SLUR(M)-py: A SLURM Powered Pythonic Pipeline for Parallel Processing of 3D (Epi)genomic Profiles

Cullen Roth[1,*], Vrinda Venu[2], Sasha Bacot[1],
Christina R. Steadman[2], and Shawn R. Starkenburg[1]

[1]Los Alamos National Laboratory, Genomics and Bioanalytics, Los Alamos, New Mexico
[2]Los Alamos National Laboratory, Climate, Ecology and Environment, Los Alamos, New Mexico
[*]Corresponding Author: croth@lanl.gov

June 2024

Abstract

There is increasing demand to quickly process multiple types of sequencing-based data to completely capture epigenetic alterations and associated changes in chromatin structure underlying cellular responses. Furthermore, the need for a set of bioinformatic tools that leverage high performance computing and parallelization for processing omics data from many experiments has become apparent. Here we present SLUR(M)-py: a flexible command line tool (written in Python) that leverages the Simple Linux Utility for Resource Management system (SLURM) to process, align, and analyze sequencing data from three-dimensional and epigenomic assays in a high-performance computing environment. SLUR(M)-py is designed with host-pathogen infection experiments in mind, and contains unique scripts and functions that automate calls to SLURM for processing paired-end sequenced reads from chromatin characterization experiments, including whole-genome, ChIP-seq, ATAC-seq and Hi-C. ATAC-seq and Hi-C data from viral infection experiments as well as data from the ENCODE project are utilized to demonstrate processing speed, which outpace current high-performance computing pipelines. We explore the effect of dropping duplicate sequenced reads in ATAC-seq data and demonstrate how SLUR(M)-py can be used for quality control and to detect artifacts in Hi-C experiments from viral infection experiments. Finally, we utilize SLUR(M)-py to explore the dynamics of inter-chromosomal contacts in mammalian cells exposed to vaccinia virus, the vaccine for smallpox.

Keywords

SLURM, Python, High Performance Computing, Parallelization, ATAC-seq, Hi-C

Introduction

Within the nucleus, the shape, looping, and interactions between chromatin are critical to cellular function, development, and for mounting a response to a pathogen or other environmental stimuli [1–14]. A myriad of sequencing technologies, based on pair-end, short read sequencing, are designed to probe various characteristics of chromatin, including shape, expression, and chemical modification. Assays including chromatin conformation capture followed by sequencing (Hi-C) and accessible chromatin by sequencing (ATAC-seq) are used to study the three-dimensional (3D) architecture and the accessibility of DNA within cells, respectively [15–19]. Additionally, chromatin immunoprecipitation (ChIP-seq) assays can probe the genome for specific chemical modifications made to histone tails, indicative of permissive or repressive regions of DNA [20–22]. Together, each of these sequencing paradigms provide complementary information important to researchers when exploring a hypothesis pertaining to epigenetics. Thus, the ability to quickly integrate, characterize and standardize data from these sequencing approaches in epigenomic investigations (including their complete processing, analysis, and visualization) is critical to understand chromatin dynamics during cellular development and responses to external stimuli.

While each genomic sequencing strategy has a specific benefit, caveats, and requirements for analysis, the expected output of most sequencing assays are the same, paired-end sequencing data. From the perspective of a bioinformatician, across experiments and assays, the processing of paired-end sequencing data (trimming of adaptors from reads, removal of low-quality reads—such as reads with repetitive, low-complexity sequence—, alignment to a reference genome, marking and removal of duplicate alignments, filtering of poorly aligned or non-unique alignments, and removal of artifacts) is similar between the aforementioned genomic and epigenomic assays. As such, co-processing these data through a single informatics pipeline can simplify downstream analysis and visualization. Some software tools and pipelines have been written for this purpose. For example, the Juicer pipeline has automated the analysis and generation of interacting chromatin maps (`.hic` files) from paired-end sequencing files to characterize chromatin loops and topological associated domains (TADs) [23, 24]. HiCExplorer and Fan-c offer the research community alternative pipelines for generating Hi-C maps, data analysis, and visualization [25–27]. For epigenomic assays like ATAC- and ChIP-seq, the ENCODE project has published guidelines and recommended software for experiments and subsequent generation of sequencing files (`.bam`) for appropriate, high-quality analysis [18, 22]. Additionally, software such as ATAC-graph provides automated processing, diagnostics, and quality control plots of ATAC-seq data [28]. Investigators will often cobble these tools together with other bioinformatic tools to analyze the myriad of sequencing data that is generated within a single study [29, 30].

Given the low barrier to entry, flexibility, and open-source nature of Python, it is an ideal coding language to process paired-end sequencing reads from epigenomic and chromatin conformation experiments. Several existing pipelines leverage Python but are only geared towards processing a single data type. For example, HiCExplorer and Fan-c are powerful informatic, Pythonic tools for the processing, analysis, and visualization of Hi-C data. While their Pythonic nature makes these tools more accessible, they do not scale well with larger data sets. Using these tools to process large sets of data from human cell lines or model systems can prove problematic, when more than one billion reads are recommended for fully resolved genomes [4, 31].

Across institutions, researchers regularly rely on a high-performance computing environment to quickly process data. Popular high-performance computing environments for data processing within the bioinformatic community include Amazon Cloud Computing and Nextflow [32]. Alternatively, web-based applications have also been developed for the processing of sequencing datum [25, 33–36]. One of the most popular, open-source software tools, designed to manage a high-performance computing environment is the Simple Linux Utility for Resource Management system (SLURM) [37]. SLURM is simple to use and scalable for large computational jobs [38]. Given these features and the user-friendly nature of SLURM, it has become an accepted and integral tool for high performance computing, capable of managing small to large servers. SLURM empowers users to submit and run parallel jobs while also establishing dependencies, allowing for the creation of truly end-to-end, automated pipelines [38]. For example, the Juicer pipeline leverages SLURM to manage, submit, and run parallel processes for generating `.hic` files from Hi-C experiments [23]. Additionally, the HiC-Pro pipeline contains a parallel mode which can submit jobs to SLURM to create and run parallel bioinformatic tasks [39]. To our knowledge, there is currently no other published high-performance computing enabled pipeline for the processing and analysis of most other sequencing data from chromatin characterization experiments such as whole-genome, ChIP-seq, ATAC-seq, or RNA-seq.

Here we present SLUR(M)-py (pronounced slurpy; the M is silent), a Python-based pipeline designed for a high-performance computing environment that processes the alignments, filtering, and analysis of various types of paired-end sequencing data. This Pythonic pipeline auto-generates and submits scripts to SLURM. SLURM then schedules and parallelizes bioinformatic processes underlying the analysis of omics data, greatly improving the speed of common bioinformatic tasks. The novelty of SLUR(M)-py is its ability to consume paired-end sequencing data from several different sequencing strategies, including whole-genome, ATAC-, ChIP-, and Hi-C sequencing. SLUR(M)-py also generates quality control metrics and useful images/graphs for further analysis and publication.

Using the SLUR(M)-py platform, we reanalyzed ATAC-seq and Hi-C data from our previous publications [40, 41] as well as Hi-C data from the ENCODE project [42], and compared the outputs generated by SLUR(M)-py to previous in silico efforts. Reanalysis with SLUR(M)-py improved computational speeds,

outpacing our previous informatic analysis. Moreover, the data products produced from the reanalysis with SLUR(M)-py qualitatively and quantitatively compare to previous outputs made using the Juicer pipeline.

Through our reanalysis, we also demonstrate SLUR(M)-py's utility in viral infection experiments. In one of our most recent studies, we infected kidney cells from the African Green Monkey (VERO cell line) with the double stranded, cytoplasmic DNA virus, vaccinia virus [41]. Hi-C and ATAC-seq experiments were conducted in a paired manner to measure changes in chromatin organization and accessibility, respectively across a 24-hour time-course. Outputs from SLUR(M)-py allow for the interrogation of contacts between chromosomes as a function of viral infection. The contacts between chromosomes do not appear altered as a function of viral infection, with the greatest fold change in contacts between chromosomes occurring at 24 hours post infection. The viral infection data is also used to demonstrate additional tools democratized within the SLUR(M)-py pipeline, including compartment correction and contact distance decay estimation for Hi-C data.

Results

Pipeline overview

SLUR(M)-py combines the best of currently available bioinformatic tools and provides parallelization for quickly aligning and processing sequenced reads for studying chromatin structural dynamics (Table 1). These include MACS2 [43–45], fastp [46], bwa [47], samtools and pysam [48], samblaster [49], bio-Python [50], Pandas and Dask Dataframes [51]. These tools were specifically chosen for their speed, efficiency, community familiarity, and capabilities. These tools are combined to form three central scripts to process whole-genome sequencing, ChIP-seq, ATAC-seq, and Hi-C data types. Each of the three scripts within SLUR(M)-py only require a set of paired-end reads and a path to a reference file (in `.fasta` format) that has been indexed via bwa (Figure 1). SLUR(M)-py will attempt to index the input reference file if no associated bwa index file(s) are initially detected. Like the Juicer pipeline, SLUR(M)-py encodes submissions of scripts to SLURM to efficiently manage parallelized processes within the pipeline. However, SLUR(M)-py includes additional scripts to process information that is complementary to Hi-C including ATAC-seq and ChIP-seq data (Figure 1A and Figure 1B). Quality control and diagnostic plots are also automatically generated when whole-genome sequencing, ATAC-seq, ChIP-seq, or Hi-C datasets are provided.

The computing environment and dependencies SLUR(M)-py requires are listed on the hosting Github repository and can be easily installed via Python's package, dependencies, and environment manager, *conda* [52]. When installing (and running SLUR(M)-py) on a computing cluster, no specific node type is required. Scripts generated by SLUR(M)-py and submitted to SLURM are node agnostic and do not require a graphical processing unit (gpu) to perform tasks. By default, SLUR(M)-py will instruct SLURM to submit jobs to terabyte (tb) node(s); however, any type of node may be specified with the `-P` argument in calls to the SLUR(M) scripts. After installation, SLUR(M)-py can be linked within a given project directory (Figure 2A). This project directory contains a sub directory named "fastqs" which holds the paired fastq files (in gzipped compressed format [53]) from a sequencing experiment. Multiple replicates (i.e., more than one pair of fastq files) may be held within this directory. Unlike other pipelines (for example see Table 1), the SLUR(M)-py workflow can merge replicates (by adding the "`--merge`" flag at the command line) or keep replicates separate throughout processing, which is the default behavior.

To begin a run, the three alignment scripts within SLUR(M)-py require, 1) a path to a reference file in `.fa` or `.fasta` format (set by the `-r` argument) and, 2) the presence of gzipped fastq files (fastq.gz) within the fastqs subdirectory. The basic alignment script in SLUR(M)-py, `wgs.py`, (Figure 2B) utilizes the `.fastq` preprocessing software fastp to quickly split input read pairs into smaller sets. At this stage, low quality reads may be filtered and removed from processing. This subprocess may be skipped with the addition of

the "`--skip-fastp`" argument. The number of splits generated by fastp is controlled via the optional "`-F`" argument (default = 64). For each split formed by fastp, the *bwa mem* algorithm is used to align paired-end reads. Parallelized runs of bwa are called using SLURM, greatly reducing overall computational load and run times. Post alignment, read pairs are split into four categories based on their alignment patterns; those paired reads (1) both mapping to the target genome, (2) pairs of reads with only one read mapping to the genome (and the other unmapped) (3) those pairs of reads with one or both reads mapping to the mitochondrial sequence, and (4) unmapped read pairs. These sets are then combined across splits into singular files for each category. The basic SLUR(M)-py procedure ends with the counting of reads within these files and the removal of temporary files. The final output of this process is a binary alignment map file (`.bam`).

In the event of SLURM crashing, node failure, or an error, SLUR(M)-py contains a check pointing system within each of the workflows. This feature was included to save data generated at intermittent steps within the pipeline(s) and allow for restarts at steps within the individual protocols. For example, when attempting to process whole-genome sequencing data with the `wgs.py` protocol (Figure 2B), after splitting input read pairs with fastp—which is arguably one of the most time-consuming steps within any of the SLUR(M)-py scripts—users may need to restart alignments with bwa. This can be done without again splitting input read pairs by passing "`-R bwa`", which would direct a SLUR(M)-py script (in this instance the `wgs.py` script) to pick up at the step of alignment. In the event of an error, the SLUR(M)-py tools also include a "checkwork" function to identify sources of errors saved within the debug directory (Figure 2E). For completely restarting runs, all the scripts within SLUR(M)-py can be passed the "`--restart`" flag should users wish to completely erase and reset an attempt. After completing runs of with SLUR(M)-py, a "clean" functionality is included within each script to remove large intermittent and temporary files generated during processing and then compress the final, large text files and outputs with the gzip command.

ATAC-seq and ChIP-seq scripts

As stated, SLUR(M)-py was initially generated for the analysis of sequencing data from experiments such as ATAC-seq and ChIP-seq (Figure 2B and 2C). For this purpose, we have developed the `peaks.py` script within SLUR(M)-py (Figure 2C). This script is like the wgs.py process but includes additional steps unique to ATAC-seq experiments. In addition to performing parallelized alignments on subsets of reads, `peaks.py` includes marking of duplicate alignments (via samblaster), filtering of alignments—further processing those that pass mapping quality thresholds (default map-q = 30)—and detection of loci enriched for mapped reads (i.e., peak calling) via MACS2. This script generates a filtered `.bam` file as well as peak files from MACS2. ATAC-seq peak calling is generally performed without an input control. Adding an input control file, in `.bam` format, to the call of the `peaks.py` script will initiate the ChIP-seq mode (Figure 2C).

Hi-C script

The final script (named hic.py) within SLUR(M)-py is designed to process sequenced, paired-end reads from Hi-C experiments (Figure 2D). Given the underlying chemistry and preparation of Hi-C sequencing, the fragments from these experiments require additional, unique processing [31]. Pandas and Dask Dataframes are leveraged to quickly process alignments (in Python) after performing splitting and mapping with fastp and bwa, respectively. Briefly, alignments are prefiltered, separating alignments into sets of read pairs mapping to the genome, going unmapped, and mapping to the mitochondria; the latter two groups are set aside from analysis. The read pairs mapping to the reference genome are further formatted into properly mapping Hi-C contacts. These are filtered to remove those read pairs with dangling ends (after enzymatic digestion), that present as self-circles, or are the likely result of ligation errors [31]. After concatenating files, the pairs-text file with Hi-C contacts is sorted (from left to right) by genomic position. At this stage, the users also have the option to mark and remove duplicates. The final output of the `hic.py` script is a pairs-text file that is compatible with the suite of tools from the Juicer pipeline (to form a `.hic` file) or HiCExplorer (to generate a `.mcool` or `.cool` file [54]). After running any of the three SLUR(M)-py scripts, the results are recorded within the "aligned" directory (Figure 2E), which contains a `.bam` (for the `wgs.py` and `peaks.py` scripts) or `.txt` (from `hic.py`) file.

Table 1: Processing features within SLUR(M)-py and published Hi-C pipelines

	SLUR(M)-py	Juicer [23]	HiCExplorer [27]	Fan-c [26]	HiC-Pro [39]
Multi-omic	Yes	No	No	No	No
SLURM management	Yes	Yes	No	No	Yes
Pythonic basis & filtering	Yes	No	Yes	Yes	Yes
Replicate separating	Yes	No	No	No	Yes
Arima compatible	Yes	Yes	Yes	No	Yes
Dovetail compatible	Yes	Yes	No	No	Yes
Enzymatic libraries[a]	Yes	Yes	Yes	Yes	Yes
Calls *bwa mem* with the `-5SPM` option[b]	Yes	Yes	No	Yes	No

[a] Restriction site based libraries: MboI, DpnII, Sau3AI, and HindIII

[b] The `-5SPM` option sets the *bwa mem* command to ignore pairing, skip mate rescue, mark shorter split hits as secondary, and take the alignment with the smallest coordinate as the primary alignment. Often referred to as the "Hi-C option".

Automation of diagnostic plots

Upon successful completion of a run, a diagnostics directory is also generated which contains a timestamp (listing the start, end, and total run time) and a visualization (and `.csv` file) of the read alignment summary (Figure 3A). Additionally, for ATAC-seq (and ChIP-seq experiments), the fragment distributions (Figure 3B), binned genomic ranks (Figure 3C), and FrIP scores [18] (Supplementary Table S1) are also automatically calculated and reported to users within the same diagnostics directory.

Runtime analysis

To test the run time efficiency of SLUR(M)-py, we processed both ATAC-seq and Hi-C data from multiple studies [40, 41]. Experiments from Venu et al. (2024) in VERO cells include paired ATAC-seq and Hi-C samples taken at 12, 18, and 24 hours post infection with vaccinia virus. In Roth et al. (2023), ATAC-seq assays from A549 cells for several biological replicates ($n = 8$) were collected under nominal laboratory conditions (no perturbation or viral infection). Additional Hi-C data in human cell lines were gathered from the public ENCODE project and repository [42] in the form of raw fastqs and similarly reprocessed using the SLUR(M)-py pipeline. In total, these previously published data sets provided us with twenty-six ATAC-seq (Supplementary Table 1) and twenty-one Hi-C (Supplementary Table 2) samples for processing.

For ATAC-seq samples processed with the `peaks.py` script, the average run times for samples from the A549 simulation study ($n = 8$) and vaccinia virus infection experiments in VERO cells ($n = 18$) were 28 and 40 minutes, respectively, with three datasets taking a little over an hour to completely process the raw paired fastq.gz files into a binary alignment map (`.bam`) files (Supplementary Table 1). Analysis of Hi-C data ($n = 21$) was also relatively quick, finishing in less than 16 hours across all the data analyzed here (Supplementary Table 2). The run times of both ATAC-seq and Hi-C were linear on a $\log_2$ scale with respect to the number of paired-end reads (Figure 4A and Figure 4B, respectively). Indeed, modeling of SLUR(M)-py run time (as a function of counts of input read pairs) explains approximately 83.6 and 81.3% of the variation in run time for ATAC-seq and Hi-C sequencing experiments, respectively (p-value $< 10^{-7}$, Linear Regression).

Comparing Hi-C processing

Because the Juicer pipeline and our SLUR(M)-py Hi-C script both use SLURM to possess Hi-C data, we wanted to compare our strategy for Hi-C processing against the current standard. Specifically, we compared the run-time from the Juicer pipeline (running with an early exit before .hic file creation) to the run times of the SLUR(M)-py Hi-C pipeline using the experiments in the VERO cell line from Venu et al. (2024). Across

samples, all the runs with the Juicer pipeline took longer than 24 hours to converge to a pairs-txt file while all the runs using the hic.py script in SLUR(M)-py finished in less than 16 hours (Figure 4C). This increase in computational speed, and reduction in run time in SLUR(M)-py can be attributed to using fast software for read splitting (fastp) and a larger set of initial splits made on input read pairs, which decreased overall memory requirements and increased the number of parallel processes. Also, unlike the Juicer pipeline, tasks coded from SLUR(M)-py scripts do not require specific nodes (i.e. gpu nodes) for processing and alignments, allowing scripts to run across any available nodes.

After running both the Juicer and SLUR(M)-py Hi-C pipelines on the VERO Hi-C data, we generated .hic files using the processed results from these pipelines and the `pre` command from the suite of Juicer tools [23]. The Hi-C spector reproducibility score [55] was then used to quantify and compare the similarity between SLUR(M)-py v.s. Juicer processed Hi-C data. Overall, the spector reproducibility scores calculated between Hi-C data processed by SLUR(M)-py and Juicer are biologically reproducible, with a genomic median score ranging from 0.944 to 0.976 across samples ($n = 12$). Across these Hi-C samples, most individual chromosomes displayed reproducibility scores higher than 0.80, signifying high reproducibility between SLUR(M)-py and Juicer per chromosome [55].

For two samples (both infected with vaccinia virus), a single chromosome, chromosome 19 (NC_023660.1), displayed scores below 0.80 (Supplementary Figure S1). Examination of read pairs processed by SLUR(M)-py and juicer revealed that Juicer retains reads that are chimeric (mapping to more than one chromosome in a single read) whose mate goes unmapped, while the Hi-C script in SLUR(M)-py drops chimeric mapping read(s), with an unmapped mate. This observation and difference in processing, along with the infection of vaccinia virus, lead to the low reproducibility scores for this chromosome in these samples. We discuss the detection of ligation events within Hi-C assays between the contig representing the vaccinia virus and the Vero genome later within this manuscript. Overall, visual inspection of Hi-C maps confirms the high reproducibility between results generated by our SLUR(M)-py pipeline and the Juicer pipeline (Figure 5).

Duplicate marking

During initial library preparation, PCR amplification can lead to the propagation of duplicate sequenced fragments. Alternatively, a single amplification cluster can be incorrectly identified as multiple clusters by the optical sensor, leading to the creation of optical duplicates. Most current practices suggest that duplicate fragments should be identified and removed to eliminate bias and the inclusion of artifacts in downstream analyses [18, 22, 23, 40, 48, 56]. However, for some omics analyses, marking duplicates may be unwarranted. For example, during genetic variant calling from whole genome sequencing, a recent study has

suggested that marking and removing duplicates is memory intensive and unnecessary [57]. Similarly, another study examining duplicates within ATAC-seq data discovered that many duplicate fragments (identified by position) are not always the result of PCR amplification but true sequenced fragments created by the same, small open regions being sequenced multiple times [58]. To provide duplicate identification and removal, SLUR(M)-py utilizes samblaster within the `peaks.py` script when analyzing paired-end reads from ATAC-seq (or ChIP-seq) experiments. Given that users may wish to skip this step, duplicate marking and removal is optional within SLUR(M)-py, controlled by the addition of the "`--skip-dedup`" flag at the command line.

To test the effect on processing times and the change in results when skipping duplicate marking and removal during the analysis of ATAC-seq data, we ran several timing tests using the ATAC-seq data (in A549 cell line) from Roth et al. (2023). Skipping duplicate marking and removal reduced the overall processing time of this data ($n = 8$) by approximately 5.30 minutes on average (Figure 6A, p-value < 0.008, Wilcoxon signed-rank test). While this greatly reduced processing times of these experiments, the overall effect on peak counts was unclear. In most samples the number of peaks detected by MACS2 increased when duplicate read pairs were retained (Figure 6B), adding an average (median) of nearly 3,990 peaks to the peak counts across experiments in the A549 cell line. Yet in two (of the eight) A549 samples, fewer peaks were identified when retaining duplicates during processing. The FRiP scores across all samples were unaltered by marking and removing duplicate reads (Figure 6C). A note of caution to users, for ATAC-seq experiments, excluding duplicate marking and removal may reduce the total processing time but affects the number of significant peaks identified via MACS2.

Compartment score correction

For Hi-C data, from `.hic` files, tools such as Fan-c and Juicer can estimate chromosome compartment scores which are used in turn to determine A (open) and B (closed) compartment assignments [31, 41]. For each chromosome, these assignments are usually estimated from the first eigenvector taken from principal component analysis on the Hi-C contacts [31]. Convention defines regions with positive and negative eigenvalues as being from the open (A) and closed (B) chromatin compartments, respectively [4, 15, 31].

Given that there is often more than one orthogonal representation of the principal components, the estimates of compartment scores need to be examined, and in some cases corrected, to properly assign A and B compartments across the genome [25, 26, 41]. Within SLUR(M)-py, we have developed a compartment correction algorithm that utilizes Fan-c, the narrow peak file from MACS2 and an ATAC-seq experiment, and a locally weighted scatterplot smoothing (LOWESS) model [59] to correct and reorient compartment assignments. The underlying logic behind this procedure is that open chromatin regions belonging to the A

(open) compartment should have regions with more ATAC-seq peaks than those in B (closed) compartments. Within SLUR(M)-py, the script `compartmentcorr.py` fits a LOWESS model relating eigenvalues from Fan-c to the number of significant peaks from an input ATAC-seq experiment, at a given genomic resolution, per chromosome (Figure 7A). The last, right-most values from the LOWESS model are examined, and if those are below zero, then they are reoriented, and corrected compartment scores and assignments are returned, via multiplying the input eigenvalues by a negative one (Figure 7B). This procedure is then repeated separately per chromosome by `compartmentcorr.py` (Supplementary Figure S2).

Inter-chromosomal interaction measurements

To broadly quantify the interaction between chromosomes, SLUR(M)-py provides an estimate of the inter-chromosomal contact scores [15, 60]. The inter-chromosomal contact score, as described in [15] Lieberman-Aiden et al. (2009) and Duan et al. (2010), is a broad quantification of the interaction frequency between two chromosomes. Specifically, this score is a ratio calculated by dividing the number of observed contacts between two chromosomes by the expected number of contacts between them, given the total number of inter-chromosomal contacts involving said chromosomes. For visualization of these scores, SLUR(M)-py generates a heatmap using the Seaborn plotting package in python (Figure 8A) [61].

When examining the inter-chromosomal contacts within Hi-C data from Venu et al. (2024), we detected an artifact of the ligation events between the Vaccinia virus and the Green Monkey genome. Designed with host-pathogen infection experiments in mind, SLUR(M)-py can filter out alignments mapping to pathogen contigs, assuming the pathogen contig has been added as an additional contig within the reference genome. These artifacts were present in both the Hi-C and ATAC-seq data, the number of detected vaccinia virus fragments was proportional to the number of sequenced reads (p-value < 0.05, $R^2 = 0.923$, Supplementary Figure S3). Within the Hi-C experiments, these fragments make up a small portion (less than one percent) of the ligation events and only appeared in assays from infected cells (Supplementary Figure S4). Read pairs mapping to the vaccinia virus contig are recoverable from ATAC-seq experiments. However, no read pairs were identified that span vaccinia virus and genome in this assay.

The apparent ligation between chromosomes and virus within the Hi-C experiment was further explored. Specifically, we examined the anchoring points of paired reads, isolating pairs with one read mapping to the virus contig and the other belonging to the Green Monkey genome. In depicting the mapping of these coordinates, we discovered that the supposed ligation events between the virus and genome are uniformly distributed (Supplementary Figure S5). Furthermore, across chromosomes, the proportions of paired reads connecting the virus and genome are linear with respect to the size of the chromosome (p-value $< 2^{-10}$,

$R^2 = 0.92$, Supplementary Figure S6). These observations, along with historic, experimental evidence that vaccinia virus does not enter the nucleus, led us to conclude that these observed ligation events are false positive, inter-chromosomal contacts. From these events, we can predict a lower bound for the number of false ligation events in our experiments, making up approximately 0.345 percent of observed inter-chromosomal ligations (Supplementary Figure S7A). Furthermore, these false positive events were independent from the number of sequenced reads within Hi-C experiments (p-value > 0.05, Supplementary Figure S7B).

Broadly, the number of ligation events between chromosomes was approximately 10% (of the total ligation events) within each Hi-C experiment from the VERO cell line (Supplementary Figure S4). This did not significantly vary between samples infected with the vaccinia virus and controls. Here we performed additional, deeper analysis of the interactions captured between chromosomes using a combination of the output from our SLUR(M)-py, Hi-C script and PyDESeq2 [62] The contacts between chromosomes were quantified by generating contiguous bins, 500 kb in size, left to right, along each chromosome per sample, generating 841,610 unique, inter-chromosomal bins (Figure 8B). The number of contacts between pairs of chromosomes were counted per bin and used as inputs into PyDESeq2 to search for differentially altered inter-chromosomal contacts due to vaccinia virus infection.

Collapsing across time, we observed poor separation of infected and control samples when performing principal component analysis on the binned inter-chromosomal counts (Supplementary Figure S8A). Similarly, across the infection time-course only 66 of 841,610 bins (less than one percent) formed between interacting chromosomes were identified as significantly differential (adjusted p-value < 0.001, absolute fold-change > 1, Supplemental Figure S8A). Performing linear contrasts and examining differential inter-chromosomal contacts between chromosomes at each hour post infection (hpi), we saw no effect at 12 hpi (Supplementary Figure S8B), a minimal effect at 18 hpi (Supplementary Figure S8C), and the greatest separation of infected and control samples at 24 hpi (Supplementary Figure S8D). At this final time point (24 hpi), we identified only a small number of inter-chromosomal regions (13 and 138) with significant control and infected biased contacts (respectively) between chromosomes (p-value < 0.001, absolute fold-change > 1, Figure 8C, Supplementary Figure S8D). This result, the identification of only a small number of differential inter-chromosomal regions, suggests that changes in contacts between chromosomes are not a hallmark of vaccinia virus infection.

Conclusions and Future Directions

SLUR(M)-py is a SLURM powered, pythonic pipeline for performing parallel processing of paired-end sequencing profiles prepared from epigenomic experiments. SLUR(M)-py contains protocols and code written in python for analyzing DNA sequences from many of the standard assays used within epigenomic studies. These pythonic scripts allow users to process paired-end sequenced reads from whole-genome, Hi-C, ChIP-, and ATAC-seq assays using just SLUR(M)-py. In this regard, SLUR(M)-py is a multi-omic pipeline, designed to run within a high-performance computing environment, that eliminates the need for researchers to install and maintain multiple pipelines.

Underlying SLUR(M)-py is unique Python code that interacts with the simple Linux utility for resource management system, SLURM. With SLURM, one can parallelize, stack, and schedule dependencies necessary for end-to-end omics analysis. Unlike currently published pipelines, SLUR(M)-py is not dependent upon a gpu node for processing. SLUR(M)-py can schedule jobs to run on any node type integrated within a system managed by SLURM. By default, runs of SLUR(M)-py will seek a tb node. All the results presented within this paper were generated using tb nodes. Our experience has shown that jobs submitted to gpu nodes tend to complete faster, which is expected given the difference between tb and gpu nodes. The flexibility of node choice allows users to run multiple submissions of SLUR(M)-py across several nodes, which is especially beneficial for high performance computing environments with a heterogeneous node architecture.

We developed SLUR(M)-py using the sequencing data from Hi-C and ATAC-seq assays collected from mammalian cells during viral infection experiments [41]. Unlike current high-performance computing bioinformatics pipelines, the names of the contig representing the viral vector, mitochondrial DNA, or sex chromosomes (chrX and chrY for example) can be passed to SLUR(M)-py as additional inputs. This feature allows users to monitor the level of contamination from these sources or to mask their alignments, which are removed early on in processing, saving computational time and cost. Conversely, a list of select contigs from within a reference genome can be passed to SLUR(M)-py to limit analysis to those chromosomes of interest. Furthermore, these features provide flexibility to the SLUR(M)-py workflows which are not restricted to just human systems and cell lines with expected chromosome names such as chr1, chr2, and chrM, representing the first to autosomes and the mitochondrial contig (just to name a few examples). Indeed, SLUR(M)-py is designed to be species agnostic. As an example of this, with SLUR(M)-py both Hi-C and ATAC-seq data were reprocessed from human cell lines (taken from the ENCODE consortium) and the African Green Monkey (*Chlorocebus sabaeus*), VERO cell line, produced by Venu et al. (2024). In both cases SLUR(M)-py auto-generated the metrics for quality control, produced the expected data packages for further analysis, and rendered figures for publication without error.

SLUR(M)-py demonstrates quick run-times relative to other pipelines. To our knowledge, Juicer is the only other Hi-C workflow that programs and submits tasks to SLURM. For comparison, we reanalyzed our previously published Hi-C data [41], which was initially processed using the Juicer pipeline. While direct comparison is difficult to make (given differences in software, hardware architecture, and pipeline design), the run times of SLUR(M)-py were faster than runs with the Juicer pipeline when reprocessing our data from viral infection experiments. Additionally, across the genome, the Hi-C maps generated with the outputs from the SLUR(M)-py pipeline match those Hi-C maps made using the Juicer pipeline with high reproducibility.

Using SLUR(M)-py we set out to explore the interactions between chromosomes within the VERO cell line. SLUR(M)-py provides scripts for calculating the interaction between chromosomes, and much like human cell lines [15], we see that smaller chromosomes interact more often with each other than with larger chromosomes in *C. sabaeus*. We expanded on this analysis along chromosomes (across 500 kb chromosome bins) to identify regions with differential inter-chromosomal contacts due to viral infection. Across the infection time course, this new analysis showed little difference in inter-chromosomal contacts due to infection with vaccinia virus. The largest difference in inter-chromosomal contact frequencies was seen at 24 hours post infection. Thus, unlike the dynamics in loop structures (due to infection) observed along chromosomes [41], the changes in contacts between chromosomes across time are not associated with viral infection. For other studies seeking to explore inter-chromosomal dynamics and their role in responding to viral infection, we have generated a set of scripts within SLUR(M)-py to aid in this analysis.

When analyzing contacts between chromosomes, we utilized SLUR(M)-py to explore artifacts that appear in those experiments from Venu et al. (2024). Specifically, we identified several ligation events between the contig representing the vaccinia virus and the chromosomes of the Green Monkey genome. The DNA of the vaccinia virus does not translocate into the nucleus, and therefore, its DNA does not directly interact with the host chromosomes [63–65]. It follows that crosslinking between the host and vaccinia genomes after formaldehyde exposure during Hi-C library preparation is impossible. Thus, we determined these events (which are rare) are due to ligation errors produced after crosslinking. We estimated the rate of these false ligation events that occur within Hi-C experiments, is less than one percent. This finding should provide investigators with higher confidence that inter-chromosomal interactions are not dominated by random noise. For viral infection studies, SLUR(M)-py scripts have the option to remove these false contacts. We have also included scripts within SLUR(M)-py for calculating genome-wide distance decay profiles and conducting compartment correction for Hi-C data.

For the ATAC-seq data analyzed here—with an average of 21 million paired-end reads—the average run times were a little over half an hour (36.4 minutes). With these data and short runtimes, we investigated the effect on the average runtime when skipping duplicate marking and removal. The effects on runtimes were

minimal, skipping duplicate marking (and removal) is estimated to save users, on average, a little over five minutes. Given the work put into samblaster (the duplicate marking software used within the ATAC-seq script) there seems little room to improve the speed of this process. However, skipping duplicate filtering led to an increased count of significant loci detected within each ATAC-seq sample (nearly an additional four-thousand peaks). It has been demonstrated that the nature of ATAC-seq leads to an overestimate of duplicate read pairs [58]. Briefly, small accessible regions create a bias in the final set of sequenced regions. While these regions are biologically reproducible, marking duplicates simply by removing fragments with duplicate positions (the traditional method) affects the total end counts of detected significant loci in ATAC-seq data [58]. Currently, the SLUR(M)-py pipeline is unable to retain duplicate read-pairs based on position and is only able to remove them. Future applications will try to estimate those read pairs that are marked as duplicates yet are biologically valid and worthy of retention. Thus, retaining or removing duplicates within a run of SLUR(M)-py we leave up to the users, controlled by the inclusion of the "`--skipdup`" flag at the command line.

In closing, SLUR(M)-py is designed to be a flexible pipeline for quickly processing data from epigenomic experiments. Additional tools and features are currently under development. For example, future versions of SLUR(M)-py will be able to process long read sequences from Oxford Nanopore and Pacbio sequencing [66, 67]. Protocols for processing sequencing data like RNA-seq, CUT&RUN sequencing [68–70], and genetic variant detection are also under development. While not explicitly tested, SLUR(M)-py should be able to process data from single cell experiments too, assuming the input data is in the form of paired-end sequences. SLUR(M)-py is publicly available and hosted on Github and able to be copied, forked, or modified by the scientific community. It is our hope that SLUR(M)-py contributes towards 3D genomic studies of chromatin conformation and architecture.

Methods

Data included, preparation, and processing

The raw data from ATAC-seq and Hi-C sequencing experiments generated from Venu et al. (2024) and used in this study are available in the SRA repository, under the BioProject accession number PRJNA1037174. For details on the viral infection experiments in VERO cells infected with the vaccinia virus see Venu et al. (2024). Using the SLUR(M)-py, pipelines (`peaks.py` and `hic.py`) the paired-end reads from ATAC-seq and Hi-C (respectively) experiments were aligned to the Green Monkey (*Chlorocebus sabaeus*) reference genome (GCF_000409795.2_Chlorocebus_sabeus_1.1_genomic.fasta from NCBI). The sequence of the Vaccinia virus (Accession No. U94848, downloaded from GenBank) was added to this reference genome as an additional contig. This reference file (in fasta file format) was then indexed using bwa. For the VERO data sets (both ATAC-seq and Hi-C) the call to the `peaks.py` and `hic.py` scripts (respectively) within SLUR(M)-py were modified (using the `-G` parameter) to include only the 29 fully assembled autosomes (Chromosomes 1 through 29) and the two sex chromosomes (X and Y) with NCBI contig names NC_023642.1 – NC_023672.1. The calls to `peaks.py` and `hic.py` scripts were also modified to include the removal of reads mapping to the Green Monkey mitochondrial contig (`-M` NC_008066.1). Apart from the inclusion of the `-G` and `-M` parameters, all other default settings in SLUR(M)-py scripts were utilized for analysis of the VERO samples.

Details on the creation, handling, and sequencing of ATAC-seq experiments using the A549 cell line presented here in timing and duplicate marking analysis are included within Roth et al. (2023). Briefly, raw, paired-end sequenced reads from eight biological replicates were downloaded from the sequence read archive from the bioproject PRJNA975595 with accession numbers SRR24717527–SRR24717534. Using the ATAC-seq alignment script (`peaks.py`) with default settings, read pairs per sample were aligned to the telomere-to-telomere human reference genome, version 2 [71].

Other data used

Nine sets of paired-end sequences representing Hi-C data collected on human cell lines were taken from the ENCODE project and data repository [22, 42]. These included experiments conducted within the A549, HepG2, IMR90, and K562 cell lines. Please refer to Supplementary Table 2 for the list of ENCODE accession numbers for each sample. These data were each aligned to the telomere-to-telomere human reference genome, version 2 [71] using the `hic.py` script in SLUR(M)-py with default settings.

Hi-C analysis

For comparative analysis, the Hi-C samples from experiments within the VERO cell line were aligned to the *C. sabaeus* reference genome (modified with the addition of the vaccinia contig (as described above) using the Juicer pipeline [23]. For processing of each replicate with Juicer, the early exit parameter (`-E`) was also added to end the processing of Hi-C contacts after the creation of the merged, sorted no duplicates text file but prior to the creation of an `.hic` file. The overall processing times of these runs were calculated using the time stamps on output logs made within Juicer's debug directory.

Post processing, `.hic` files were generated with the `pre` command from Juicer tools (version = 1.22.01), using both the text-based outputs from Juicer and `hic.py` from SLUR(M)-py workflows as inputs. Default settings of both pipelines (Juicer and `hic.py`) were used, and designed to match as best as possible (for example map-q >= 30). To compare the `.hic` files made using the Juicer pipeline to those made by our `hic.py` script, the spector reproducibility score [55] was calculated for the main autosomes and sex chromosomes at 500 kb resolution across the sampled Hi-C maps ($n = 12$). Runs of SLUR(M)-py auto-generate timestamps and calculate total run time(s). For runs with the Juicer pipeline, we estimated the total run time from start and end timestamps of logs within the Juicer debug output.

Timing and testing of ATAC-seq analysis

For the analysis of processing times when analyzing ATAC-seq and Hi-C data using SLUR(M)-py, the auto-generated timestamps across runs were transformed into total minutes and hours then regressed against the number of input, paired-end reads using a linear model. For testing the effect of duplicate marking and removal on overall runtimes and processing of ATAC-seq samples, the `peaks.py` script within SLUR(M)-py was re-run twice for each sample, with and without marking duplicates, post alignment with bwa by passing the combination of the `-R` parameter with the input "mark" and the `--skip-dedup` flag. Time stamps (autogenerated by the `peaks.py` script) from these runs were brought into Python for calculating the change in processing time. The auto generated diagnostic files, which contain information on the number of peaks and summits detected with MACS2, for each run were similarly analyzed.

Inter-chromosomal analysis

From the raw output from `hic.py`, the number of Hi-C contacts between chromosomes were counted and used to calculate the inter-chromosomal interaction score (IS) following methods from and Lieberman-Aiden et al. (2009) and Duan et al. (2010). Briefly this is calculated for a pair of chromosomes i and j using the following formula: $\mathrm{IS}(i,j) = N_{ij}/((N_i/N)(N_j/N)(N))$. Where N_i and N_j are the number of inter-

chromosomal contacts involving chromosomes i and j (respectively), N_{ij} is the number of inter-chromosomal contacts between chromosomes i and j, and N is the total number of inter-chromosomal contacts [15, 60]. These scores were visualized using Seaborn's heatmap function in Python on a $\log_2$ scale [61].

For visualization of inter-chromosomal contacts, contacts between chromosomes were counted every 250 kb. These inter-chromosomal bins were used in a binomial model from Kaufmann et al. (2015) to identify significant interactions every 250 kb between chromosomes [72]. The Circos, circle plot library [73] in Python was used to plot significant 250 kb bins (binomial test, p-value > 0.001) between chromosomes.

For differential, inter-chromosomal analysis of Vaccinia infected VERO samples, the inter-chromosomal contacts every 250 kb were counted across replicates and samples and compared between vaccinia infected and control Hi-C maps at 12, 18, and 24 hours post infection. Specifically, the counts of binned inter-chromosomal contacts were used as input into PyDESeq2 [62]. Linear contrasts were performed at each time-point post infection. An adjusted, p-value threshold of p-value < 0.001 and absolute fold-change greater than one were used to establish significant differential contacts between chromosomes.

Software availability

The code base for SLUR(M)-py is currently under development and hosted on Github at:
https://github.com/SLUR-m-Py/SLURPY

Acknowledgements

We would like to thank members of the Genomics and Bioanalytics Group and the Biosystems Administrators at Los Alamos National Laboratory for their helpful comments on this manuscript. We would also like to acknowledge and thank the ENCODE Consortium and the members of the Job Dekker lab of University of Massachusetts, the Erez Aiden lab of Baylor University, and the Tim Reddy lab of Duke University for generating the Hi-C data sets used within this study. The work conducted within this paper was funded in part by the Los Alamos National Laboratory Directed Research Grant (20210082DR) awarded to SRS. This material is also based upon work supported by the U.S. Department of Energy, Office of Science, through the Biological and Environmental Research (BER) and the Advanced Scientific Computing Research (ASCR) programs under contract number 89233218CNA000001 to Los Alamos National Laboratory (Triad National Security, LLC).

Author contributions

CR develops and maintains SLUR(M)-py and its associated software. VV tested, proposed user features, provided feedback, and assisted in development of SLUR(M)-py software. CR and VV interpreted and summarized Hi-C results. SB added code, performed debugging, and edits to the SLUR(M)-py pipeline and documentation. All authors provided edits and comments on the manuscript. CRS and SS provided funding and materials for the research presented within this paper. CR performed the analysis and generated figures. CR and SS wrote the manuscript.

Conflicts of interest

The authors declare no conflicts of interest.

Figures

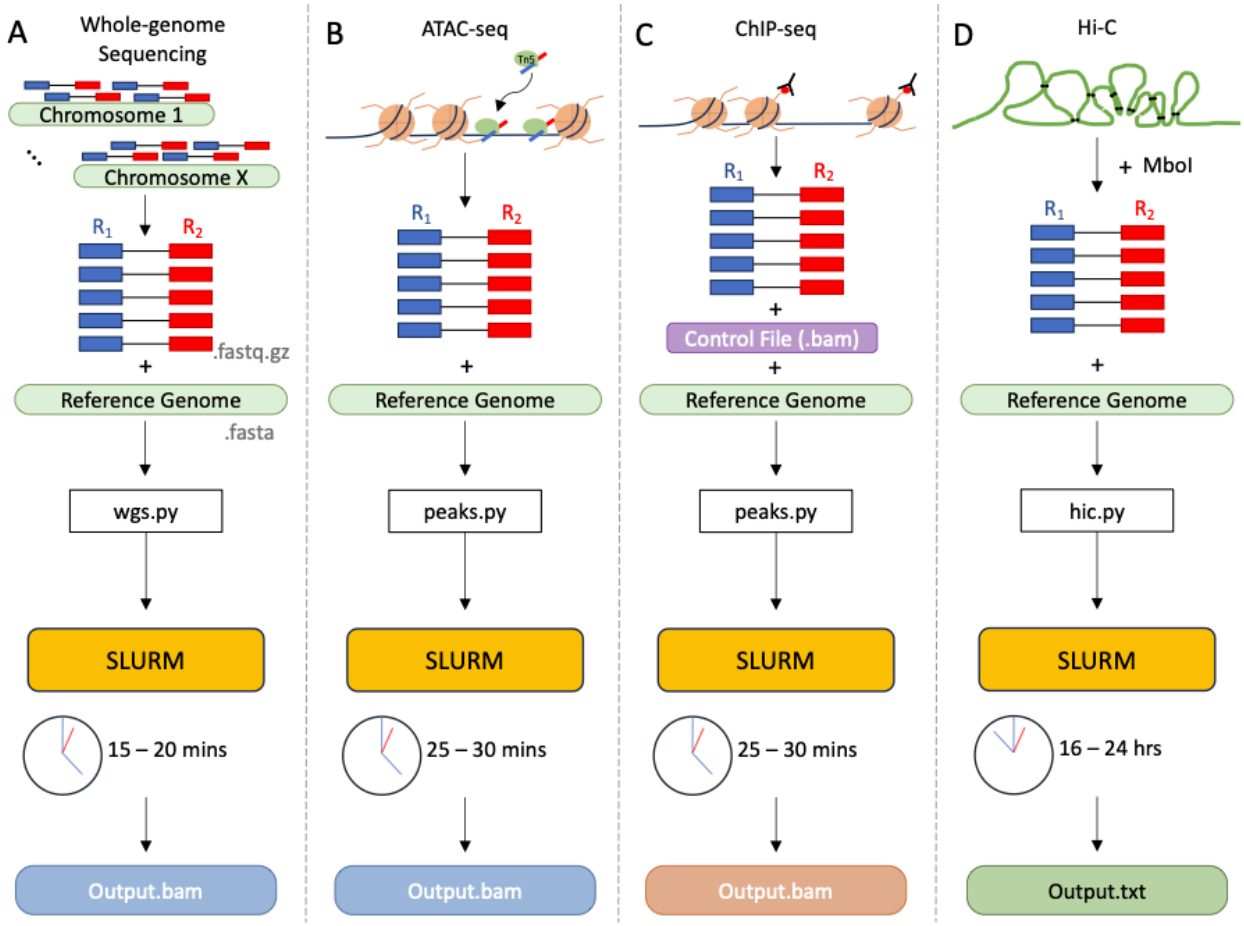

Figure 1: Overview of SLUR(M)-py; a high-performance computing, multi-omic platform for parallelized processing of paired-end reads to study chromatin dynamics. **A**) The basic SLUR(M)-py workflow for processing paired-end, whole-genome sequences. **B**) ATAC-seq targets open regions of the genome with Tn5, generating paired-end sequencing reads. These reads (in `.fastq.gz` format) and a reference genome (`.fasta`) are the minimum requirements passed to SLUR(M)-py's ATAC-seq script (`peaks.py`). From `peaks.py`, jobs are passed to SLURM to parallelize alignments, set dependencies, and que downstream processes, quickly generating output as binary alignment maps (`.bam`). **C**) ChIP-seq experiments target modified histone tails to generate paired-end reads after sequencing. For processing ChIP-seq experiments, the `peaks.py` script requires the addition of a control file (in `.bam` format). **D**) Overview of inputs for the Hi-C script (`hic.py`). Paired-end reads from Hi-C experiments, along with the name of the fragmentation enzyme, and the reference genome are passed to `hic.py` as inputs.

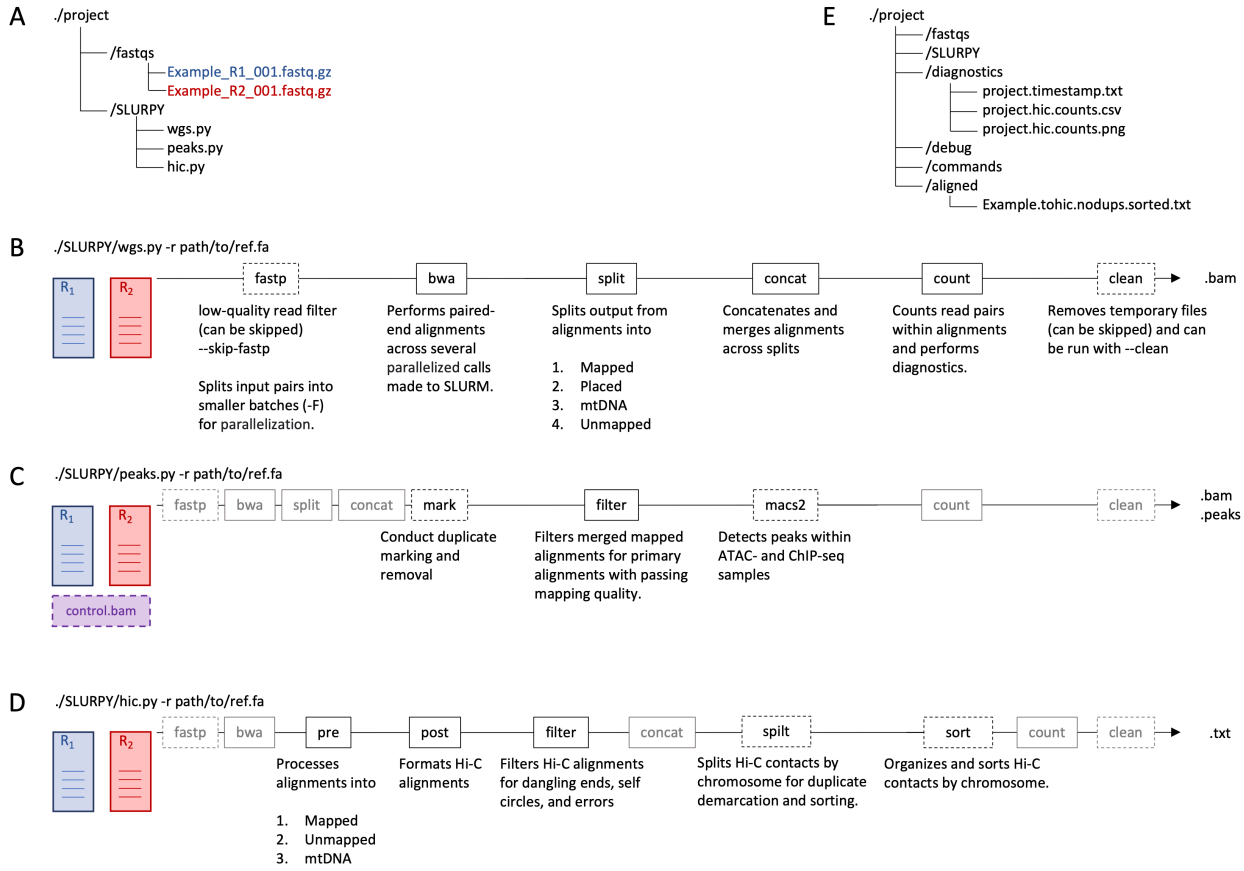

Figure 2: Example project directory and outline of SLUR(M)-py pipeline(s). **A**) Directory structure. Paired `fastq.gz` files (within the fastqs subdirectory) contain paired-end reads as input into SLUR(M)-py. The directory contains all the scripts and functions associated with SLUR(M)-py and can be soft-linked as a subdirectory. **B**) The basic alignment script, `wgs.py`. Each box represents a step within the script; dashed boxes are optional (via adjustment in the call to `wgs.py`). In the basic processing pipeline, paired *fastqs.gz* files are split (with fastp) into smaller sets for parallelization. Fastp conducts initial quality control (QC), removing low-quality reads, and produces diagnostic reports. Across splits of paired `fastqs.gz` files, parallelized calls to bwa align paired reads to a reference genome (set by the `-r` argument). A split function then segregates alignments into files for mapped, placed, mitochondrial (mtDNA) mapping, and unmapped pairs. A concatenation function organizes and combines splits into singular sets for each input. A penultimate counting function provides QC of results. **C**) The Pythonic processing pipeline (`peaks.py`) for ATAC-seq data. Much like the basic processing procedure (shown in **B**) alignments are marked for duplicates, filtered, and fed into MACS2 for peak calling. One or more control file(s) (`.bam`) can be fed as input(s) for ChIP-seq experiments. **D**) A parallelized processing pipeline for Hi-C data. Post splitting and alignment, alignments are sorted (mapped, unmpped, and mtDNA mapping), formatted into Hi-C records, filtered for errors, and duplicates are marked and removed. A pairs-txt file (`.txt`) is generated for further processing. E) Changes and output files within the example project directory (shown in **A**) after running Hi-C analysis.

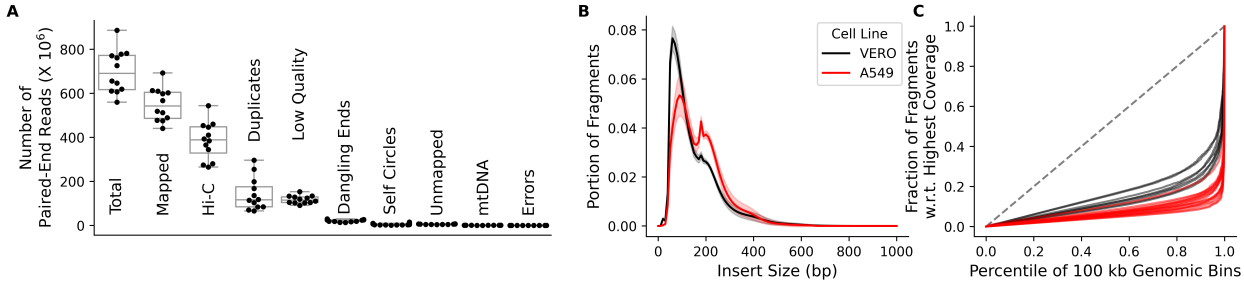

Figure 3: SLUR(M)-py generated diagnostic plots. **A**) Alignment counts plot (generated via `hic.py`) displaying mapping patterns of reads (across the VERO Hi-C samples). From left to right, the total read pairs, those mapped to the genome, valid Hi-C contacts, duplicate read pairs, low quality mappings, and Hi-C errors (dangling ends, self-circles, unmapped, mitochondrial, and ligation errors, left to right). **B**) Fragment distribution and **C**) ranked genomic-bin coverage plots autogenerated by SLUR(M)-py pipeline. Black and red colors depict samples from VERO and A549 cell lines. Shaded regions in **B** show 90% confidence intervals of the mean (solid curves).

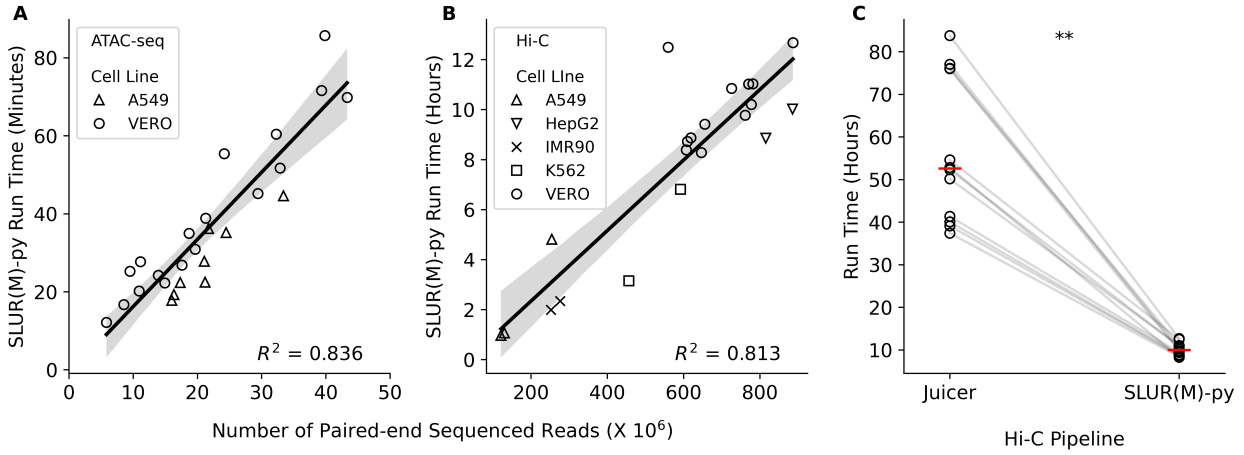

Figure 4: SLUR(M)-py run time analysis. **A**) The run times (y-axis, minutes) of VERO and A549 ATAC-seq samples (circles and triangles, respectively) as a function of the number of paired-end reads (x-axis). **B**) The run times (y-axis, hours) of Hi-C samples in VERO (circles) and human cell lines (x's, triangles, and squares). In **A** and **B**, a black line and gray shaded region represent a linear regression model and its 95% confidence interval (respectively) relating the run time of the atac.py and hic.py scripts (**A** and **B**, respectively) to the number of input reads; these models explain 83.6% and 81.3% (R^2) of the variation in run time (**A** and **B**, respectively). **C**) Distribution of run times (y-axis, hours) when processing VERO Hi-C data from Venu et al. (2024) with Juicer and SLUR(M)-py (x-axis) pipelines. Red horizontal lines mark the mean run times, black lines connect samples (** p-value < 0.001, Wilcoxon signed-rank test).

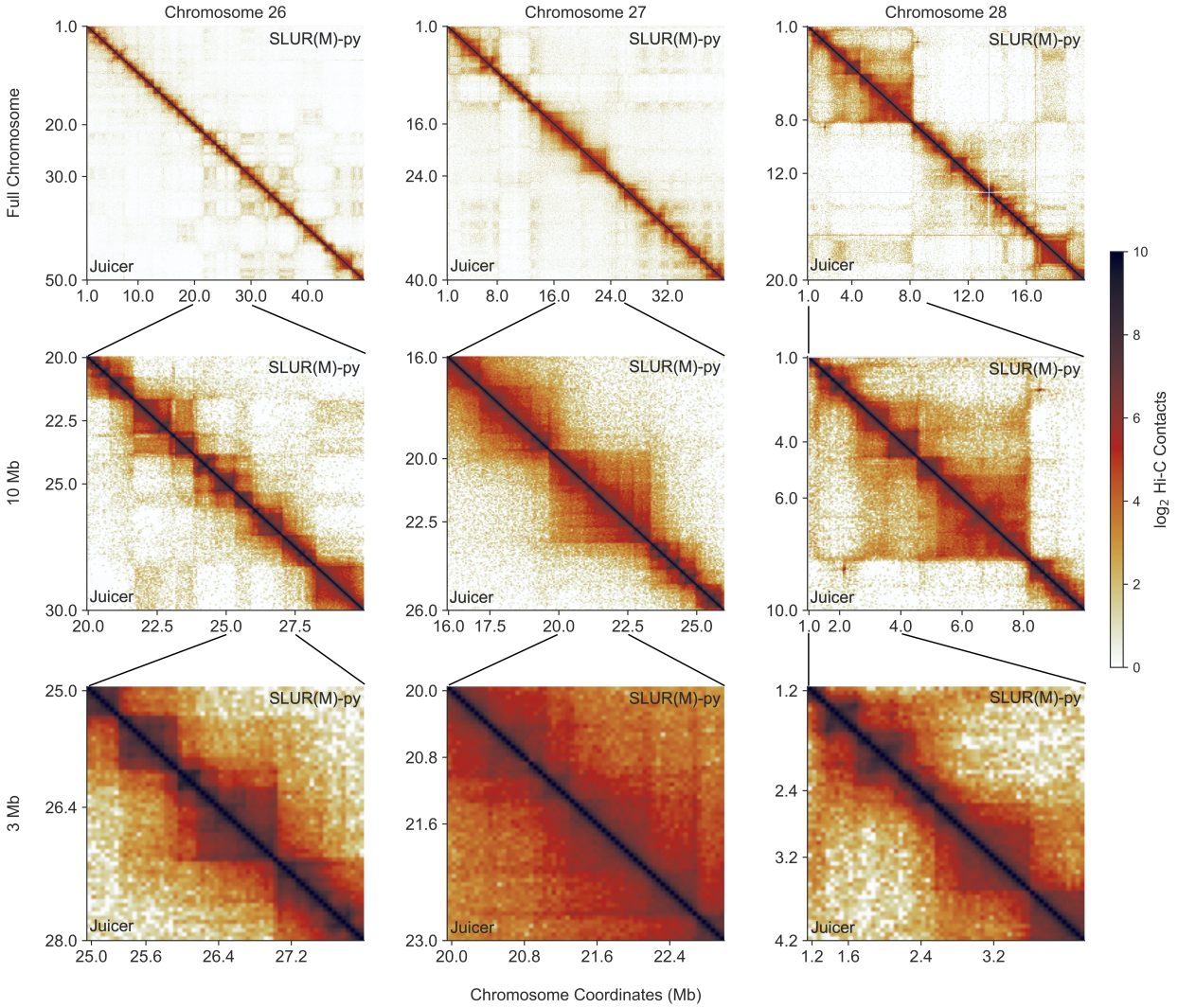

Figure 5: Comparison of Hi-C maps generated by Juicer and SLUR(M)-py pipelines. Example Hi-C maps for chromosomes 26, 27, and 28 (left to right, respectively) of the VERO genome, at scales of the full chromosome, 10 Mb, and 3 Mb (top to bottom, respectively). The Hi-C maps made from Juicer or SLUR(M)-py appear on the lower and upper diagonals, respectively. Yellow to black colors depict increasing contact frequency ($\log_2$) along the chromosomes.

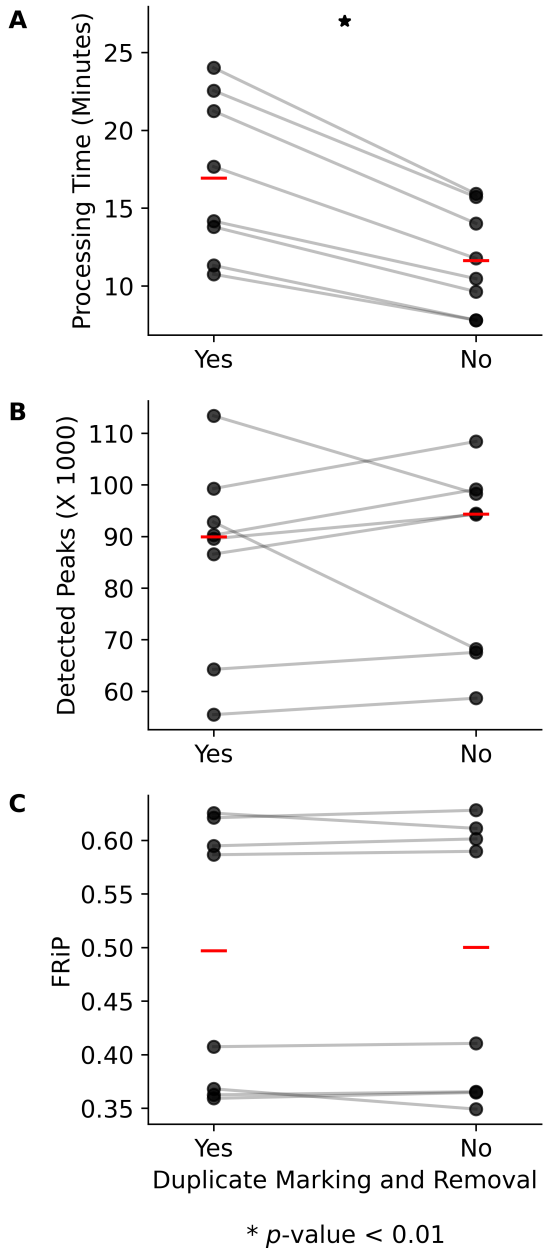

Figure 6: The effect of marking and removing duplicates on processing times, peak counts, and FRiP scores across ATAC-seq experiments. **A**) The number of minutes (y-axis, post alignment) for marking and removing duplicates, filtering alignments, and calling MACS2 across ATAC-seq experiments in the A549 cell line (* p-value < 0.01, Wilcoxon signed-rank test). **B**) The number of detected peaks (via MACS2) in each ATAC-seq experiment (y-axis). **C**) The fraction of reads in peaks (FRiP, y-axis) calculated by SLUR(M)-py. The x-axis is shared in each panel and denotes the change in processing time, number of detected peaks, or FRiP (**A**, **B**, or **C** respectively) when including or excluding duplicate marking and removal. Red horizontal lines mark the average of processing times, detected peaks, and FRiP scores; the harmonic mean was used in **C**.

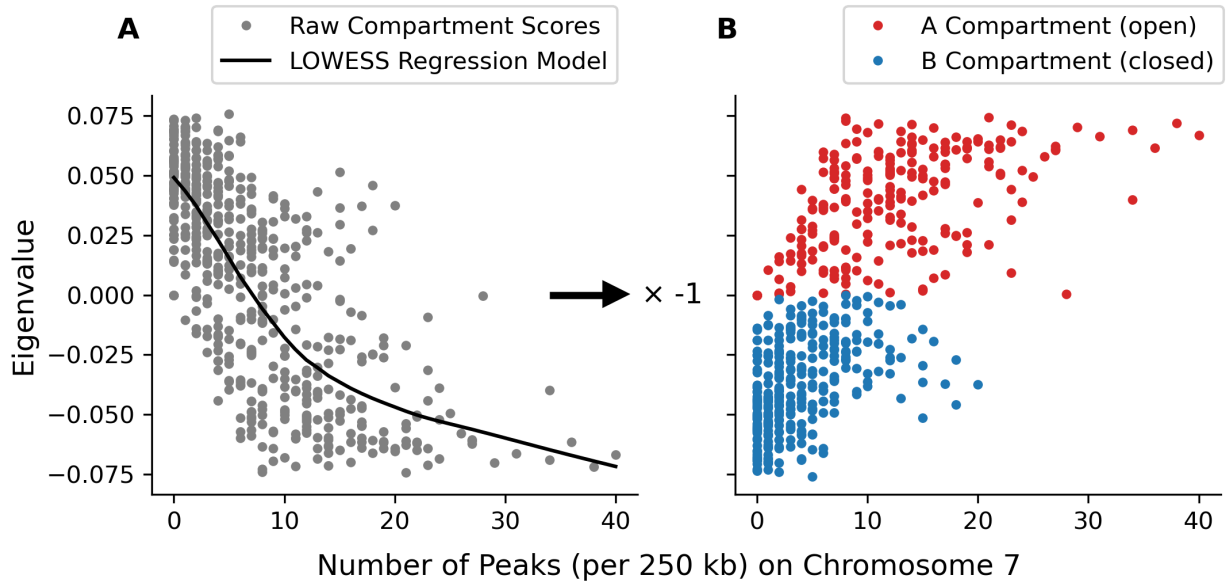

Figure 7: Compartment score correction with LOWESS regression. **A**) Raw compartment scores (gray) estimated by the first eigenvalue from PCA (y-axis) on Hi-C map(s) for chromosome 7 plotted against the number of detected peaks from an associated ATAC-seq experiment (x-axis). A black line represents a Locally Weighted Scatterplot Smoothing (LOWESS) model relating the eigenvalues to peak counts. **B**) The correction of compartment scores (multiplying by -1), implying that regions that are within the A compartments (red) tend to have more open chromatin (larger values on the x-axis) compared to closed regions in the B compartment (blue).

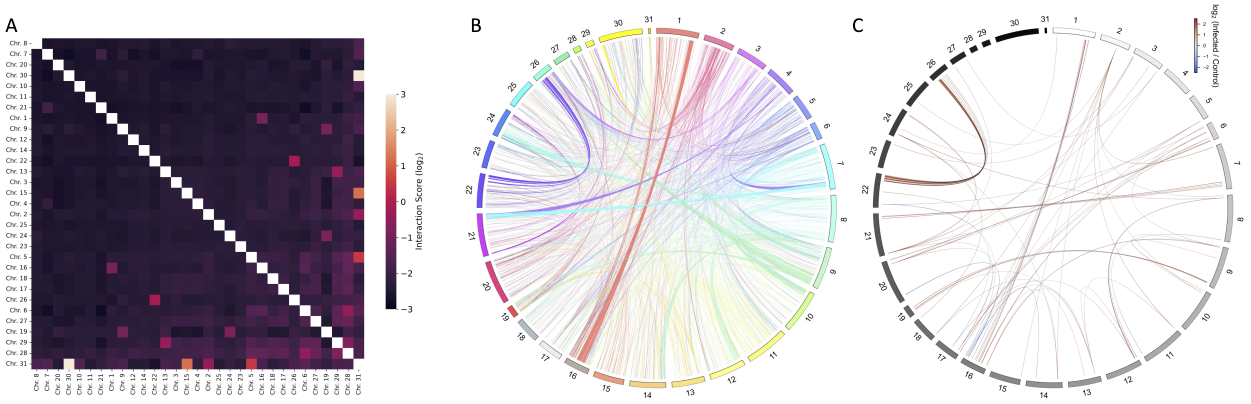

Figure 8: Inter-chromosomal analysis automated within the SLUR(M)-py pipeline. **A**) The interaction scores ($\log_2$) between chromosomes (x- and y-axis, sorted longest to shortest, left to right) are measured as the ratio of contacts between a pair of chromosomes and the expected number of inter-chromosomal contacts of those chromosomes. **B**) Significant inter-chromosomal interactions between VERO chromosomes. Colored arcs represent 250 kb regions with a significant number of Hi-C contacts between chromosomes as measured via a genome-wide binomial model. **C**) The fold change of inter-chromosomal interactions with significant differences between control Hi-C map and the Hi-C map from VERO cells post 24 hours of exposure to the vaccinia virus.

References

[1] Ferrari R, et al. Epigenetic reprogramming by adenovirus e1a. Science. 2008;321(5892):1086–1088.

[2] Kleinjan DA, et al. Long-range gene control and genetic disease. Advances in Genetics. 2008;61:339–388.

[3] Turner BM. Epigenetic responses to environmental change and their evolutionary implications. Philosophical Transactions of the Royal Society B: Biological Sciences. 2009;364(1534):3403–3418.

[4] Rao SS, et al. A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. Cell. 2014;159(7):1665–1680.

[5] Teferi WM, et al. The vaccinia virus K7 protein promotes histone methylation associated with heterochromatin formation. PLoS One. 2017;12(3):e0173056.

[6] Menachery VD, et al. MERS-CoV and H5N1 influenza virus antagonize antigen presentation by altering the epigenetic landscape. Proceedings of the National Academy of Sciences. 2018;115(5):E1012–E1021.

[7] Zheng H, et al. The role of 3D genome organization in development and cell differentiation. Nature Reviews Molecular Cell Biology. 2019;20(9):535–550.

[8] Tsai K, et al. Epigenetic and epitranscriptomic regulation of viral replication. Nature Reviews Microbiology. 2020;18(10):559–570.

[9] Oudelaar AM, et al. The relationship between genome structure and function. Nature Reviews Genetics. 2021;22(3):154–168.

[10] Balnis J, et al. Persistent blood DNA methylation changes one year after SARS-CoV-2 infection. Clinical Epigenetics. 2022;14(1):94.

[11] Loda A, et al. Gene regulation in time and space during X-chromosome inactivation. Nature Reviews Molecular Cell Biology. 2022;23(4):231–249.

[12] Kee J, et al. SARS-CoV-2 disrupts host epigenetic regulation via histone mimicry. Nature. 2022;610(7931):381–388.

[13] Wang R, et al. SARS-CoV-2 restructures host chromatin architecture. Nature Microbiology. 2023;8(4):679–694.

[14] Dekker J, et al. Spatial and temporal organization of the genome: Current state and future aims of the 4D nucleome project. Molecular Cell. 2023;.

[15] Lieberman-Aiden E, et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. Science. 2009;326(5950):289–293.

[16] Buenrostro JD, et al. ATAC-seq: a method for assaying chromatin accessibility genome-wide. Current Protocols in Molecular Biology. 2015;109(1):21–29.

[17] Paulsen J, et al. Chrom3D: three-dimensional genome modeling from Hi-C and nuclear lamin-genome contacts. Genome Biology. 2017;18:1–15.

[18] Yan F, et al. From reads to insight: a hitchhiker's guide to ATAC-seq data analysis. Genome Biology. 2020;21:1–16.

[19] Reiff SB, et al. The 4D Nucleome Data Portal as a resource for searching and visualizing curated nucleomics data. Nature Communications. 2022;13(1):2365.

[20] Barski A, et al. High-resolution profiling of histone methylations in the human genome. Cell. 2007;129(4):823–837.

[21] Barski A, et al. Genomic location analysis by ChIP-Seq. Journal of Cellular Biochemistry. 2009;107(1):11–18.

[22] Landt SG, et al. ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia. Genome Research. 2012;22(9):1813–1831.

[23] Durand NC, et al. Juicer provides a one-click system for analyzing loop-resolution Hi-C experiments. Cell Systems. 2016;3(1):95–98.

[24] Robinson JT, et al. Juicebox. js provides a cloud-based visualization system for Hi-C data. Cell Systems. 2018;6(2):256–258.

[25] Wolff J, et al. Galaxy HiCExplorer 3: a web server for reproducible Hi-C, capture Hi-C and single-cell Hi-C data analysis, quality control and visualization. Nucleic Acids Research. 2020;48(W1):W177–W184.

[26] Kruse K, et al. FAN-C: a feature-rich framework for the analysis and visualisation of chromosome conformation capture data. Genome Biology. 2020;21:1–19.

[27] Wolff J, et al. Loop detection using Hi-C data with HiCExplorer. Gigascience. 2022;11:giac061.

[28] Lu RJH, et al. ATACgraph: profiling genome-wide chromatin accessibility from ATAC-seq. Frontiers in Genetics. 2021;11:618478.

[29] Ramírez F, et al. deepTools: a flexible platform for exploring deep-sequencing data. Nucleic Acids Research. 2014;42(W1):W187–W191.

[30] Lopez-Delisle L, et al. pyGenomeTracks: reproducible plots for multivariate genomic datasets. Bioinformatics. 2021;37(3):422–423.

[31] Lajoie BR, et al. The Hitchhiker's guide to Hi-C analysis: practical guidelines. Methods. 2015;72:65–75.

[32] Di Tommaso P, et al. Nextflow enables reproducible computational workflows. Nature Biotechnology. 2017;35(4):316–319.

[33] Ramírez F, et al. deepTools2: a next generation web server for deep-sequencing data analysis. Nucleic Acids Research. 2016;44(Web Server issue):W160.

[34] Li PE, et al. Enabling the democratization of the genomics revolution with a fully integrated web-based bioinformatics platform. Nucleic Acids Research. 2017;45(1):67–80.

[35] Kerpedjiev P, et al. HiGlass: web-based visual exploration and analysis of genome interaction maps. Genome Biology. 2018;19:1–12.

[36] Afgan E, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Research. 2018;46(W1):W537–W544.

[37] Yoo AB, et al. Slurm: Simple linux utility for resource management. In: Workshop on job scheduling strategies for parallel processing. Springer; 2003. p. 44–60.

[38] Jette MA, et al. Architecture of the Slurm Workload Manager. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer; 2023. p. 3–23.

[39] Servant N, et al. HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. Genome Biology. 2015;16:1–11.

[40] Roth C, et al. Improved quality metrics for association and reproducibility in chromatin accessibility data using mutual information. BMC Bioinformatics. 2023;24(1):441.

[41] Venu V, et al. Multi-omics analysis reveals the dynamic interplay between Vero host chromatin structure and function during vaccinia virus infection. Communications Biology. 2024;7(1):721.

[42] Consortium EP, et al. An integrated encyclopedia of DNA elements in the human genome. Nature. 2012;489(7414):57.

[43] Zhang Y, et al. Model-based analysis of ChIP-Seq (MACS). Genome Biology. 2008;9:1–9.

[44] Feng J, et al. Identifying ChIP-seq enrichment using MACS. Nature Protocols. 2012;7(9):1728–1740.

[45] Gaspar JM. Improved peak-calling with MACS2. BioRxiv. 2018; p. 496521.

[46] Chen S, et al. fastp: an ultra-fast all-in-one FASTQ preprocessor. Bioinformatics. 2018;34(17):i884–i890.

[47] Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv preprint arXiv:13033997. 2013;.

[48] Danecek P, et al. Twelve years of SAMtools and BCFtools. Gigascience. 2021;10(2):giab008.

[49] Faust GG, et al. SAMBLASTER: fast duplicate marking and structural variant read extraction. Bioinformatics. 2014;30(17):2503–2505.

[50] Chapman B, et al. Biopython: Python tools for computational biology. ACM Sigbio Newsletter. 2000;20(2):15–19.

[51] McKinney W, et al. Data structures for statistical computing in Python. In: SciPy. vol. 445; 2010. p. 51–56.

[52] Rolon-Mérette D, et al. Introduction to Anaconda and Python: Installation and setup. Quantitative Methods for Psychology. 2016;16(5):S3–S11.

[53] Deutsch P. GZIP file format specification version 4.3; 1996.

[54] Abdennur N, et al. Cooler: scalable storage for Hi-C data and other genomically labeled arrays. Bioinformatics. 2020;36(1):311–316.

[55] Yan KK, et al. HiC-spector: a matrix library for spectral and reproducibility analysis of Hi-C contact maps. Bioinformatics. 2017;33(14):2199–2201.

[56] McKenna A, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research. 2010;20(9):1297–1303.

[57] Ebbert MT, et al. Evaluating the necessity of PCR duplicate removal from next-generation sequencing data and a comparison of approaches. BMC Bioinformatics. 2016;17:491–500.

[58] Zhu T, et al. ATAC-seq with unique molecular identifiers improves quantification and footprinting. Communications Biology. 2020;3(1):675.

[59] Cleveland WS. Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association. 1979;74(368):829–836.

[60] Duan Z, et al. A three-dimensional model of the yeast genome. Nature. 2010;465(7296):363–367.

[61] Waskom ML. Seaborn: statistical data visualization. Journal of Open Source Software. 2021;6(60):3021.

[62] Muzellec B, et al. PyDESeq2: a python package for bulk RNA-seq differential expression analysis. Bioinformatics. 2023;39(9):btad547.

[63] Hruby DE, et al. Vaccinia virus replication I. Requirement for the host-cell nucleus. Journal of Virology. 1979;29(2):705–715.

[64] Moyer RW. The role of the host cell nucleus in vaccinia virus morphogenesis. Virus Research. 1987;8(3):173–191.

[65] Greseth MD, et al. The life cycle of the vaccinia virus genome. Annual Review of Virology. 2022;9:239–259.

[66] Schöpflin R, et al. Integration of Hi-C with short and long-read genome sequencing reveals the structure of germline rearranged genomes. Nature Communications. 2022;13(1):6470.

[67] Zhong JY, et al. High-throughput Pore-C reveals the single-allele topology and cell type-specificity of 3D genome folding. Nature Communications. 2023;14(1):1250.

[68] Skene PJ, et al. An efficient targeted nuclease strategy for high-resolution mapping of DNA binding sites. elife. 2017;6:e21856.

[69] Zheng Xy, et al. Low-input chromatin profiling in Arabidopsis endosperm using CUT&RUN. Plant Reproduction. 2019;32:63–75.

[70] Wu SJ, et al. Single-cell CUT&Tag analysis of chromatin modifications in differentiation and tumor progression. Nature Biotechnology. 2021;39(7):819–824.

[71] Nurk S, et al. The complete sequence of a human genome. Science. 2022;376(6588):44–53.

[72] Kaufmann S, et al. Inter-chromosomal contact networks provide insights into Mammalian chromatin organization. PloS One. 2015;10(5):e0126125.

[73] Krzywinski M, et al. Circos: an information aesthetic for comparative genomics. Genome Research. 2009;19(9):1639–1645.

Supplementary Materials

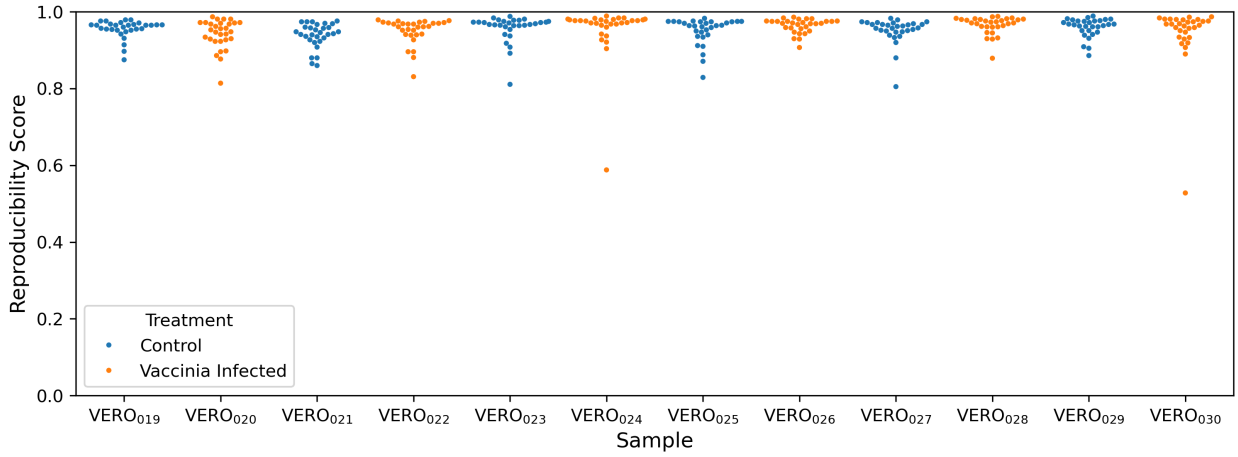

Figure S1: Reproducibility scores between Hi-C maps made by Juicer and SLUR(M)-py. Across Hi-C samples from Venu et al. (2024), (x-axis) the spector reproducibility (y-axis) score was calculated between Hi-C maps (per chromosome) made by the Juicer pipeline and those processed by the SLUR(M)-py pipeline. Dots within swarm plots represent scores per chromosome for a given sample. Blue and orange colors denote control samples and those samples infected with vaccinia virus. Only two infected samples show a score below 0.80 for chromosome 19 (NC_023660.1).

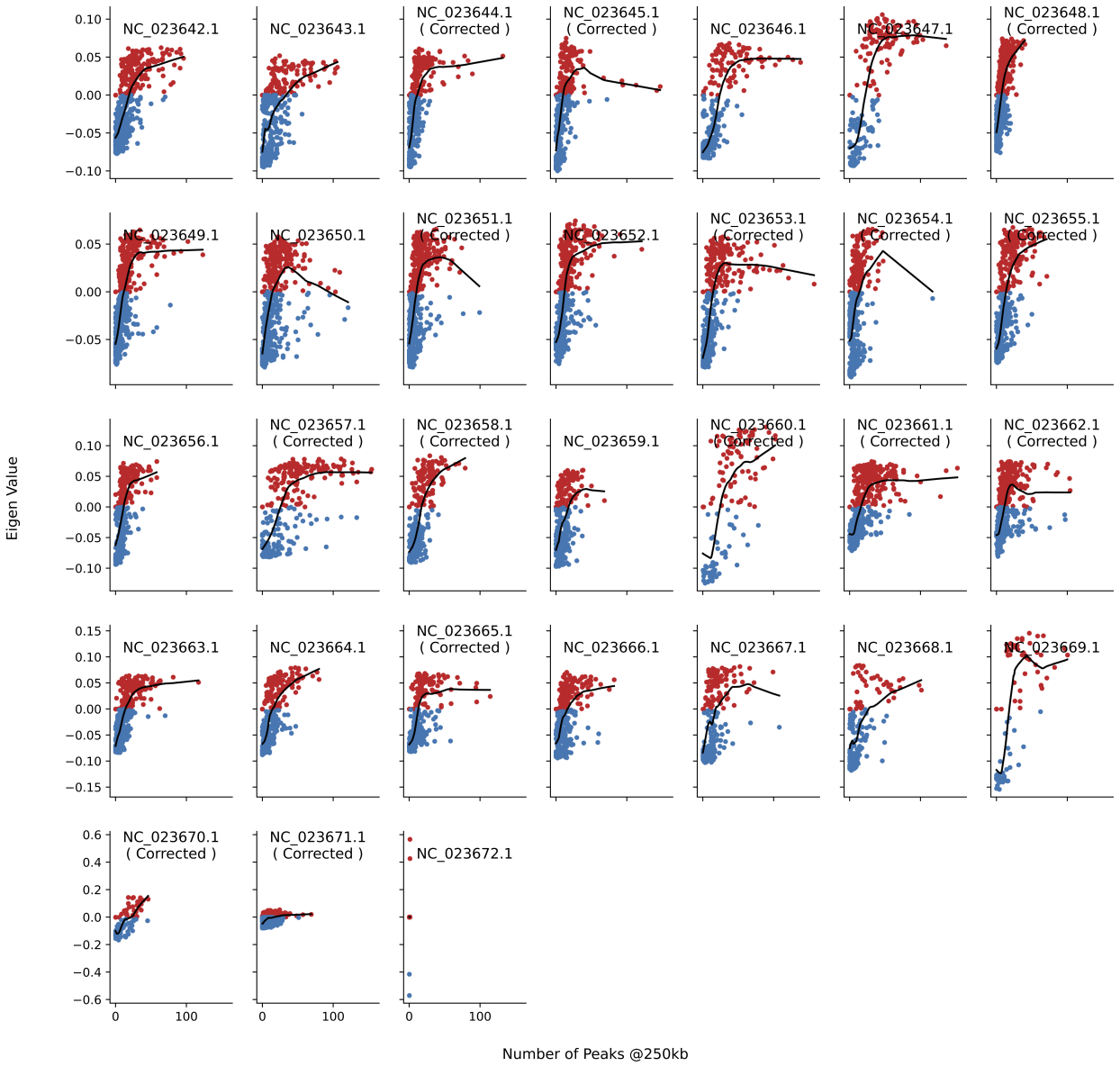

Figure S2: Hi-C compartment score correction and assignment. For the 31 chromosomes (subplots) within the VERO reference genome, the compartment scores (y-axis) are plotted against the number peaks within 250 kb genomic regions (x-axis). Compartments assigned as open (A) or closed (B) are then depicted in red and blue, respectively. A black line represents a LOWESS model. If compartment assignments were corrected with our `compartmentcorr.py` script the "Corrected" label appears within the subplot title, under the chromosome name. The `compartmentcorr.py` script auto-generates and saves the above plots for diagnostics.

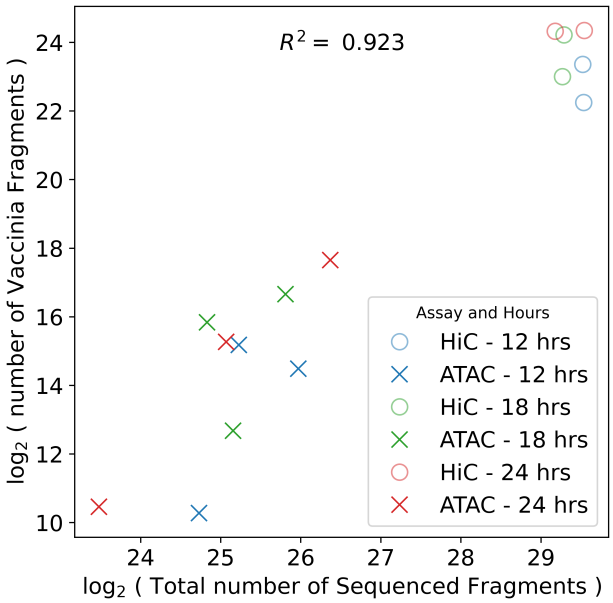

Figure S3: Counts of sequenced vaccinia fragments across ATAC-seq and Hi-C experiments. The count of sequenced fragments (y-axis) mapped to the vaccinia contig as a function of the total number of sequenced fragments (x-axis). The Xs and Os mark counts from an ATAC-seq or Hi-C experiment, respectively. Blue, green, and red colors denote the number of hours post infection. There is a significant linear relationship between the total identified vaccinia fragments and the number of sequenced reads in sequencing experiments on infected cells (p-value < 0.01, $R^2 = 0.923$).

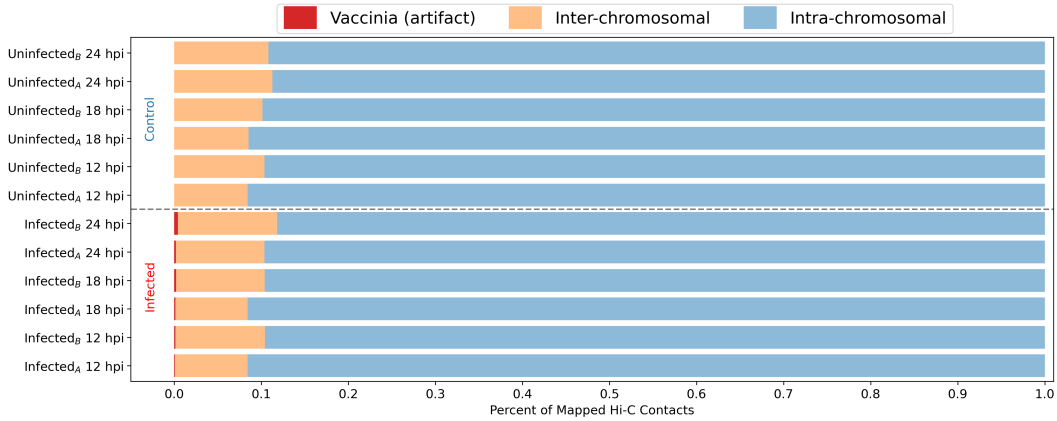

Figure S4: Inter- and intra-chromosomal contacts in control and infected Hi-C experiments. The portions (x-axis) of mapped Hi-C contacts are colored blue or orange across experiments (y-axis) for groups of paired-end reads mapping within or between chromosomes (respectively). Within infected experiments (bottom six experiments) the small portion of paired-end reads mapping between the genome and the vaccinia contig are marked in red. These events represent miss-ligation events between the genome and the viral DNA occurring post cross-linking within the cellular nucleus.

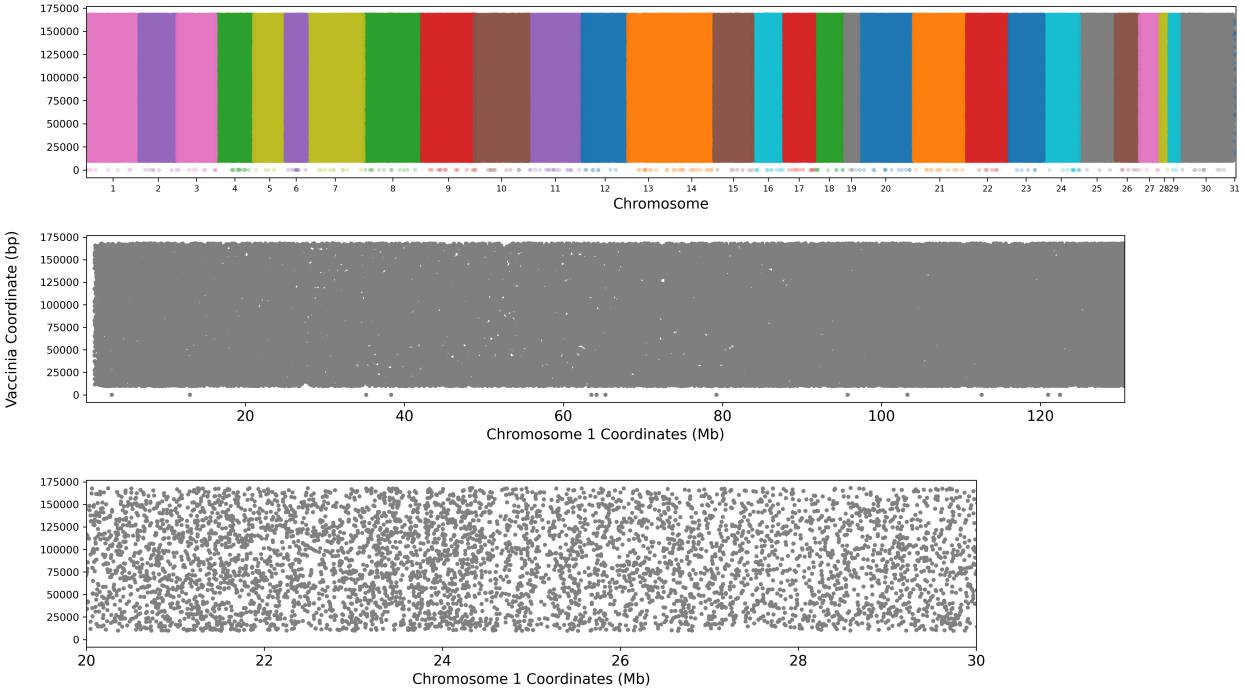

Figure S5: Mapping positions of artificial, miss-ligation events between vaccinia virus and genome. **Top**: Colored dots mark the locations of a ligation event between the double stranded vaccinia virus and the genome. The locations of these events along the vaccinia contig and the green monkey chromosome are marked along the y- and x-axis, respectively. **Middle**: The locations of artificial ligation events between the vaccinia virus and chromosome 1 of the Green Monkey genome. Each gray dot represents the positioning of a single ligation event between the genome (x-axis) and the viral contig (y-axis). **Bottom**: A 10 Mb region along chromosome 1 displaying ligation events and their locations between the genome (x-axis) and the viral contig (y-axis). At all three resolutions, genomic (**top**), chromosomal (**middle**), and 10 Mb (**bottom**), the ligation between genomic and viral DNA is uniformly distributed.

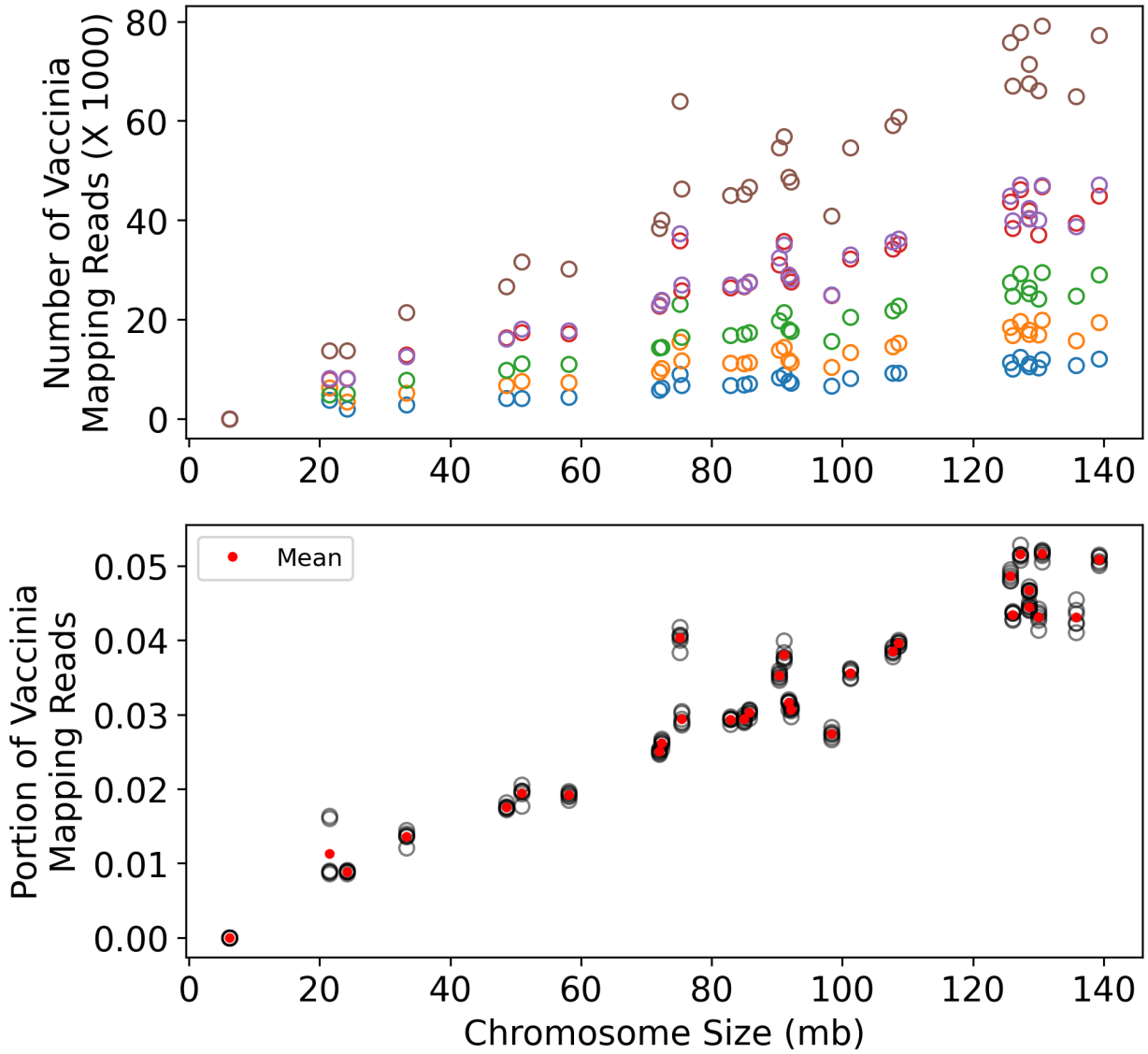

Figure S6: Counts of vaccinia mapped reads across the Green Monkey chromosomes. **Top**: Across replicated sequencing experiments (separated by colors) The number of identified read pairs mapping between the vaccinia contig and a given Green Monkey chromosome (y-axis) is plotted as a function of chromosome size. **Bottom**: The portions (y-axis, black circles) of vaccinia mapping reads as a function of chromosome size (x-axis). The mean portion across experiments is displayed via red dots.

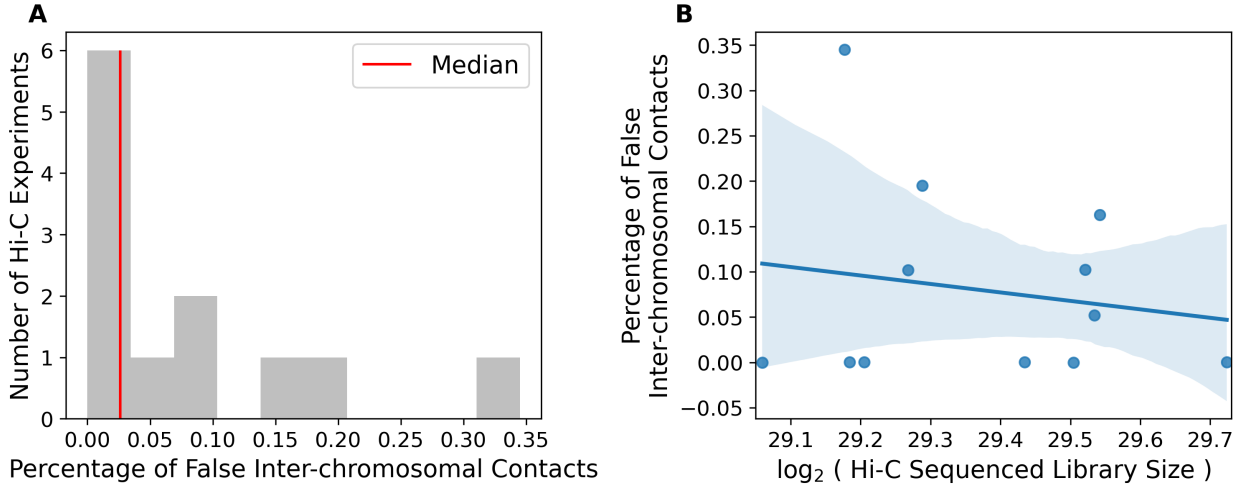

Figure S7: Percent of false positive inter-chromosomal contacts. **A**: Distribution of the percent of ligation events between the vaccinia contig and Green Monkey genome determined to be false positive inter-chromosomal contacts. The median percentage across experiments was 0.0261 (red vertical line). **B**: The percentage of false inter-chromosomal contacts (y-axis) versus the sequenced reads within each Hi-C library (x-axis). Blue line and shaded region represent a regression model and its 95% confidence interval relating the false inter-chromosomal contacts to the $\log_2$ of the read size. There is no significant linear relationship between the false positive percentage and number of reads (p-value > 0.05, $R^2 = 0.029$).

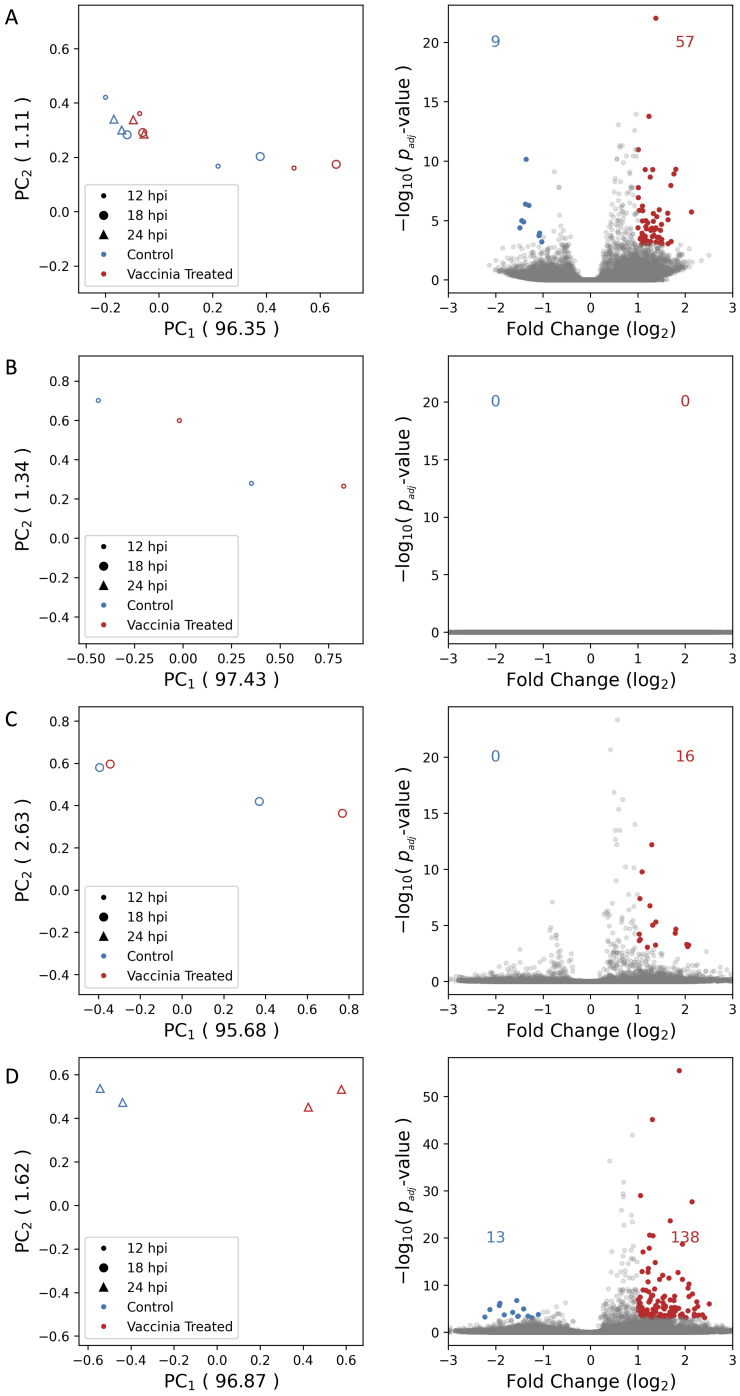

Figure S8: Analysis of inter-chromosomal contacts in VERO cells across vaccinia infection time-course. **A**) **Left**: the principal component analysis of 500 kb binned counts of contacts between chromosomes across vaccinia infected samples (red) and controls (blue) throughout time at 12, 18, and 24 hours post infection (hpi) (dots, circles, and triangles, respectively). **Right**: Differential analysis of inter-chromosomal contacts. Red and blue colors depict binned contacts between chromosomes increasing or decreasing due to vaccinia infection, respectively, across time. **B-D**) Principal component analysis (left) and differential analysis (right) of inter-chromosomal contacts at 12, 18, and 24 hpi.

.

Table S1: Processing, Mapping, and Peak Statistics of ATAC-seq Samples

Cell Line	Run Time	Paired-end Reads	Mapped (q 30)	Peaks	Summits	FRiP	Genome Bases Covered (percent)	Source
A549	0:19:17	16351381	11619607	89580	114555	0.5865	43693765 (1.402)	[40]
A549	0:22:28	21204135	10945957	90265	104479	0.3592	46231355 (1.483)	[40]
A549	0:17:48	16021189	8514473	64258	76240	0.3625	27657701 (0.887)	[40]
A549	0:44:35	33406352	16987737	92792	108354	0.3681	51768427 (1.661)	[40]
A549	0:22:23	17318402	7460950	55477	64813	0.4074	31944717 (1.025)	[40]
A549	0:36:13	21740996	20690406	99288	125802	0.5948	66678633 (2.139)	[40]
A549	0:27:50	21074428	13940998	113389	139868	0.6253	59522589 (1.909)	[40]
A549	0:35:11	24469297	17429501	86591	109579	0.621	65863808 (2.113)	[40]
VERO	0:27:43	11125133	15688912	112610	144830	0.4859	79265606 (2.889)	[41]
VERO	0:24:14	13919640	7743792	81198	102970	0.439	49251432 (1.795)	[41]
VERO	0:55:23	24173485	30454312	132871	208144	0.4061	114774486 (4.183)	[41]
VERO	0:34:59	18691465	12217888	100913	134291	0.308	69594515 (2.542)	[41]
VERO	0:25:15	9492136	12803176	114561	147878	0.3904	95182764 (3.476)	[41]
VERO	0:12:10	5848626	3948012	67851	71947	0.2171	29596890 (1.079)	[41]
VERO	0:38:51	21299383	19135402	113191	152489	0.6799	60565293 (2.207)	[41]
VERO	0:30:53	19644978	5754557	81310	98461	0.4512	37448317 (1.365)	[41]
VERO	0:16:43	8547245	7805617	104767	127810	0.7415	47891003 (1.745)	[41]
VERO	0:22:16	14905800	3326216	49718	60790	0.4342	22312359 (0.813)	[41]
VERO	0:20:12	10941701	9714255	99634	126874	0.7038	47536681 (1.732)	[41]
VERO	0:26:49	17601470	4288293	62048	73774	0.3361	29699129 (1.082)	[41]
VERO	1:25:40	39831755	39798249	245706	354836	0.3814	110789009 (4.037)	[41]
VERO	0:51:44	32864186	13174253	115644	156562	0.5307	51187658 (1.865)	[41]
VERO	1:00:24	32288959	26392861	122195	166423	0.6018	61006806 (2.223)	[41]
VERO	0:45:12	29424885	8528669	67641	88676	0.3599	28950015 (1.055)	[41]
VERO	1:11:34	39334143	30132783	146057	201135	0.5834	71671681 (2.612)	[41]
VERO	1:09:48	43316533	17979081	123494	169877	0.4162	55772205 (2.032)	[41]

.

Table S2: Cell lines, Read Counts, and SLUR(M)-py Run Times of Hi-C data

Sample Name	Paired-end Reads	Run Time (HH:MM:SS)	Valid Hi-C Contacts	Fragmentation Enzyme	Cell Line	Source
$A549_{000}$	130305900	01:03:45	78143020	HindIII	A549	ENCLB222WYT
$A549_{001}$	121585833	00:58:13	74012587	HindIII	A549	ENCLB571HTP
$A549_{002}$	254857397	04:48:50	172165621	MboI	A549	ENCLB571GEP
$HepG2_{003}$	815725922	08:51:10	739526295	HindIII	HepG2	ENCLB022KPF
$HepG2_{004}$	885833793	10:01:18	771404739	HindIII	HepG2	ENCLB625TGE
$IMR90_{005}$	252725458	01:59:11	173539638	MboI	IMR90	ENCLB476ZIC
$IMR90_{006}$	276809208	02:20:03	190530643	MboI	IMR90	ENCLB512HIP
$K562_{007}$	591854553	06:48:44	363306188	MboI	K562	ENCLB470UUQ
$K562_{008}$	456757799	03:08:49	305186793	MboI	K562	ENCLB941YRT
$VERO_{019}$	559483569	12:29:35	265274243	Arima	VERO	[41]
$VERO_{020}$	777613496	10:12:05	274526191	Arima	VERO	[41]
$VERO_{021}$	761567093	09:46:21	344542129	Arima	VERO	[41]
$VERO_{022}$	646594007	08:17:01	280411179	Arima	VERO	[41]
$VERO_{023}$	886507172	12:40:27	544176258	Arima	VERO	[41]
$VERO_{024}$	770418243	11:01:13	459796419	Arima	VERO	[41]
$VERO_{025}$	725697285	10:50:32	453580308	Arima	VERO	[41]
$VERO_{026}$	781644457	11:01:34	446320262	Arima	VERO	[41]
$VERO_{027}$	610190915	08:43:18	384589159	Arima	VERO	[41]
$VERO_{028}$	655739644	09:24:57	392837985	Arima	VERO	[41]
$VERO_{029}$	619039813	08:52:24	410112037	Arima	VERO	[41]
$VERO_{030}$	607026371	08:23:21	365787228	Arima	VERO	[41]