9498	0.0001	0.9498	0.9496	0.9499
Kraken2Minimap2_nanopore	0.9510	0.0001	0.9510	0.9507	0.9511
Kraken2	0.9503	0.0001	0.9504	0.9501	0.9505

4.4 Human contamination in public datasets

Real Illumina and Nanopore datasets of human oral and gut microbiomes were used for evaluation of the tools.

BLAST [4] was used to confirm the outputted/mapped reads.

Versions of the software used:

- Bowtie2 [48] version 2.4.5
- Minimap2 [52] version 2.24
- HoCoRT version 1.0.0

Datasets (from NCBI SRA database) used:

- Illumina dataset
  NCBI accession code: SRR18498477
  Submitted to NCBI by: Radboud university medical center
  Study: Gut microbiome in viral-suppressed people living with HIV
  Instrument: Illumina HiSeq 2000
  Strategy: WGS
  Source: METAGENOMIC
  Selection: RANDOM
  Layout: PAIRED

- Nanopore dataset
  NCBI accession code: SRR9847864
  Submitted to NCBI by: Stanford University
  Study: Nanopore long read assembly of the human gut metagenome
  Instrument: MinION
  Strategy: WGS
  Source: METAGENOMIC
  Selection: RANDOM
  Layout: SINGLE

The sequencing reads were downloaded and extracted using sra-tools [64].

```
prefetch SRR18498477 && fastq-dump --split-files SRR18498477.sra
prefetch SRR9847864 --max-size 30GB && fastq-dump SRR9847864.sra
```

The sequences were aligned using the Bowtie2 [48] (in end-to-end mode) and Minimap2 [52] (in Nanopore mode) HoCoRT pipelines. The same indexes used in 6.4 were used here as well. The indexes were built using "hocort index" from the GRCh38 patch 13 human genome.

```
hocort map bowtie2 -f false -p end-to-end -x bt2_idx/GRCh38
-i SRR18498477_1.fastq SRR18498477_2.fastq
-o SRR18498477_1_bowtie2.fastq SRR18498477_2_bowtie2.fastq

hocort map minimap2 -f false -p nanopore -x mn2_idx/GRCh38
-i SRR9847864.fastq -o SRR9847864_minimap2.fastq
```

The mapped sequences were converted from FastQ to FASTA format using

VSEARCH. [58] The original sequences were also converted to FASTA. The paired-end files were concatenated together after conversion.

```
vsearch --fastq_filter SRR9847864_minimap2.fastq
--fastaout SRR9847864_minimap2.fasta

cat SRR18498477_1.fasta SRR18498477_2.fasta > SRR18498477.fasta
```

The BLAST [4] human genome index was used when finding alignments with blastn.

```
update_blastdb.pl --num_threads 12 --decompress human_genome
```

The FASTA files were mapped using blastn in megablast mode. Only the alignments with an expect value of 1e-10 or better were outputted. BLAST [4] alignment count was limited to 1 alignment per input sequence. The expect value 1e-10 was chosen to only output highly significant alignments.

```
blastn -num_threads 16
-db human_genome/GCF_000001405.39_top_level
-query SRR18498477_bowtie2.fasta -task megablast
-out report_illumina_human.out -outfmt 6 -evalue 1e-10
-subject_besthit -max_target_seqs 1
```

The number of alignments in the output files was counted using the command wc.

```
wc -l report.out
```

4.4.1 Results

The first table 4.27 presents the number of sequencing reads in the relevant files. The second table 4.28 presents the number of alignments BLAST [4] found when mapping both the original files, but also the Bowtie2 [48] and Minimap2 [52] output files to the human genome index.

The results show that BLAST [4] found more sequences which align to the human genome than both Bowtie2 [48] and Minimap2 [52] HoCoRT pipelines. There were also sequences in the Bowtie2 [48] and Minimap2 [52] output files which BLAST [4] did not align to the human genome.

To conclude, human sequences were found in public Illumina HiSeq and Nanopore datasets by both BLAST [4], and the HoCoRT Bowtie2 [48] and Minimap2 [52] pipelines.

Table 4.27: Total number of sequencing reads. The "Original" column represents the total number of reads in the original file/-s. The "Bowtie2 output" and "Minimap2 output" columns represent the number of reads in the HoCoRT [48] and Minimap2 [52] output file/-s. The percentage in the parentheses represents the percentage of the total in the original file/-s.

Dataset	Original	Bowtie2 output	Minimap2 output
SRR18498477 (Illumina)	46,273,568 (100%)	9,704 (0.0002%)	12,252 (0.0002%)
SRR9847864 (Nanopore)	9,741,166 (100%)	—	3,288 (0.0003%)

Table 4.28: Human alignments found by blastn in megablast mode with expect value equal to or lower than 1e-10. The "Original" column represents the total number of alignments found in the original file/-s. The "Bowtie2 output" and "Minimap2 output" columns represent the number of human alignments found by BLAST [4] in the HoCoRT Bowtie2 [48] and Minimap2 [52] output file/-s. The percentage in the parentheses represents the percentage of the total in the original file/-s.

Dataset	Original	Bowtie2 output	Minimap2 output
SRR18498477 (Illumina)	10,970 (100%)	9,583 (87%)	10,451 (95%)
SRR9847864 (Nanopore)	154 (100%)	—	99 (64%)

4.5 Comparison with DeconSeq

In all of the previous Illumina results, paired end sequencing reads were used. Deconseq [59] does not support paired end reads, therefore, this evaluation test was therefore carried out independently from the previous evaluation tests.

Two datasets were used; one Illumina HiSeq and one Illumina MiSeq unpaired simulated gut microbiome (1% human, 99% other reads) datasets. These datasets were generated with seed 123 using InSilicoSeq [35]. The datasets were generated using the pre-built HiSeq and MiSeq error models available in InSilicoSeq [35]. The dataset generation source code is available at https://github.com/ignasrum/hocort-eval. The HoCoRT Bowtie2 [48] pipeline was run in end-to-end mode.

Versions of the software used:

- Bowtie2 [48] version 2.4.5
- Deconseq [59] version 0.4.3
- InSilicoSeq version 1.5.4
- HoCoRT version 1.0.0

DeconSeq [59] is about 34 to 49 times slower than HoCoRT Bowtie2 [48] pipeline in end-to-end mode (tables 4.29 and 4.30). The accuracy of the two tools was mostly the same, where both achieved 0.99 accuracy (tables 4.29 and 4.30). DeconSeq [59] had lower precision than HoCoRT Bowtie2 [48] pipeline, with the difference being about 0.06 to 0.11 (tables 4.29 and 4.30). The sensitivity was almost equal, with DeconSeq [59] being slightly ahead, e.g. 0.99974 vs 1.00000 (table 4.30).

Table 4.29: HiSeq dataset consists of 5 million unpaired reads, and was generated using InSilicoSeq [35] using seed 123. The dataset contains 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 1 run (1 run x 1 dataset). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. Runtime was measured in seconds. All numbers are rounded.

Tool	runtime	accuracy	precision	sensitivity
Bowtie2_end-to-end	49.8	0.99947	0.94934	0.99992
Deconseq	1677.2	0.99869	0.88432	0.99994

Table 4.30: MiSeq dataset consists of 5 million unpaired reads, and was generated using InSilicoSeq [35] using seed 123. The dataset contains 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each row represents 1 run (1 run x 1 dataset). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. Runtime was measured in seconds. All numbers are rounded.

Tool	runtime	accuracy	precision	sensitivity
Bowtie2_end-to-end	88.7	0.99960	0.96194	0.99974
Deconseq	4384.2	0.99824	0.85028	1.00000

4.6 Optimization of parameters

There was an attempt to optimize the parameters of the aligners. It was done by manually tuning parameters, then aligning a dataset with human contamination to a human reference genome, and measuring the results. The human sequences were labelled human so it was possible to distinguish them.

The parameters used:

```
custom : --end-to-end --sensitive --score-min L,-0.4,-0.4
default : --end-to-end
```

As it can be seen in table 4.31, the custom parameters were not perfect. The runtime was faster, there were a few more false negatives, and fewer false

Table 4.31: HoCoRT Bowtie2 pipeline was tested in end-to-end mode with default and custom parameters. HiSeq simulated human oral microbiome paired-end datasets were used. Datasets consisted of 5 million paired reads, and were generated using InSilicoSeq [35] with the pre-built HiSeq error model. The datasets contained 1% human reads, and 99% "other" reads (bacteria, viral, fungi). Each cell represents the mean of 7 runs (1 run x 7 datasets). Cyan colored cells indicate the best result in a column, while red colored cells indicate the worst result. Runtime was measured in seconds.

Tool	runtime	FN	FP
Bowtie2_custom	441.1789	10.1428	2553.7142
Bowtie2_default	473.0875	2.4285	2965.1428

positives than with default parameters.

In this case, as the human sequences are considered extremely sensitive information, a compromise was made to keep the default parameters instead of trying to optimize.

A sophisticated attempt to optimize the parameters could be made by using optimization algorithms and/or machine learning. This was not attempted due to time constraints. Such an attempt could allow to both improve the runtime, and produce fewer false negatives and false positives.

Regardless, it might be impossible to optimize for all scenarios. Removal of different host contamination (e.g. human and fish contamination) in an efficient manner might prove to be difficult using the same parameters.

Chapter 5

Discussion

5.1 Key findings

A tool, HoCoRT, was developed to simplify the process of removing specific organisms from sequencing reads. It consists of pipelines which utilize existing sequence alignment tools. HoCoRT's performance was measured thoroughly.

In the BBMap [20] presentation, the author claims that BBMap [20] "has superior speed" to alternative tools such as Bowtie2 [20]. The opposite was found in the testing. It was found that Bowtie2 [48], one of the most popular sequencing read aligners, was also one of the most performant for aligning Illumina reads. Across all tests, Bowtie2 [48] was faster than BBMap [20] in the range from 4% (table 4.16) to around 3200% (table 4.16). The claim about BBMap's [20] performance was made in 2014. The improvements to Bowtie2 [48] since 2014 could explain the current findings. Bowtie2 [48] had good runtime, high accuracy, precision, and sensitivity compared to the other tested tools in both low and high percentage host contamination Illumina HiSeq and MiSeq tests (refer to section 4.3). Based on the testing in section 4.3, Bowtie2 [48] in end-to-end mode is recommended for use with Illumina reads.

A HoCoRT pipeline using Kraken2 [69] was implemented. Kraken2 [69] is a taxonomic classifier, not an aligner like Bowtie2 [48] is. It proved to be extremely fast and removed most of the human contamination, but it had much lower precision than Bowtie2 [48].

There was an idea to combine Kraken2 [69] with Bowtie2 [48], so the Kraken2Bowtie2 pipeline was implemented. This pipeline first runs Kraken2 [69], then Bowtie2 [48]. It proved to be not as useful with low percentage (1%) of host contamination as Bowtie2 [48] by itself had better performance (refer to tables 4.3, 4.6, 4.9, and 4.12). Where the

Kraken2Bowtie2 pipeline proved useful, was with high percentage (50%) of host contamination. With high percentage (50%) of host contamination it ran about 10 times faster than Bowtie2 [48] by itself, and gave similar results in terms of accuracy, precision and sensitivity (refer to tables 4.15, 4.18, 4.21, and 4.24).

HoCoRT was compared to competing solutions such as DeconSeq [59]. DeconSeq [59] uses BWA [49], and its architecture is not modular. The use of BWA [49] is hardcoded and DeconSeq [59] is not adaptable to use aligners other than BWA [49]. DeconSeq [59] also does not support paired end reads which is the standard for Illumina reads. The paired end read files have to be concatenated together before mapping. DeconSeq [59] had a bit better sensitivity than Bowtie2 [48] in end-to-end mode with HiSeq reads; 0.9999 vs 0.9996 (table 4.29), but runtime was almost 36 times slower (1677.1 / 46.7 = 35.9, table 4.29). Bowtie2 [48] also had higher accuracy and precision than DeconSeq [59]; accuracy was 0.99947 vs 0.99869 (table 4.29), precision was 0.94934 vs 0.88432 (table 4.29). The same was true for MiSeq reads, although the runtime difference was even larger; DeconSeq [59] was approximately 49 times slower (4384.2 / 88.7 = 49.42, table 4.30).

Regarding Nanopore reads, Bowtie2 [48] does not support them so a different aligner had to be used. Minimap2 [52] and Kraken2 [69] were used for host contamination removal from Nanopore reads. As shown in 4.6 and 4.10, Minimap2 [52] had a sensitivity of 1.0 in all Illumina tests (HiSeq and MiSeq), and a sensitivity close to the best in Nanopore tests (approximately 0.95). Keep in mind that some of the Nanopore reads were unidentifiable by any software which appears to have limited the maximum sensitivity to around 0.95. The downside to Minimap2 [52] was the lower precision with Illumina reads in the gut microbiome tests (0.77 to 0.82, figure 4.5 and 4.9). In conclusion, Minimap2 [52] is useful for mapping both Illumina and Nanopore sequencing reads.

HoCoRT was not only tested and benchmarked with simulated datasets, but also with real public datasets (section 4.4.1). BLAST [4] was used to confirm the findings. It was found that both Bowtie2 [48] and Minimap2 [52] HoCoRT pipelines were able to find human host contamination in both Illumina HiSeq and Nanopore real public datasets (section 4.4.1). Neither of the programs (BLAST [4], Bowtie2 [48], Minimap2 [52]) are perfect and there may be biases and faults in the results. There were sequences in the Bowtie2 [48] and Minimap2 [52] output files which BLAST [4] did not align. These sequences may be either false positives (actual microbial/other sequences which were aligned to the human genome by Bowtie2 [48] and Minimap2 [52]) or human sequences which BLAST [4] did not manage to identify, whereas Bowtie2 [48] and/or Minimap2 [52] did.

Bowtie2 [48] was also tested with the flag "–un-conc" (Bowtie2_end_to_end_un_conc and Bowtie2_local_un_conc). This flag outputs the paired reads which did not map concordantly to the index, to an output FastQ file. This way of mapping sequences has been used in scientific applications [41]. In the testing with both HiSeq and MiSeq sequencing reads, it was found that Bowtie2 [48] with the flag "–un-conc" had low sensitivity. This means that it mapped few human sequencing reads. The low sensitivity did not help it achieve much higher precision than the other tested tools (section 4.3). In contrast, the HoCoRT Bowtie2 [48] pipeline only requires one of the two reads to map to the index to classify both as mapped. This helped it achieve much higher sensitivity and accuracy, while having almost the same run-time and precision as Bowtie2 with the "–un-conc" flag.

5.2 Strengths

HoCoRT is easy to access and is free. It is open-source and is licensed under the MIT license. HoCoRT is also easy to install via Bioconda [66] or by using the provided installation helper scripts. A simple command installs HoCoRT:

```
  conda install -c bioconda hocort
```

It also provides simple and consistent command line interfaces for both index generation and sequence contamination removal.

HoCoRT is highly modular and extendable. This allows the support for new tools to be added quickly. This is important as the field of bioinformatics is advancing quickly, and new mappers and tools are being released continuously.

5.3 Limitations

One of the two obvious competing solutions is GenCof [30]. GenCof [30] offers a graphical user interface to make it easier to remove specific organisms from sequencing reads. HoCoRT is a command line only tool, and this is an obvious limitation because users must already be familiar with the command line. This limits who can use HoCoRT.

GenCof [30] also offers a way to do quality control of the input sequencing reads. HoCoRT does not offer any way to do this. This choice was made mostly because of time constraints during development. The architecture of HoCoRT remains modular and flexible, and sequencing read quality control may be added in the future. GenCof [30] may be used instead of HoCoRT when a graphical user interface is needed. Otherwise, HoCoRT offers more flexibility regarding use of different tools/pipelines. HoCoRT also supports Nanopore reads which GenCof [30] does not.

The evaluation testing was originally intended to be more extensive. There were plans to measure scalability with different numbers of computing threads, and increasingly large datasets. The evaluation testing was limited in scope due to time constraints. This leaves doing more extensive evaluation testing as a future perspective.

Another limiting factor was that all evaluation testing was only done on raw data without cleaning and processing. This allowed greater understanding of the different tools performance on a varied spectrum of read qualities. It may also have been a limiting factor on the evaluation results. For example, BWA-MEM2 [68] had consistently low precision and accuracy (relative to the other tested tools), especially in the simulated gut microbiome tests with 1% host contamination. Here, the precision of BWA-MEM2 [68] was 0.2634 in HiSeq tests (table 4.21), and 0.1032 in MiSeq tests (table 4.22). It is unclear what caused this.

Feedback was received from some representative users during the development of HoCoRT, though no extensive usability testing was done. This is not a simple thing to execute properly, and is left as a future perspective.

5.4 Future work

Currently, HoCoRT is only meant to work on single node systems. Ways to distribute the load across many nodes in a distributed system should be explored and tested. One potential solution is implemented by the NIH HPC group. They split the input files and process the resulting files on separate computing nodes using DeconSeq [59] [32]. A similar solution should suffice for HoCoRT as the input files (FastQ) are easily divisible. This should be explored further and tested.

Combination pipelines (e.g. Bowtie2HISAT2) could utilize piping to pass along data. This would improve the runtime and would require less space on the drive during the run. These pipelines currently do not utilize piping. This should be implemented and the performance improvement should be measured. Kraken2 [69] is not able to output the mapped FastQ sequences into stdout, which makes the pipelines utilizing Kraken2 [69] unable to use piping.

There was some experimentation with optimization of mapper configuration parameters. These limited tests proved to have mixed results, therefore the defaults were used in most of the testing for consistency (except BBMap_fast in results section 4.3). The parameters were optimized in a simple way by random experimentation. If more sophisticated optimization algorithms were to be used, it is believed that the results and runtimes could be improved further.

The current version of HoCoRT (version 1.0.0) is representative of the work done in this thesis. Some of the pipelines are slow and inaccurate and should be removed in the future. More tools should also be tested, for example, Magic-BLAST [17] seems to be promising. Magic-BLAST [17] appears to be better at intron-discovery than HISAT2 [43], and this may or may not translate to host contamination removal, it should therefore be tested.

The developer documentation should be improved. A documentation website similar to Bioconda's should be made. For example, the Bioconda's website has descriptions for how to contribute, all the technical documentation, and user documentation in one place. The current HoCoRT documentation website at https://ignasrum.github.io/hocort/ only hosts technical documentation.

Some of the existing tools mentioned in section 1.5 were not tested and are not supported by HoCoRT version 1.0.0. These tools include SNAP [71] and Centrifuge [44]. They should be tested and possibly implemented if they prove to be useful. The main reason why they were not tested and implemented was lack of time. There are many existing tools and it would take a lot of time to test all of them thoroughly. Therefore, only the most popular tools were chosen for the testing to limit the scope.

As mentioned in section 5.3, the scope of the testing was limited. More extensive evaluation and usability testing should be done. The user interface should be improved according to feedback gotten from the users.

In section 4.1.9, a potential HoCoRT input data quality control script/tool was mentioned. This should be implemented if there is a demand for it. Although, there are already several good solutions such as BBDuk [19] and Trimmomatic [16].

Many pipelines combining different tools (e.g. Bowtie2HISAT2) were initially implemented and are still available in HoCoRT version 1.0.0, but will be removed in later versions. This decision was made on the basis of performance and ease of use. Pipelines such as Bowtie2HISAT2 depend on the performance of both tools. For example, if one of the tools has low precision then the combination pipeline will also have low precision. Sensitivity could be improved by using a combination pipeline, but many of the tools had a sensitivity of 1.0 by themselves. Therefore, there is no point in keeping most of the combination pipelines. Many of the average-performance pipelines should be removed to improve the ease of use of HoCoRT.

Lastly, as mentioned previously under limitations, HoCoRT does not have a graphical interface. A simple Python interface could be implemented by utilizing the HoCoRT Python library as the backend. This would make HoCoRT even more accessible as people who find it hard to use command line tools would be able to use HoCoRT.

Bibliography

1 *Advantages of paired-end and single-read sequencing*. (n.d.). Retrieved December 5, 2022, from https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html

2 Albrecht, J. P. (n.d.). How the GDPR will change the world, 4.

3 Alessandrini, V. (2016). Intel threading building blocks. *Shared memory application programming* (pp. 307–339). Elsevier. https://doi.org/10.1016/B978-0-12-803761-4.00011-3

4 AltschuP, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, *215*, 403–410.

5 *BBMap - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/bbmap

6 *BBMap - sourceforge page*. (n.d.). Retrieved May 1, 2021, from https://sourceforge.net/projects/bbmap/

7 *BBMap - user guide*. (n.d.). Retrieved May 1, 2021, from https://jgi.doe.gov/data-and-tools/software-tools/bbtools/bb-tools-user-guide/bbmap-guide/

8 Behjati, S., & Tarpey, P. S. (2013). What is next generation sequencing? *Archives of disease in childhood - Education & practice edition*, *98*(6), 236–238. https://doi.org/10.1136/archdischild-2013-304340

9 Bharti, R., & Grimm, D. G. (2021). Current challenges and best-practice protocols for microbiome analysis. *Briefings in Bioinformatics*, *22*(1), 178–193. https://doi.org/10.1093/bib/bbz155

10 *Bioconda homepage*. (n.d.). Retrieved May 14, 2021, from https://bioconda.github.io/

11 *BLAST - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/blast

12 *BLAST - download guide*. (n.d.). Retrieved May 1, 2021, from https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download

13 *BLAST - download page*. (n.d.). Retrieved May 2, 2021, from ftp://ftp.ncbi.nih.gov/blast/executables/release/LATEST

14 *BLAST - NCBI*. (n.d.). Retrieved May 1, 2021, from https://blast.ncbi.nlm.nih.gov/Blast.cgi

15 Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. https://doi.org/10.1145/2723872.2723882

16 Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: A flexible trimmer for illumina sequence data. *Bioinformatics*, *30*(15), 2114–2120. https://doi.org/10.1093/bioinformatics/btu170

17 Boratyn, G. M., Thierry-Mieg, J., Thierry-Mieg, D., Busby, B., & Madden, T. L. (2019). Magic-BLAST, an accurate RNA-seq aligner for long and short reads. *BMC Bioinformatics*, *20*(1), 405. https://doi.org/10.1186/s12859-019-2996-x

18 *Bowtie2 - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/BenLangmead/bowtie2

19 Bushnell, B. (2014a). BBTools software package. *URL http://sourceforge.net/projects/bbmap*, *578*, 579.

20 Bushnell, B. (2014b). BBMap: A fast, accurate, splice-aware aligner. https://www.osti.gov/biblio/1241166

21 *BWA-MEM2 - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/bwa-mem2

22 *BWA-MEM2 - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/bwa-mem2/bwa-mem2

23 Cai, X., Langtangen, H. P., & Moe, H. (2005). On the performance of the python programming language for serial and parallel scientific computations. *Scientific Programming*, *13*(1), 31–56. https://doi.org/10.1155/2005/619804

24 *Centrifuge - bioconda*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/centrifuge

25 *Centrifuge - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/DaehwanKimLab/centrifuge

26 *Centrifuge - homepage*. (n.d.). Retrieved May 1, 2021, from http://www.ccb.jhu.edu/software/centrifuge/

27 Cerutti, F., Bertolotti, L., Goldberg, T. L., & Giacobini, M. (2012). Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): Application and theory. *BioData Mining*, *4*(1), 20. https://doi.org/10.1186/1756-0381-4-20

28 Church, G. M., & Gilbert, W. (n.d.). (DNA methylation/UV crosslinking/filter hybridization/immunoglobulin genes), 5.

29 *Contamination in sequence databases*. (n.d.). Retrieved March 19, 2021, from https://www.ncbi.nlm.nih.gov/tools/vecscreen/contam/

30 Czajkowski, M. D., Vance, D. P., Frese, S. A., & Casaburi, G. (2019). GenCoF: A graphical user interface to rapidly remove human genome contaminants from metagenomic datasets (I. Birol, Ed.). *Bioinformatics*, *35*(13), 2318–2319. https://doi.org/10.1093/bioinformatics/bty963

31 *DeconSeq - GitHub page*. (n.d.). Retrieved May 2, 2021, from https://github.com/kiwiroy/deconseq

32 *DeconSeq: DECONtamination of SEQuence data using a modified version of BWA-SW*. (2020, August 20). Retrieved May 15, 2022, from https://hpc.nih.gov/apps/DeconSeq.html

33 de Koning, A. P. J., Gu, W., Castoe, T. A., Batzer, M. A., & Pollock, D. D. (2011). Repetitive elements may comprise over two-thirds of the human genome (G. P. Copenhaver, Ed.). *PLoS Genetics*, *7*(12), e1002384. https://doi.org/10.1371/journal.pgen.1002384

34 *GenCoF - GitHub page*. (n.d.). Retrieved May 2, 2021, from https://github.com/MattCzajkowski/GenCoF

35 Gourlé, H., Karlsson-Lindsjö, O., Hayer, J., & Bongcam-Rudloff, E. (2019). Simulating illumina metagenomic data with InSilicoSeq (J. Hancock, Ed.). *Bioinformatics*, *35*(3), 521–522. https://doi.org/10.1093/bioinformatics/bty630

36 *HISAT2 - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/hisat2

37 *HISAT2 - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/DaehwanKimLab/hisat2

38 *HISAT2 - manual page*. (n.d.). Retrieved May 1, 2021, from http://daehwankimlab.github.io/hisat2/manual/

39 Huang, W., Li, L., Myers, J. R., & Marth, G. T. (2012). ART: A next-generation sequencing read simulator. *Bioinformatics*, *28*(4), 593–594. https://doi.org/10.1093/bioinformatics/btr708

40 *Information about GRCh37.p13* [Human genome assembly GRCh37.p13]. (n.d.). Retrieved April 21, 2022, from https://www.ncbi.nlm.nih.gov/grc/human/data?asm=GRCh37.p13

41 Ji, Y., Abrams, N., Zhu, W., Salinas, E., Yu, Z., Palmer, D. C., Jailwala, P., Franco, Z., Roychoudhuri, R., Stahlberg, E., Gattinoni, L., & Restifo, N. P. (2014). Identification of the genomic insertion site of pmel-1 TCR α and β transgenes by next-generation sequencing (Z. Gong, Ed.). *PLoS ONE*, *9*(5), e96650. https://doi.org/10.1371/journal.pone.0096650

42 Johnson, K. V.-A. (2020). Gut microbiome composition and diversity are related to human personality traits. *Human Microbiome Journal*, *15*, 100069. https://doi.org/10.1016/j.humic.2019.100069

43 Kim, D., Paggi, J. M., Park, C., Bennett, C., & Salzberg, S. L. (2019). Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature Biotechnology*, *37*(8), 907–915. https://doi.org/10.1038/s41587-019-0201-4

44 Kim, D., Song, L., Breitwieser, F. P., & Salzberg, S. L. (2016). Centrifuge: Rapid and sensitive classification of metagenomic sequences. *Genome Research*, *26*(12), 1721–1729. https://doi.org/10.1101/gr.210641.116

45 Koster, J., & Rahmann, S. (2012). Snakemake–a scalable bioinformatics workflow engine. *Bioinformatics*, *28*(19), 2520–2522. https://doi.org/10.1093/bioinformatics/bts480

46 *Kraken2 - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/kraken2

47 *Kraken2 - GitHub manual page*. (n.d.). Retrieved May 2, 2021, from https://github.com/DerrickWood/kraken2/wiki/Manual

48 Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with bowtie 2. *Nature Methods*, *9*(4), 357–359. https://doi.org/10.1038/nmeth.1923

49 Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, *25*(14), 1754–1760. https://doi.org/10.1093/bioinformatics/btp324

50 Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

51 Li, H. (2013, May 26). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM [Number: arXiv:1303.3997]. Retrieved May 15, 2022, from http://arxiv.org/abs/1303.3997

52 Li, H. (2018). Minimap2: Pairwise alignment for nucleotide sequences (I. Birol, Ed.). *Bioinformatics*, *34*(18), 3094–3100. https://doi.org/10.1093/bioinformatics/bty191

53 Li, H., & Durbin, R. (2010). Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, *26*(5), 589–595. https://doi.org/10.1093/bioinformatics/btp698

54 Liu, T., Chen, C.-Y., Chen-Deng, A., Chen, Y.-L., Wang, J.-Y., Hou, Y.-I., & Lin, M.-C. (2020). Joining illumina paired-end reads for classifying phylogenetic marker sequences. *BMC Bioinformatics*, *21*(1), 105. https://doi.org/10.1186/s12859-020-3445-6

55 *Minimap2 - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/minimap2

56 *Minimap2 - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/lh3/minimap2

57 Quigley, E. M. (2017). Basic definitions and concepts: Organization of the gut microbiome. *Gastroenterology Clinics of North America*, *46*(1), 1–8. https://doi.org/10.1016/j.gtc.2016.09.002

58 Rognes, T., Flouri, T., Nichols, B., Quince, C., & Mahé, F. (2016). VSEARCH: A versatile open source tool for metagenomics. *PeerJ*, *4*, e2584. https://doi.org/10.7717/peerj.2584

59 Schmieder, R., & Edwards, R. (2011). Fast identification and removal of sequence contamination from genomic and metagenomic datasets (F. Rodriguez-Valera, Ed.). *PLoS ONE*, *6*(3), e17288. https://doi.org/10.1371/journal.pone.0017288

60 Sjoberg, D. I., Johnsen, A., & Solberg, J. (2012). Quantifying the effect of using kanban versus scrum: A case study. *IEEE Software*, *29*(5), 47–53. https://doi.org/10.1109/MS.2012.110

61 *SNAP - bioconda page*. (n.d.). Retrieved May 1, 2021, from https://anaconda.org/bioconda/snap-aligner

62 *SNAP - GitHub page*. (n.d.). Retrieved May 1, 2021, from https://github.com/amplab/snap

63 *SNAP - microsoft page*. (n.d.). Retrieved May 1, 2021, from https://www.microsoft.com/en-us/research/project/snap/

64 *SRA-toolkit*. (n.d.). Retrieved May 16, 2022, from https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software

65 Tharwat, A. (2021). Classification assessment methods. *Applied Computing and Informatics*, *17*(1), 168–192. https://doi.org/10.1016/j.aci.2018.08.003

66 The Bioconda Team, Grüning, B., Dale, R., Sjödin, A., Chapman, B. A., Rowe, J., Tomkins-Tinch, C. H., Valieris, R., & Köster, J. (2018). Bioconda: Sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, *15*(7), 475–476. https://doi.org/10.1038/s41592-018-0046-7

67 Ursell, L. K., Metcalf, J. L., Parfrey, L. W., & Knight, R. (2012). Defining the human microbiome. *Nutrition Reviews*, *70*, S38–S44. https://doi.org/10.1111/j.1753-4887.2012.00493.x

68 Vasimuddin, M., Misra, S., Li, H., & Aluru, S. (2019). Efficient architecture-aware acceleration of BWA-MEM for multicore systems. *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 314–324. https://doi.org/10.1109/IPDPS.2019.00041

69 Wood, D. E., Lu, J., & Langmead, B. (2019). Improved metagenomic analysis with kraken 2. *Genome Biology*, *20*(1), 257. https://doi.org/10.1186/s13059-019-1891-0

70 Yang, C., Chu, J., Warren, R. L., & Birol, I. (2017). NanoSim: Nanopore sequence read simulator based on statistical characterization. *GigaScience*, *6*(4). https://doi.org/10.1093/gigascience/gix010

71 Zaharia, M., Bolosky, W. J., Curtis, K., Fox, A., Patterson, D., Shenker, S., Stoica, I., Karp, R. M., & Sittler, T. (2011). Faster and more accurate sequence alignment with SNAP. *arXiv:1111.5572 [cs, q-bio]*. Retrieved March 19, 2021, from http://arxiv.org/abs/1111.5572

72 Zhang, H. (2016). Overview of sequence data formats [Series Title: Methods in Molecular Biology]. In E. Mathé & S. Davis (Eds.), *Statistical genomics* (pp. 3–17). Springer New York. https://doi.org/10.1007/978-1-4939-3578-9_1

Appendix A

Documentation

A.1 End-user documentation

A.1.1 Installation

Installation with Bioconda

Install with Bioconda:

```
conda install -c bioconda hocort
```

Manual installation

First ensure that there is no conda environment called "hocort".

Now download the necessary files:

```
wget https://raw.githubusercontent.com/ignasrum/hocort/main/install.sh &&
wget https://raw.githubusercontent.com/ignasrum/hocort/main/environment.yml
```

After downloading the files, run the installation bash script to install HoCoRT:

```
bash ./install.sh
```

The installation is done. Activate the Conda environment:

```
conda activate hocort
```

A.1.2 Using HoCoRT

Pipeline naming

Pipelines are named after the tools they utilize. For example, the pipeline bowtie2 uses Bowtie2 [48] to map the reads, and kraken2bowtie2 first classifies using Kraken2 [69], then maps using Bowtie2 [48].

Building indexes

Indexes are required to map sequences, and may be built either manually or with "hocort index" which simplifies the process. A Bowtie2 [48] index may built using "hocort index" with the following command:

```
hocort index bowtie2 --input genome.fasta --output dir/basename
```

If one wishes to remove multiple organisms from sequencing reads, the input fasta should contain multiple genomes.

```
cat genome1.fasta genome2.fasta > combined.fasta
```

Paired end run

To map reads and output mapped/unmapped reads use the following command:

```
hocort map bowtie2 -x dir/basename -i input1.fastq input2.fastq
-o out1.fastq out2.fastq
```

Single end run

Exactly as above, but with one input file and one output file.

```
hocort map bowtie2 -x dir/basename -i input1.fastq -o out1.fastq
```

Compressed input/output

Most pipelines support .gz compressed input and output. No extra configuration is required aside from having ".gz" extension in the filename.

Removing host contamination

The filter "–filter true/false" argument may be used to switch between outputting mapped/unmapped sequences. For example, if the reads are contaminated with human sequences and the index was built with the human genome, use "–filter true" to output unmapped sequences (everything except the human reads).

Extracting specific sequences

The filter "–filter true/false" argument may also be used to extract specific sequences. First, the index should be built with the genomes of the organisms to extract. Second, the sequencing reads should be mapped with the "–filter false" argument to output only the mapped sequences (sequences which map to the index containing genomes of the specific organisms).

Advanced usage

Importing and using HoCoRT in Python

HoCoRT can be imported in Python scripts and programs with "import hocort". This allows precise configuration of the tools being run.

```
import hocort.pipelines.bowtie2 as bowtie2

idx = "dir/basename"
seq1 = "in1.fastq"
seq2 = "in2.fastq"
out1 = "out1.fastq"
out2 = "out2.fastq"
# options is passed to the aligner/mapper, this allows precise configuration
options = ["--local", "--very-fast-local"]

returncode = bowtie2.run(idx,
                         seq1,
                         out1,
                         seq2=seq2,
                         out2=out2,
                         options=options)
```

Passing arguments to the underlying tools

It is possible to pass arguments to the underlying tools by specifying them in the -c/–config argument like this:

```
hocort map bowtie2 -c="--local --very-fast-local --score-min G,21,9"
```

A.2 Developer documentation

A.2.1 Adding support for an aligner/classifier

All aligner classes are in hocort/aligners. To add support for a new aligner, create a file named after the aligner in hocort/aligners, e.g. bowtie2.py. Look at the abstract base class (ABC) hocort/aligners/aligner.py for methods which have to be implemented. Also look at the already implemented aligner classes to understand how to implement a new one.

There are three methods; "build_index", "align", and "index_interface". The "build_index" method builds a command which is executed by hocort/execute.py. The "align" method also builds a command which is executed by hocort/execute.py. The "index_interface" method provides a command line interface for building an index for the relevant aligner.

The files for classifiers are in hocort/classifiers. Everything is the same as for the aligners, except for the output and the "align" method being called "classify". Instead of SAM or BAM output, the classifiers output FastQ.

A.2.2 Adding a pipeline

The pipeline files are in hocort/pipelines. Look at the "Pipeline" abstract base class at hocort/pipelines/pipeline.py. It has two abstract methods which need to be overridden. The "run" method executes the pipeline. It builds commands using the aligner, classifier, and parser classes, and executes them using hocort/execute.py. The "interface" method provides a command line interface for the pipeline.

A.2.3 Create tests

Create tests for the newly implemented classes. The tests should be placed in hocort/tests. Any necessary test data is in hocort/tests/test_data. The tests are implemented using pytest.

A.2.4 How to contribute

First, fork the project. Implement the changes, and run tests using pytest. Submit a pull request if the tests pass. Don't forget to describe what you have done.

A.2.5 Docstrings

All the functions of HoCoRT have docstrings. pdoc3 was used to generate the developer documentation from docstrings. The documentation can be found in /docs. To generate the documentation, run the script named gen_docs.sh.

A.2.6 Note on logging

All parts of HoCoRT utilize the default Python package called logging to log information. The logging module is initialized by hocort/logging.py in hocort/interface.py. To obtain the logger in a certain file you created, run the following code:

```
import logging

logger = logging.getLogger(__file__)
```

A.2.7 Note on versioning

The version of HoCoRT is specified in hocort/version.py. This file has two variables defined; __version__ and __last_modified__. These are imported and accessed in other parts of HoCoRT by running the following code:

```
import hocort.version as version

print(version.__version__)
print(version.__last_modified__)
```

A.2.8 Note on external dependencies

The external dependencies are specified in hocort/dependencies.py. The name and command used to test the dependency are specified in a dictionary. hocort.dependencies is imported in hocort/interface.py, and the external dependency check are executed:

```
import sys
import hocort.dependencies as dep

if not dep.check_external_dependencies():
    sys.exit(1)
```