2. Methods

2.1 Samples

The following protocol and design were carried out by Dafni Glinos, a PhD student in the lab and Dr. Natalia Kunowska, a Senior Research Assistant in the lab.

1. Setup: The ChIPmentation-seq (ChM-seq) protocol (Schmidl et al. 2015) was performed on 100,000 sonicated Treg cells. After sorting and isolating Treg cells, those same cells were resuspended in full medium (IMDM, 10%FCS) at 1-2 million cells per ml.
2. ChIP-seq protocol:
    a. ChIP-seq libraries were generated using Illumina TruSeq index tags, while ChM-seq libraries were generated using the Nextera dual index tags.
    b. They cross linked cells by resuspending in a 1% formaldehyde solution.
    c. Following a 5 minute incubation at 37℃ the cells were quenched with glycine for 5 minutes to achieve a 125 mM concentration.
    d. They lysed and sonicated cells using the iDeal ChIP-seq kit for histones (Diagenode) as instructed by the manufacturer.
    e. They performed immunoprecipitation (IP) using the same kit. For this step, they used ChIP-seq grade antibodies against the histone marks H3K27ac (Diagenode), H3K27me3 (Abcam), H3K4me1 (Active Motif) and H3K4me3 (Active Motif).
    f. They prepared sequencing libraries as instructed by the iDeal ChIP-seq protocol. When doing ChIPmentation-seq, the two previous steps were combined by adding the Nextera Tn5 transposase after the IP step.

3. Sequencing: Samples were loaded for sequencing on a HiSeq 2500 instrument and V4 chemistry using standard 75 bp paired-end reads. In total there were 14 ChIPmentation-seq libraries that averaged 62 million reads per sample.
4. Data pre-processing: After data acquisition, the Sequencing Facility at the Wellcome Sanger Institute demultiplexed the sequencing data and mapped it to the human genome. Sequencing files were released in CRAM format.

2.2 ChIP-seq data processing

I re-mapped the reads from build 37 (GRch37) to the human reference genome build 38 (GRCh38) using bwa mem algorithm (Li 2013), and converted the aligned reads into BED (Kent et al. 2002) format. Only uniquely mapped reads were kept and duplicates were removed using samtools version 1.3.1 (H. Li et al. 2009).

The mapped reads were built into a signal track in bedgraph format using MACS2 (Zhang et al. 2008). The method uses input data i.e. control data, which is generated by treating samples the same way in the ChIP-seq protocol but without the addition of the antibody against the target epitope. The method then computes if the local average read coverage is statistically different compared to the control background. This data is referred to as observed data. This step is meant to render the output signal tracks to correct for any biases in the different samples.

2.3 Imputation reference panel

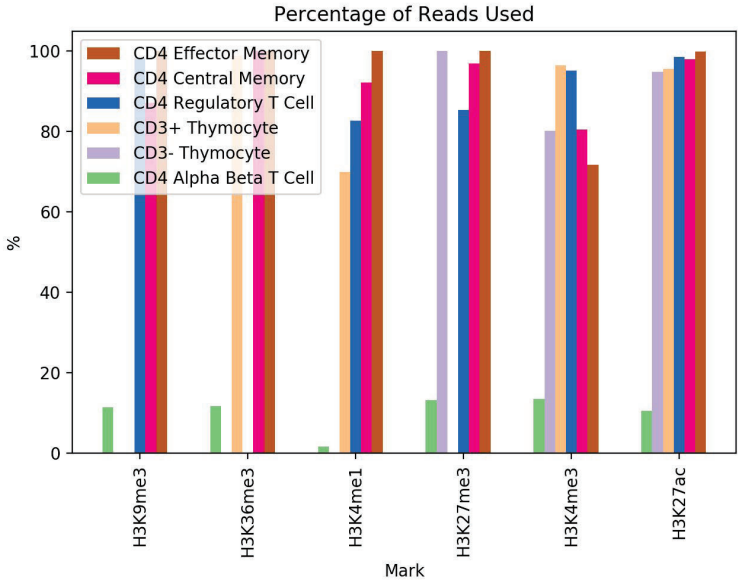
ChromImpute takes the information from the reads as input and converts it into smaller fragments, in the *Convert* step. Specifically, during the *Convert* step the software takes the signal track and averages the read coverage over 25 bp bins across the genome.

The construction of the reference panel was performed in two steps. First I built the reference panel with the Roadmap (Bernstein et al. 2010) and ENCODE data

(ENCODE Project Consortium 2012; ENCODE Project Consortium 2004) as specified in the ChromImpute paper (Ernst & Kellis 2015). For that I used 129 cell types and 8 histone marks: DNase, H3K27ac, H3K27me3, H3K36me3, H3K4me1, H3K4me3, H3K9ac, H3K9me3. Second, I added the BLUEPRINT (Adams et al. 2012) data to the reference. For that I selected cell types within the general T cell population: $CD4^+$ Effector Memory, $CD4^+$ Central Memory, $CD4^+$ Regulatory T cell, $CD3^+$ Thymocyte, $CD3^-$ Thymocyte, $CD4^+$ Alpha Beta T cell. I used any given mark that was available, and the marks that I used were: H3K27ac, H3K27me3, H3K36me3, H3K4me1, H3K4me3, H3K9ac, H3K9me3.

The number of reads for each of the cell types varied greatly; most of the reads came from $CD4^+$ Alpha Beta T cells as these represented the vast majority of the total samples. I aimed to have between 100 to 160 million reads per mark per cell type as that was the general baseline required to prevent any bias towards a specific cell type/mark combination. If there were more than 160 million reads available, I would randomly subsample the different read files to reach the total of 160 million reads. For example, the $CD4^+$ Alpha Beta T cells had 31 samples with over one billion reads. Thus the percentage of reads sampled was far less than for other cell types **(Figure 2.1.A)**. Overall, I constructed the reads based on what was available; many cell types had a lower number of reads (Tregs, Effector Memory). If there were less than 160 million reads in the entire cell type, I used them all in the reference **(Figure 2.1.B)**.

A) Percentage of reads used



B) Number of reads sampled

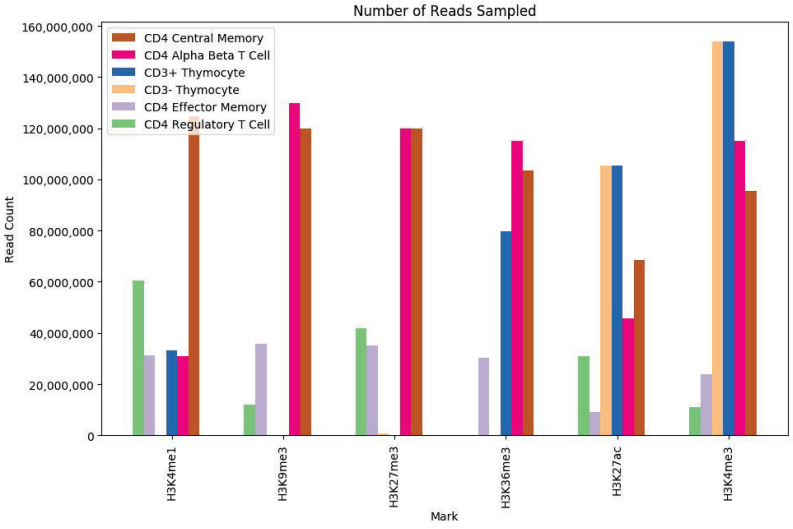


Figure 2.1 Reads assembly broken down by cell type and histone mark The Blueprint reference panel reads were assembled using downloaded FASTQ (Cock et al. 2010) files from: https://www.ebi.ac.uk/ega/user/login. The files were then converted into

BAM (H. Li et al. 2009) format and combined and added to the existing consortium. **A)** describes the percentage of reads used in the reference and **B)** illustrates the total number of reads sampled in each histone mark.

2.4 Imputation

There are four steps in the imputation pipeline. The first step is to compute the global distance between each of the observed data signal tracks and the reference signal tracks for the same histone mark, using a genome wide signal correlation between the imputed and observed tracks. For this, I used the *ComputeGlobalDist* method in ChromImpute. The obtained output was a list of samples ranked by the correlation with the observed sample signal track in the reference panel for the given histone mark. The second step, *GenerateTrainData*, generates training data by using the signal tracks from the same sample with different marks. This step is meant to capture any specific sample information and requires that at least 2 histone marks are assayed for any given sample. The training data is then combined with the most similar samples, up to ten samples, based on the same histone mark. In the third step, the model is trained using regression trees, this corresponds to the *Train* command. Each of the different signal tracks are transformed into a regression tree that is used to compute imputed signal at a given position. Finally in the fourth step, the different trees are then applied, *Apply*, onto the observed signal track and using an ensemble approach, the information from each tree is combined and averaged. The final result is a signal track with imputed signal. There is no correction for read depth or any biases, and samples that may be deeply sequenced may impute in a different way than samples with low sequencing depth.

2.5 Evaluating reference panel

In order to understand the effectiveness of the reference panel, I performed a signal track recovery analysis to evaluate which marks performed best during imputation. The marks with the best recovery were H3K27ac, H3K4me3 and H3K4me1 as there is the most reference data for these marks. For that I use the *Eval* tool, provided by

the ChomImpute. The output of this analysis are several metrics for each sample and histone mark combination, including: the fraction of the observed top *x* percent signal track in the imputed top *x* percent signal track and vice versa; the genome wide correlation between the observed and imputed signal tracks; the area under the ROC for predicting the top *x* imputed signal with the observed signal; and the area under the ROC for predicting the top *x* observed signal with the imputed signal. Here *x* represents a value that a user can specify for ChromImpute.

2.6 Downstream computation of observed and imputed peaks

In order to call peaks, and generate the initial signal track, I used the MACS2 (Zhang et al. 2008) software. To generate signal tracks I used the commands: *macs2 callpeak -t <index_file> -f BEDPE -n <temp_peaks_file> -g hs --nomodel -B --SPMR* along with *macs2 bdgcmp -t <bdg_reads> -c <bdg_input> -o <bedgraph_output> -m ppois -S <s_value>*. In order to call narrow peaks I used this command: *macs2 bdgpeakcall -i <bedgraph_input> -c <cutoff> -l <min_length> -g <max_gap> --outdir <output_dir> -o <output_peak_file>*. To call broad peaks use the same command but swap *bdgpeakcall* with *bdgbroadcall*.

I used BEDTools (Quinlan & Hall 2010) to analyze the generated peaks. To compare which peaks were shared as well as unique to the imputed and observed datasets I used the *intersect,* command. An example command to find 20 percent overlap in two sets of peaks is: *bedtools intersect -a Observed.bed -b Imputed.bed -f 0.2 -r -wo > 20percentInBoth.bed*. To convert BAM files to BED format I used the following command: *bedtools bamtobed -bedpe -i <bam_file> > <bed_file>*.

I used ChIPSeeker (Yu et al. 2015) to provide gene annotations and follow up with functional analysis on the different peak files. I also used the UCSC known genes version 3.4.0 to annotate genes (Hsu et al. 2006). Exentensive documentation can be found here: *https://bioconductor.org/packages/release/bioc/vignettes/ChIPseeker/inst/doc/ChIPseeker.html*.

I used several Python libraries for analysis: Pybedtools (Dale et al. 2011) as an API for BEDTools, Pandas (Mc Kinney n.d.) for data table manipulation, matplotlib (Hunter 2007) and seaborn (Waskom et al. 2014) for plotting and graphs, PyPlink to handle PLINK (Purcell et al. 2007) files with respect to genotype data, Pathos (McKerns et al. 2012) for multiprocessing and code optimization, NumPy (Oliphant 2006) and SciPy (Millman & Aivazis 2011; Oliphant 2007) for statistics and general mathematics. I used iPython (Perez & Granger 2007) and Jupyter Notebooks (Kluyver et al. 2016) for quick code prototyping as well. I used Pandas to handle all of my data, which allowed for ease to use relational database concepts with tables loaded in RAM (Random Access Memory). I used SKlearn (Pedregosa et al. 2011) to perform a Linear Regression as well as normalization within the generated Pandas dataframes.

I used the UCSC Genome Browser (Kent et al. 2002) to manually verify peak overlap and compare and contrast observed and imputed peak information.

The code for the ChromImpute pipeline along with every Python script I developed for analysis is available at:
https://gitlab.internal.sanger.ac.uk/TrynkaLab/ChromImpute.