Phippery: A phage immunoprecipitation sequencing pipeline to characterize antibody targets

Jared G. Galloway

June 15, 2020

Abstract

A globalized world has drafted more urgency and attention to the danger of pathogens in parallel to amazing discoveries which proliferate the effectiveness of our own immune system to combat disease. The adaptive immune system interacts with and identifies pathogens by complex mechanisms of cell digestion, selection and mutation, a process which produces molecular locks on specialized cells, most commonly, antibodies. The pathogen's surface proteins act as a key, and are flagging to be destroyed when the complimentary antibody cells bind. Identifying both the binding specificities of sample antibody repertoires (usually from blood serum), as well as the knowing the key for pathogens of interest remains an outstanding and challenging problem in serology. Addressing these questions, a method known as Phage Immuno-Precipitation Sequencing, or **PhIP-Seq**, has presented itself as a promising protocol for investigation of antibody targets. This method allows us to observe and quantify the protein-protein interactions between a set of antibodies and artificial epitopes displayed on phages (phage vectors). By designing and synthesizing phages displayed proteins, researchers can model the span of potential epitopes encoded by any locus in a pathogen's genome. The protocol ultimately results in an *enrichment* matrix where each entry represents the number of peptide-antibody binding events observed when serum is exposed to the phage library. Notably, this protocol requires clever experimental design, sensitive wet-lab technique, a bioinformatics pipeline, and powerful statistical tools to parse signal from various sources of noise. In this paper, we will present computational tools to help deal with the bioinformatics, and book-keeping as well as set the stage for analysis by providing a well-defined data structure to organize the entire dataset. Specifically, we provide a **Nextflow** analysis pipeline that takes in demultiplexed next generation sequencing files and produces a coherent dataset to be queried Finally, we present some figures produced by running the pipeline with an anonymous, empirical dataset focused on SARS-CoV-2.

Introduction

The immune system is most simply divided into two biological mechanisms which we call the *innate* and *adaptive* immune systems [4]. The *innate* immune system is primarily made of informed by cells, such as leukocytes, which classify an object as either foreign or native to the body. When a foreign object is detected, the body immediately responds with a set of non-specific deterrents such as inducing a fever. If the pathogen survives these initial defence mechanisms, the body's second wave of defence is a more targeted attack known as the adaptive immune response. The relies upon generating antibodies which can identify a singular pathogen which induced the response. Antibodies primary job is to "tag" a pathogen which can then signal a specialized cell, such as a natural killer cell (NKC) to destroy it later. The ability to rapidly respond to the same pathogen after the initial infection is known as *immunity* and it is a product of the adaptive immune system "memory". When designing a vaccine for a specific virus, the memory of the adaptive immune system is leveraged by provoking the immune response with a neutralized representation of the virus. This needs to be similar enough to the real virus that it allows the body to generate the right cell machinery, without exposing the individual to a dangerous pathogen. It follows that knowledge of antibody binding specificities would inform a great deal about the natural defences, or lack of, an individual possesses. Further, identification of the molecular key, often referred to as the *epitope*, as the primary identifier of a specific pathogen to an antibody could greatly simplify the process of vaccine design. Obtaining this information remains an outstanding challenge in the field of serology, but several advances have recently been made to help us investigate [3].

For linear epitopes, we know that the oligonucleotide sequence encoding for this protein lives somewhere where in the genome of the pathogen. By synthesizing the span of potential epitope-encoding nucleotides across the exon's of a pathogen genome, researchers can leverage Next Generation Sequencing (NGS) to quantify the binding specificities of sample antibodies [1, 5]. This is a novel technique known as Phage Immuno-Precipitation Sequencing, or **PhIP-Seq** [8]. The PhIP-Seq protocol first requires that a researcher design a peptide library from a set of pathogen genome loci of interest. The libraries are most commonly generated using a sliding window across a locus producing oligonucleotide "tiles" encoding for a peptide. The size and stride of the nucleotide tiles is designed using specialized software before they are synthesized. These nucleotide chains are then cloned into a phage with the hope that it models the potential molecular key on the surface of a pathogen. When mixed with antibodies, phages that have bound with an antibody are "precipitated" out of the mixture using magnetic beads with a technique referred to as Immuno-Precipitation (IP). The key idea being that the repertoire of phages caught on the beads should then be representative of antibody targets from the sample. The samples themselves can come from a variety of sources and individuals most fitting to the experiment being run. It should be noted that in nature, we expect the "epitope" of any particular pathogen to be either *linear* or *conformational*. Conformational epitopes are proteins which have a *tertiary* structure, meaning the protein has folded and the epitope can no longer be represented by the linear chain of nucleotides encoding for the protein. The nature of this protocol means that we can only identify *linear* epitopes as the phage-displayed peptides are sequential in practice.

After the IP step as been performed, researchers can then quantify the number of phages caught on the beads using barcoded short-read alignment. From a broad perspective, the analysis of these data consists primarily of demultiplexing the samples, then aligning the peptides to the reference library. By computing the number of read hits for a specific peptide sequence, we generate a count of each specific peptide and sample binding event observed. The final result is in the form of a matrix, X, where (heuristically) there is a row, i, for each peptide and a column, j, for each sample. Concretely, $X_{j,i}$ = the number of reads from sample i, which contain the peptide j.

Doing analysis for this protocol presents two challenges: (1) This protocol requires an immense amount of book-keeping and computational power to produce coherent results. (2) From this protocol, there are a variety of sources we can expect add noise to the resulting data, these are including but not limited to; IgG content of a sample, "immuno-dominant" antibodies, sequencing bias, amplification bias, and phage display (total amount of each phage in the phage library) [12]. Here, approach the first problem by presenting and describing a novel **Nextflow** pipeline constructed to handle all bookkeeping and data processing steps in parallel [2]. The pipeline shows promising results and – when tested on an empirical dataset focused on SARS-CoV-2 – completes over 900 data processing steps in less than ten minutes. Additionally, we provide some preliminary results analyzing the output from this pipeline using a python package, **phippery**, specifically designed for the organization and inspection of the pipeline output.

Methods

In data science, a pipeline is loosely defined set of data processing steps to be run in sequence. Pipeline managers such as **Snakemake** and **Nextflow** compile the dependencies of one process to the next based on the input [10, 6]. Ultimately this results in a Directed Acyclic Graph (DAG) of data processing steps and their respective execution order. In the context of PhIP-Seq, the nature of many seperate sample alignments to different peptide-encoding nucleotide sequence references makes this workflow a perfect candidate for a workflow manager which can perform these steps in parallel on a high performance computer (HPC). For this specific pipeline we use Nextflow paired specialized docker containers to produce a reproducible and highly-parallelized workflow.

Nextflow is primarily defined by a set of *Processes*, *Channels*, and *Operators*. Each process block defines a singular data processing step, given some inputs, using either a command line script or one of a few popular data-driven scripting languages. Each process execution is run in a containerized environment where the inputs are staged in an independent temporary directory making it autonomous to the rest of the pipeline execution steps. The pipeline manager handles all process inputs and outputs through *Channels* so the individual processes don't conflict with each other in terms of file system or necessary software dependencies. A *Channel* can be thought of as a First-In-First-Out (FIFO) queue and acts as the only mode

of communication between the data processing steps. Processes which define their input from a channel are executed as soon as something is put in the queue. *Operators* allow us to merge, split, or transform the items placed in a channel to configure a nicely formatted output usually containing files and metadata needed for a downstream process. Figure 1 shows the DAG of our pipeline from input metadata and sequencing file to a specialized python object containing the respective counts and metadata for each sample and peptide [11]. In this section, we will discuss the pipeline inputs, the individual steps of the pipeline, and the resulting output.

Pipeline Input - Data

To configure the run of a pipeline the user must specify the path of JSON configuration file. This file contains information about where the pipeline can find sample and peptide metadata, the sample fastq files for each experiment, and lastly, some information about read and reference nucleotide length. Using this, the pipeline automatically parses the JSON file and configures the steps needed to generate the output. This approach allows the user to organize their data how they like, and let the pipeline manager structure pipeline output..

Configuration file There a few key-value pairs that are required to be in the configuration file. An example configuration file for a simulated dataset is shown in Listing 1. First the user must specify an "output_dir" key with the value as a file path where the would like the results of the pipeline to be published. The "samples" key must specify a valid path to a sample metadata csv described below. Each sample is tied directly to one of the keys specified in "experiments", thus, the path to the base directory holding all sample files must be specified for *each* experiment. Similarly, each sample specifies a reference library it should be aligned against, this is the "reference" column. Additionally, there are a few other parameters which can be used to specify read and reference oligo length for alignment schemes. It should be clear that only a *single* sample metadata csv is specified, while *multiple* peptide references can be specified. The pipeline will output a phippery PhipData object for each reference containing the respective samples as described in the sample metadata file.

```
{
    "output_dir" : "simulate_ones/",

    "samples" : "simulate_ones/samples/sample_metadata.csv",

    "experiments" : {
        "expa" : "simulate_ones/experiments/expa/some/path/",
        "expb" : "simulate_ones/experiments/expb/some/path/"
    },

    "references" : {
        "refa" : "simulate_ones/references/peptide_metadata_a.csv",
        "refb" : "simulate_ones/references/peptide_metadata_b.csv"
    }
}
```

Listing 1: An example JSON configuration file for a simulated dataset.

Sample metadata To specify which samples are included in the pipeline run, the researcher who designs the experiment compiles a comma-separated file specifying all relevant metadata for each Immuno-Precipitation experiment run. Table 1 shows an example of this sample metadata table for simulated data. There are four required fields that must be specified for each sample (row) in the file; (1) A unique identifier in the form of a single integer, specified in the "ID" column. (2) The file *pattern* which specifies the filenames for all technical replicates of a single sample in the "fastq_pattern" column. (3) An "experiment" name which, in the config file, ties the sample to the location of the base directory where the fastq file is expected to be found. (4) A "reference" name which specifies the index the samples should be aligned to.

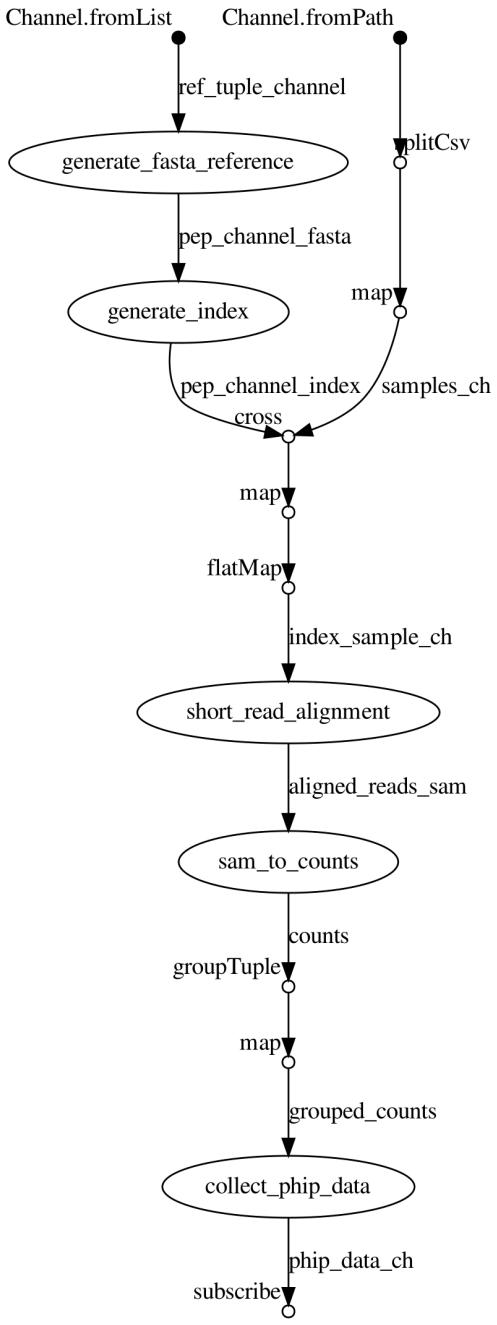


Figure 1: The Directed Acyclic Graph (DAG) representation of our pipeline workflow. Here, each bubble represents a single data processing step to be run in temporary directory inside a respective docker container. The arrows specify **Nextflow** "channels" which direct and organize output from one process to the next. Dots between channel arrows specify where a **Nextflow** "operator" was used to organize and collect the output of a set of channels into another set of channels.

While this presents only *required* fields, it's generally useful to add any other relevant metadata connected to each sample such as sample type, species etc. Any and all fields can be used for downstream analysis when querying the dataset.

ID	fastq_pattern	reference	experiment
0	sample-*-0	refa	expa
1	sample-*-1	refa	expa
2	xeno-AE-122-*-R1.3.3.0	refa	expa
3	xeno-AE-122-*-R1.3.3.1	refa	expa
4	xeno-AE-122-*-R1.3.3.2	refa	expa
5	xeno-AE-122-*-R1.3.3.3	refb	expb
6	xeno-AE-122-*-R1.3.3.4	refb	expb
7	johnny-boy-*-0	refb	expb
8	johnny-boy-*-1	refb	expb
9	johnny-boy-*-2	refb	expb

Table 1: An example sample metadata table containing the requires fields for a simulated dataset.

Peptide metadata For each unique reference listed in sample metadata file, we must specify a peptide metadata comma-separated file in order to build an index. As seen in Table 2, there only two required fields; (1) A unique identifier in the form of a single integer, specified in the "ID" column. (2) the "Oligo" column specifying the nucleotide encoding for a specific peptide. Currently, we expect the index oligo sequences, if there at all, be lowercase and the peptide encoding sequence be uppercase. Similarly to the sample metadata table, it is generally useful to include other information about the peptide such the virus strain, and some genomic positional information.

ID	Oligo
0	gcatcagtaggctgcgtaGGGATTAGGCGGACCTCCATGAATACCG...
1	gcatcagtaggctgcgtaTGTAGGCAAGGAGCAACACTTCTTCTTT...
2	gcatcagtaggctgcgtaGAGAATGGGCCAGGAATGATCTACTGTC...
3	gcatcagtaggctgcgtaCGTGTCAAAAACTGCGTATTTACGAAGA...
4	gcatcagtaggctgcgtaTTTCAGATCCTACCATTTGTGTCCTTAA...
5	gcatcagtaggctgcgtaCCGAGTTCGTATTTTTACAAATCCCGGA...
6	gcatcagtaggctgcgtaTATTTAATGAGTGTGAGGCAAAGTTGTT...
7	gcatcagtaggctgcgtaTTTACGCTCAGCAAGCGTAGCTAGCATG...
8	gcatcagtaggctgcgtaTCAAACAGGTTACGACACAAAGAACGCC...
9	gcatcagtaggctgcgtaTCGCATTCTCCTGGCCTACTTACAGTTC...

Table 2: An example peptide metadata table containing the requires fields for a simulated dataset.

Data Processing Steps

There are a number of individual data processing steps that take place when running this pipeline. The number of times each process is run is determined by the number of references and samples. Concretely, if there are S samples, each with T technical replicates and R references, then we can expect $2ST + 3R$ separate processes to be run. In this section, we will briefly review the unique data processing steps in the pipeline. The pipeline begins by defining two input channels; the first channel defines the references to be generated and their respective peptide metadata tables, the second channel parses each sample in the sample

metadata file, finds all technical replicates for that sample, and merges with the first channel. In short, the pipeline the performs sequence alignment, counts generation, and counts merging.

generate fasta reference The first step in the pipeline is to parse the configuration file for all listed references and respective peptide metadata files. For each reference found, the pipeline executes a "generate_fasta_reference" process. The primary job for this step is to produce a fasta representation of the respective peptide metadata table in preparation for index generation. Simply put, this step simply converts the ID into a header followed by the nucleotide sequence encoding for each peptide in the table. A container for the **phippery** python package is used as the primary execution step. For cluster submission, we request only a single core with a modest amount of memory because the software being used isn't multithreaded.

Index generation For samples to be aligned to the reference, a specific binary index must be generated. In this step, we use **bowtie-build** to generate the index which each respective sample will be aligned to [7] The five binary files that are output are nested in a subdirectory dir named "¡reference name¿_index". This process is run in a public **bowtie** container hosted on Quay and *can* be multithreaded.

Short read alignment The bulk of the work performed in the pipeline happens during short-read sequence alignment of every technical replicate to it's respective reference peptide library. After the references have been built and the sample metadata file has been parsed, we merge the channels into a new channel, named *index_sample_ch*. Taking in metadata, and reference for a single technical replicate, we perform end-to-end short read alignment using **bowtie** [7]. Unlike most alignment scenarios, we are not looking for a read *within* a reference sequence, but rather our nucleotide encoding of peptide references in a sample read. Concretely, we trim the 3' (low-quality) end of each sample read by the difference in length between the reference and the read. For example, if our read length is 125bp and out peptide references tiles are 117bp, the we trim 8bp from the right side of the read and look and perform alignment.

```
zcat ${respective_replicate_path} |
bowtie --trim3 ${trim} -n ${num_mm} -l ${tile_length} \
--tryhard --nomaqround --norc --best --sam --quiet \
${index}/${ref_name} - \
> ${ID}.${replicate_number}.sam
```

Listing 2: Short-read alignment parameters

The output from this step is a sam file named named by sample id, and technical replicate number. Each line in the sam file gives alignment information about the same reads.

alignment counting After the samples have been aligned, we need to compute the number of reads which aligned to each separate peptide. Here, we use **samtools-idxstats** tool to parse each sam file and output a two column tsv for each respective technical replicate [9]. Before computing the counts, we must convert to a binary format (BAM), index, and sort the samfile to prep it for **samtools-idxstats**.

```
samtools view -u -@ 4 \${sam_file} | \
samtools sort -@ 4 - > ${ID}.${replicate_number}.bam
samtools sort -@ 4 ${ID}.${replicate_number}.bam \
-o ${ID}.${replicate_number}.sorted
mv ${ID}.${replicate_number}.sorted ${ID}.${replicate_number}.bam
samtools index -b ${ID}.${replicate_number}.bam
samtools idxstats ${ID}.${replicate_number}.bam | \
cut -f 1,3 | sed "/^*/d" > ${ID}.${replicate_number}.tsv
```

Listing 3: samtools processing steps

The tab-separated values file for each sample contains two columns; the first being the peptide "ID" number, and the second is the number of alignments reported with no more than ${num_mm} mismatches as described in the alignment step.

merging counts and metadata The final step of the pipeline needs to collect *all* the counts files for a single reference, and merge them into a coherent dataset. To do this, we use custom code in **phippery** designed to merge all sample counts given a list of tsv files described in the previous step. First the code looks for all technical replicates of a single sample, then computes correlation and takes the mean (or optionally, the sum) of the sample before merging into the final counts table by peptide ID. This means that each individual sample will be presented as a single column in the counts table, and the rows will represent peptide counts. Finally, this step appends technical replicate info onto sample metadata and produces a **phippery.PhipData** object described in more detail below.

Pipeline Output

When doing analysis on the resulting counts from an experiment, it's imperative that we have access to all metadata for each sample and peptide. this is because the metadata defines all interesting aspects and subsets of the entire counts table. Often, we would like to do things like; compare and contrast counts between sample groups, normalize counts by negative controls, or even look for cross reactivity between virus strains. Ultimately, we have three tables that constitute the entirety of a dataset for a given peptide reference library; (1) the *counts* table where the peptide enrichments numbers are held for each sample, (2) the *peptide metadata* table containing things like nucleotide encoding and information about there that tile derived from, biologically, and (3) the *sample metadata* table giving a way to make sense of enrichment we see across different sample types. The python package we created, **phippery**, centers around a python Object, **PhipData**, defined primarily by these three tables. Additionally, it provides a useful and tested array of methods to perform set operations on the entire dataset. The key to keeping these three tables organized and trivial to query is the fact the peptide metadata and counts share the same index columns, and the sample metadata index matches the counts columns. Each of these tables are queried in together by the unique identifiers ("ID" column) specified in the metadata tables that was initially provided to the pipeline. Figure 2 visualizes this relationship.

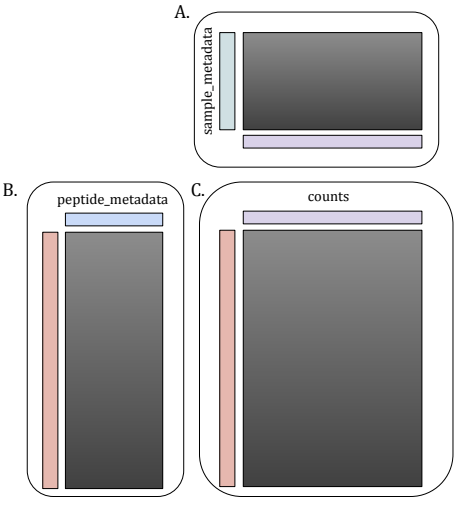
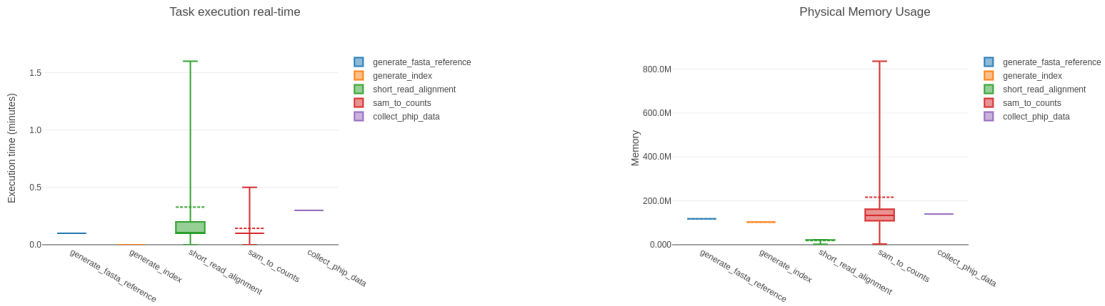


Figure 2: Cartoon of a PhipData Python Object. Most notably, the *orange* rectangles in tables B and C represent the synchronized index for peptide metadata and row counts. The *purple* rectangles seen in A and C represent the synchronized sample metadata index with the counts column headers.



(a) Distribution of total process execution time for each process definition

(b) Distribution of total process memory usage for each process definition

Pipeline results on an anonymous dataset

The pipeline described above has thus far been used on a novel set of empirical datasets focused on SARS-CoV-2. For the purpose of example, we will include some figures produced from one of these pipeline runs. However, due to the unpublished, private and preliminary nature of the data thus far, we will not be including any defining names, details, or interpretations of the data itself. Instead, we will simply present the figures and captions generally explaining what they represent as these exemplify much of the pipeline explanation above.

Amazingly for this dataset, the pipeline managed and executed more than 900 separate processes on a local HPC and completed the entire workflow in less than 7 minutes. Each process was farmed out as a separate job with time, cpu, and memory allocation specific to the process being run. There we're four unique docker containers which handled the software dependencies for each one of the data processing steps during execution. In Figure 3a, we can see that the majority of the individual processes took less than 30 seconds to run. Run completely in parallel, this explains how such a massive amount of computation can finish so quickly taking full advantage of HPC infrastructure. Unsurprisingly, the sequence alignment step takes the largest amount of time, while index generation is nearly instant. In contrast to time, 3b shows short read alignment takes the *least* amount of memory on average – clearly showing the advantage of **Bowtie** for short read alignment.

As a first look at the results, we can very easily plot the raw counts in the matrix as a heatmap. Figure 4 shows this for our empirical dataset across 6 different individual sequencing experiments all included in the singular pipeline run. It seems as though the experiments $1, 2$ and 4 have consistent looking horizontal bands, where the others are quite noisy. This pattern conforms to what we observe when plotting sample-specific technical replicate correlation in Figure 5. In other words, samples from experiments $3, 5$ and 6 were not did not show reproducibility, and thus we can expect high quality results in the enrichment matrix. The bars showing correlation for technical replicates are colored their factor status in one of the many sample metadata columns.

Finally, in Figure 6, we show *Standardized Enrichment* across all samples for a single experiment. Concretely, this means we divide columns by the column sum to get sample specific enrichment *frequencies* before dividing by a library input control and subtracting a "beads-only" mock IP negative control. We then group samples by a their respective status in another sample metadata columns to observe group differences as seen in Figure 7

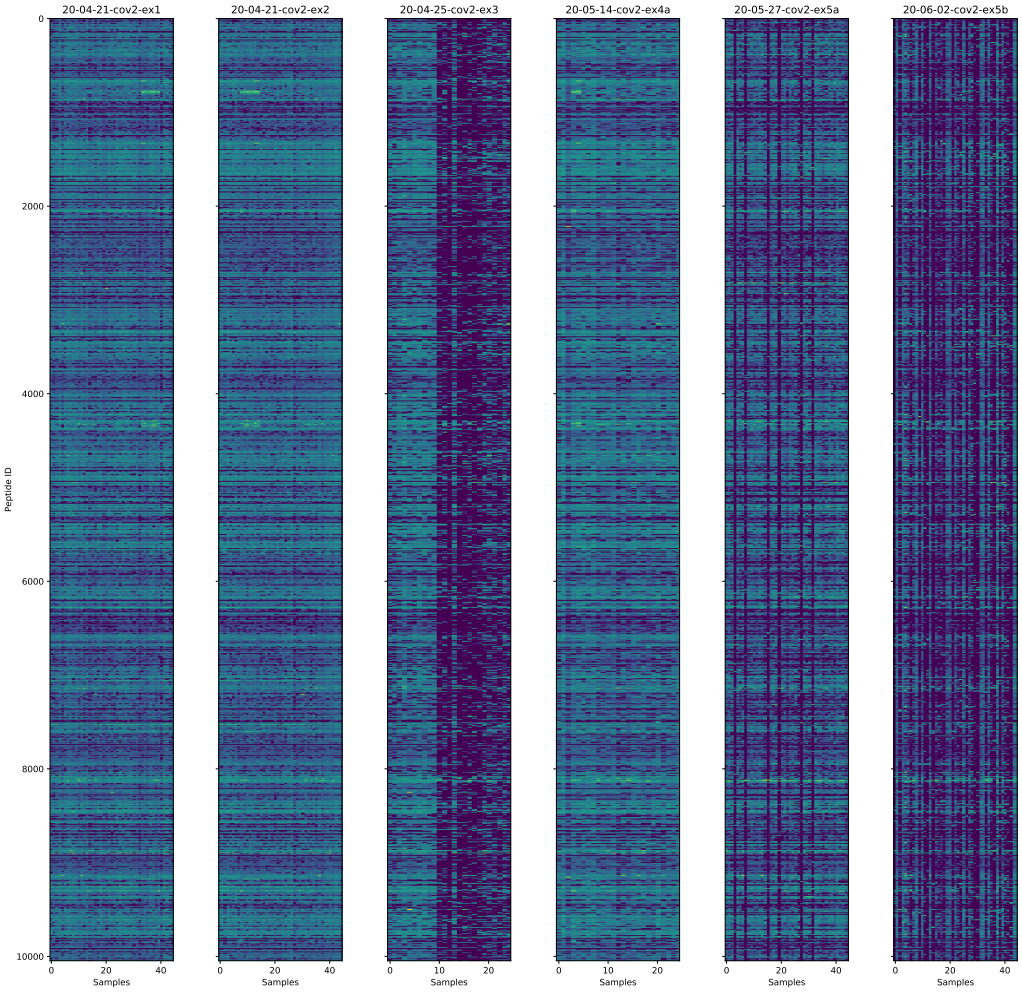


Figure 4: A heapmap showing the raw counts matrix, grouped by experiment.

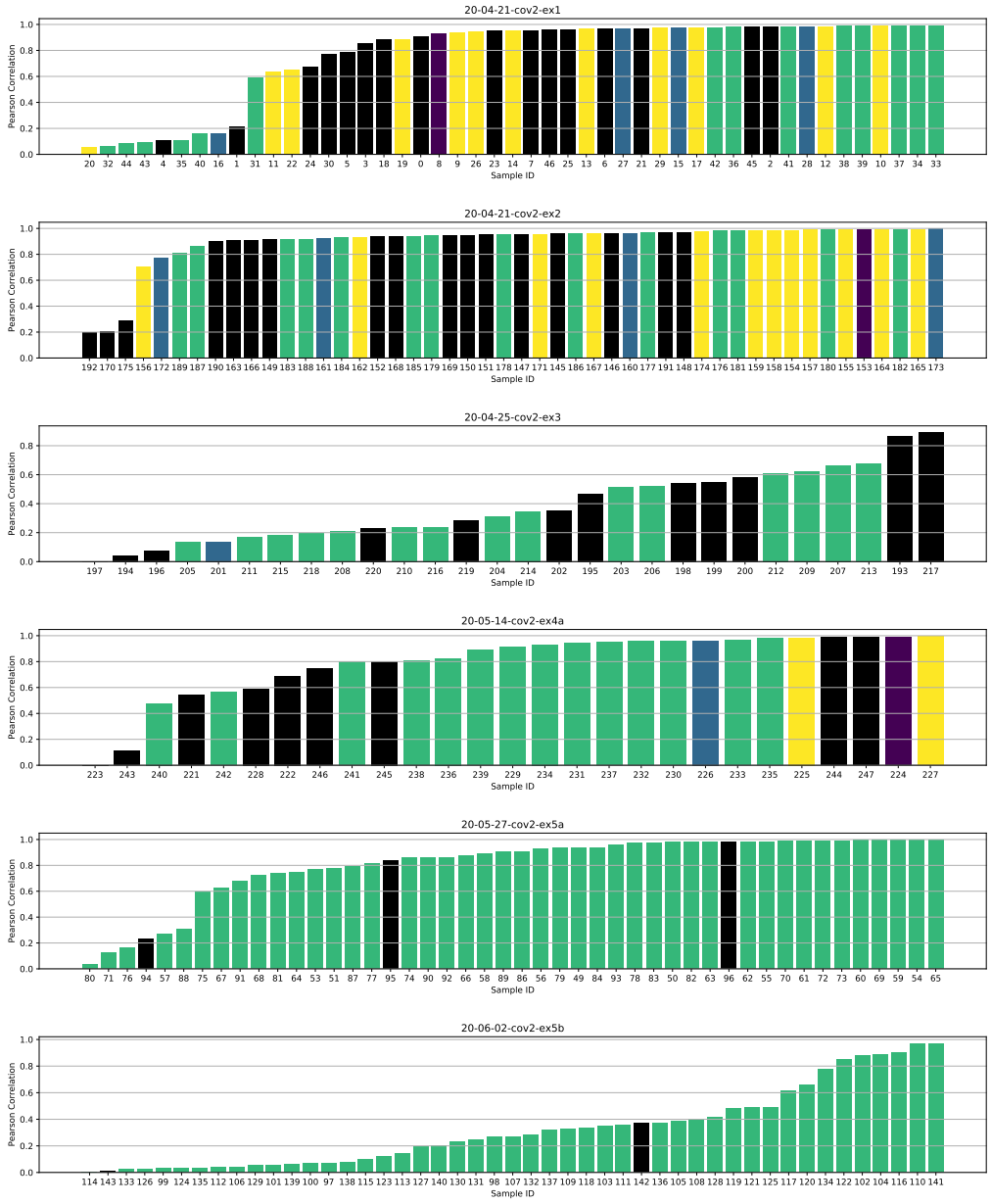


Figure 5: Technical replicate correlation for each sample with greater than one replicate, grouped by experiment and colored by a sample metadata column.

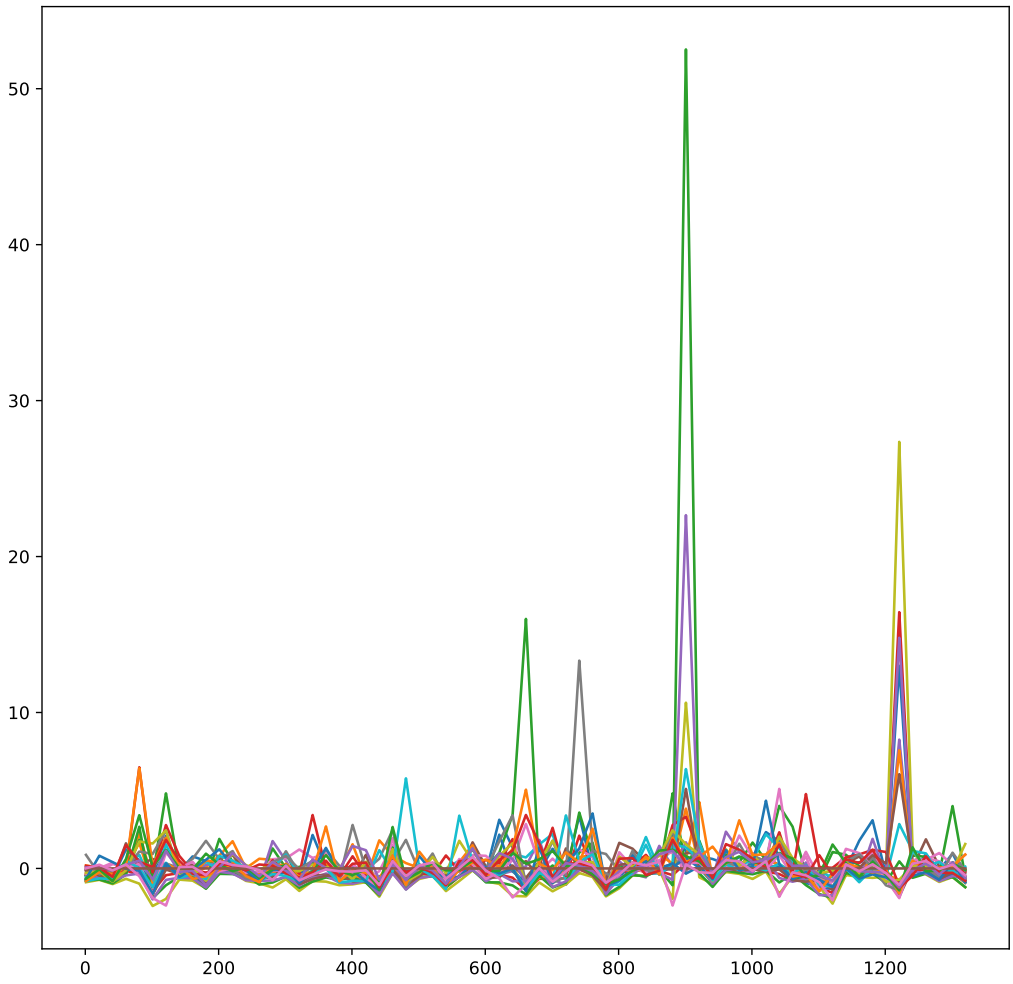


Figure 6: Standardized enrichment for strain ¡X¿, plotted as a function of tile genomic location. Each line represents a single samples enrichment.

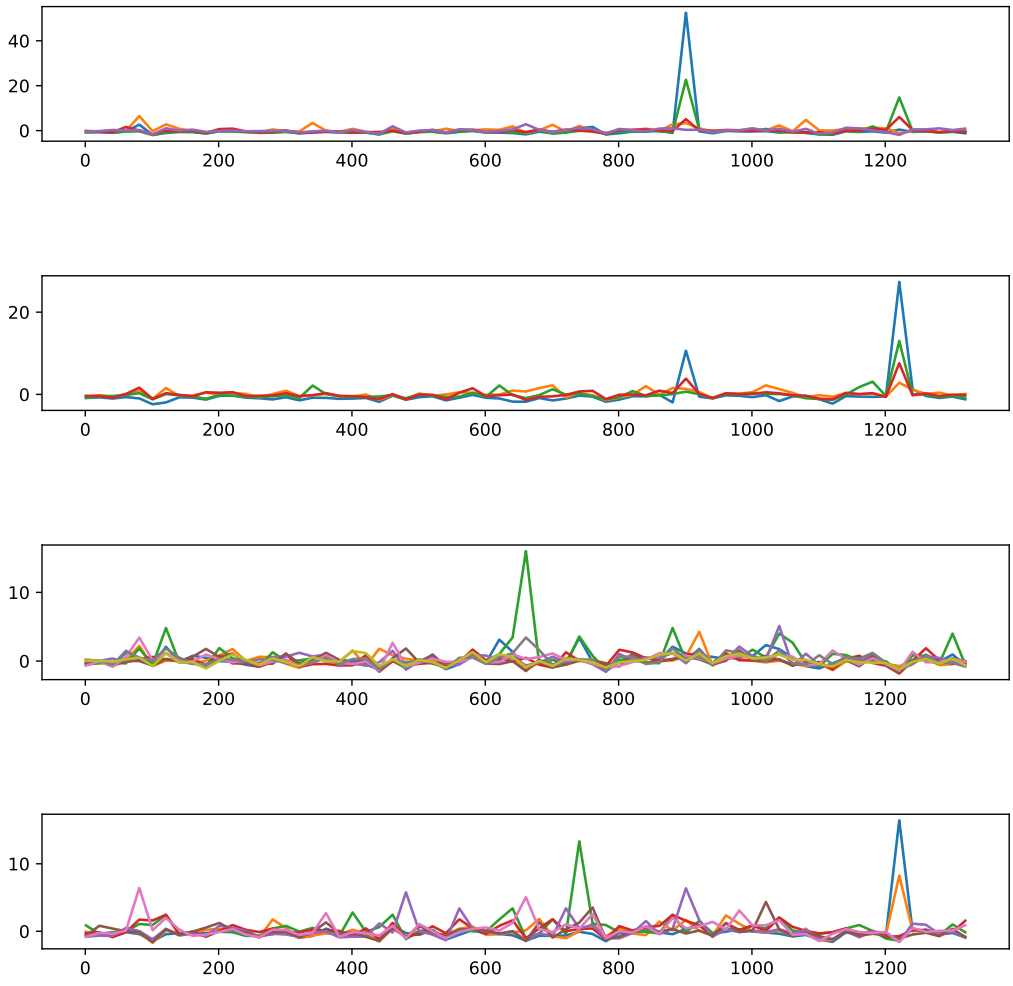


Figure 7: Standardized enrichment for strain ¡X¿, plotted as a function of tile genomic location. The samples are then grouped by a column in the sample metadata column.

References

[1] Jody D. Berry, Kevin Hay, James M. Rini, Meng Yu, Linfa Wang, Francis A. Plummer, Cindi R. Corbett, and Anton Andonov. Neutralizing epitopes of the sars-cov s-protein cluster independent of repertoire, antigen structure or mab technology. *mAbs*, 2(1):53–66, 2010. PMID: 20168090.

[2] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, Apr 2017.

[3] Laura E. Doepker, Cassandra A. Simonich, Duncan Ralph, Mackenzie M. Shipley, Meghan Garrett, Theodore Gobillot, Vladimir Vigdorovich, D. Noah Sather, Ruth Nduati, Frederick A. Matsen, and Julie M. Overbaugh. Diversity and function of maternal hiv-1-specific antibodies at the time of vertical transmission. *Journal of Virology*, 94(9), 2020.

[4] Aakanksha Jain and Chandrashekhar Pasare. Innate control of adaptive immunity: Beyond the three-signal paradigm. *Journal of immunology (Baltimore, Md. : 1950)*, 198(10):3791–3800, May 2017. 28483987[pmid].

[5] Saahir Khan, Rie Nakajima, Aarti Jain, Rafael Ramiro de Assis, Al Jasinskas, Joshua M. Obiero, Oluwasanmi Adenaiye, Sheldon Tai, Filbert Hong, Donald K. Milton, Huw Davies, Philip L. Felgner, and . Analysis of serologic cross-reactivity between common human coronaviruses and sars-cov-2 using coronavirus antigen microarray. *bioRxiv*, 2020.

[6] Johannes Kster and Sven Rahmann. Snakemakea scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 08 2012.

[7] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, Mar 2009.

[8] H. Benjamin Larman, Zhenming Zhao, Uri Laserson, Mamie Z. Li, Alberto Ciccia, M. Angelica Martinez Gakidis, George M. Church, Santosh Kesari, Emily M. LeProust, Nicole L. Solimini, and Stephen J. Elledge. Autoantigen discovery with a synthetic human peptidome. *Nature Biotechnology*, 29(6):535–541, Jun 2011.

[9] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079, Aug 2009. 19505943[pmid].

[10] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), mar 2014.

[11] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[12] Tiezheng Yuan, Divya Mohan, Uri Laserson, Ingo Ruczinski, Alan N. Baer, and H. Benjamin Larman. Improved analysis of phage immunoprecipitation sequencing (phip-seq) data using a z-score algorithm. *bioRxiv*, 2018.