COMBINING DNA METHYLATION WITH DEEP LEARNING IMPROVES SENSITIVITY AND ACCURACY OF EUKARYOTIC GENOME ANNOTATION

Gregory J. Zynda

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Luddy School of Informatics, Computing, and Engineering
Indiana University

April 2020

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Mehmet Dalkilic, Ph.D.

Haixu Tang, Ph.D.

Yuzhen Ye, Ph.D.

Donald Williamson, Ph.D.

Matthew W. Vaughn, Ph.D.

April 7th, 2020

Copyright © 2020

Gregory J. Zynda

This work is dedicated to

My wife, Mel,

for her love and support

ACKNOWLEDGEMENTS

I am grateful to my advisor, Mehmet Dalkilic, for seeing potential in an eager undergrad, who was passionate about bioinformatics, but without direction. He allowed me to explore the field, discover my expertise, and reach this achievement.

I would also like to thank my committee members, Yuzhen Ye and Donald Williamson, for their valuable time and feedback. In particular, Haixu Tang, who exposed me to real-world bioinformatics at the CGB, and Matthew Vaughn, who gave me a home at TACC to continue my research and contribute to his.

I am especially grateful to my wife, Mel, who encouraged and supported me throughout this journey. When problems arose, my friends, Robert, Steve, and James were always willing to listen and help me work through them. Finally, I want to thank my parents, Laura and Greg, for their love, support, and genuine interest.

Gregory J. Zynda

Combining DNA methylation with deep learning improves sensitivity and accuracy
of eukaryotic genome annotation

The genome assembly process has significantly decreased in computational complexity since the advent of third-generation long-read technologies. However, genome annotations still require significant manual effort from scientists to produce trustworthy annotations required for most bioinformatic analyses. Current methods for automatic eukaryotic annotation rely on sequence homology, structure, or repeat detection, and each method requires a separate tool, making the workflow for a final product a complex ensemble.

Beyond the nucleotide sequence, one important component of genetic architecture is the presence of epigenetic marks, including DNA methylation. However, no automatic annotation tools currently use this valuable information. As methylation data becomes more widely available from nanopore sequencing technology, tools that take advantage of patterns in this data will be in demand.

The goal of this dissertation was to improve the annotation process by developing and training a recurrent neural network (RNN) on trusted annotations to recognize multiple classes of elements from both the reference sequence and DNA methylation. We found that our proposed tool, RNNotate, detected fewer coding elements than GlimmerHMM and Augustus, but those predictions were more often correct. When predicting transposable elements, RNNotate was more accurate than both RepeatMasker and RepeatScout. Additionally, we found that RNNotate was significantly less sensitive when trained and run without DNA methylation, validating our hypothesis. To our best knowledge, we are not only the first group to use recurrent neural

networks for eukaryotic genome annotation, but we also innovated in the data space by utilizing DNA methylation patterns for prediction.

Mehmet Dalkilic, Ph.D.

Haixu Tang, Ph.D.

Matthew W. Vaughn, Ph.D.

Donald Williamson, Ph.D.

Yuzhen Ye, Ph.D.

CONTENTS

Acknowledgements v

Abstract vi

Abbreviations xxiv

1. Contribution 1

2. Introduction 3

2.1 Genome annotation 4

2.2 Computational annotation 5

2.2.1 Annotation of repetitive elements 5

2.2.2 Annotation of genes 7

2.2.3 Ensemble methods for annotation 8

2.3 Sequence tagging with machine learning 9

3. Methodology 11

3.1 Collecting requirements and building a specification 11

3.1.1 Input Specification - Unclassified 11

3.1.2 Output Specification - Training and Predictions 13

3.1.3 Data Shape - Input and Output 16

3.2 Developing data access interfaces 19
3.2.1 FASTA access 19
3.2.2 Methylation alignment and access 20
3.2.3 Annotation access 21
3.3 Developing a flexible model 22
3.3.1 Convolutional section 23
3.3.2 Recurrent section 23
3.3.3 Hidden section 26
3.3.4 Loss functions 26
3.4 Data traversal methods 27
3.4.1 Independent traversal 28
3.4.2 Stateful traversal 29
3.4.3 Stranded traversal 29
3.5 Aggregation and decoding of output 32
3.6 Data sources 34
3.7 Determining optimal model architecture 34
3.7.1 Collecting local error metrics 35
3.7.2 Hyperparameter sweep of model 35
3.8 Final training 36
3.9 Evaluation metrics 37

4. Results and Discussion 39
4.1 Initial results 39
4.2 Hyperparameter sweep 40

4.3 Long training 44
4.4 Base level evaluation of transcripts 47
4.4.1 Base level performance of exon and gene predictions 48
4.4.2 Base level performance of mRNA predictions 50
4.4.3 Base level performance of CDS predictions 52
4.4.4 Overall base level performance 52
4.5 Base level evaluation of transposable elements 53
4.6 Region level evaluation of transcripts 55
4.6.1 Characteristics of transcription regions 58
4.7 Region level evaluation of transposable elements 59
4.8 Impact of DNA methylation on prediction 62
4.9 Investigating model performance 65
4.9.1 Regional exon methylation 67
4.9.2 Regional mRNA methylation 69
4.9.3 Regional CDS methylation 69
4.9.4 Regional gene methylation 71
4.9.5 Performance conclusions from methylation 71
4.10 Runtime performance 77
4.11 Issues limiting portability 78

5. Future Work **80**
5.1 Removing strand specific categories from output 80
5.2 Improving data variety 80
5.3 Optimizing hyperparameter search 81

5.4 K-mers to improve the prediction of coding features 82
5.5 Incorporating additional data dimensions 82

6. Conclusions **84**

7. Appendix **86**

7.1 Software 86
7.2 Data 86

8. Additional Work **88**

8.1 Feature Frequency Profiles for Automatic Sample Identification using PySpark 88
8.1.1 Abstract 88
8.1.2 Introduction 89
8.1.3 Related work 91
8.1.4 Spark For Genomics 92
8.1.5 Approach 94
8.1.6 Classifying Reads 96
8.1.7 Methodology 97
8.1.8 Results 98
8.1.9 Conclusion 99
8.1.10 Availability 101
8.2 Repliscan: a tool for classifying replication timing regions 101
8.2.1 Abstract 101
8.2.2 Background 102
8.2.3 Implementation 105

8.2.4 Results and Discussion 122

8.2.5 Conclusions 131

8.2.6 Availability 132

Bibliography **132**

Curriculum Vitae

LIST OF TABLES

3.1 The numerical nucleotide mapping code for encoding sequences into a numerical format for use by model. 12

3.2 RNNotate input specification detailing the order, type, and range for each dimension of the input. 13

3.3 Annotation features type tracked by RNNotate. The first two columns denote the feature's strand-specific indices, the third column tracks the data type, and the fourth column provides a description. The final two rows enable sub-classification levels of TE related features (bold), which are numerical and unstranded. 15

3.4 Numerical codes used in index 64 of output specification (Table 3.3) for collected TE orders. 17

3.5 Numerical codes used in index 65 of output specification (Table 3.3) for collected TE superfamilies. 18

3.6 Features predicted by RNNotate when low-frequency features are excluded. Bold features can still allow the prediction of TE sub-classifications. 19

3.7 Size comparison between normal and Meth5py files. Original size in megabytes (MB), Meth5py size in MB, and space savings in percent. 21

3.8 Strengths and weaknesses of different ways to process batches (Independent, Stateful) and sequences (Forward, Bidirectional). 25

3.9 Example independent classifications. Given category counts at 3 indices, output classifications are demonstrated for two thresholds (t): 0.5 and 0.4. 32

3.10 Final output GFF3 specification in tabular format. Each row represents a column in the GFF3 tab-delimited format. Columns 1, 4, 5, and 7 describe the location of a prediction. Column 2 designates the source of the prediction. Column 3 is the feature type, and corresponds to a category in Table 3.3. Columns 6 and 8 correspond to score and phase, which are not used, and left empty with “.”. 33

3.11 Architecture parameter permutations for testing. 36

4.1 Hyperparameter multiple regression on validation MSE at epoch 20. Bold P-values are significant (P ¡ 0.05). 43

4.2 The top 5 models at epoch 20, sorted by median validation MSE and median validation IQR. 43

4.3 Corresponding median mean squared error (MSE) and interquartile range (IQR) for both Chromosome 1 (Training) and Chromosome 3 (Validation). Data was sorted by validation MSE and IQR. 45

4.4 Base level performance measures. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each tool, feature type, and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), and precision (PREC) were also calculated. Glimmer stands for GlimmerHMM in this table. 49

4.5 Average of base-level sensitivity (SENS), specificity (SPEC), and precision (PREC) across features by tool and strand. Bold values represent the best tool for each performance/strand (column) category. . . 53

4.6 Base level performance measures for TEs. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each tool and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), precision (PREC), and accuracy (ACC) were also calculated, with top values in bold for each strand. . 54

4.7 Region level performance of transcription elements. True positive (TP), false positive (FP), and false negative (FN) counts were calculated for each tool, feature type, and strand combination in total regions. Sensitivity (SENS) and precision (PREC) were also calculated based on region counts. 57

4.8 Region level performance of transposable elements. True positive (TP), false positive (FP), and false negative (FN) counts were calculated for each tool, feature type, and strand combination in total regions. Sensitivity (SENS) and precision (PREC) were also calculated based on region counts. 60

4.9 Training and validation error after training without methylation. Corresponding median mean squared error (MSE) and interquartile range (IQR) for both Chromosome 1 (Training) and Chromosome 3 (Validation). Data was sorted by validation MSE, and the best (lowest) 10 values were kept. 62

4.10 Base level performance measures of RNNotate. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each model version, feature type, and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), and precision (PREC) were also calculated with top values in bold. All comparisons were unstranded made against the Chromosome 3 of the Araport11 annotation. 64

4.11 t-test of Transcription methylation differences. Tabulation of MSE values from each methylation context. For each category of MSE values (TP-FP and TP-FN), a one-tailed t-test was performed to compare methylation differences of features originating from the reverse strand (-) and the forward strand (+). 75

4.12 t-test of TE methylation differences. Tabulation of MSE values from each methylation context. For each category of MSE values (TP-FP and TP-FN), a one-tailed t-test was performed to compare methylation differences of features originating from the reverse strand (-) and the forward strand (+). 76

4.13 Runtime and resource requirements of tools. For each tool, the wall clock runtime was recorded in “minutes:seconds” along with the maximum memory usage in gigabytes (GB). 77

8.1 Accuracy of read classification from sparkmer and Kraken. 99

8.2 Example coverage values to demonstrate replication timing classification. 120

LIST OF FIGURES

2.1 Growth of "Complete" virus (green), prokaryote (gold), and eukaryote (blue) genomes at NCBI over time by year. 4

3.1 Genome browser view of interleaved TEs in the Araport11 annotation. 16

3.2 The performance of querying annotation data for increasing region sizes was compared between HDF5 (blue) and the interval tree format (yellow). 22

3.3 Illustration of model architecture possibilities. Data flows from the inputs at the bottom upward to the outputs at the top. The optional convolutional layer with "same" padding and variable width (3 shown) is represented in **orange**. The optional stateful recurrent states are represented by the **blue** regions flanking the **purple** recurrent section. The optional bidirectional layer is represented by the semi-transparent flow in the **purple** recurrent layer. The configurable hidden layers are represented in **green**. The final output layer is represented in **yellow**. 24

3.4 Starting from comparing a binary matrix to itself, a percentage of total values are randomly changed to 0 or 1, and compared to the original with the binary cross-entropy (blue) and mean squared error (gold) loss functions. 27

3.5 Illustration of independent batch traversal. The different sequence colors are only used to visually separate subsequences. Batches can contain any number of sequences. 28

3.6 Illustration of stateful batch traversal. Below the chromosome line, is an example of online stateful learning with one sequence per batch. The chromosome in the *Batched Learning* example is processed with a batch size of 2, so it is traversed in 2 subsequences with 50% overlap. 30

3.7 The “Genomic Data” row illustrates both stranded genomic data, and annotation features. The “Traversal Coordinates” row illustrates discrete sections of data. The “Unstranded Traversal” row demonstrates what information is present in each data batch. The “Stranded Traversal” row demonstrates how a chromosome is processed and which strands of annotation data are present. 31

3.8 Left: The comparison figure will be 2-D with half of the parameters represented on the y-axis, and the other half represented on the x-axis. Right: Each circle will represent a model, and poor performing models with large IQRs or medians will blend in with the background by being small and similar to the background. 37

4.1 Hyperparameter comparison at epoch 5. Model parameters are represented on the bordering tables. Model error represented by circle color, where light circles have high error and dark circles have low error. A model’s error distribution is inversely represented by each circle’s area, where a large circle has a tight IQR and small circles have a large IQR. The top 5 models are annotated with their rank and a red circle. . . 41

4.2 Hyperparameter comparison at epoch 20. Model parameters are represented on the bordering tables. Model error represented by circle color, where light circles have high error and dark circles have low error. A model’s error distribution is inversely represented by each circle’s area, where a large circle has a tight IQR and small circles have a large IQR. The top 5 models are annotated with their rank and a red circle. 42

4.3 Error during long training. The top two plots show error rate along the training and testing chromosomes. Initial epochs start in blue and transition to red for late epochs. The epoch with the lowest median test MSE is plotted in gold. The bottom plot contrasts the training (blue) and testing (red) MSE distributions of across epochs. 46

4.4 Base level performance of exons and genes. Left - Venn diagram showing the logical relations between predictions of Araport11 Chromosome 3 exons by RNNotate and Augustus. Right - Venn diagram showing the logical relations between exon predictions of Araport11 Chromosome 3 genes. 50

4.5 Base level performance of mRNA. Top-Left - Venn diagram showing the logical relations between predictions of Araport11 Chromosome 3 mRNA by RNNotate and GlimmerHMM from the forward strand. Top-Right - Logical relations between mRNA predictions by RNNotate and GlimmerHMM from the reverse strand. Bottom-Center - Logical relations between mRNA predictions by RNNotate and GlimmerHMM from both strands. 51

4.6 Reciprocal overlap of CDS features. Histogram of reciprocal overlap for each predicted CDS feature from each of the 3 tools against Araport11 CDS regions. The vertical red line represents the 75% threshold used for "true-positive" classifications. 56

4.7 Descriptive comparison of CDS predictions. Left figure shows the nucleotide proportion distributions for CDS predictions on either strand by tool (RNNotate, GlimmerHMM, Augustus) or reference (Araport11). Right figure shows the sqrt(length) distributions of CDS predictions on either strand by tool or reference. 59

4.8 Reciprocal overlap of transposable elements. Histogram of reciprocal overlap for each predicted TE from each of the 3 tools against TEs in the Araport11 annotation. The vertical red line represents the 75% threshold used for true-positive classifications. 60

4.9 Descriptive comparison of TE predictions. Left figure shows the nucleotide proportion distributions for TE predictions on either strand by tool (RNNotate, RepeatMasker, RepeatScout) or reference (Araport11). Right figure shows the sqrt(length) distributions of CDS predictions on either strand by tool or reference. 61

4.10 Error during long training without methylation. The top two plots show error rate along the training and testing chromosomes. Initial epochs start in blue and transition to red for late epochs. The epoch with the lowest median test MSE is plotted in gold. The bottom plot contrasts the training (blue) and validation (red) MSE distributions of across epochs. 63

4.11 Base level logical relations between Araport11 Chromosome 3 features and bases predicted by RNNotate and RNNotate without methylation (w/o Methylation). 66

4.12 Genome browser view of Araport11 gene AT3G19670.2 (blue), and its prediction by Augustus (green), GlimmerHMM (orange), RNNotate (red), and RNNotate without methylation (purple). 67

4.13 Methylation around exon predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue). 68

4.14 Methylation around mRNA predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue). 70

4.15 Methylation around CDS predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue). 72

4.16 Methylation around gene predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue). 73

4.17 Effect of model architecture on memory consumption by batch. The left figure shows total memory usage after each training batch for three different model architectures. The right figure shows the memory increase after each training batch for three different model architectures. In the figure legends, “lstm=1,2” designates 1 or 2 LSTM layers and “td=1,2” designates 1 or 2 time-distributed hidden layers. 78

8.1 Number of complete genomes archived by NCBI genome. 90

8.2 PySpark workflow for reference k-mer counting. 93

8.3 PySpark workflow for classifying input reads. 95

8.4 Histogram illustrating the default partitioning of the reads RDD after k-mer calculation. 97

8.5 Runtime of sparkmer on increasing numbers of executors. 100

8.6 Overview of the cell cycle. Cell division takes place in two stages: interphase and mitosis. Interphase is when a cell copies its genome in preparation to physically divide during mitosis. Interphase starts with cell growth and preparation for DNA synthesis in Gap (G1). After G1, DNA is replicated in regions during the Synthesis (S) phase. The cell then transitions into a second growth phase - Gap 2 (G2). When the cell has finished growing, the cell divides into two daughter cells in Mitosis (M). 102

8.7 Repliscan workflow. Diagram of the preliminary alignment and quality control methods at the top, and the Repliscan methods at the bottom. 107

8.8 Replication signal and sampling uncertainty. The top two graphs show raw and windowed replication signal across *A. thaliana* chromosome 3. The bottom two graphs show raw and windowed replications signals at 18.5-19.0 megabases from the top view as represented by the gray selection area. The red bars represent sampling uncertainty ($\sqrt{\lambda}$ for Poisson distributions). 108

8.9 Normalized and transformed replication signals. Violin plots showing how the normalized and aggregated *A. thaliana* chromosome 3 replication signals from G1, early (E), middle (M), and late (L) S-phase data was bounded from $[0, \infty)$. We separately experimented with with log transforms to make the distributions more normal-like, and square root transforms to stabilize the spread. 110

8.10 Outlying coverage in chromosome 3. Based on the normal distribution fit (yellow) to the log transformed coverage distribution of early (E), middle (M), and late (L) S-phase data, windows that fall in the tails shaded in gray are removed from the analysis. 113

8.11 Smoothing comparisons. **A** - Noise (green) is added to an original signal (purple), and then smoothed with a 4 unit (40 point) moving average (orange), a 5 unit (25% subset) LOESS (red), and a level 3 Haar wavelet (blue). Both the moving average and LOESS spread out the peaks and artificially lowered signal amplitudes, while the Haar wavelet keeps bounds and peak heights close to the original. **B** - The *A. thaliana* middle S-phase normalized signal (green), is smoothed with a moving average (orange), LOESS (red), and the level 3 Haar wavelet (blue) for comparison. 116

8.12 Replication threshold from coverage. The upper plot shows how much of *A. thaliana* chromosome 3 will be kept for downstream analysis as a function of the signal threshold. The lower plot shows the chromosome coverage differential as a function of the threshold. The vertical red line in each plot marks the optimal threshold of 0.92. 120

8.13 Comparison of *A. thaliana* and *Z. mays* segmentation. Following the segmentation legend on the right, *A. thaliana* chromosome 3 (top) and *Z. mays* chromosome 10 (bottom) have been classified into segmentation regions by Repliscan. The large white regions in the *A. thaliana* figure are unclassified regions due to high or very low signal. Below each replication segmentation is a depiction of the chromosome, with the centromere location marked in yellow [112], [113]. 123

8.14 Composition of replication segmentation. The segment composition shows that replication in *A. thaliana* is skewed towards early S replication, while *Z. mays* has an even distribution across early, middle, and late S. We can also see that the non-sequential early-late (EL) and early-middle-late (EML) classifications comprise a very small proportion of the classified segments in both cases. 124

8.15 Segment size distribution. Boxplots for every combination of replication time, illustrating the distribution of segment sizes. Early (E) and mid-late (ML) S were largest in *A. thaliana*, while early and late (L) were largest in *Z. mays*. 125

8.16 Segmentation differences in downsampled data. After downsampling the *A. thaliana* data, the accuracy of median (top) and sum (bottom) aggregation, and outlier detection using log gamma, none (NA), normal, square root gamma, and whiskers. Inflection points in the differences are labeled with black diamonds. 126

8.17 Unconverged log gamma fit. Most of the data is removed when the iterative fitting function fails to converge with the log transformed gamma distribution. Instances like this produce the spikes of differences in Figure 8.16. 127

8.18 Human fibroblast Repli-seq. 50 kilobase sliding window replication signals (blue) reproduced from Hansen *et al.*, published "BJ-G1_segment" regions, and 50 kilobase Repliscan results (bottom). 129

8.19 *D. melanogaster* KC167 Repli-Seq. Reproduction of the LOESS smoothed continuous replication profile (Lubelsky LOESS), and the thresholded, discrete early (blue) and late timing domains (Lubelsky > 0.5) from original Lubelsky *et al.* study. Repliscan segmentation results with Early (Early, Early-Mid) and Late (Mid-Late, Late) replication (2S), and Early, Early-Mid, Mid-Late, and Late replication (4S) configuration with 10 kilobase windows. 130

LIST OF LISTINGS

4.1 Run command for long training session 44
4.2 Command for running GlimmerHMM 47
4.3 Command for running Augustus 47
4.4 Command for running RepeatMasker 53
4.5 Command for running RepeatScout 53

ABBREVIATIONS

NCBI: National Center for Biotechnology Information

RNN: Recurrent neural network

CNN: Convolutional neural network

LSTM: Long short-term memory

GRU: Gated recurrent unit

CPU: Central processing unit

GPU: Graphics processing unit

CG: When a cytosine (C) nucleotide is sequentially followed by a guanine (G) nucleotide in the 5' to 3' direction.

CHG: When a cytosine (C) nucleotide is sequentially followed by any nucleotide except guanine (H = C,T,A), which is then followed by a guanine (G) nucleotide in the 5' to 3' direction.

CHH: When a cytosine (C) nucleotide is sequentially followed by any two nucleotides except guanine (H = C,T,A) in the 5' to 3' direction.

CDS: Coding DNA sequence

TE: Transposable element

DNA: Deoxyribonucleic acid

RNA: Ribonucleic acid

RNA-seq: RNA sequencing

CHIP-seq: Chromatin Immunoprecipitation Sequencing

SAM: Sequence Alignment/Map file format

BAM: Binary version of SAM file format

TimEx: Time of replication

Repli-seq: Replication label incorporation sequencing

Edu: 5-Ethynyl-2'-deoxyuridine

BrdU: 5-Bromo-2'-deoxyuridine

NGS: Next generation sequencing

SRA: Sequence read archive

G1: Gap 1 of cell division

G2: Gap 2 of cell division

S: Synthesis phase of cell division

E: Early S-phase replication

M: Middle S-phase replication

L: Late S-phase replication

WB: Whisker bounds

1. Contribution

The genome assembly process has significantly decreased in computational complexity since the advent of third-generation long-read technologies. However, genome annotations still require significant manual effort from scientists to produce trustworthy annotations required for, most bioinformatic analyses. Current methods for automatic eukaryotic annotation rely on sequence homology, structure, or repeat detection, and each method requires a separate tool, making the workflow for a final product a complex ensemble. The goal of this dissertation was to improve the annotation process by training a recurrent neural network (RNN) on trusted annotations to recognize multiple classes of elements.

While deep neural network tools exist for making annotation predictions on prokaryotic genomes [1], none exist for eukaryotic genomes. Such tools have been infeasible due to the greater size and complexity of eukaryotic genomes. which are both more complex and an order of magnitude larger. One important component of genetic architecture is the presence of epigenetic marks, including DNA methylation. However, no automatic annotation tools currently use this valuable information. As methylation data becomes more widely available from nanopore sequencing technology [2], [3], tools that take advantage of patterns in this data will be in demand. This dissertation evaluated the effect of including DNA methylation as an annotation indicator, and found that it increased prediction sensitivity of all element categories.

This culminating annotation tool, RNNotate, was developed to support both Python 2 and 3, and execution on CPUs and GPUs. Continuous and high-coverage testing during development ensured all output was consistent as features were added and functions were optimized. RNNotate also supports distributed execution with Horovod [4] to further reduce the time to solution for large eukaryotic genomes and to efficiently utilize modern high-performance computing clusters. RNNotate is currently available in a documented and open software repository and through the conda package manager for other researchers to easily deploy to either reproduce the findings of this dissertation or perform new analyses of their own.

In addition to these direct outcomes, this work has made several peripheral contributions to the bioinformatics community. First, the BSMAP methylation caller was optimized and parallelized to enable faster analysis. Second, programming interfaces for randomly accessing methylation data were created and published. Third, a numerical specification and programming interfaces for reading and writing annotations numerically was created. Lastly, a well documented, flexible framework for analyzing genomic data with recurrent neural networks was developed, validated on multiple types of hardware, and published for other researchers to create their own models and investigate their own hypotheses.

2. Introduction

For a novel organism to be studied genetically, the genome of that organism must be first assembled and then annotated. A genome assembly is a complete and contiguous picture of an organism's genome, ideally comprised of whole chromosome molecules. After assembly is complete, researchers can then discover which regions of DNA encode for proteins and other elements. This identification process and culminating result is genome annotation.

While most biological research is dependent on both the genome assembly and annotation to serve as reference points for making comparisons, the annotation, in particular, maps out which regions are functionally significant to the biological processes in the organism. An annotation is created by comparing the assembly sequence to known sequences of repeats and genes from related organisms. There are automated tools for performing this search, but manual intervention still results in higher fidelity, making it preferred. The manual process is extremely expensive in terms of time and expertise, so annotations are often improved over time as experts study and contribute to them.

2.1 Genome annotation

Similar to Moore's law, sequencing technology improves year after year, and third-generation long read technologies enable the assembly of eukaryotic genomes in as little as two days [5], [6]. This exponential trend can be measured by tracking the number of completed genomes archived at the National Center for Biotechnology Information (NCBI) as shown in Figure 2.1 [7]. Expanding our genome library is still relevant because genomes from new organisms expose us to novel and diverse biological functions and also allows us to observe more evolutionary patterns in greater detail.

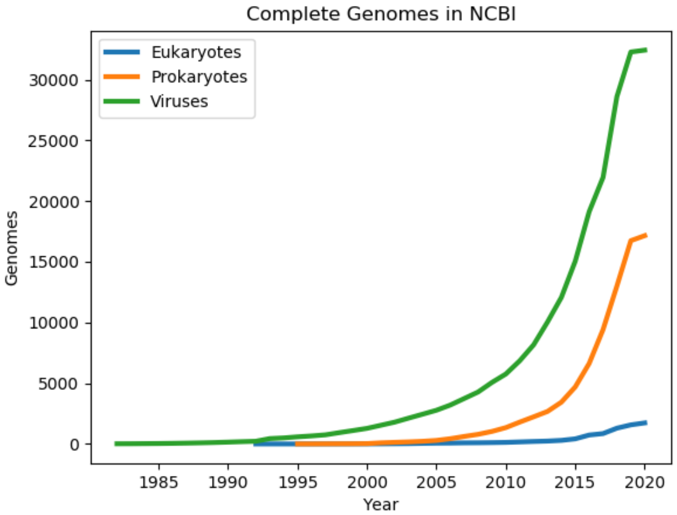

Fig. 2.1: Growth of "Complete" virus (green), prokaryote (gold), and eukaryote (blue) genomes at NCBI over time by year.

When an organism is studied at the genetic level, there are usually two prerequisites: an assembled reference genome of a trusted quality [8], and a corresponding annotation. Genome annotations are generally produced in two stages: the computational phase and then the manual annotation phase [9]. While the new long read technologies are accelerating genome assembly, the process for annotating genomes has fallen behind due to complicated computational pipelines and the manual curation bottleneck.

The computation phase attempts to make both intrinsic and extrinsic predictions of both repetitive elements and protein coding genes. The manual annotation phase was traditionally done by hand, where humans would review evidence for each predicted gene to decide on structures, but this is becoming more automated to incorporate more information and accelerate the process as software becomes more sophisticated and more data becomes available [10].

2.2 Computational annotation

In the computation phase, the annotation pipelines for both NCBI [11] and Ensembl [12] first detect and filter out repetitive elements. Masking, or removing, these repetitive subsequences reduces the overall complexity of the reference and prevents coding repeats like retrotransposons from confounding gene detection. Second, genes are predicted through numerous methods and then functionally annotated from databases of curated genes from other organisms.

2.2.1 Annotation of repetitive elements

Transposable elements (TEs) are DNA sequences that can “jump” and replicate throughout their host genome [13]. Their repetitive and prolific presence increases the difficulty of genome assembly [14], sequence alignment [15], and genome annotation [16]. The detection and classification of TEs is crucial since they comprise significant portions of eukaryotic genomes [17] and their transposition can induce large-scale genome rearrangement. Current methods to identify repetitive and transposable elements can be categorized into three main categories: homology search, structure recognition, and repeat discovery.

2.2.1.1 Homology search Homology search methods are extrinsic, meaning they depend on databases of prior information to make predictions. Tools like TESeeker [18] and RepeatMasker [15] require external databases of known repetitive elements in order to recognize those present in a genome. Since this method relies on known and curated data, it can lead to precise results. However, the dependence on prior information makes the use of this class of tool inappropriate on new genomes which may contain many novel families of transposable elements.

2.2.1.2 Structure recognition Structure recognition scans a genome for coding sequences structured like specific types of transposable elements (DNA-only transposon, retroviral-like retrotransposons, and non-retroviral retrotransposons) [19]. For example, LTR STRUC scans a genome and recognizes a coding pattern specific to LTR retrotransposons [20]. Since structure recognition tools identify transposable elements based on specific patterns or products to enable mobility, they are suitable for application to novel genomes. However, since they rely on recognizing the open reading frames of coding products, genes are often incorrectly reported as transposable elements.

2.2.1.3 Repeat discovery Repeat discovery methods like those of RepeatScout [16] detect highly repeated novel transposable elements based on frequency and repeated patterns. This method is best suited for the *de novo* identification of transposable elements in novel genomes since classifications are based on outlying patterns in the input genome. While generally applicable, this method is susceptible to reporting false-positives from non-transposing repeats (tandem repeats, segmental duplications, and satellites).

2.2.2 Annotation of genes

After repetitive elements are detected and masked, genes are annotated through assembled transcript alignment, RNA-evidence, and structure prediction. All three methods are intrinsic and can detect coding sequences based on how DNA is transcribed into RNA and the rules programmed into their models, without curated data sources.

2.2.2.1 Assembled transcript alignment The annotation method of assembled transcript alignment begins with the deep sequencing of RNA from an organism. These sequencing reads are then assembled into contiguous gene transcripts [10], [21]. The assemblies for each of these can then be aligned back to the genome assembly of DNA to reveal the genes and their individual exons. Since this protocol directly samples genes, it is the most trusted and popular method of gene annotation, but it does require that the genes be expressed. The requirement of deep sequencing requirement also makes the method cost-prohibitive, especially since the genome assembly already required extremely deep sequencing coverage.

2.2.2.2 RNA-evidence While not as exact as transcript alignment, genes can also be annotated through direct RNA-evidence without assembly. In this method, the RNA from an organism is sequenced at a relatively low coverage depth and then directly aligned back to the genome annotation. The genes can then be annotated based on differences in coverage by tools like Cufflinks [22] and ESTAnnotator [23]. While this method is lower in cost and computation complexity than assembling transcripts, it still requires that a gene be actively expressed for annotation.

2.2.2.3 Structure prediction The final, and least popular, method for gene annotation is structure prediciton. This is a *de novo* method that requires no additional sequencing and makes predictions purely from the genome reference sequence. Structure prediction Tools such as GlimmerHMM [24], GeneMark-ES [25], and Augustus [26] use probabilistic models to detect genes by recognizing valid open reading frames based on how nucleotides code for amino acids during translation. This method category returns the most false-positives, but can be run on any genome assembly, making it the most cost-effective. Unlike the other two methods, structure prediction works on DNA, so it does not require that a gene be actively expressed.

2.2.3 Ensemble methods for annotation

Each of the mentioned annotation methods and tools have their own strengths and weaknesses, so most annotation protocols utilize multiple methods and aggregate the results to either improve the confidence of consensus classifications or filter out false positives. Many tools have also been developed as top-level orchestration scripts that run multiple classifiers in the background [10], [27], [28]. While these tools take the effort out of decoding and combining the results from multiple tools, they are often difficult to install, maintain, and use due to the number of required software dependencies. For instance, REPET has 13 sub-tool dependencies, each with their own library dependencies. To improve the usability and reproducibility software, the bioinformatics community has built over 7000 packages in the Bioconda package repository. Even though REPET is one of the most trusted methods for TE annotation, it is not present in Bioconda due to the difficulty of its installation.

2.3 Sequence tagging with machine learning

Machine learning with neural networks has become accessible through standardized libraries [29]–[32], and research has demonstrated that these models can make inferences from whole collections of data which humans often miss. New tools comprised of deep neural networks have recently arisen to also improve the annotation of genomes. DeepAnnotator uses a recurrent neural network to detect genes in prokaryotic sequences 100 bases at a time [1]. DeepRibo incorporates ribosome profiling signals to improve the identification of genes [33]. DeepTSS is a convolutional neural network designed to detect transcription start sites in 299 base pair sequences [34]. Da Cruz et al. created a convolutional neural network to classify the order and superfamily of transposable elements [35]. A recurrent neural network (RNN) has also been developed to classify RNA sequences as either coding or non-coding [36]. However, none have attempted eukaryotic genome annotation. Our tool, RNNotate, begins this work and explores the usage of a recurrent neural network to take genomic input to produce an annotation.

While the standard signal for detecting actively transcribing genes is RNA-seq, there are many other sequencing protocols for detecting other targets. Bisulfite-sequencing is a protocol to reveal DNA methylation, a type of epigenetic modification to DNA. In plants, DNA methylation is the addition of a methyl group to a cytosine making 5-methylcytosine [19]. This means that not all cytosines are equal, adding a new dimension for genomic exploration. Previous studies of DNA methylation have also shown that different classes of genetic elements have their own unique DNA methylation signature, and both the mechanism and pattern of DNA methylation has been found to differ between euchromatin and heterochromatin [37]. Euchromatin is associated with physically accessible gene sequences on the chromosome arms and heterochromatin is repeat-heavy and densely packed into the centromere and other physical features like knobs [38], [39]. It has also been found that mCHH islands enforce boundaries between active chromatin around genes and transposons in maize [40]. This work explores the effect of including DNA methylation as an additional dimension to facilitate whole genome annotation predictions for both genes and transposable elements.

3. Methodology

3.1 Collecting requirements and building a specification

The tool has been designed in such a way that it can be trained on the annotations from multiple organisms, and then generate a valid general feature format (GFF3) annotation file which can be consumed by other tools [41]. Since the goal of this project is a model to predict known features, a vocabulary and specification was predefined for both the input and output after surveying the data supplied in reference (FASTA), methylation (BAM), and annotation (GFF3) files from trusted data archives: NCBI [11], Ensembl [42], and Phytozome [43].

3.1.1 Input Specification - Unclassified

Recurrent models in Keras and Tensorflow 1.14+ expect time series input, where each timestep has the same data structure [30], [44]. This requires that any descriptive information be encoded into each timestep instead of existing once in a data header. Each input sequence is a contiguous region from the reference, where the nucleotide and methylation data have been encoded into a 2-dimensional numerical format at the base pair level.

We included four dimensions from the FASTA reference sequence: a single-base nucleotide code to communicate the sequence (Table 3.1), the relative location along the chromosome for location based trends, the genome ploidy to account for heterozygosity, and the assembly quality since scaffolds will have a different landscape and fidelity from whole chromosome molecules. The specification for these categories can be seen in Table 3.2.

Tab. 3.1: The numerical nucleotide mapping code for encoding sequences into a numerical format for use by model.

Nucleotide	**Numerical Code**
A	0
G	1
T	2
C	3
N	4
Other	4

Previous studies have used k-mer based vocabularies to detect reading frames [1], but in traditional natural language processing research, there is a trade-off between processing individual characters and whole words (k-mers). Character-level models are larger, but they are capable of learning subtler rules that word-based models miss [45]. Initial toy-scale prototypes were able to detect valid reading frames from single nucleotide characters, so we continued with them in hopes that enough subtle rules could be learned to accurately predict a useful annotation.

The methylation data at each base pair location consists of six values: the frequency and read count for each of the three methylation context in plants CG, CHG, CHH. The methylation frequency is a floating-point number in the range [0, 1], and the read count, with range [0, inf], conveys a measure of certainty for the frequency value through sequencing depth. The read count does not require normalization for separate samples since uncertainty at a single base is independent of the read count at the next.

Tab. 3.2: RNNotate input specification detailing the order, type, and range for each dimension of the input.

Index	Data type	Range	Description
0	uint-4	[0, 4]	Nucleotide code (Table 3.1)
1	float-32	(0, 1]	Fraction of chromosome (first character is 1/len)
2	float-32	[0,1]	CG methylation ratio
3	uint-8	[0, 256]	Number of CG reads
4	float-32	[0,1]	CHG methylation ratio
5	uint-8	[0, 256]	Number of CHG reads
6	float-32	[0,1]	CHH methylation ratio
7	uint-8	[0, 256]	Number of CHH reads
8	uint-4	[0, 16)	Ploidy
9	uint-4	[0, 3]	Assembly Quality: (Unkonwn, Contig, Scaffold, Chromosome)

3.1.2 Output Specification - Training and Predictions

The model is trained using supervised methods, so the output specification not only defines what can be predicted, but also what can be learned. To determine the necessary vocabulary to encode a variety of organisms, annotations from the archives Ensembl [12] and Phytozome [43], and projects like TAIR [46] and Gramene [47] were surveyed to compound a list of common elements and metadata to consistently support the analysis of new data. The full list of 64 features chosen to be recognized by RNNotate can be seen in Table 3.3. In addition to common features like exons, rare features such as tmRNA were included in case they had distinct characteristics that set them apart from standard featureless DNA.

All annotations from these data archives were in the GFF3 format, which is the standard file format for storing features. The GFF3 format is a tab-delimited text file expressing the coordinates and metadata of a single feature interval per line. The features in a GFF3 file may not only overlap, but also be dependently linked into a hierarchy such as multiple exons belonging to a gene. To reduce the complexity of the output specification, the dependency information was ignored and only the location and feature type was tracked. To allow for a single base to belong to multiple overlapping features, like the previous gene and exon relationship, the classification of each position is represented categorically with a binary vector. The coding strand is captured by representing each category twice in the binary vector, the first for the forward strand, and the second for the reverse as shown in Table 3.3. Since this binary categorical specification is limited to single gene on each (+/-) strand at each base, only the longest version is represented instead of each alternative splicing model. This rule applies to all features of the same category that overlap. Several of the categories in the binary categorical vector are types of repetitive elements, two additional numerical values (Table 3.3) were included to track their sub-classifications.

The survey of annotation archives found that both Ensembl and Phytozome strip repetitive elements from their annotations [12]. While this hinders the study of transposable elements, it makes downstream analyses like gene annotation and alignment easier. Luckily, they are present at the project level in both TAIR, Araport11 [48], and Gramene. Since different projects utilized different workflows to classify repetitive elements, there were discrepancies in their naming. Translation tools were created and included with RNNotate to decode project-specific names into a universal set, which can be adapted for future organisms. Tools currently exist to predict both the order

Tab. 3.3: Annotation features type tracked by RNNotate. The first two columns denote the feature's strand-specific indices, the third column tracks the data type, and the fourth column provides a description. The final two rows enable sub-classification levels of TE related features (bold), which are numerical and unstranded.

+ Index	- Index	Data type	Description
0	32	bool	CDS - Coding sequence
1	33	bool	RNase_MRP_RNA
2	34	bool	SRP_RNA - Signal recognition particle
3	35	bool	antisense_RNA
4	36	bool	antisense_lncRNA
5	37	bool	biological_region
6	38	bool	chromosome
7	39	bool	contig - Originates from a contiguous region
8	40	bool	exon
9	41	bool	five_prime_UTR
10	42	bool	gene
11	43	bool	lnc_RNA - Long non-coding RNA
12	44	bool	mRNA - Messenger RNA
13	45	bool	miRNA - MicroRNA
14	46	bool	ncRNA - Non-coding RNA
15	47	bool	ncRNA_gene - Does not encode proteins
16	48	bool	pre_miRNA - Drosha processing remnant
17	49	bool	pseudogene
18	50	bool	pseudogenic_exon
19	51	bool	pseudogenic_tRNA
20	52	bool	pseudogenic_transcript
21	53	bool	rRNA - Ribosomal RNA
22	54	bool	region - Genomic region
23	55	bool	snRNA - Small nuclear RNA
24	56	bool	snoRNA - Small nucleolar RNA
25	57	bool	supercontig - Contigs combined into scaffolds
26	58	bool	tRNA - Transfer RNA
27	59	bool	three_prime_UTR
28	60	bool	tmRNA - Transfer messenger RNA
29	61	bool	**transposable_element**
30	62	bool	**transposable_element_gene**
31	63	bool	**transposon_fragment**
	64	uint-8	transposable_element Order
	65	uint-8	transposable_element Superfamily

and superfamily of repetitive elements and both of these subcategories were present in the Araport11 and Gramene annotations. While TEs are more frequent in areas like the heterochromatin, these predictions often do not overlap, but are interleaved as shown in Figure 3.1.

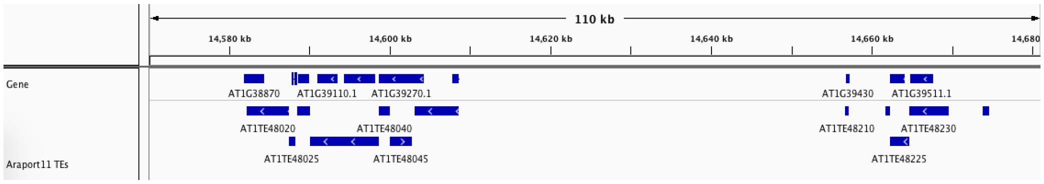

Fig. 3.1: Genome browser view of interleaved TEs in the Araport11 annotation.

This rule may not apply to the entire genome, and the landscape may be an artifact of the tools used to generate the annotation. Since investigating or resolving these occurrences is beyond the scope of this work, only two additional numerical fields were included in the output specification to encode for them (Table 3.3). This means there are top-level stranded TE (+ and -) categories, along with integer values that encode for a common set of orders (Table 3.4) and superfamilies (Table 3.5).

3.1.3 Data Shape - Input and Output

Depending on the model architecture, recurrent neural networks may require both the input and output data shapes down to the batch size be statically defined at runtime when the neural network is compiled. To process multiple sequences in parallel, input sequences of shape [S, I] are batched together into 3-dimensional matrices of shape [B, S, I]. Where B represents the number of sequences in each batch, S represents the base pair length of each sequence, and I represents the dimensionality (fields) of the input at each base pair - 4 reference and 6 methylation values totalling 10. The

Tab. 3.4: Numerical codes used in index 64 of output specification (Table 3.3) for collected TE orders.

Value	**Name**	**Description**
0	Unassigned	Order was not specified or is unknown
1	DNA	Dna transposon
2	LINE	Long interspersed nuclear repeat
3	LTR	Long terminal repeat
4	Low_complexity	Tandem repeats, polypurine, and AT-rich regions
5	RC	Rolling circle replication
6	Retroposon	Repetitive DNA reverse transcribed from RNA
7	rRNA	Ribosomal RNA
8	Satellite	Large arrays of tandemly repeating DNA
9	Simple_repeat	Micro-satellites
10	SINE	Short interspersed nuclear elements
11	snRNA	Small nuclear RNA
12	TIR	Terminal inverse repeats
13	tRNA	Transfer RNA

corresponding output will also have shape [B, S, O], where O is the dimensionality of the output at each base pair (64 + 2 = 66).

In the event TE sub-classifications and low-frequency features are too sparse for the model to effectively learn, two separate options for respectively excluding each group were implemented. When the non-binary TE sub-classifications are excluded, the output dimensionality is reduced by 2 (64), and the model switches to the steeper binary cross-entropy loss function. When low-frequency features are excluded, only the features listed in Table 3.6 are predicted, reducing the output to dimension to 8, or 10 (8+2) when TE sub-classifications are included.

Tab. 3.5: Numerical codes used in index 65 of output specification (Table 3.3) for collected TE superfamilies.

Value	**Name**
0	Unassigned
1	Cassandra
2	Caulimovirus
3	centr
4	CMC-EnSpm
5	Copia
6	En-Spm
7	Gypsy
8	HAT
9	hAT-Ac
10	hAT-Charlie
11	hAT-Tag1
12	hAT-Tip100
13	Harbinger
14	Helitron
15	L1
16	L1-dep
17	Mariner
18	MuDR
19	MULE-MuDR
20	PIF-Harbinger
21	Pogo
22	RathE1_cons
23	RathE2_cons
24	RathE3_cons
25	Tc1
26	TcMar-Mariner
27	TcMar-Pogo
28	TcMar-Stowaway
29	tRNA
30	solo

Tab. 3.6: Features predicted by RNNotate when low-frequency features are excluded. Bold features can still allow the prediction of TE sub-classifications.

+ Index	- Index	Data type	Feature
0	8	bool	CDS
1	9	bool	exon
2	10	bool	five_prime_UTR
3	11	bool	gene
4	12	bool	mRNA
5	13	bool	three_prime_UTR
6	14	bool	transposable_element
7	15	bool	transposable_element_gene

3.2 Developing data access interfaces

The final model will need to be trained on subsequences that cover entire genomes multiple times. Most bioinformatics formats assume that files will be streamed from start to finish, which makes the selection of specific regions computationally expensive. This characteristic requires the development of efficient random-access interfaces to reference, bisulfite, and annotation data.

3.2.1 FASTA access

The FASTA format can be indexed to allow for efficient random access from disk via samtools [49] or other libraries [50], but sequential region access was improved by caching large regions in memory, which is updated infrequently. Random search patterns fall back to separate read requests to the filesystem, so the large cache is not constantly updated. This interface also translates each possible nucleotide code from the FASTA format specification into a numerical value for the model so the transformation happens early, and only a reduced representation of the data is held in memory.

3.2.2 Methylation alignment and access

Data produced by the Bisulfite protocol looks identical to data from DNA sequencing [51], so both the alignment process and quality control steps are similar. For this work, Bisulfite sequencing reads were aligned to their respective reference using BSMAPz [52], and only properly-paired, uniquely mapping reads were kept for calling methylation frequencies. While the mapping process has been able to adapt and borrow from performant DNA aligners, the methylation callers that aggregate the alignment records into site-specific methylation ratios were single-core and memory intensive. To ensure that methylation results from this study could be quickly reproduced without requiring large-memory hardware, the methylation caller for BSMAP was optimized to efficiently call methylation frequencies from read alignments in parallel [53]. Since methylation can only take place at specific sites (CG, CHG, CHH, where H = C,A,T), methylation callers produce sparse tabular output which lists the frequency a cytosine is methylation by location. The format is both legible and browsable, but randomly accessing specific points or whole regions is computationally expensive. A new methylation file format and API, Meth5py [54], was created to losslessly store methylation calls in fixed-width, sparse, compressed, and indexed data structures for efficient seek and read operations. The Hierarchical Data Format (HDF5) format was ideal for this workload since each chromosome data matrix can be stored as a separate dataset, while compression and caching are handled behind the scenes by the library itself [55]. Once converted into this new format, regions of methylation data can be extracted and converted into the numerical format required by the model with a 52.6% space savings (Table 3.7).

Tab. 3.7: Size comparison between normal and Meth5py files. Original size in megabytes (MB), Meth5py size in MB, and space savings in percent.

File	Original (MB)	Meth5py (MB)	Savings (%)
th.bam.mr	2098	976	53
th_M1-C_Rt.bam.mr	2038	960	53
th_M1-C_St.bam.mr	1909	921	52
th_M2-C_Rt.bam.mr	2029	956	53
th_M2-C_St.bam.mr	1949	933	52

3.2.3 Annotation access

The format that the annotation, or target, data will be presented to the model has been defined, but this is a dense format which maps well to the base pair level recurrent neural network. Most genomes are functionally sparse, with elements that can span thousands of base pairs when they do occur. Simply converting an annotation into this dense format and holding it in memory would allow for the fastest processing, but the space trade-off is expensive since every base has 66 values. Two alternatives are storing the dense format on disk in HDF5 format, similar to the methylation data, or storing it in chromosome-level interval trees.

Converting the data once to an HDF5 format means that a target region simply needs to be queried and then presented to the model. No additional conversion would be necessary, but the data would be coming from disk with latency. An interval tree is a data structure designed to store intervals and efficiently answer which intervals overlap with a given point or range query. Using such a structure would allow the annotation to efficiently live in memory, but selected intervals would need to be converted to the dense format before going to the model.

To determine which method would be best, both formats were developed. The TAIR10 annotation was converted to each format. The HDF5 arrays were stored on solid state disk (SSD) and the interval trees for each chromosome were held in memory. Regions ranging from 100 to 10,000,000 base pairs in length were randomly chosen and dense model arrays were generated from each data structure. The results in Figure 3.2 showed that the interval tree structure was always faster than the HDF5 format for all lengths even though it required an additional decoding step to the dense format.

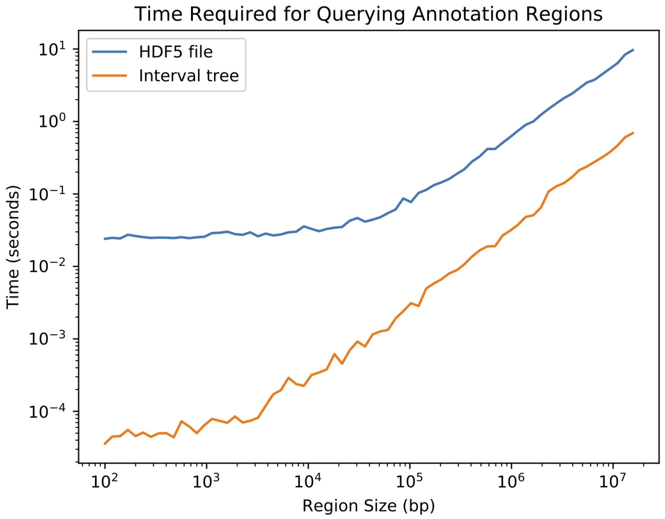

Fig. 3.2: The performance of querying annotation data for increasing region sizes was compared between HDF5 (blue) and the interval tree format (yellow).

3.3 Developing a flexible model

While there are recommended practices and models proven capable for certain workloads like image classification or sentiment analysis, there is little guidance for learning entire genomes. Model development began around a recurrent neural network to tag, or annotate, the input data. Since it was impossible pre-determine the best architecture for this research, the model was designed to be extremely flexible for experimentation through trial and error. Since hundreds to thousands of models will need to be tested for different research questions, structure configuration takes place on the command line interface (CLI) at runtime instead of requiring that new code be tediously generated for each test. While the model will have numerous architectures, they all conform to the following structure: an optional convolutional layer of variable width, a section of recurrent layers (purple in Figure 3.3), an optional section of hidden layers (green in Figure 3.3), and a final output section for generating the correct output dimensionality (yellow in Figure 3.3).

3.3.1 Convolutional section

To improve the chances of recognizing short patterns such as coding frames or microsatellites, the input can first be transformed through a convolutional neural network (CNN) layer [56]. The CNN neurons use a rectified linear unit (ReLU) activation and a convolution width chosen at runtime, which both require tuning depending on the target. A pooling layer was not used after the convolution since the model needs to generate a single prediction for each input base instead of a single tag or description for a collection.

3.3.2 Recurrent section

The recurrent section is designed to be extremely flexible to give the model as much power as possible while also being efficient. Each input point is processed by

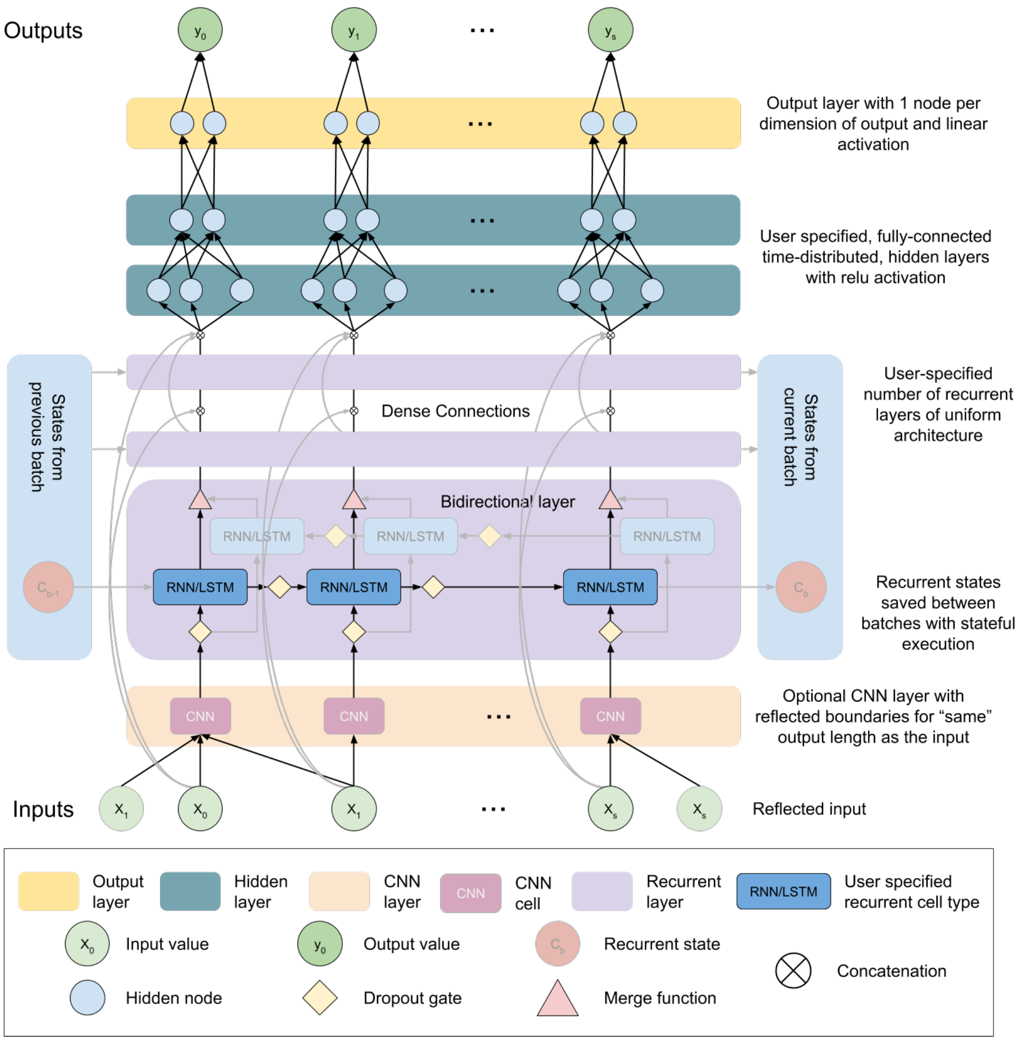
Outputs
y0
y1
...
ys
Output layer with 1 node per dimension of output and linear activation
User specified, fully-connected time-distributed, hidden layers with relu activation
States from previous batch
Dense Connections
States from current batch
User-specified number of recurrent layers of uniform architecture
Bidirectional layer
RNN/LSTM
Cb-1
Cb
Recurrent states saved between batches with stateful execution
CNN
Optional CNN layer with reflected boundaries for "same" output length as the input
Inputs
x1
x0
x1
xs
xs
Reflected input
Output layer
Hidden layer
CNN layer
CNN cell
Recurrent layer
User specified recurrent cell type
Input value
Output value
Recurrent state
Concatenation
Hidden node
Dropout gate
Merge function

Fig. 3.3: Illustration of model architecture possibilities. Data flows from the inputs at the bottom upward to the outputs at the top. The optional convolutional layer with "same" padding and variable width (3 shown) is represented in **orange**. The optional stateful recurrent states are represented by the **blue** regions flanking the **purple** recurrent section. The optional bidirectional layer is represented by the semi-transparent flow in the **purple** recurrent layer. The configurable hidden layers are represented in **green**. The final output layer is represented in **yellow**.

either an RNN, LSTM, or GRU cell [57] and utilize GPU-accelerated versions when appropriate hardware is detected by TensorFlow. Each cell also has a configurable number of internal neurons with hyperbolic tangent (tanh) activation functions. The chain of recurrent cells can process input batches either independently or statefully, and sequences either in the forward direction or bidirectionally, which have different strengths (Table 3.8). Independent RNNs process sequences as sets of dependent inputs, but multiple sequences can be processed in parallel very quickly. Stateful RNNs circumvent the sequence length limit imposed by vanishing-gradients by passing the cell state between batches at the cost of concurrency.

Tab. 3.8: Strengths and weaknesses of different ways to process batches (Independent, Stateful) and sequences (Forward, Bidirectional).

RNN Type	**Pro**	**Con**
Independent	Fast	Inappropriate for long patterns
Stateful	Ideal for long sequences	Unable to process long sequences in parallel
Forward	Requires fewer cells	Unable to use information later in the sequence
Bidirectional	Access to future information	Slower and unable to be stateful

In standard RNNs, independent and stateful, sequences are processed sequentially by input unit, so information only flows from past to future. Bidirectional RNNs are another RNN architecture type that can pass information from future timepoints back to earlier times in a sequence (Figure 3.3) by processing a sequence both forward and backward. To help prevent overfitting during training, both recurrent dropout and several types of regularization can also be configured in the recurrent layer. Once the recurrent architecture has been defined, multiple layers can be stacked like a multilayer perceptron to increase the overall power.

The recurrent section can also be repeated and densely connected to potentially learn complex patterns and relationships. Normally, deep models are limited by vanishing gradients, but input and output signals are passed to these repeated sections with dense connections inspired by DenseNet [58]. These connections concatenate the original input to the beginning of each recurrent section, along with broadcasting the output of each recurrent section forward as shown in Figure 3.3. This type of architecture allows each recurrent section to learn separate patterns since they all have access to the original input, while also having the option to reuse the output from previous sections.

3.3.3 Hidden section

After the recurrent section, multiple dense, hidden layers of variable neurons with tanh activation can be defined. These layers are time-distributed and applied independently to each RNN timestep. Lastly, the final output layer is always a time distributed hidden layer with a neuron for each output dimension and either linear activation to allow for non-binary output values when classifying TE sub-classification or sigmoid for all binary outputs if not.

3.3.4 Loss functions

By default, the model is evaluated using the mean squared error (MSE) function to allow for both multiple classification labels and the integer values representing singularity TE order and super families in the output. When the TE sub-categories are excluded from the output, the steeper binary cross-entropy (logloss) loss function is used for evaluating the model during training. Since the binary cross-entropy function only expects 0 and 1 output values, the final output hidden layer also had to be from linear to sigmoid activation. While stratifying transposable elements into separate classification bins theoretically yields more information, the binary cross-entropy loss function is much steeper than the mean squared error function as demonstrated in Figure 3.4 and may accelerate convergence.

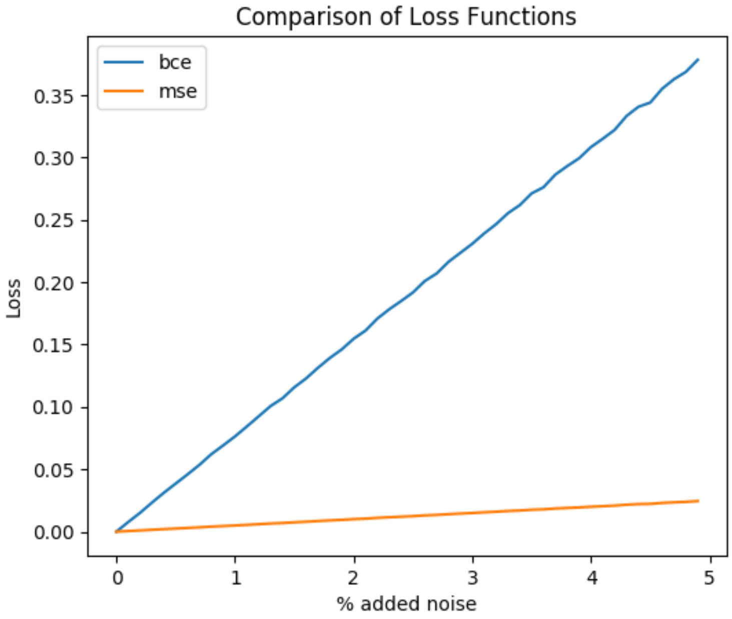

Fig. 3.4: Starting from comparing a binary matrix to itself, a percentage of total values are randomly changed to 0 or 1, and compared to the original with the binary cross-entropy (blue) and mean squared error (gold) loss functions.

3.4 Data traversal methods

For the model to predict the classification of each base at least once, the entire genome must be passed through the model at least once. Physical memory limits and vanishing-gradients prevent whole chromosomes from being processed as singular input sequences. These limitations required that the chromosomes be segmented and individually classified. Since the recurrent layers in this model support stateful execution, there are two traversal methods available: independent and stateful batches.

3.4.1 Independent traversal

Independent batches are generated in a sliding window method. A window of the specified sequence length will slide a specified number of offset bases as shown in Figure 3.5. To prevent bases from accidentally being skipped, this slide distance must be less than or equal to the sequence size. Initially, each region in a batch was retrieved individually, but this bottlenecked the analysis since large batches with significant overlap were targeted with independent traversal. For i/o efficiency, a single long region is fetched from disk for each batch. This long region is then reshaped, without modifying the actual memory, into a new view of the correct sequence length and offset. In the case of distributed execution, "worker-1" will process the first batch, "worker-2" will process the second batch, and so on.

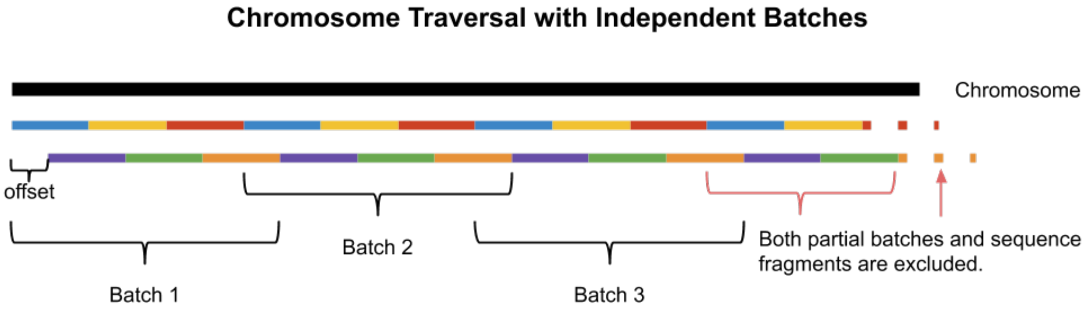
Chromosome Traversal with Independent Batches
Chromosome
offset
Batch 1
Batch 2
Batch 3
Both partial batches and sequence fragments are excluded.

Fig. 3.5: Illustration of independent batch traversal. The different sequence colors are only used to visually separate subsequences. Batches can contain any number of sequences.

3.4.2 Stateful traversal

With stateful execution, subsequences of a huge sequence are consecutively processed with state information flowing forward through time. The simplest way to statefully traverse a chromosome is the “online” method, where the first batch contains a single sequence for bases 1-100, the second batch contains a single sequence for bases 101-200, and so on until the end of the chromosome. The other is to divide the main sequence into multiple sequences which are each statefully processed as shown in Figure 3.6. To enable some robustness to the classification, stateful sequences overlap with their neighbor by 50%. An example of this would be taking a sequence of 99 bases and processing it in batches of 2 with the ranges 1-66 and 34-99, where the two sequences overlap over the range 34-66. When this type of traversal is distributed, the original batch size is divided amongst workers so the total batch size does not increase, which would increase the number of subdivisions and shorten the dependence chain.

Since stateful batches are spread across such long regions, it was not possible to retrieve the necessary data with a single read and then reshape it as with the independent batches. To prevent the analysis from bottlenecking during stateful traversals, multiple data-retrieval workers are used to accelerate data retrieval and transformation.

3.4.3 Stranded traversal

This current method processes chromosomes in the forward direction only. Predictions are generated for both the forward and reverse strand, but the model will

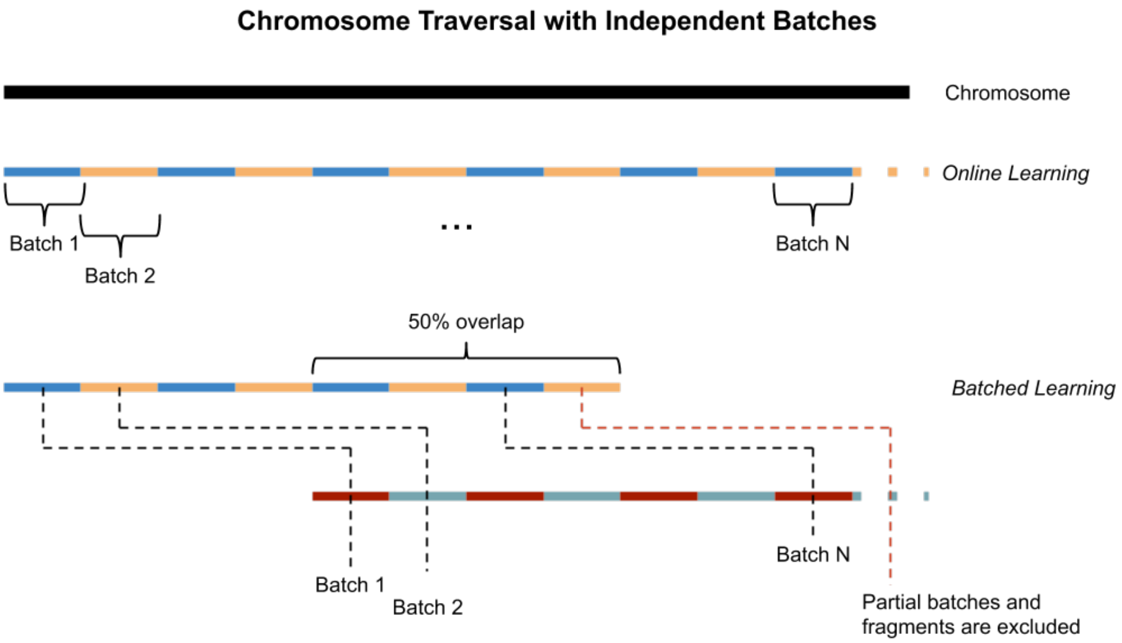

Fig. 3.6: Illustration of stateful batch traversal. Below the chromosome line, is an example of online stateful learning with one sequence per batch. The chromosome in the *Batched Learning* example is processed with a batch size of 2, so it is traversed in 2 subsequences with 50% overlap.

need to not only learn to predict the structure of a gene in the forward direction based on the input data,

`5'UTR -> ORF -> 3'UTR`

but the reverse direction as well.

`3'UTR <- ORF <- 5'UTR`

To allow the model to apply the same rules to coding elements irrespective of their origin, a stranded traversal method was implemented.

The stranded traversal begins with the same DNA and methylation input as an unstranded one, except target data from the reverse strand of the annotation is masked. This means that only features originating from the forward strand can be learned from the forward traversal along the chromosome. After processing the entire forward direction of a chromosome, data is then generated in the reverse direction, where the last coordinate is now the first. The queried nucleotide sequence is also reverse complemented, and all training data from the annotation is masked on the forward strand. Once again, this forces all features originating from the reverse strand be predicted during the reverse traversal. Figure 3.7 illustrates and compares this process to a standard forward-only traversal. To prevent the model from being biased towards a specific strand because it was always learned last, the batches from each strand were interleaved for an even distribution.

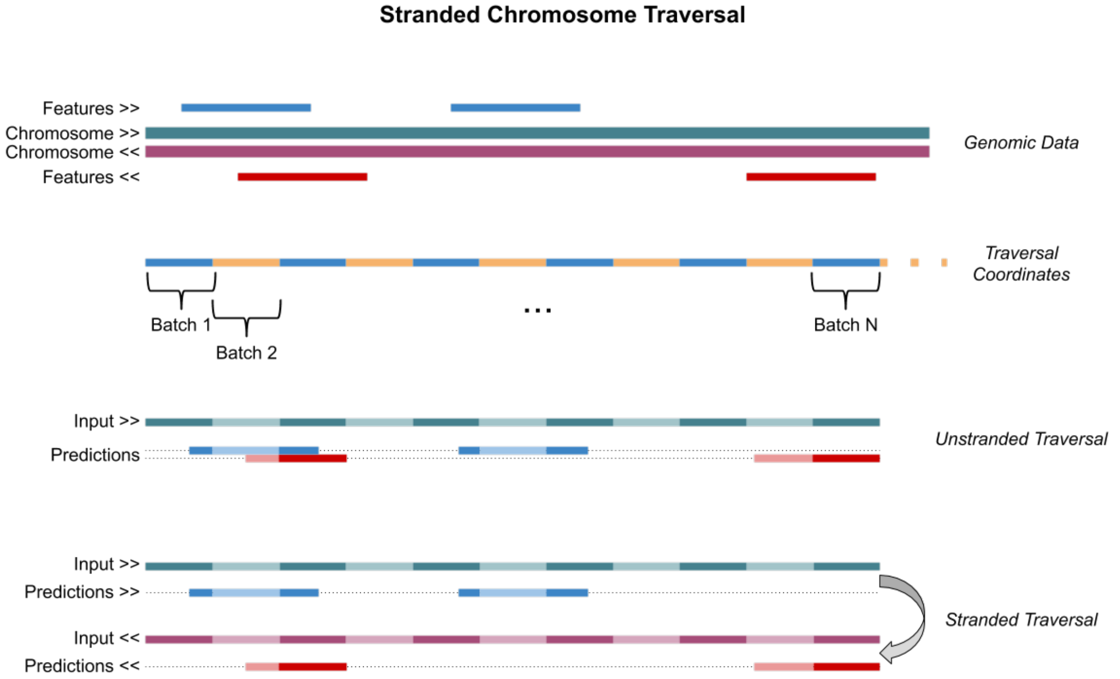

Fig. 3.7: The “Genomic Data” row illustrates both stranded genomic data, and annotation features. The “Traversal Coordinates” row illustrates discrete sections of data. The “Unstranded Traversal” row demonstrates what information is present in each data batch. The “Stranded Traversal” row demonstrates how a chromosome is processed and which strands of annotation data are present.

3.5 Aggregation and decoding of output

With the exception of online traversals, the model will produce overlapping predictions. These overlapping predictions are tracked so the final classification of each base comes from multiple predictions, which can be aggregated for a more robust result. Due to the characteristics of independent and stateful batches, two different aggregation methods were designed for optimal classification power.

Since the predictions from independent batches are also independent, any overlapping predictions from the input traversal are aggregated to filter out marginal results. To ensure that this aggregation step does not interfere with the model residing in memory, results are stored on disk in HDF5 format. The datasets are used to track the occurrence of each feature type along with a total count since non-coding regions do not have an explicit class in the specification and will not increase a feature count. After all batches have been processed, the aggregation arrays are crawled base by base to assemble contiguous regions. Since each base can be apart of multiple features in the annotation, simply choosing the most popular category with arguments of the maxima (argmax) is not used. Instead, a category is output if its fraction of the total is above a specified threshold as demonstrated in Table 3.9.

Tab. 3.9: Example independent classifications. Given category counts at 3 indices, output classifications are demonstrated for two thresholds (t): 0.5 and 0.4.

Index	**Category**			**Total**	**Classification**	
	A	**B**	**C**		**t=0.5**	**t=0.4**
0	3	0	0	6	A	A
1	3	3	0	6	A,B	A,B
2	3	3	4	7	C	A,B,C

Stateful batches are handled differently from independent batches because they make predictions based on the information from the previous batch, and this prior information, makes later predictions more trustworthy than early ones. Because of this difference, no proportion method is used, and early predictions are overwritten with more trustworthy classifications from later predictions. To prevent the aggregation data from bogarting model memory, HDF5 files are used to efficiently store data on disk.

Once prediction is finished, the HDF5 files are crawled, and contiguous regions are tracked and stored in a feature list. These feature lists are then scanned, and smoothed by filling gaps smaller than 50 bases and removing features smaller than 50 bases. This process is analogous to removing salt and pepper noise from an image. After smoothing each feature category, data is finally decoded into the standard, tab-delimited GFF3 format, with the following columns (Table 3.10).

Tab. 3.10: Final output GFF3 specification in tabular format. Each row represents a column in the GFF3 tab-delimited format. Columns 1, 4, 5, and 7 describe the location of a prediction. Column 2 designates the source of the prediction. Column 3 is the feature type, and corresponds to a category in Table 3.3. Columns 6 and 8 correspond to score and phase, which are not used, and left empty with ".".

Column	**Information**
1	Chromosome
2	RNNotate (origin of prediction)
3	Feature type
4	Start coordinate
5	End coordinate
7	Strand
9	Attributes: unique ID, TE order, TE superfamily

3.6 Data sources

RNNotate was run on open data from A. thaliana for reproducibility and accessibility. Input data came from both the Araport11 reference assembly [48] and bisulfite-sequencing reads from wild-type stem and root samples from study PRJEB6701 [59] for DNA methylation values. The reads were aligned with BSMAPz using the ZED protocol [60], which requires a base-quality of 20, excludes repeated equal-quality hits, uses random seed 77345 for reproducibility, and considers up to 10,000 seeds per read. After alignment, samples were merged to maximize coverage and minimize sampling uncertainty. Methylation frequencies were then called using properly-paired reads with unique alignments. For training and evaluation, the model uses the Araport11 annotation, and transposable element identities from TAIR10 [46].

3.7 Determining optimal model architecture

The model architecture was specifically designed to be flexible so an optimal architecture could be discovered for this research. The only way to identify the optimal architecture is by comparing results from numerous configurations. The model returns loss values during training, but these are only suitable for debugging for two reasons. First, the model currently supports two different loss functions, mean squared error and binary cross-entropy, whose values are incomparable. Second, a single loss value is calculated for a whole batch. This is fine for small independent batches of data, but stateful batches span large regions, so the loss metric will not represent a local region.

3.7.1 Collecting local error metrics

To get an accurate measure of error for an entire chromosome, the model pauses training and, using the same data traversal, makes predictions on the data. Then, regardless of the actual loss metric used for training, the mean squared error is calculated for each output sequence, and temporarily stored. The error is then averaged in non-overlapping 1 kilobase windows and stored in tab-separated values (TSV) format, which can be plotted as shown by Figure 9.2.1. By default, the model does this every 5 epochs during the training process, but this frequency can be configured at runtime for different levels of learning resolution.

3.7.2 Hyperparameter sweep of model

Initial testing found independent batches, regularization, and short sequences to perform poorly, so the hyperparameter sweep will explore the 648 permutations of arguments in Table 3.11. Each permutation will be trained on *A. thaliana* Chromosome 1 and 2, and validated on Chromosome 3 for an unbiased performance estimate. Data was split into 512 base sequences, and 608 sequences per batch. Each model was then trained over 20 epochs, and evaluated every 5th epoch.

After training, the performance of each permutation was evaluated by the median and interquartile range (IQR) from chromosome-wide MSE values for both the training and validation chromosomes - yielding 4 summary values per model. Both the median and IQR are robust metrics which convey the center and spread of the chromosome error distributions. The median is the 50th percentile of the error distribution, and represents the most likely value without being affected by extreme outliers. The

Tab. 3.11: Architecture parameter permutations for testing.

Parameter	**Arguments**
CNN width	0, 3, 5, 6
LSTM neurons	32, 64, 128
LSTM layers	1, 2, 3
Learning rate	0.01, 0.001, 0.0001
Dropout rate	0, 0.3
Dense blocks	0, 2, 4
Total Permutations	**648**

IQR is the difference between the 75th and 25th percentiles, and describes the spread of half the data while also being robust against outliers.

The collected median and IQR values from the training and validation data were used to create a two-dimensional comparison plot to highlight argument effects and trends as shown in Figure 3.8. The comparison figure will be 2-D with half of the parameters represented on the y-axis, and the other half represented on the x-axis. Each circle will represent a model, and poor performing models with large IQRs or medians will blend in with the background by being small and similar to the background.Tables can be sorted to show specific ranks, but a scatter plot such as this highlights superior performance while also grouping parameters visually.

3.8 Final training

Once an effective model architecture was identified, it was trained on the full data collection, *A. thaliana* chromosomes 1, 2, 4, and 5, for 200 epochs. The comparison tests were trained on a subset of the data for 20 epochs for a relatively quick comparison of how each model learned, while this training session allows a model to reach

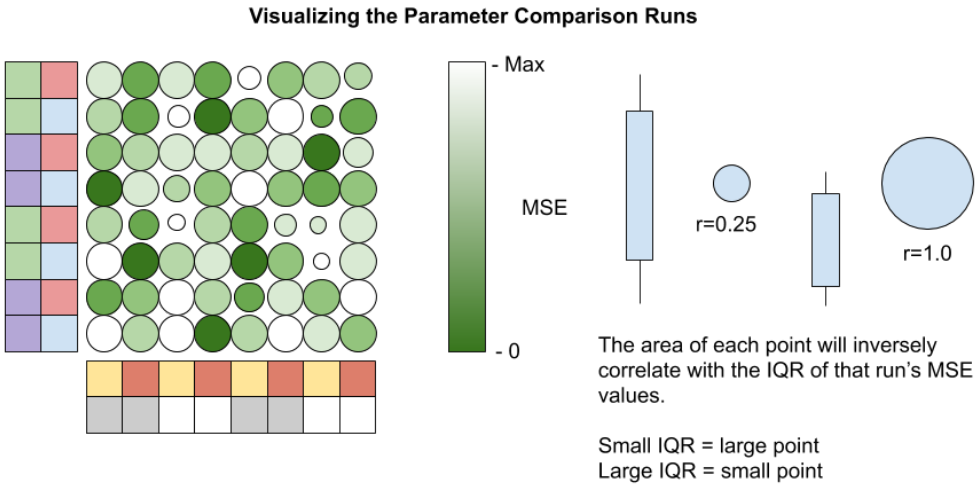
Visualizing the Parameter Comparison Runs
- Max
MSE
- 0
r=0.25
r=1.0
The area of each point will inversely correlate with the IQR of that run's MSE values.
Small IQR = large point
Large IQR = small point

Fig. 3.8: Left: The comparison figure will be 2-D with half of the parameters represented on the y-axis, and the other half represented on the x-axis. Right: Each circle will represent a model, and poor performing models with large IQRs or medians will blend in with the background by being small and similar to the background.

its full potential on the provided data. Once again, chromosome 3 was withheld from training and only used for validation. Since the model may diverge or overfit during training, which would cause the last epoch to not be the best when evaluated on the validation data, the model was again checkpointed and MSE values were collected after every 5th epoch.

3.9 Evaluation metrics

After training, the output annotation from the model with the lowest validation MSE and IQR was evaluated against the original Araport11 annotation [48]. The following statistical performance measures were used for evaluation at both the base pair and region level [61], [62]: TP, FP, TN, FN, Sensitivity, Specificity, and Precision. These two resolution levels of information were previously used by Baidouri et al. for evaluating the performance of their TE annotation pipeline, and were easily adopted to all features predicted by RNNotate [63]. Regions were determined to be "true-positive" if they had at least 75% reciprocal overlap with a region of the same type and strand from the control annotation. Nucleotide proportion and length distribution characteristic metrics were also generated for each region to compare the composition of predictions from different sources.

In addition to comparisons against the known annotation, RNNotate was benchmarked against the performance of other tools to measure usefulness. No other tools exist for making predictions from DNA methylation, so we focused on *de novo* prediction tools: GlimmerHMM [24] and Augustus [26] transcription predictions, and RepeatScout [16] and RepeatMasker [15] for transposable element predictions. Tools like gffcompare [64] and ParsEval [65] exist for comparing annotations, but they focus on transcription features. This is both too specific, where they compare fine grain differences like gene isoforms and exon chains, but also ignore all features not related to protein-coding genes. RNNotate was designed to classify as many curated features as possible, so differannotate was developed as a generic option for calculating our performance metrics and characteristics and finally generating Venn diagrams of logical relations and box plots of characteristic distributions [62].

4. Results and Discussion

4.1 Initial results

All parameters were tested during development and testing, but those which were identified early on as inefficient or ineffective were not included in the final hyperparameter sweep. Independent batches were always inferior to stateful since they were limited in size. LSTMs also performed better than both RNN and GRU cells, which told us the long-term memory was useful with our long sequences. Bidirectional recurrent layers required an order of magnitude more memory, and were outperformed by unidirectional models for all data traversals. The TE sub-classifications and full annotation categories were found to be too sparse, and features like genes, CDS, and TEs dominated the learning process. When data was only processed in the forward (unstranded) direction, the model recognized very few features on the reverse strand. We found both regularizers and batch normalization to have negligible effect due to a combination of our batch sizes and learning rates. We also found dense connections scaled better and conveyed more information than a deep hidden section.

4.2 Hyperparameter sweep

The hyperparameter sweep was run in 7 hours on 31 compute nodes of Longhorn, an IBM cluster with four Nvidia V100 cards per node, at the Texas Advanced Computing Center. Each individual model in the parameter space was trained for 20 epochs and evaluated every 5th epoch by generating classification predictions on the input sequences and then calculating the mean squared error for each individual sequence. This not only increased the evaluation resolution, but was crucial for accurately capturing the error in large and diverse batches of data. Two visual comparison summaries generated from this data at epochs 5 and 20 can be seen below in Figures: 4.1 and 4.2.

Several models with a 0.01 learning rate had very high error rates in Figure 4.2, but some models also managed to perform well with the same learning rate. Beyond these outliers, there were no obvious visual trends in Figure 4.1 and Figure 4.2 since the ineffective parameters were excluded from the hyperparameter sweep.

Parameters were also encoded into a simple numerical format to facilitate a multiple regression analysis in python with statsmodels [66]. Simple polar values were converted to 0 when False and 1 when True. Learning rate was converted inversely: 0.01:0, 0.001:1, 0.0001:2. The number of dense blocks, CNN width, layers, and neurons were sorted and encoded sequentially with the smallest value being 0 and incrementing by 1. The multiple regression found both the learning rate and the number of dense blocks to have a significant effect on the model's ability to predict the validation data as shown in Table 4.1. While not significant by having a p-value greater than 0.05, the number of neurons each recurrent cell and layers in each recurrent

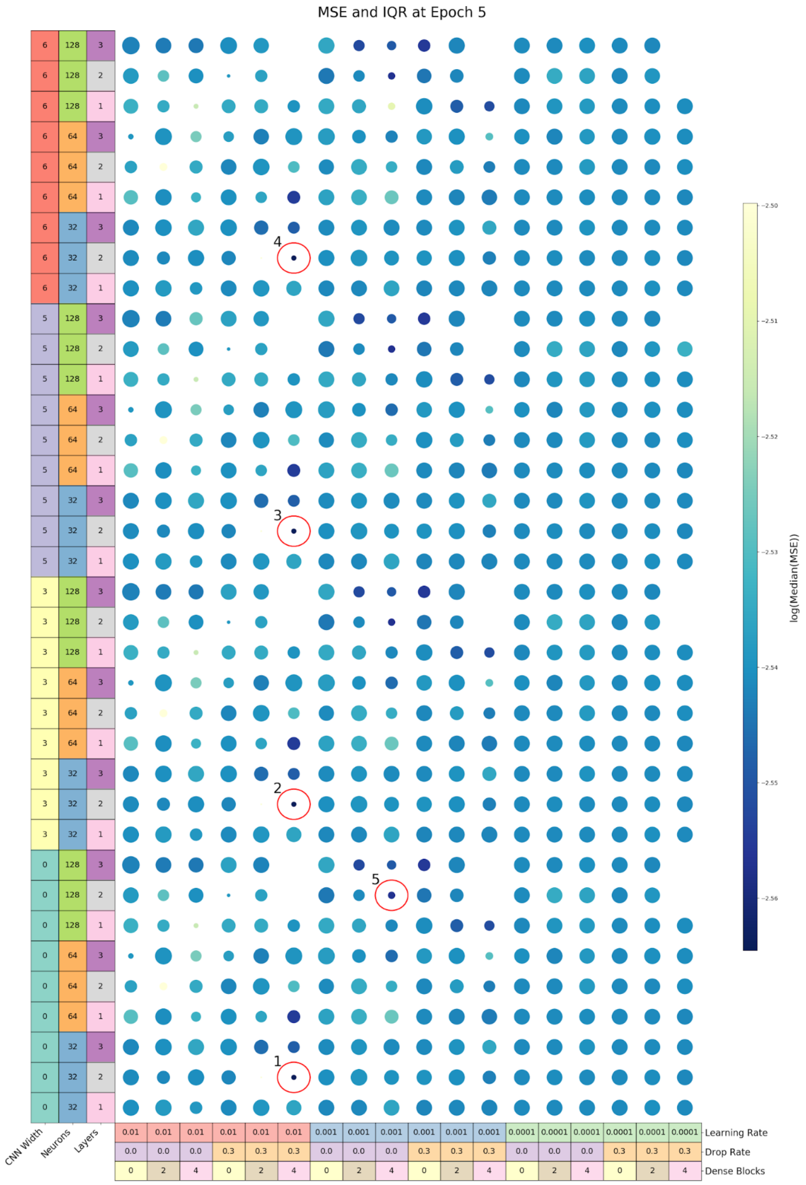

Fig. 4.1: Hyperparameter comparison at epoch 5. Model parameters are represented on the bordering tables. Model error represented by circle color, where light circles have high error and dark circles have low error. A model's error distribution is inversely represented by each circle's area, where a large circle has a tight IQR and small circles have a large IQR. The top 5 models are annotated with their rank and a red circle.

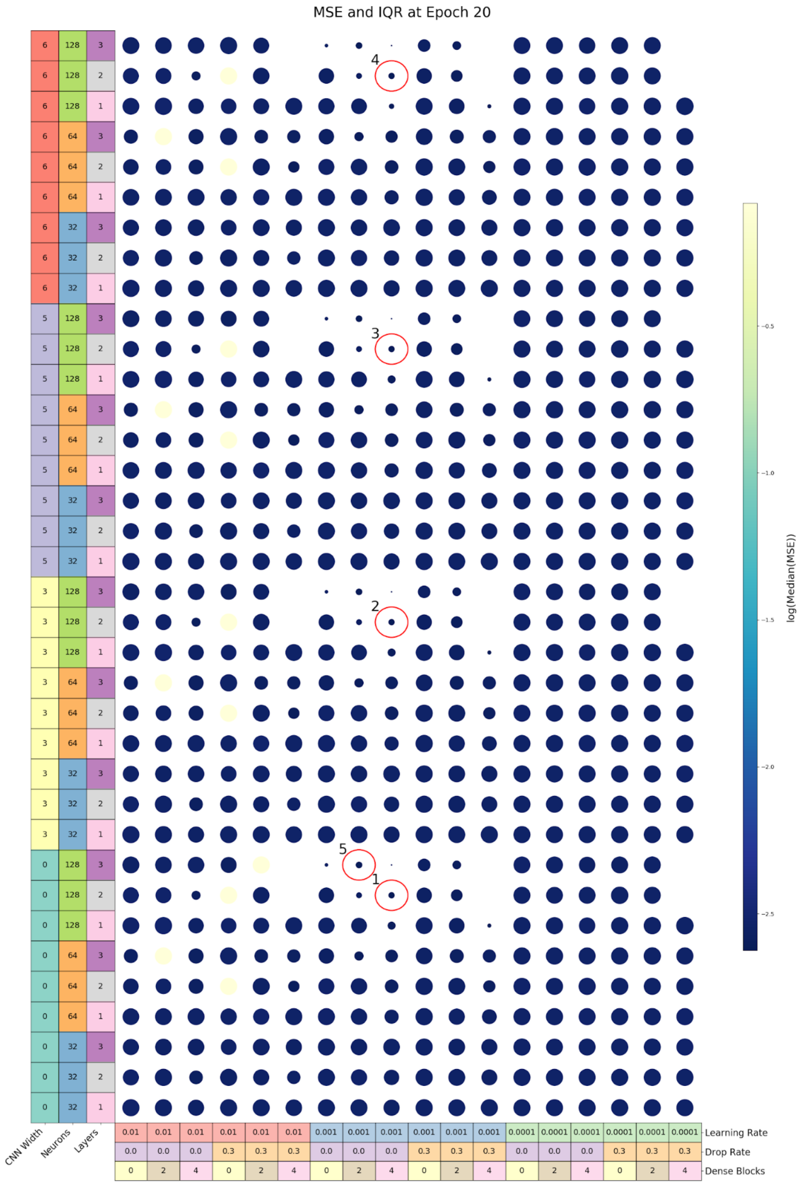

Fig. 4.2: Hyperparameter comparison at epoch 20. Model parameters are represented on the bordering tables. Model error represented by circle color, where light circles have high error and dark circles have low error. A model's error distribution is inversely represented by each circle's area, where a large circle has a tight IQR and small circles have a large IQR. The top 5 models are annotated with their rank and a red circle.

block were also found to have a mild effect on the prediction capability.

Tab. 4.1: Hyperparameter multiple regression on validation MSE at epoch 20. Bold P-values are significant (P ¡ 0.05).

	Coefficients	Standard Error	t-Statistic	P-value
const	0.1104	0.015	7.359	0
CNN Width	-0.0016	0.004	-0.384	0.701
Neurons	0.0109	0.006	1.862	0.063
Layers	0.0103	0.006	1.787	0.074
Learning Rate	-0.026	0.006	-4.514	**7.61E-06**
Drop Rate	0.0151	0.009	1.595	0.111
Dense Blocks	-0.0145	0.006	-2.481	**0.013**

The actual MSE and IQR values from the top five models (highlighted in red in Figure 4.2) are in Table 4.2 below. The top four models all had the same median MSE and IQR values, along with the same parameters for each argument except for the CNN width. Since the models performed the same for all CNN widths, including zero (no CNN), it was decided to simply exclude the CNN for the optimal model to reduce the overall complexity and maximize the throughput.

Tab. 4.2: The top 5 models at epoch 20, sorted by median validation MSE and median validation IQR.

Rank	1	2	3	4	5
CNN Width	0	3	5	6	0
LSTM Neurons	128	128	128	128	128
LSTM Layers	2	2	2	2	3
Learning Rate	0.001	0.001	0.001	0.001	0.001
Drop Rate	0	0	0	0	0
Dense Blocks	4	4	4	4	2
MSE	0.072457	0.072457	0.072457	0.072457	0.072460
IQR	0.055270	0.055270	0.055270	0.055270	0.054531

4.3 Long training

This project originally targeted both CPU and GPU hardware with the ability to distribute the computation across multiple compute nodes when possible to support as many computing environments as possible. For the final long training session however, we ended up relying on a single V100 GPU for the final long training session for two reasons. First, TensorFlow's CPU implementation of recurrent neural networks leaked memory after every batch that was processed, which eventually caused every long training session to fail when running on a CPU. Second, huge batch sizes increased the training and classification throughput, but hindered the learning process.

The optimal model with four densely connected recurrent blocks, two layers of 128-neuron LSTM layers per recurrent block, and a learning rate of 0.001 was then trained for 200 epochs on *A. thaliana* Chromosomes 1, 2, 4, and 5 using Command 4.1. There were two differences between this training session and the hyperparameter sweep: The model was trained on chromosomes 1, 2, 4, and 5 instead of only 1 and 2, and the model was trained for 200 epochs instead of 20.

Listing 4.1: Run command for long training session

```
$ RNNotate –R Araport11.fa –D out_dir –N long_train –M th.bam
   .mr –P 2 −o 512 −v train −f –A Araport11.gff3 –E 200 –B
   608 −−stranded –L 512 –C lstm −n 128 −l 2 −r 0.001 −d 0 −−
   conv 0 –H 0 −S −−dense 4 −−fewer —noTEMD −−train 1,2,4,5
   −−test 3 −−every 5
```

The MSE and IQR values were, again, collected every 5 epochs, and the results from the top five epochs can be seen in Table 4.3. The model had the lowest median MSE at training epoch 20, but the final MSE value was different than the hyperparameter search because more chromosomes were included for training. The model continued to decrease in training error past epoch 20, but error increased on the validation data as visualized in Figure 4.3 and shown in Table 4.3.

Tab. 4.3: Corresponding median mean squared error (MSE) and interquartile range (IQR) for both Chromosome 1 (Training) and Chromosome 3 (Validation). Data was sorted by validation MSE and IQR.

	Training		Validation	
Epoch	MSE	IQR	MSE	IQR
20	0.041	0.044	0.069	0.063
15	0.057	0.059	0.070	0.062
25	0.030	0.033	0.071	0.064
5	0.073	0.042	0.075	0.041
10	0.071	0.051	0.075	0.050
35	0.024	0.023	0.076	0.054
75	0.016	0.013	0.077	0.054
70	0.017	0.014	0.077	0.059
30	0.051	0.059	0.078	0.068
45	0.020	0.017	0.078	0.061
90	0.014	0.010	0.079	0.059
55	0.031	0.043	0.080	0.065
105	0.013	0.009	0.080	0.057
115	0.013	0.008	0.080	0.057
50	0.024	0.028	0.080	0.067
85	0.015	0.011	0.080	0.062
40	0.043	0.052	0.080	0.069
60	0.019	0.016	0.081	0.061
95	0.015	0.011	0.081	0.056
80	0.014	0.010	0.081	0.061
100	0.039	0.057	0.081	0.065
155	0.012	0.010	0.082	0.069
120	0.014	0.010	0.082	0.062
65	0.019	0.019	0.082	0.062
110	0.014	0.010	0.082	0.064
140	0.011	0.007	0.083	0.060
150	0.012	0.009	0.083	0.061
160	0.011	0.008	0.084	0.064
135	0.016	0.025	0.084	0.065
125	0.015	0.013	0.084	0.066
130	0.015	0.018	0.085	0.065
145	0.011	0.007	0.085	0.059
165	0.924	0.033	0.921	0.031
170	0.924	0.033	0.921	0.031
175	0.924	0.033	0.921	0.031
180	0.924	0.033	0.921	0.031
185	0.924	0.033	0.921	0.031
190	0.924	0.033	0.921	0.031
195	0.924	0.033	0.921	0.031
200	0.924	0.033	0.921	0.031

MSE was then plotted by location across both the training and testing chromosomes in the top of Figure 4.3 to reveal any trends tied to location. In the "Training" plot in Figure 4.3, the model learns to recognize the repeat-heavy heterochromatin [39], but those rules must not be general since there was not a similar dip in the testing data. Overall, the error rate was level across the testing chromosome, which shows that the model did not excel at predicting either repeats in the heterochromatin, or genes in the euchromatin.

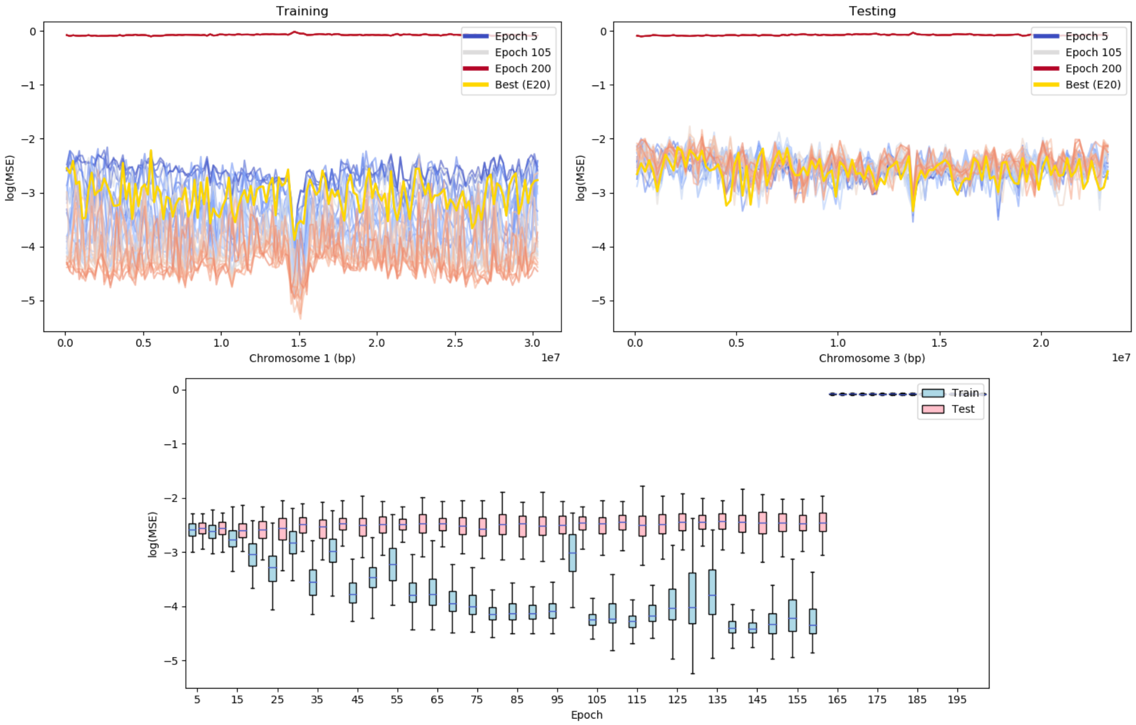

Fig. 4.3: Error during long training. The top two plots show error rate along the training and testing chromosomes. Initial epochs start in blue and transition to red for late epochs. The epoch with the lowest median test MSE is plotted in gold. The bottom plot contrasts the training (blue) and testing (red) MSE distributions of across epochs.

The box plots in Figure 4.3 track the error rate over the entire training process. While the error on the training data continued to decrease until epoch 160, the model performed best on the validation (Testing) data at epoch 20. The box plots do show that the solution space of this model is complex and managed to break out of its local minimum at epoch 160, and did not recover. RNNotate was then restored to the model at epoch 20, and the *A. thaliana* genome was processed in entirety in 16 minutes using 16 gigabytes of memory.

While the raw error rate does reveal a picture of overall performance, it does not convey specific strengths or weaknesses of the model. The classifications were split into two groups, transcripts and transposable elements, and compared at both the base and region level against the known annotation and trusted *de novo* tools.

4.4 Base level evaluation of transcripts

The resulting GFF3 file produced by RNNotate, at the 20th epoch, was compared against the Araport11 annotation and both GlimmerHMM (Command 4.2) and Augustus (Command 4.3) predictions, which are both used for predicting transcripts.

Listing 4.2: Command for running GlimmerHMM

```
$ glimmerhmm Araport11.fa trained_dir/arabidopsis -g -o
    glimmerhmm.gff3
```

Listing 4.3: Command for running Augustus

```
$ augustus --strand=both --singlestrand=false --gff3=on --
    sample=0 --progress=false --UTR=on --uniqueGeneId=true --
    species=arabidopsis Araport11.fa > augustus.gff3
```

While RNNotate attempted to predict every element curated in Araport11, GlimmerHMM predicted

$$\{\texttt{CDS, mRNA}\}$$

elements and Augustus predicted

{`CDS, exon, gene, intron, start_codon, stop_codon, transcript,`
`transcription_end_site, transcription_start_site`}

of which only CDS elements were common between all tools, and {exon, mRNA, and gene} were common between RNNotate and at least one tool. Performance metrics were calculated at the base pair (bp) level for each element, strand, tool combination and compounded into Table 4.4.

4.4.1 Base level performance of exon and gene predictions

Starting with exons in Table 4.4, RNNotate detected 47% (sensitivity) of the exon bases in the Araport11 annotation while Augustus detected 80%, and both tools had similar precision rates. Even with the stranded traversal, RNNotate predicted exons with a higher sensitivity on the reverse strand (54%) than on the forward strand (32%), hinting that either the batches or data was still biased towards specific strands. In contrast, the methods of Augustus predicted exons with near equal performance from either strand. Similar to exons, the sensitivity rate of gene predictions by RNNotate (54%) was lower than Augustus for all strands (92%), however RNNotate had a higher specificity (96%) and precision (95%) than Augustus (62% and 75%) for all strands. This result shows that RNNotate went beyond learning open reading frames and was able to predict actual genic regions (Figure 4.4, which were extensively validated with RNA-Seq in the Araport11 annotation.

Tab. 4.4: Base level performance measures. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each tool, feature type, and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), and precision (PREC) were also calculated. Glimmer stands for GlimmerHMM in this table.

	Strand	Tool	TP	FP	TN	FN	SENS	SPEC	PREC
exon	+/-	RNNotate	5.41	0.74	11.19	6.11	0.47	0.94	0.88
		Augustus	9.35	1.11	10.82	2.18	0.81	0.91	0.89
	+	RNNotate	1.86	0.67	17.02	3.9	0.32	0.96	0.74
		Augustus	4.52	0.56	17.13	1.25	0.78	0.97	0.89
	-	RNNotate	3.27	0.49	16.92	2.77	0.54	0.97	0.87
		Augustus	4.75	0.63	16.79	1.29	0.79	0.96	0.88
mrna	+/-	RNNotate	7.88	0.51	8.2	6.86	0.53	0.94	0.94
		Glimmer	11.4	0.86	7.86	3.34	0.77	0.9	0.93
	+	RNNotate	2.7	0.72	15.43	4.61	0.37	0.96	0.79
		Glimmer	5.5	0.42	15.73	1.81	0.75	0.97	0.93
	-	RNNotate	4.82	0.38	15.44	2.82	0.63	0.98	0.93
		Glimmer	5.85	0.49	15.33	1.79	0.77	0.97	0.92
cds	+/-	RNNotate	3.43	0.57	16.22	3.24	0.52	0.97	0.86
		Glimmer	6.21	1.53	15.27	0.46	0.93	0.91	0.8
		Augustus	6.4	1.75	15.04	0.27	0.96	0.9	0.79
	+	RNNotate	1.05	0.32	19.91	2.18	0.33	0.98	0.77
		Glimmer	3	0.74	19.49	0.23	0.93	0.96	0.8
		Augustus	3.08	0.86	19.37	0.14	0.96	0.96	0.78
	-	RNNotate	2.3	0.39	19.61	1.16	0.67	0.98	0.85
		Glimmer	3.21	0.79	19.21	0.25	0.93	0.96	0.8
		Augustus	3.31	0.9	19.1	0.15	0.96	0.96	0.79
gene	+/-	RNNotate	6.98	0.41	10.11	5.96	0.54	0.96	0.95
		Augustus	11.84	3.97	6.55	1.1	0.92	0.62	0.75
	+	RNNotate	2.25	0.47	16.47	4.27	0.35	0.97	0.83
		Augustus	5.71	2.02	14.92	0.81	0.88	0.88	0.74
	-	RNNotate	4.5	0.36	16.31	2.29	0.66	0.98	0.93
		Augustus	6.01	2.08	14.59	0.79	0.88	0.88	0.74

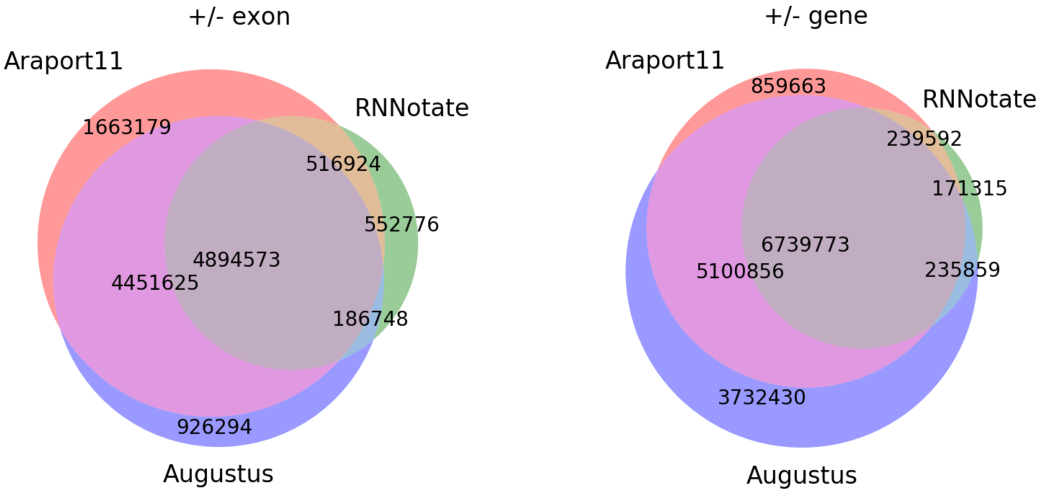

Fig. 4.4: Base level performance of exons and genes. Left - Venn diagram showing the logical relations between predictions of Araport11 Chromosome 3 exons by RNNotate and Augustus. Right - Venn diagram showing the logical relations between exon predictions of Araport11 Chromosome 3 genes.

4.4.2 Base level performance of mRNA predictions

Similar to exons and genes, RNNotate detected 53% of the mRNA bases while GlimmerHMM detected more at 77%. Strand bias was again present with mRNA predictions from RNNotate sequences, where predictions from the forward strand had a 37% sensitivity, while predictions from the reverse strand had 63% sensitivity. Similar to exon results, the specificity and precision rates for mRNA predictions from RNNotate were slightly higher, at 94% and 94% respectively, than GlimmerHMM with 90% and 93% respectively.

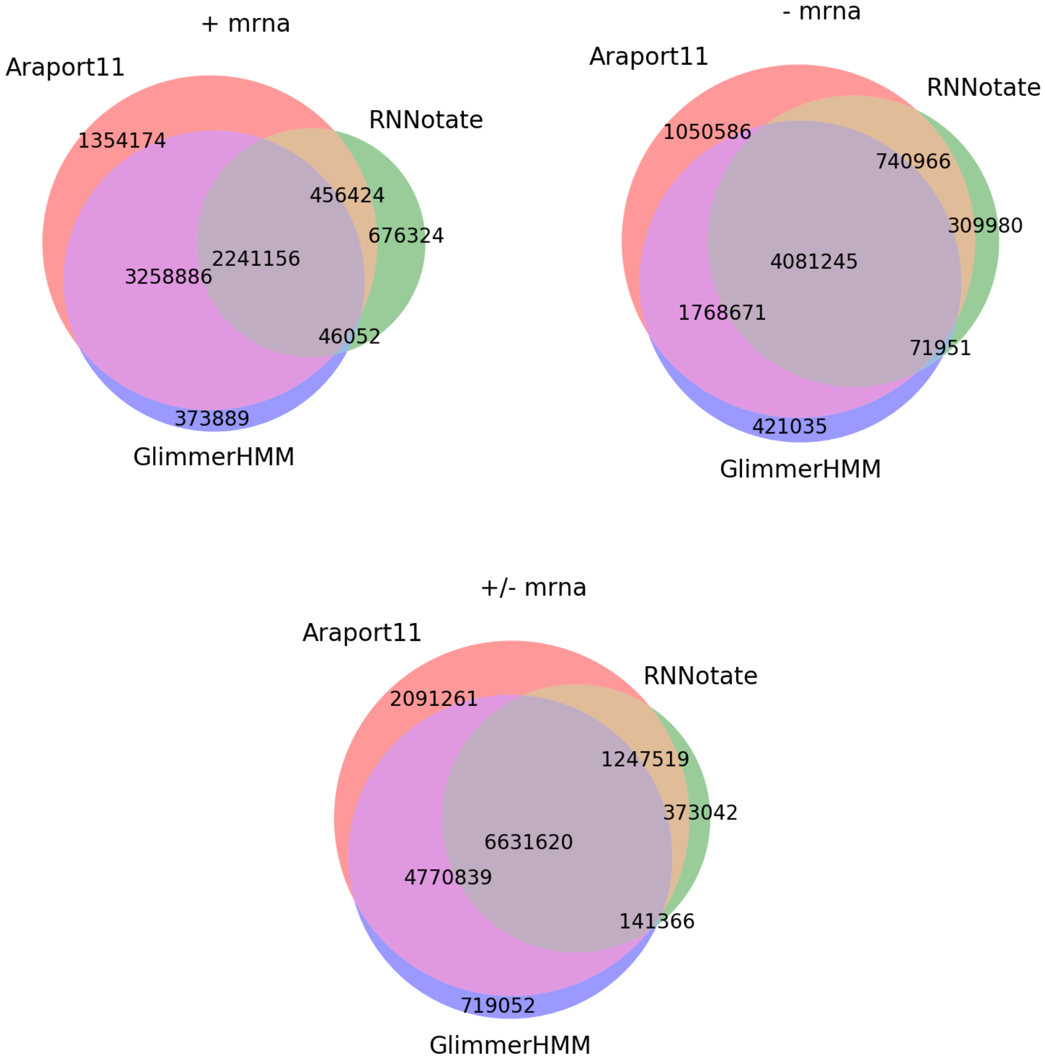

Fig. 4.5: Base level performance of mRNA. Top-Left - Venn diagram showing the logical relations between predictions of Araport11 Chromosome 3 mRNA by RNNotate and GlimmerHMM from the forward strand. Top-Right - Logical relations between mRNA predictions by RNNotate and GlimmerHMM from the reverse strand. Bottom-Center - Logical relations between mRNA predictions by RNNotate and GlimmerHMM from both strands.

4.4.3 Base level performance of CDS predictions

All three tools predicted CDS regions, and previous trends continued. RNNotate had a lower sensitivity rate at 52% than both GlimmerHMM and Augustus, respectively at 93% and 96%. Predictions from RNNotate were also biased towards the reverse strand, where sensitivity was at 33% on the forward strand and 67% on the reverse. Even though sensitivity was lower, RNNotate always had equal or higher specificity (97%) and precision (86%) rates than GlimmerHMM (91% and 80%) and Augustus (90% and 79%) for predictions from both strands.

4.4.4 Overall base level performance

Averaging the values of each feature category from Table 4.4 into Table 4.5 helps elucidate the trends found while analysing the results for each individual feature category. RNNotate had the lowest sensitivity rate for predicting transcription features out of all the tools, with results biased towards the reverse strand. However, even though RNNotate predicted fewer bases for each category, its specificity and precision was equal or higher than the other tools. Beyond the raw numbers, this result is impressive since GlimmerHMM and Augustus, which were trained on the entire *A. thaliana* annotation, predict a subset of the features, while RNNotate predicts all of them, making the rules generated during training fairly complex. Future versions of RNNotate will work towards reducing strand bias through modified batching and altered data representations.

Tab. 4.5: Average of base-level sensitivity (SENS), specificity (SPEC), and precision (PREC) across features by tool and strand. Bold values represent the best tool for each performance/strand (column) category.

	SENS			SPEC			PREC		
	+/-	+	-	+/-	+	-	+/-	+	-
RNNotate	0.51	0.34	0.63	**0.95**	**0.97**	**0.98**	**0.91**	0.78	**0.89**
GlimmerHMM	0.85	0.84	0.85	0.91	**0.97**	0.97	0.87	**0.87**	0.86
Augustus	**0.9**	**0.87**	**0.88**	0.81	0.94	0.93	0.81	0.8	0.8

4.5 Base level evaluation of transposable elements

RepeatMasker (Command 4.4) and RepeatScout (Command 4.5) were run on the Araport11 reference assembly and the open repeat library Dfam_Consensus 3.1 [67] for homology searches to generate TE predictions. Both tools output predictions called “similarity”, which were treated as “transposable_element” features for comparison against the original Araport11 annotation and the predictions generated by RNNotate. The base-level performance results were compounded below in Table 4.6.

Listing 4.4: Command for running RepeatMasker

```
$ RepeatMasker -e ncbi -pa 24 -species arabidopsis -dir . -
   gff Araport11.fa
```

Listing 4.5: Command for running RepeatScout

```
$ build_lmer_table -sequence Araport11.fa -freq out.freq -l
   11
$ RepeatScout -sequence Araport11.fa -output out.rep.fa -freq
    out.freq -l 11 -goodlength 75
$ cat out.rep.fa | filter-stage-1.prl > out.rep.filt1.fa
$ RepeatMasker -pa 24 -s -dir . -lib out.rep.filt1.fa
  Araport11.fa
$ cat out.rep.filt1.fa | filter-stage-2.prl --cat Araport11.
  fa.out --thresh 10 > out.rep.filt2.fa
$ RepeatMasker -pa 24 -s -dir . -lib out.rep.filt2.fa -nolow
  -norna -no_is -gff Araport11.fa
```

Tab. 4.6: Base level performance measures for TEs. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each tool and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), precision (PREC), and accuracy (ACC) were also calculated, with top values in bold for each strand.

Strand	Tool	TP	FP	TN	FN	SENS	SPEC	PREC	ACC
+/-	RNNotate	1.53	0.15	18.37	3.41	0.31	**0.99**	**0.91**	**0.85**
	RepeatMasker	0.13	0.28	18.24	4.81	0.03	**0.99**	0.31	0.78
	RepeatScout	1.59	0.36	18.16	3.35	**0.32**	0.98	0.82	0.84
+	RNNotate	0.78	0.42	20.49	1.77	**0.31**	**0.98**	**0.65**	**0.91**
	RepeatMasker	0.06	0.34	20.56	2.49	0.03	**0.98**	0.16	0.88
	RepeatScout	0.48	0.59	20.32	2.08	0.19	0.97	0.45	0.89
-	RNNotate	0.47	0.09	20.91	1.98	0.19	**1**	**0.84**	**0.91**
	RepeatScout	0.47	0.57	20.43	1.99	0.19	0.97	0.45	0.89

Starting with strand agnostic classifications (+/-), RepeatMasker performed the worst with few true positives, twice as many false-positives, and many false negatives. While RepeatMasker was clearly the worst performing, both RNNotate and RepeatScout were fairly equivalent. RepeatScout had a slightly higher sensitivity rate than RNNotate (32% vs 31%), but RNNotate had half as many false-positive predictions, making it more precise.

On the forward strand (+), RepeatMasker performed the worst out of the three tools again. RNNotate performed the best, with twice the sensitivity rate as RepeatScout (31% vs 19%). As with the strand agnostic results, RNNotate was equally specific as RepeatScout, but more precise with its predictions (65% vs 45%).

On the reverse strand (-) there were no predictions from RepeatMasker, since it only generated predictions on the forward strand. RNNotate and RepeatScout once again had equivalent sensitivity rates (19% and 19%). However, RepeatScout returned 6 times as many false-positive bases, making it half as precise as RNNotate (45% vs 84%).

These results show that RNNotate was more sensitive when predicting elements on the forward strand (31% vs 19%), which is a reverse of the bias it showed when predicting transcription elements. As the methods for training RNNotate improve, we plan to investigate whether this bias arose from the training process or there is a real biological mechanism confounding the prediction. RepeatScout had a lower strand-specific precision, showing it also had trouble determining the correct strand of its transposable element predictions. This could mean both tools have room for improvement, and possibly that the transposable elements reported in the annotation are not perfect. Transcription elements can be verified through RNA-seq, but transposable elements cannot be sensed directly, only recognized when they move or by their patterns.

4.6 Region level evaluation of transcripts

While base level metrics give an idea of overall performance, the actual predictions may not reflect complete features. Metrics generated from the contiguous regions of each prediction are better for conveying the fidelity, and therefore usefulness, of the output annotation predictions. Predicted regions from each of the three tools (RNNotate, RepeatMasker, and RepeatScout) were determined to be true-positive based on their overlap with CDS regions in the Araport11 annotation. Figure 4.6 shows the reciprocal overlap distribution for the predictions from each tool. Both GlimmerHMM and Augustus had more perfect (100%) matches, but RNNotate had fewer predictions with no (0%) overlap.

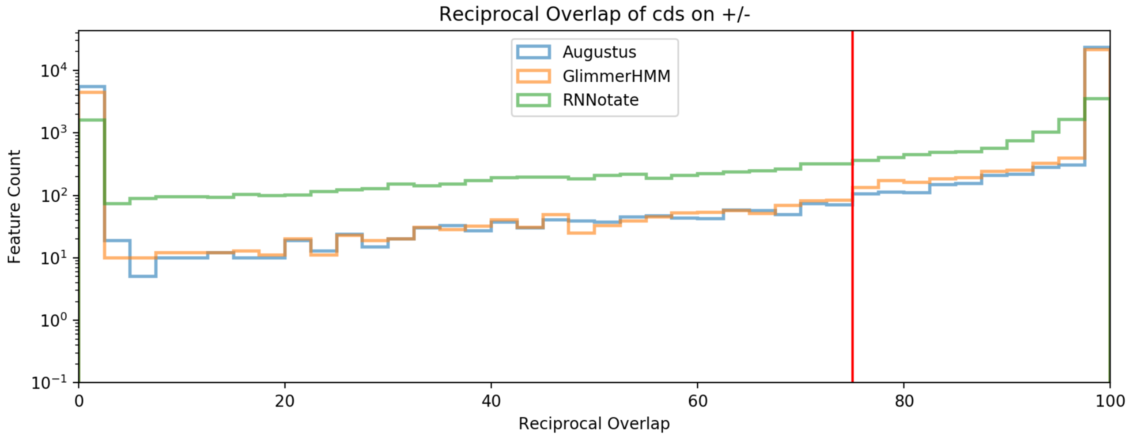

Fig. 4.6: Reciprocal overlap of CDS features. Histogram of reciprocal overlap for each predicted CDS feature from each of the 3 tools against Araport11 CDS regions. The vertical red line represents the 75% threshold used for “true-positive” classifications.

To determine if the predictions match up to known elements and serve as a useful annotation, region-based performance metrics were calculated with differannotate. Since sensitivity and precision were both usually below 90% for predictions from all three tools, and there were no obvious trends in 4.6 for determining a cutoff, a liberal 75% reciprocal overlap between a known and predicted region was required to be counted as a true-positive result. Due to the sparse nature of features, the false-positive category was excluded since the regions were extremely large. This exclusion meant that only the sensitivity (SENS) and precision (PREC) rates could be calculated in Table 4.7.

Tab. 4.7: Region level performance of transcription elements. True positive (TP), false positive (FP), and false negative (FN) counts were calculated for each tool, feature type, and strand combination in total regions. Sensitivity (SENS) and precision (PREC) were also calculated based on region counts.

Element	Strand	Tool	TP	FP	FN	SENS	PREC
exon	+/-	RNNotate	9353	9954	29930	0.24	0.48
		Augustus	23619	9259	15664	0.6	0.72
	+	RNNotate	3096	4649	16505	0.16	0.4
		Augustus	11669	4541	7932	0.6	0.72
	-	RNNotate	6198	5364	13484	0.32	0.54
		Augustus	11922	4746	7760	0.61	0.72
mrna	+/-	RNNotate	2098	6398	7323	0.22	0.25
		GlimmerHMM	2900	3537	6521	0.31	0.45
	+	RNNotate	729	3417	3977	0.16	0.18
		GlimmerHMM	1436	1799	3270	0.31	0.44
	-	RNNotate	1361	2989	3354	0.29	0.31
		GlimmerHMM	1462	1740	3253	0.31	0.46
cds	+/-	RNNotate	9707	6526	20685	0.32	0.6
		GlimmerHMM	23571	5426	6821	0.78	0.81
		Augustus	25198	6442	5194	0.83	0.8
	+	RNNotate	3092	2782	12001	0.21	0.53
		GlimmerHMM	11633	2722	3460	0.77	0.81
		Augustus	12417	3212	2676	0.82	0.79
	-	RNNotate	6559	3800	8740	0.43	0.63
		GlimmerHMM	11931	2711	3368	0.78	0.82
		Augustus	12774	3237	2525	0.84	0.8
gene	+/-	RNNotate	1909	4955	4635	0.29	0.28
		Augustus	3631	2010	2913	0.56	0.64
	+	RNNotate	633	2401	2626	0.19	0.21
		Augustus	1797	977	1462	0.55	0.65
	-	RNNotate	1263	2567	2022	0.38	0.33
		Augustus	1824	1043	1461	0.56	0.64

While predictions from RNNotate showed good performance based on the base pair resolution metrics, both Augustus and GlimmerHMM were superior at predicting correct regions of their respective categories. RNNotate did make some correct predictions, but they were lost in large quantities of false positives. Taking a look at strand agnostic (+/-) gene predictions in Table 4.7, RNNotate found more than twice as many false positives as true. This shows that while RNNotate had a 95% precision rate when classifying base pairs, they were too irregular and did not represent whole features even after gap filling and filtering.

4.6.1 Characteristics of transcription regions

All three tools predicted CDS regions, so nucleotide proportion and length statistics were generated on these regions in Figure 4.7. Both the base composition and length distribution of predicted CDS regions show that regions produced by RNNotate were different in both size and composition than those in the Araport11 annotation and those predicted by GlimmerHMM and Augustus.

The nucleotide proportions of both GlimmerHMM and Augustus matched those in Araport11 in Figure 4.7, where the proportion of A matched T and G matched C, but A and T both occurred more frequently than G and C. Looking at the RNNotate proportion distributions, the A and T proportions are equivalent, but the C proportion is higher and unsynchronized from the G proportion.

The length distributions of CDS predictions in Figure 4.7 once again show that predictions from GlimmerHMM and Augustus matched the actual distribution from Araport11. CDS regions predicted by RNNotate were as long as the 3 other sources, but many short CDS sequences were absent.

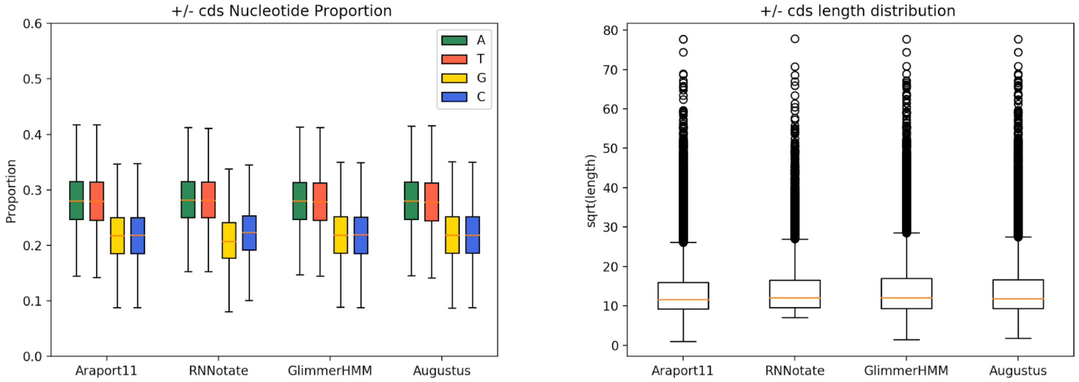

Fig. 4.7: Descriptive comparison of CDS predictions. Left figure shows the nucleotide proportion distributions for CDS predictions on either strand by tool (RNNotate, GlimmerHMM, Augustus) or reference (Araport11). Right figure shows the sqrt(length) distributions of CDS predictions on either strand by tool or reference.

4.7 Region level evaluation of transposable elements

For each of the three tools, predicted regions were determined to be true-positive based on their reciprocal overlap with TEs in the Araport11 annotation. Figure 4.8 shows that RepeatMasker had the fewest high-overlap predictions, and many false-positives that could not be matched to any Araport11 TEs. RNNotate had more high-overlap predictions than RepeatMasker, but no perfect (100%) matches. RepeatScout had the most high-overlap matches, but was even with RNNotate with low-overlap false-positives. Once again, a threshold of 75% reciprocal overlap was used to classify predictions as either true or false positives, which is represented by the red vertical line in Figure 4.8, and performance metrics were calculated and collected in Table 4.8.

As with the base pair metrics, RepeatMasker was the lowest performing tool of the three, with only 12 correctly predicted transposable elements. RNNotate was 10 times better with 119, leading to a 1.8% sensitivity rate and 4.1% precision rate.

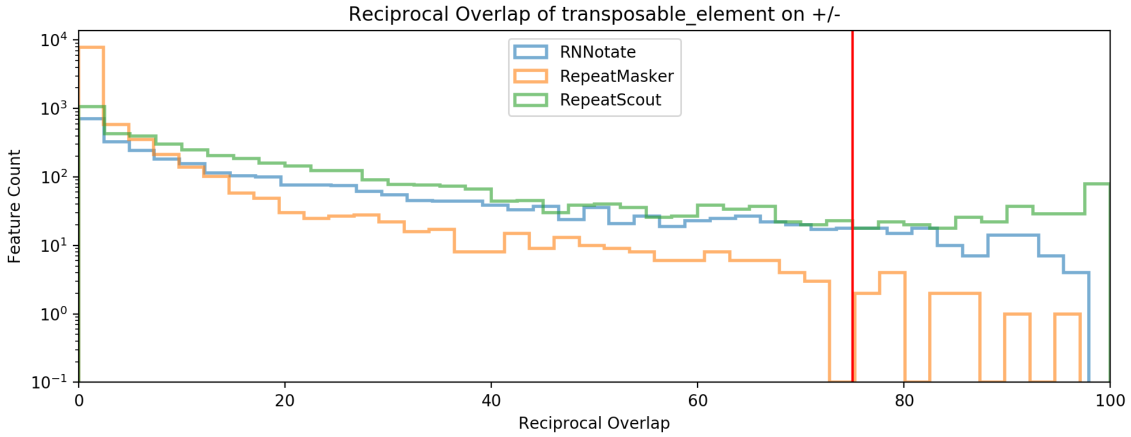

Fig. 4.8: Reciprocal overlap of transposable elements. Histogram of reciprocal overlap for each predicted TE from each of the 3 tools against TEs in the Araport11 annotation. The vertical red line represents the 75% threshold used for true-positive classifications.

While this may not seem impressive, RepeatScout, which performed best, had a 4.6% sensitivity rate and a 6.6% precision rate. To determine if the FP were too small to match a TE, or simply wrong, descriptive statistics were generated in Figure 4.9.

Tab. 4.8: Region level performance of transposable elements. True positive (TP), false positive (FP), and false negative (FN) counts were calculated for each tool, feature type, and strand combination in total regions. Sensitivity (SENS) and precision (PREC) were also calculated based on region counts.

Strand	Tool	TP	FP	FN	SENS	PREC
+/-	RNNotate	119	2797	6342	0.018	0.041
	RepeatMasker	12	9653	6449	0.002	0.001
	RepeatScout	300	4237	6161	0.046	0.066
+	RNNotate	65	1821	3219	0.02	0.034
	RepeatMasker	8	9657	3276	0.002	0.001
	RepeatScout	85	2140	3199	0.026	0.038
-	RNNotate	38	992	3139	0.012	0.037
	RepeatScout	89	2223	3088	0.028	0.038

]

The nucleotide proportions in Figure 4.9 show that both RepeatScout and RNNotate both matched the actual proportions present in the Araport11 TE annotation. RepeatMasker returned regions which were extremely biased towards adenine (A) nucleotides and thymine (T) nucleotides, meaning it may have classified simple repeats in the DNA as transposable elements.

The length distributions in Figure 4.9 showed that predictions from both RepeatScout and RNNotate were once again in sync with actual transposable elements present in the Araport11 annotation. RepeatMasker had the least realistic TE lengths, where all except a few outliers were smaller than 75% (Q1) of those present in the Araport11 annotation.

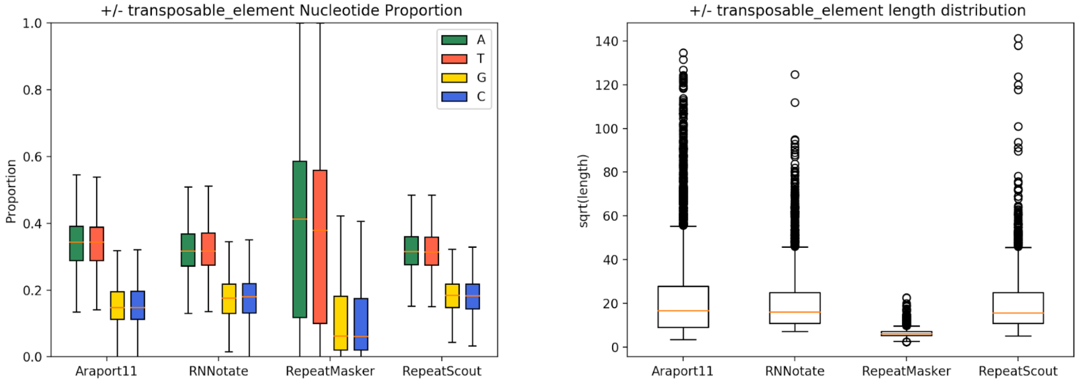

Fig. 4.9: Descriptive comparison of TE predictions. Left figure shows the nucleotide proportion distributions for TE predictions on either strand by tool (RNNotate, RepeatMasker, RepeatScout) or reference (Araport11). Right figure shows the sqrt(length) distributions of CDS predictions on either strand by tool or reference.

4.8 Impact of DNA methylation on prediction

One of the main reasons this research was undertaken was to determine if the inclusion of DNA methylation actually provided a useful dimension for predicting genomic features. It has been shown that the model can produce better or comparable results to trusted tools when it is included, but it is still unknown if the inclusion of DNA methylation impacted the results overall. To answer this question, the optimal model was once again trained for 200 epochs with the same parameters, except the methylation data was excluded. Evaluation also took place every 5th epoch, and the top 10 loss values were recorded for both the training and validation data in Table 4.9.

Tab. 4.9: Training and validation error after training without methylation. Corresponding median mean squared error (MSE) and interquartile range (IQR) for both Chromosome 1 (Training) and Chromosome 3 (Validation). Data was sorted by validation MSE, and the best (lowest) 10 values were kept.

		Training Data		Validation Data	
Rank	Epoch	MSE	IQR	MSE	IQR
1	135	0.0622	0.0502	0.0714	0.0521
2	115	0.0621	0.0492	0.0718	0.0534
3	170	0.0618	0.0532	0.0719	0.0544
4	150	0.0519	0.0553	0.0719	0.0659
5	145	0.0613	0.049	0.0735	0.0517
6	165	0.0684	0.0484	0.0744	0.052
7	140	0.0687	0.0402	0.0745	0.0413
8	160	0.0636	0.0511	0.0745	0.0524
9	185	0.0685	0.0561	0.0745	0.0563
10	120	0.0626	0.0509	0.0748	0.0524

When excluding the dimension of DNA methylation, the model performed optimally after training for 135 epochs (Table 4.9), which was almost 6 times the number of epochs used for the optimal model with DNA methylation. The error distribution box plots in Figure 4.10 show that there was no significant learning improvement over time. The fact the model never began overfitting to the data like in Figure 4.3, where training and validation error began to diverge, could mean that the model was memorizing specific methylation patterns, and not the relative location included in the input.

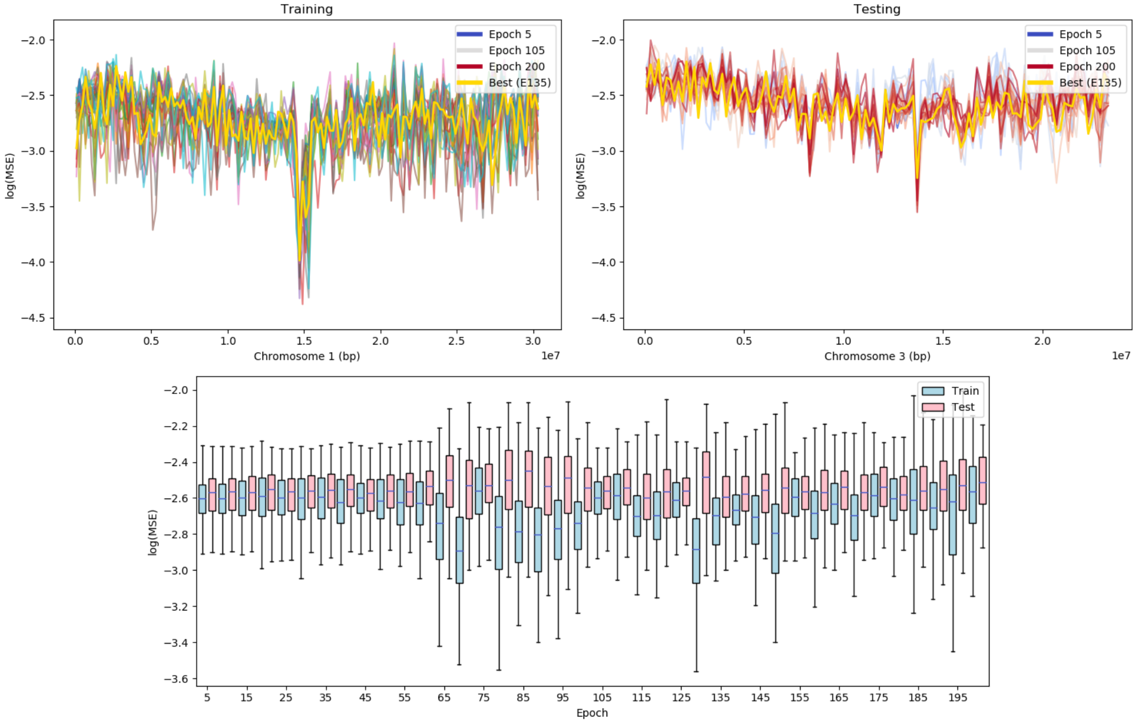

Fig. 4.10: Error during long training without methylation. The top two plots show error rate along the training and testing chromosomes. Initial epochs start in blue and transition to red for late epochs. The epoch with the lowest median test MSE is plotted in gold. The bottom plot contrasts the training (blue) and validation (red) MSE distributions of across epochs.

The output from this model was then cleaned with the same gap filling and filtering rules originally used. Afterwards, results were compared to Araport11 and those when run with DNA methylation with differannotate. Performance metrics for all feature categories were collected in Table 4.10, where the “RNNotate” sample is the original, and “w/o Methylation” is when DNA methylation was excluded. Strand specific results (+,-) were excluded since previous results determined that RNNotate to have strand bias in its predictions.

Tab. 4.10: Base level performance measures of RNNotate. True positive (TP), false positive (FP), true negative (TN), false negative (FN) counts were calculated for each model version, feature type, and strand combination in megabases. Sensitivity (SENS), specificity (SPEC), and precision (PREC) were also calculated with top values in bold. All comparisons were unstranded made against the Chromosome 3 of the Araport11 annotation.

Element	Tool	TP	FP	TN	FN	SENS	SPEC	PREC
3’ UTR	RNNotate	0.42	0.15	21.74	1.14	**0.27**	0.99	0.74
	w/o Methylation	0.12	0.02	21.87	1.45	0.08	**1**	**0.84**
3’ UTR	RNNotate	0.14	0.06	22.09	1.17	**0.11**	**1**	0.7
	w/o Methylation	0.03	0.01	22.14	1.28	0.02	**1**	**0.78**
exon	RNNotate	5.41	0.74	11.19	6.11	**0.47**	0.94	0.88
	w/o Methylation	3.9	0.37	11.56	7.62	0.34	**0.97**	**0.91**
mrna	RNNotate	7.88	0.51	8.2	6.86	**0.53**	0.94	0.94
	w/o Methylation	6.88	0.32	8.4	7.87	0.47	**0.96**	**0.96**
cds	RNNotate	3.43	0.57	16.22	3.24	**0.52**	0.97	0.86
	w/o Methylation	2.2	0.27	16.52	4.47	0.33	**0.98**	**0.89**
TE	RNNotate	1.53	0.15	18.37	3.41	**0.31**	0.99	**0.91**
	w/o Methylation	1	0.1	18.42	3.94	0.2	**1**	**0.91**
gene	RNNotate	6.98	0.41	10.11	5.96	**0.54**	0.96	0.95
	w/o Methylation	6.08	0.26	10.26	6.86	0.47	**0.98**	**0.96**
TE gene	RNNotate	0.89	0.15	21.17	1.25	**0.42**	0.99	0.86
	w/o Methylation	0.61	0.09	21.22	1.53	0.29	**1**	**0.87**

For all eight feature categories, RNNotate was more sensitive and made more positive predictions when DNA methylation was included in the input data. However, the model that resulted from training without DNA methylation was slightly more specific and precise with its classifications, showing that signals from DNA methylation not only resulted in more positives, but more false-positive predictions.

The feature category that was least affected by the exclusion of DNA methylation were genes, where RNNotate had a sensitivity of 54% with methylation and 47% without it. The prediction of "five_prime_utr" features was most affected by the exclusion of DNA methylation, where RNNotate had a sensitivity of 11% with methylation and 2% without it. Visualizing the logical relation of these two categories in Figure 4.11 illustrates the equally high precision of gene predictions from each model, while also showing the discrepancy between five prime untranslated region (five_prime_utr) predictions.

The prediction of a gene was also visualized in a genome browser in Figure 4.12. Both Augustus and GlimmerHMM generated a gene model that looked similar, but was shorter than the model in Araport11. RNNotate also generated a corresponding model, and was able to predict corresponding untranslated region (UTR) predictions in the areas that were truncated by GlimmerHMM and Augustus. The prediction generated by RNNotate without methylation data was fragmented and sparse, and did not represent the actual gene like when it was run with DNA methylation.

4.9 Investigating model performance

In an effort to understand why RNNotate was outperformed by RepeatScout, GlimmerHMM, and Augustus for certain measures of performance, and why there were strand effects in the predictions, we took a look at DNA methylation on and around predicted and known Araport11 features.

For each feature category, contiguous regions were stratified by strand to show possible differences in methylation-based indicators. DNA methylation frequencies

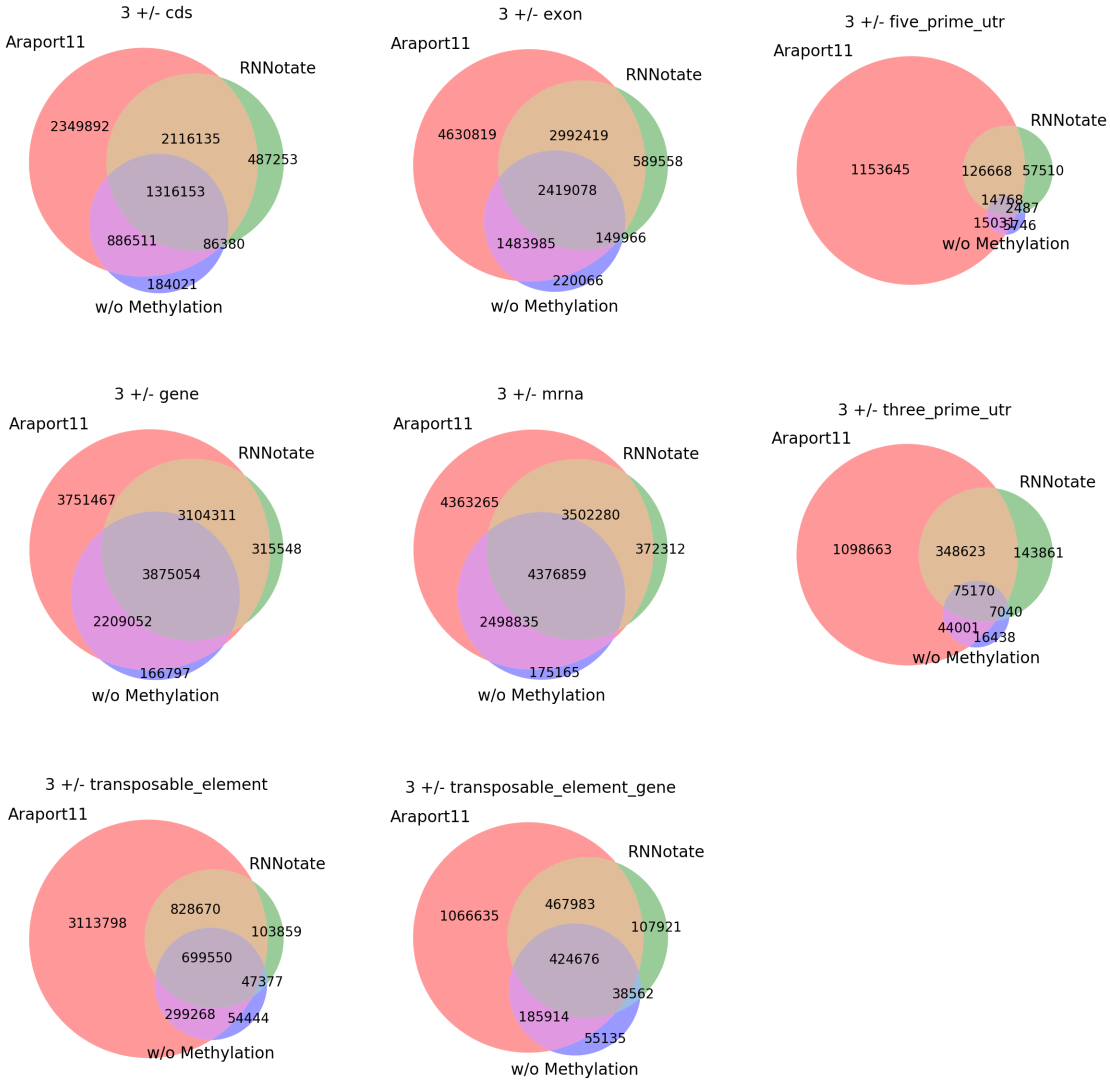

Fig. 4.11: Base level logical relations between Araport11 Chromosome 3 features and bases predicted by RNNotate and RNNotate without methylation (w/o Methylation).

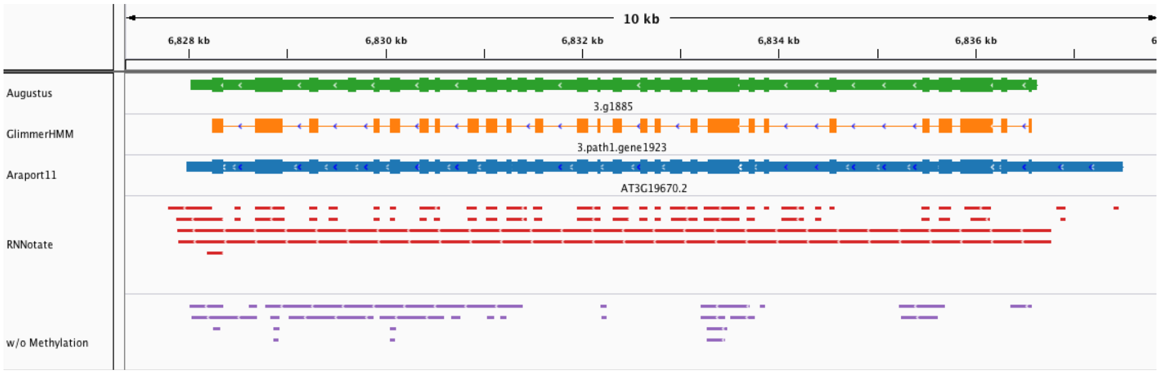

Fig. 4.12: Genome browser view of Araport11 gene AT3G19670.2 (blue), and its prediction by Augustus (green), GlimmerHMM (orange), RNNotate (red), and RNNotate without methylation (purple).

for each context (CG, CHG, CHH) were separately collected from each region, and both one kilobase (kb) up and downstream. For each of the three region categories, the positional methylation frequency values were split into 20 bins, to allow for the comparison of varying feature lengths, and averaged. The 60 values, 20 for each of the three regions, were then tracked and then averaged for each context.

4.9.1 Regional exon methylation

RNNotate was twice as sensitive at predicting exons on the reverse strand (32%) as it was on the forward strand (16%) (Table 4.7). For exons on the forward strand In Figure 4.13, false-positive predictions (gold) had a higher than average methylation frequency, also called hypermethylation, across all three contexts while also preserving the characteristic plateau in the exon region. On the reverse strand, the CG methylation patterns were similar across all exon predictions. False-positives were slightly hypermethylated for CHG and CHH methylation, but true-positive predictions had lower than average methylation frequencies, also called hypomethylation.

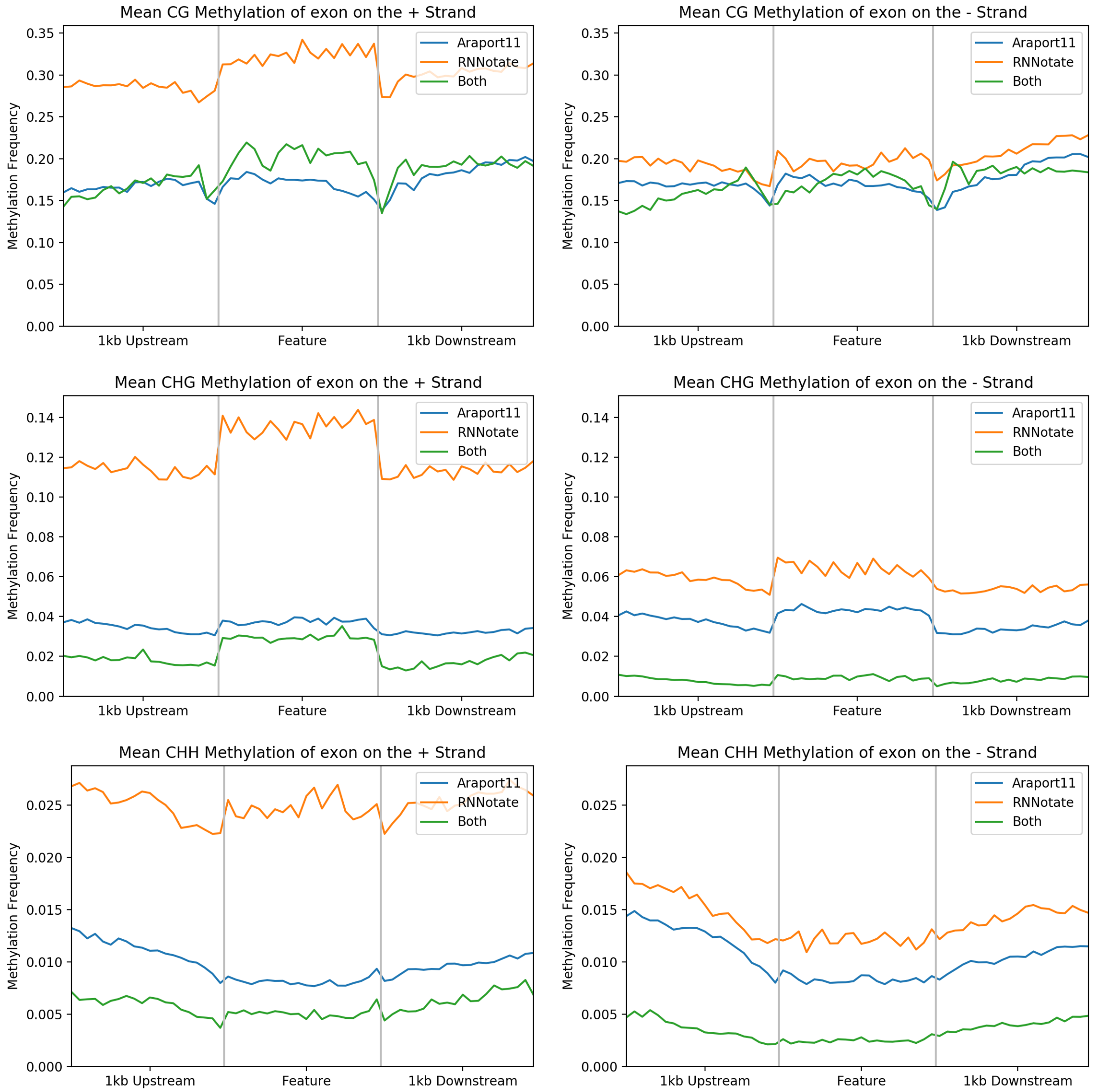

Fig. 4.13: Methylation around exon predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by “both” (green), False-positive predictions are represented by “RNNotate” orange, and False-negatives are represented by “Araport11” (blue).

4.9.2 Regional mRNA methylation

As with exon predictions, RNNotate was twice as sensitive at predicting mRNA regions on the reverse strand (29%) as the forward (16%) as shown in Table 4.7. However, unlike exons, the precision for mRNA regions was twice as high on the reverse strand (31%) as the forward (18%). Looking at methylation patterns for mRNA on the forward strand in Figure 4.14, the 729 true-positives matched the 3977 false-negatives perfectly. In contrast, the 3417 false-positives were hypermethylated and did not match the characteristic pattern. On the reverse strand, there were 1361 true-positive regions whose methylation patterns correlated with the 3354 false-negatives. There were fewer false-positives on this strand (2989), and while no contexts differed as much as those on the forward strand, both CHG and CHH were more correlated than CG methylation. These results show that the model relied more on methylation for mRNA predictions on the reverse strand than it did on the forward strand, and that indicator provided twice the sensitivity and specificity.

4.9.3 Regional CDS methylation

Once again, RNNotate was twice as sensitive on the reverse strand as it was on the forward, with respective rates of 43% and 21%. However, the margin in precision was narrower, with 65% on the reverse strand and 53% on the forward. Looking at CG methylation from both strands in Figure 4.15, the patterns between prediction classes were fairly similar. In contrast, for both CHG and CHH methylation, the valley-like patterns from true- and false-positive predictions were more similar on the reverse strand. On the forward strand, true-positive methylation patterns were

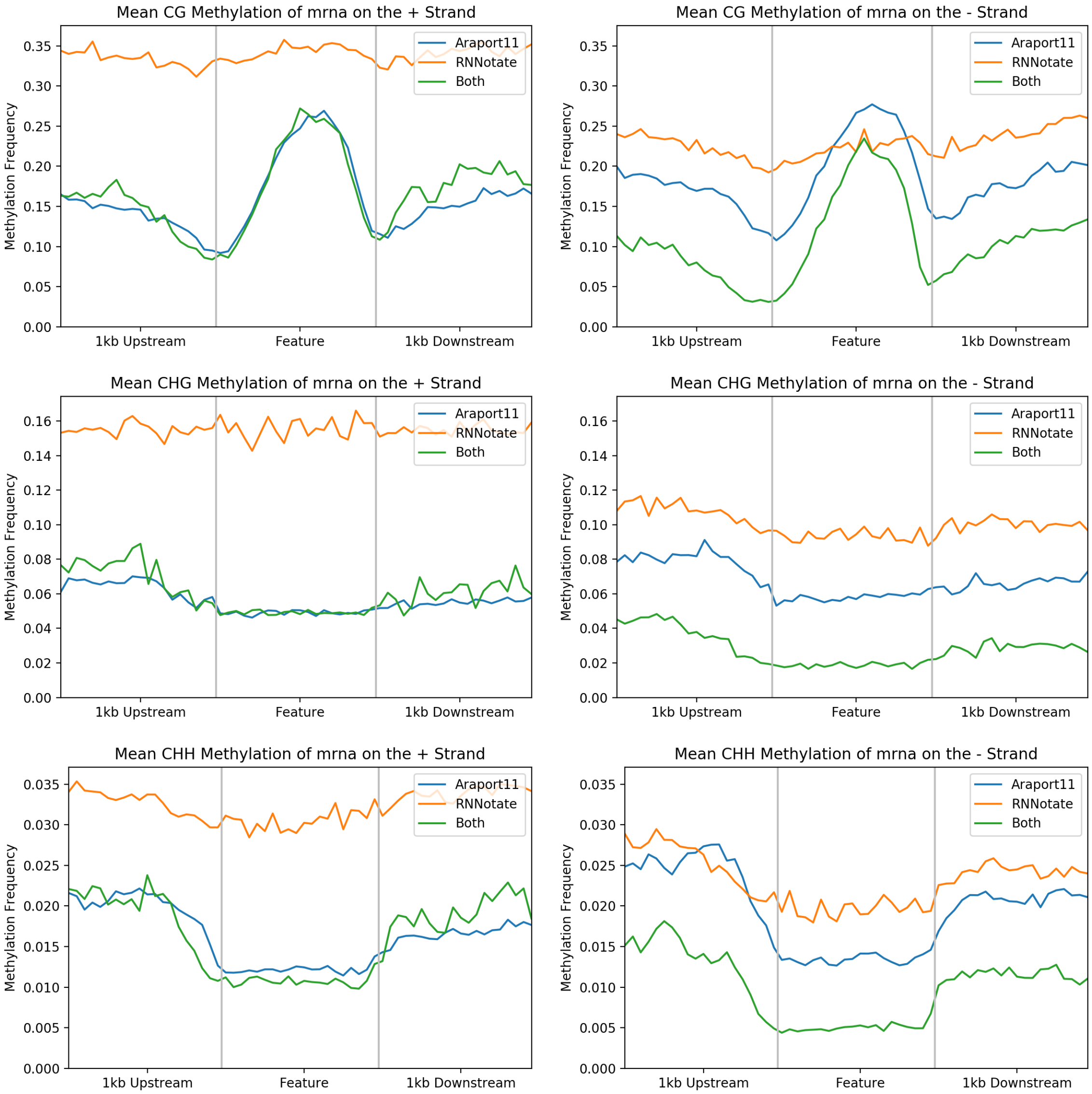

Fig. 4.14: Methylation around mRNA predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue).

similar to true-positive on the reverse, but the false-positive patterns were closer to false-negatives. These trends show that the model relied more on CHG and CHH methylation data for predictions on the reverse strand than it did on the forward. While not perfect, this increased reliance doubled the sensitivity of predictions.

4.9.4 Regional gene methylation

The gene predictions made by RNNotate were also twice as sensitive on the reverse strand (35%) as they were on forward (19%), but precision was only slightly better at 33% and 21% respectively. The patterns for CG methylation in Figure 4.16 were similar between the strands, but false-positive predictions on the reverse strand had lower methylation frequencies than the false-negative pattern. For both CHG and CHH methylation contexts, the true- and false-positive methylation patterns were more correlated on the reverse strand than they were on the forward strand. This could mean that the valley shape that these two contexts create in a gene body could have a larger effect on reverse predictions than those on the forward strand.

4.9.5 Performance conclusions from methylation

If the model is biased towards making predictions based on patterns of DNA methylation, the methylation frequency profile of true-positive (TP) would be similar to false-positive (FP) with a small difference. At the same time, the TP profile would be an exaggeration of the false-negative (FN) trend, where features with the most obvious were detected. Following this logic, if the most obvious trends were removed from the FN pool, the TP profile would be lower than FN for cases of hypomethylation

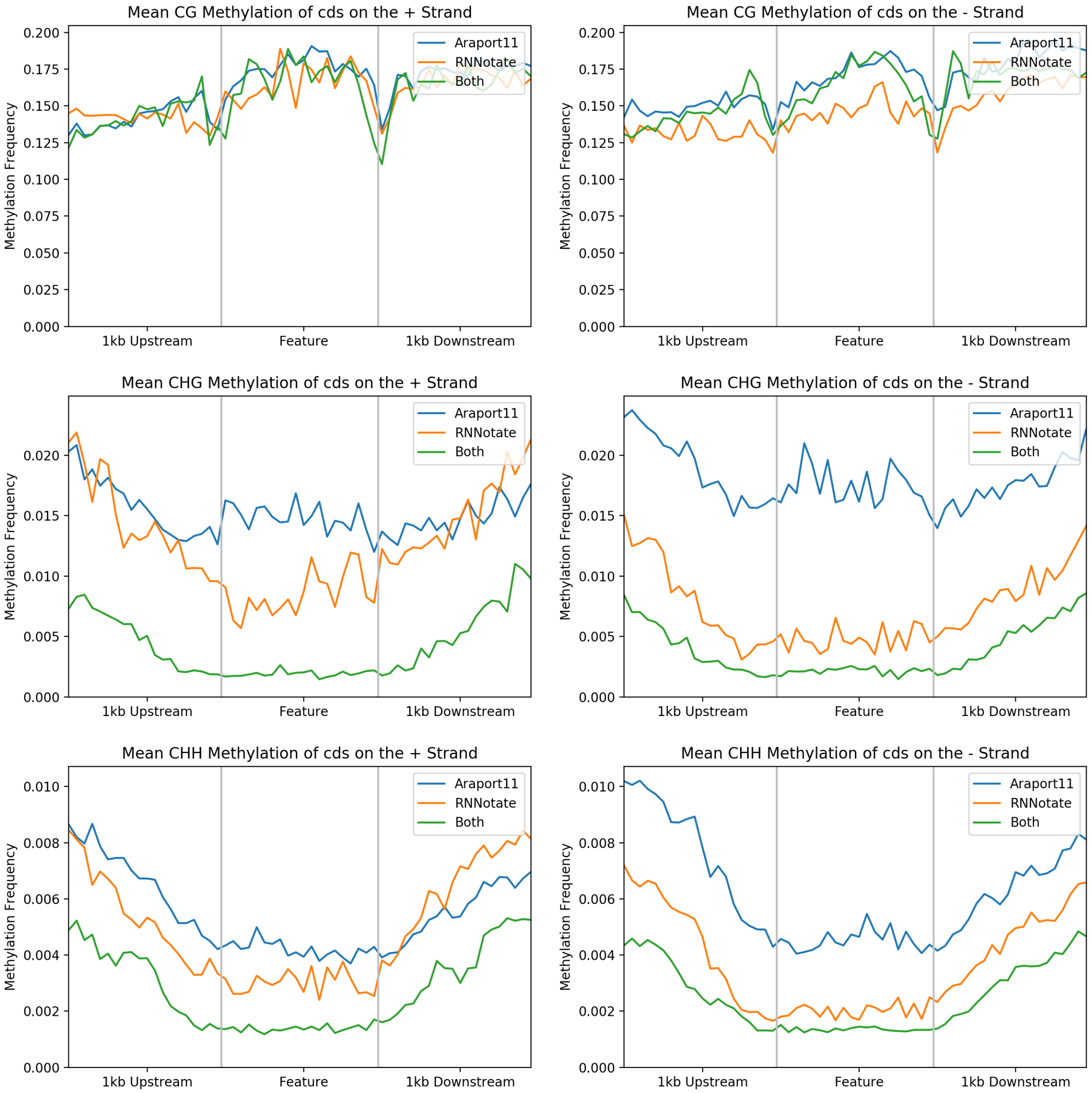

Fig. 4.15: Methylation around CDS predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue).

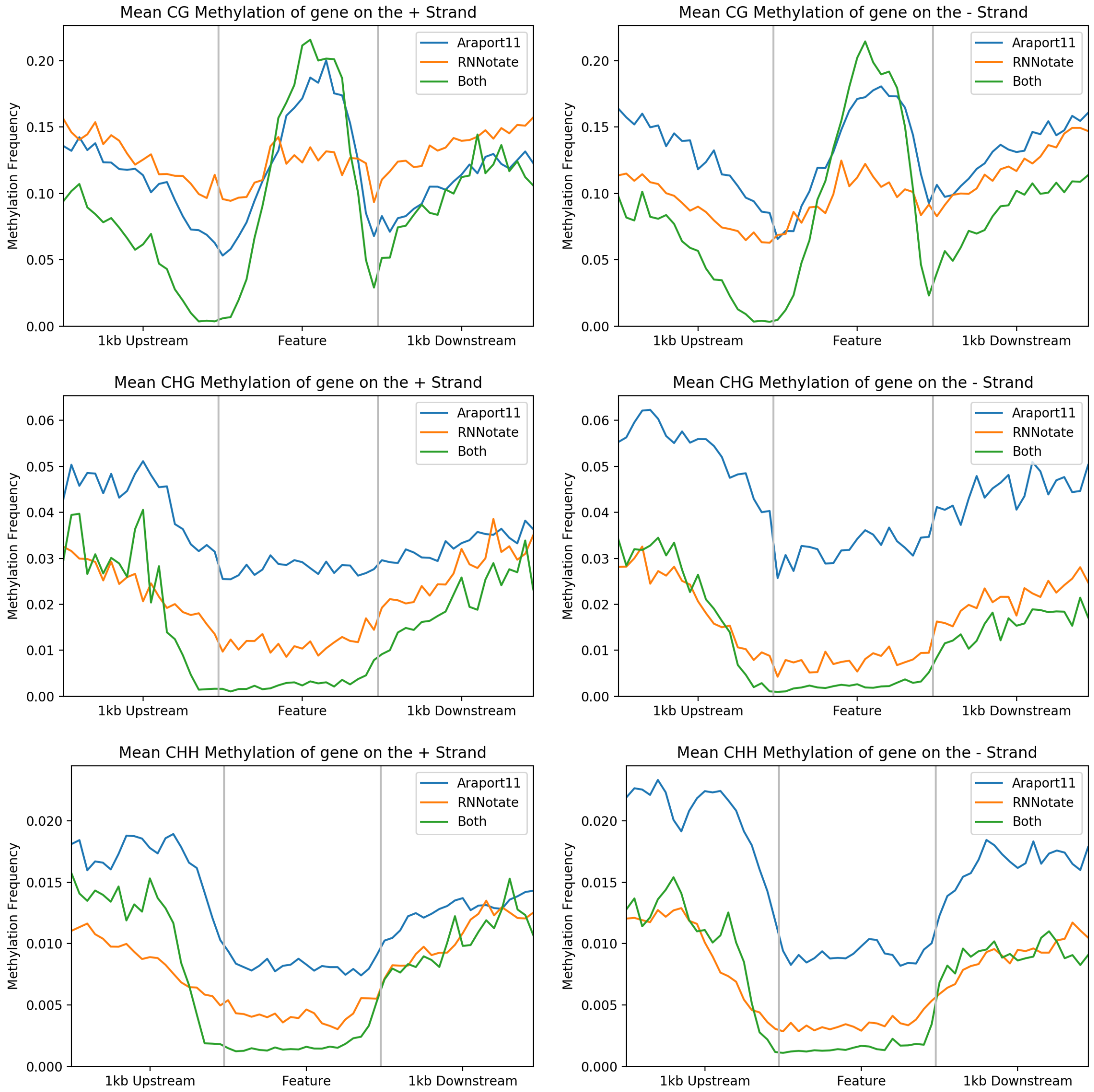

Fig. 4.16: Methylation around gene predictions. Methylation frequencies by strand and context (CG, CHG, CHH) for 3 regions: 20 averaged bins 1kb upstream from feature, 20 averaged bins for feature, and 20 averaged bins 1kb downstream from feature. The gray vertical lines serve to visually separate each region. True-positive predictions are represented by "both" (green), False-positive predictions are represented by "RNNotate" orange, and False-negatives are represented by "Araport11" (blue).

and higher than FN for cases of hypermethylation, and a large difference between the two.

Conversely, if the model tended to not rely on DNA methylation for predictions, the FP profile would be dissimilar to the TP profile. The TP profile would also be extremely similar to FN since the methylation pattern outliers were not detected and removed.

The median squared error between TP and FP profiles (Both and RNNotate) and TP and FN profiles (Both and Araport11) was calculated for each feature, context, and strand combination. A t-test was then used to determine if the error distributions between the forward (+) and reverse (-) significantly differed. The analyses of transcription elements (Table 4.11) and transposable elements (Table 4.12) were conducted separately since their methylation profiles were utilized differently.

Looking at the error values for transcription elements in Table 4.11, the mean MSE(TP-FP) was lower on reverse strand feature predictions at 0.002 than on the forward strand at 0.006. The difference between these two error distributions was also significant at 0.02. This means that when making predictions on the reverse strand, the model was over classifying regions that matched a feature's methylation pattern.

Moving to the mean MSE(TP-FN), the reverse strand was higher than the forward strand at 0.001 and 0.0003 respectively. Even though the mean MSE was 3 times greater on the reverse strand, the t-test did not find the two error distributions to be significantly different. However, since the mean was higher on the reverse strand, the model was overfit to the most extreme methylation patterns.

Tab. 4.11: t-test of Transcription methylation differences. Tabulation of MSE values from each methylation context. For each category of MSE values (TP-FP and TP-FN), a one-tailed t-test was performed to compare methylation differences of features originating from the reverse strand (-) and the forward strand (+).

		MSE(TP-FN)		MSE(TP-FN)	
		-	+	-	+
cds	CHG	1.40E-05	1.84E-04	2.08E-04	1.31E-04
	CHH	2.00E-06	7.80E-05	1.50E-05	1.27E-04
	CG	3.46E-04	5.00E-06	1.35E-04	8.00E-06
exon	CG	1.10E-03	9.92E-03	2.81E-04	1.91E-04
	CHG	2.57E-03	1.41E-02	9.09E-04	4.50E-04
	CHH	1.12E-04	3.71E-04	5.10E-05	1.60E-05
gene	CG	2.19E-03	7.40E-05	3.18E-03	4.29E-04
	CHG	3.20E-05	7.00E-06	9.49E-04	3.20E-05
	CHH	3.00E-06	3.61E-03	7.30E-05	1.34E-03
mrna	CG	1.71E-02	3.10E-02	6.26E-03	3.94E-04
	CHG	5.25E-03	9.15E-03	1.61E-03	6.10E-05
	CHH	1.77E-04	2.69E-04	9.50E-05	5.00E-06
	mean	2.41E-03	5.73E-03	1.15E-03	2.65E-04
	variance	2.38E-05	8.75E-05	3.45E-06	1.42E-07
	t-Stat	-2.1872		1.6497	
	P(T<=t)	0.0256		0.0636	

These two findings show that the model was more reliant on feature-specific patterns of DNA methylation for predictions on the reverse strand. This means that the model was more reliant and possibly overfit to the methylation patterns methylation patterns of FP predictions was more similar to the TP predictions.

Looking at the error values for transposable elements (TEs) in Table 4.12, the mean MSE(TP-FP) was lower on forward strand feature predictions at 0.0023 than on the reverse strand at 0.026. While this was almost a 10 times difference between the two, there were only 3 values for each category, so the t-test did not show a statistically significant difference. While not statistically distinct, the means show

Tab. 4.12: t-test of TE methylation differences. Tabulation of MSE values from each methylation context. For each category of MSE values (TP-FP and TP-FN), a one-tailed t-test was performed to compare methylation differences of features originating from the reverse strand (-) and the forward strand (+).

	MSE(TP-FP)		MSE(TP-FN)	
	-	+	-	+
CG	0.0553	0.0042	0.0278	0.0318
CHG	0.0179	0.0022	0.0186	0.0218
CHH	0.0053	0.0005	0.0088	0.0013
mean	0.0262	0.0023	0.0184	0.0183
variance	0.0007	0.0000	0.0001	0.0002
t-Stat	1.7114		0.0379	
P(T<=t)	0.1146		0.4866	

that reliance on DNA methylation has flipped for classifying TEs, and the forward strand is now over classifying them based on methylation patterns.

The mean MSE(TP-FN) was almost equal for TE predictions on the reverse strand (0.0184) and the forward strand (0.0183). Since there was not a significant difference between the two strands, and the error for both was larger than the 0.00115 and 0.00027 means from the transcription features, we can assume that the model was equally reliant on extreme methylation patterns when making predictions from either strand.

These results found that RNNotate was more reliant on DNA methylation signals when predicting transcription elements on the reverse strand. This aligns with aggregated base level results in Table 4.5, which saw higher sensitivity (63%) and precision (89%) for predictions on the reverse strand in comparison to the forward strand (34% and 78%). We also found that the model has separate mechanisms for classifying TEs since the analysis of DNA methylation patterns showed that forward strand, instead of reverse strand) elements were over classified based on their methylation patterns. This finding was supported by increased sensitivity on the forward strand (31% vs 19%) in Table 4.6.

4.10 Runtime performance

Both the running wall clock time and maximum memory usage were recorded for RNNotate and each of the 4 other tools used for comparison. CPU core utilization was not collected since RNNotate uses GPUs while the other tools are a mix of threaded and serial codes. The results in Table 4.13 show that RNNotate completed annotating the whole *A. thaliana* genome in 15 minutes, 42 seconds using 16.6 gigabytes (GB) of memory. While this is fairly reasonable for modern research-grade code that is used to large resource limits, all historic tools used less than a gigabyte of memory, and all except RepeatScout completed in less time.

Tab. 4.13: Runtime and resource requirements of tools. For each tool, the wall clock runtime was recorded in "minutes:seconds" along with the maximum memory usage in gigabytes (GB).

	Runtime (m:ss)	Memory (GB)
RNNotate	15:42.05	16.6
Augustus	13:17.39	0.33
GlimmerHMM	1:42.30	0.34
RepeatMasker	1:29.25	0.89
RepeatScout	17:15.47	0.92

4.11 Issues limiting portability

While the original goal of this project was to be portable and allow for the model to target both CPUs or GPUs for execution, the CPU version had to be dropped for several reasons discovered during usage. First, Tensorflow 1.14, 1.15 and 2.0 leak memory during each training batch of recurrent neural networks when running on CPUs both with and without Intel's MKL-DNN enabled at a rate proportional to model size, meaning larger models consumed more memory faster as shown in Figure 4.17.

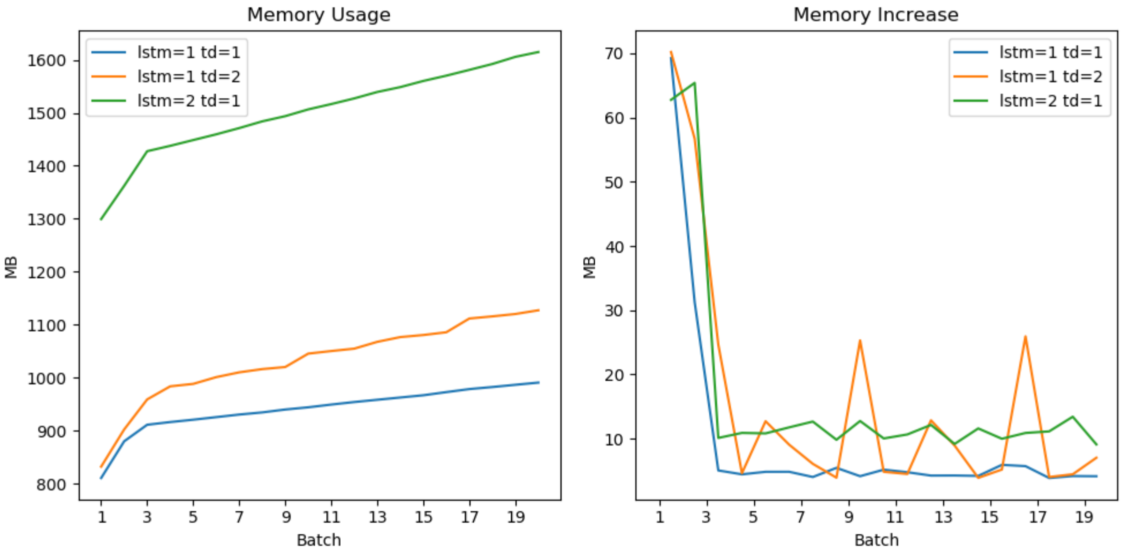

Fig. 4.17: Effect of model architecture on memory consumption by batch. The left figure shows total memory usage after each training batch for three different model architectures. The right figure shows the memory increase after each training batch for three different model architectures. In the figure legends, "lstm=1,2" designates 1 or 2 LSTM layers and "td=1,2" designates 1 or 2 time-distributed hidden layers.

This leak could be mitigated in 1.14 and 1.15 by disabling the MKL-DNN and altering the run config, but it was always present in 2.0. The leak is not present in any of the three mentioned versions when using GPUs for execution. Second, Tensorflow's XLA optimizer encountered numerical errors while computing losses on a CPU, but ran without error on a GPU using Nvidia's implementation. Third, even though CPUs are more accessible and the model handled distributed execution, the model trained much quicker on a single GPU than it did on several HPC nodes with InfiniBand interconnect and significantly more memory for the model and batches of data to reside in.

5. Future Work

5.1 Removing strand specific categories from output

The raw performance results showed that RNNotate excelled at making transcription predictions on the reverse strand, and TE predictions on the forward strand. Our follow-up analysis of DNA methylation patterns around predictions also found that the model increased the weight of DNA methylation patterns when classifying bases. This shows that the model has separate, and unequal, rules when making predictions for each strand. To remove this bias and improve performance overall, we would like to explore two changes to our model. First, we will remove the strand-specific output from the specification. Since the model will no longer be required to determine which strand the data originated from, we hope it will not develop separate processes for classification. Second, since relative location was used by the model to determine the strand of the input, we would like to experiment with removing this dimension. We believe excluding this data may also increase data reusability.

5.2 Improving data variety

Two constant challenges in this machine learning project were juggling the huge quantities of data, and traversing it in such a way that training was not biased towards a specific strand, chromosome, or feature. As demonstrated in Figure 4.3, the optimal model began to overfit to the training data after epoch 20. It is believed that performance can be further improved by diversifying the training data. First, even though it might require truncation, we would include data from multiple chromosomes in each batch. Second, data traversals currently start from the first nucleotide, and allowing starting offsets would slightly change up the batches. Lastly, since the model did not overfit the same data without DNA methylation (Figure 4.10), we would include the option to mutate a certain percentage of nucleotides and methylation values to increase reusability over more training epochs.

5.3 Optimizing hyperparameter search

While the hyperparameter sweep this research used did find an optimal model for predicting the data, there were two problems. First, it was run for 20 epochs, and the best epoch for the optimal model configuration was also found to be after the 20th epoch. Other models may have converged at a lower loss value at a later epoch, so hyperparameter optimization should be run for more epochs with future version. Second, finding the best performing model with 648 combinations of input parameters ran for 7 hours on 124 Nvidia v100 GPUs, which are server-grade and expensive, and acting as the bottleneck for future work. To reduce the resources required to identify an optimal architecture, RNNotate could benefit from a hyperparameter optimization method that does not use brute-force. Both Talos [68] and Tune [69] are third-party hyperparameter optimizers that randomly sample the hyperparameter space and iterate towards an optimal choice. Talos runs sequentially on a single computer, limiting the dataset size it can evaluate, but Tune can scale to multiple servers in the cloud or high-performance computing (HPC) cluster. Both options require significant refactoring of the RNNotate code, which prevented their incorporation in this initial version.

5.4 K-mers to improve the prediction of coding features

While the hyperparameter sweep found that convolving the input had a minimal effect on the predictive power of the model, we would like to include the option to represent the reference sequence as k-mers, which are analogous to n-grams, in future versions. We believe this feature would be worthwhile because the convolutional layer transformed the entire input, while the k-mer representation would only affect the reference base sequence. Depending on the value of "k", or width of the window, the complexity of the input data will significantly increase, so we will also explore the merits of encoders to stratify inputs.

5.5 Incorporating additional data dimensions

Draft genome studies usually begin with a reference assembled from DNA and an annotation from a combination of predictions on the DNA and RNA. The A. thaliana annotation is on the eleventh version (Araport11), and the primary source behind each improvement is the incorporation of information learned from novel data such as long iso-seq reads [70], chromatin immunoprecipitation and sequencing (CHIP-seq) [71], and other epigenetic marks.

While this initial version of RNNotate focused on incorporating DNA methylation data, generic interfaces for adding BAM and bedGraph data as model input will be developed. Both of these formats support random reads and can effectively represent the target signal from RNA-seq, CHIP-seq, and many other protocols. While each combination of input dimensions will require a separate model, it is my hope that RNNotate will be able to contribute to quality annotations at all stages of study.

6. Conclusions

RNNotate was developed as a novel method for genome annotation, which used a recurrent neural network for the actual prediction. Neural networks for prokaryotic annotation exist, but we designed efficient methods to enable eukaryotic annotation, while also exploring the effect of including DNA methylation data to improve predictions.

A hyperparameter search identified an optimal model architecture, which was trained on *A. thaliana* chromosomes 1, 2, 4, and 5. We found that RNNotate detected fewer CDS elements than GlimmerHMM and Augustus at predicting CDS elements, but those predictions were more often correct. We also found that RNNotate was more accurate than both RepeatMasker and RepeatScout at predicting transposable elements. RepeatScout was slightly more sensitive than RNNotate, but returned twice as many false-positives.

When RNNotate was trained and run without DNA methylation, its predictions were significantly less sensitive, predicting fewer positive feature targets overall and slightly increasing the precision. The gene model predictions that were originally whole when DNA methylation was included became fragmented without it as well. We also found that DNA methylation patterns were used to a greater extent in predictions on strands with superior performance.

These results from this initial version of RNNotate are promising, but not ready to replace trusted tools and pipelines; however, it could be used to supplement them. Excluding DNA methylation from the input to RNNotate demonstrated its merits for generating annotation predictions, and its inclusion from long-reads will benefit the study of novel organisms. We hope that removing strand specific output and improving the training process will remove some strand biases and improve the results overall.

In addition to the direct outcomes, this work has made several peripheral contributions to the bioinformatics community. The BSMAP methylation caller was optimized and parallelized to enable faster analysis. Programming interfaces for randomly accessing methylation data were created and published. A numerical specification and programming interfaces for reading and writing annotations numerically was created. Lastly, a well documented, flexible framework for analyzing genomic data with recurrent neural networks was developed, validated on multiple types of hardware, and published for other researchers to create their own models and investigate their own hypotheses.

7. Appendix

7.1 Software

All software developed for this research is available on GitHub:

RNNotate: `https://github.com/zyndagj/RNNotate`

differannotate: `https://github.com/zyndagj/differannotate`

BSMAPz: `https://github.com/zyndagj/BSMAPz`

7.2 Data

All data used for this research is open-source and available online:

Arabidopsis_thaliana.TAIR10.dna.toplevel.fa TAIR10 genome assembly used by Araport11 annotation. `ftp://ftp.ensemblgenomes.org/pub/plants/current/fasta/arabidopsis_thaliana/dna/Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.gz`

PRJEB6701 Study containing bisulfite-sequencing reads from wild-type stem and root samples of *A. thaliana*

Araport11_GFF3_genes_transposons.201606.gff The Araport11 annotation GFF3 file.

`https://www.arabidopsis.org/download_files/Genes/Araport11_genome_release/Araport11_GFF3_genes_transposons.201606.gff.gz`

TAIR10_Transposable_Elements.txt Identities of repeats in Araport11 annotation.

`https://www.arabidopsis.org/download_files/Genes/TAIR10_genome_release/TAIR10_transposable_elements/TAIR10_Transposable_Elements.txt`

8. Additional Work

The following sections are all quoted directly from their relevant publications. The parts included were directly contributed by the author. Full texts and references can be found with the original publications.

8.1 Feature Frequency Profiles for Automatic Sample Identification using PySpark

8.1.1 Abstract

When the identity of a next generation sequencing sample is lost, reads or assembled contigs are aligned to a database of known genomes and classified as the match with the most hits. However, any alignment based methods are very expensive when dealing with millions of reads and several thousand genomes with homologous sequences. Instead of relying on alignment, samples and references could be compared and classified by their feature frequency profiles (FFP), which is similar to the word frequency profile (n-gram) used to compare bodies of text. The FFP is also ideal in a metagenomics setting to reconstruct a mixed sample from a pool of reference profiles using a linear model or optimization techniques. To test the robustness of this method, an assortment of samples will be matched to complete references from NCBI

Genome. Since a MapReduce framework is ideal for calculating feature frequencies in parallel, this method will be implemented using the PySpark API and run at scale on Wrangler, an XSEDE system designed for big data analytics.

8.1.2 Introduction

In the realm of natural language processing the idea of n-grams, or overlapping subsequences of n items, is commonly used to summarize and model large bodies of text. Documents are broken down into word or character n-grams and the occurrence of each n-gram is tallied. Besides showing popular n-grams and relationships between entities, this is also a simple way to transform text into numerical vectors for quantitative analysis. N-grams have been previously used for genre classification [72], spam detection [73], and authorship identification [74].

Computational biology often borrows from other fields as technology progresses. Because DNA is long sequential chain of values, methods are often adapted from time-series analysis, but methods from natural language processing can be used as well. Synonymous to n-grams, computational biology has k-mers, which are subsequences of overlapping k-length nucleotides along strands of DNA. While the idea is simple, it is robust enough to be a universal statistic for genomes of greatly different sizes an origin. K-mers have been used for error detection [75], sequence assembly [76], repetitive element detection [16], sample classification [77], and genome comparison [78].

Using k-mers for sequence comparison is advantageous over common methods because they don't require alignment. Whenever large sequences are compared, local alignment methods are used to find the best match while also allowing for differences.

Alignment isn't used for comparing kmers, because they are just counted. All differences in sequence result in different overall levels or k-mers unique to a specific dataset. In the context of sample read identification, each of the hundreds of millions of reads from a next-generation sequencing run need to be aligned to a database of each possible genome. This is the same process that occurs when a read is identified with BLAST [79]. Local alignment is an $O(N^2)$ dynamic programming method, so repeating it for large numbers of references isn't ideal. The NCBI genome archives will continue to grow at an exponential rate as shown in Figure 8.1, making any alignment-free methods attractive.

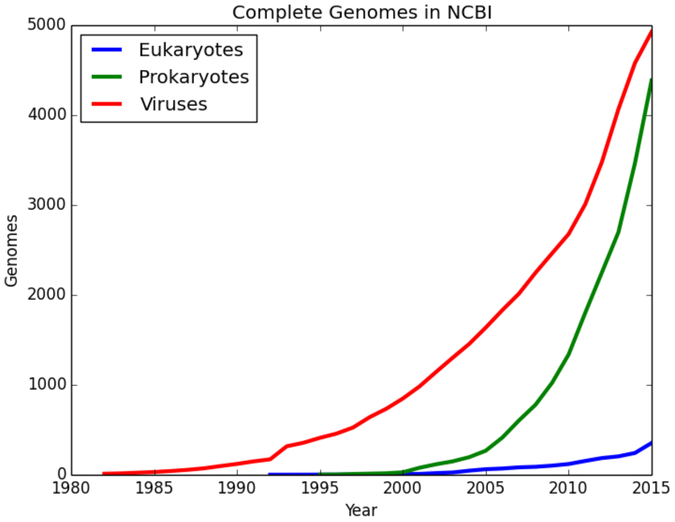

Fig. 8.1: Number of complete genomes archived by NCBI genome.

One of the most popular models for text analysis is MapReduce, which maps functions to pieces of data in parallel, and then reduces the result together. Since the preprocessing and statistical methods are very similar, we propose the use of Apache Spark for distributed k-mer frequency calculation of reference genomes and sequencing reads [80]. Not only will Spark's partitioning methods split the data between workers and to disk when there is too much to hold in memory, but it will also orchestrate the parallel computations taking place. This will circumvent the drawbacks of code that have high memory requirements and only function single-node in shared memory like jellyfish [81].

8.1.3 Related work

K-mer frequencies have been previously utilized for both genome comparison and sequencing read classification. Sims et al. utilized k-mer feature frequency profiles to compare genomes of varying lengths using the Jensen-Shannon divergence. They then constructed phylogenies based on their computed divergences which closely resembled the official taxonomic phylogenies from NCBI. Wood and Salzberg created the tool Kraken to match reads to their genus of origin based on k-mer presence. Kraken does this by constructing a database from a collection of genome references and a known taxonomy to collapse k-mers to their lowest common ancestor. Because our database is distributed across multiple executors instead of together on a single computer, our implementation skips this preprocessing step and relies on the quantity of the information in the genome references over the quality of the taxonomy.

Kraken has been shown to be very precise when classifying reads, but is inaccessible to the average user for two main reasons. First, it requires at least 70GB of system memory to run and most XSEDE systems have 2GB of memory per core. This means some kind of large-memory resource is required. Second, it is highly dependent on its database. Kraken requires that each genome be included in an accompanying taxonomy. Sequencing is getting cheaper and haploid assembly is push-button since the advent of long-read technologies [82], but curated taxonomies can lag behind. Kraken also requires that the database be rebuilt whenever samples change, and this is a time-consuming process bound by disk operations.

Apart from sequence classification and k-mer counting, here has been some recent work to support bioinformatics analysis on the Spark platform with the ADAM suite [83]. However, this is not general purpose and specifically targets the storage and analysis of specific types of files. ADAM also focuses on analyses that translate well to tabular data, like point mutations in the variant call format. These point mutations are sets of mutation coordinates, and an analysis across samples is well suited to Spark's current native data analysis abilities.

8.1.4 Spark For Genomics

Even with the push of ADAM and large companies that specialize in MapReduce like Google starting to get into the field of genomics, the platform is still gaining traction within the community. We however, saw that the feature frequency profile was ideal for transforming genomic data into vectors more suited to a MapReduce framework. We implemented sparkmer using PySpark, the Python API to the Spark platform. PySpark allows for full Spark utilization without writing any non-python code or multiple command line calls like with Hadoop Streaming. Besides MapReduce becoming more accessible, Spark has full access to python's 67,000 packages.

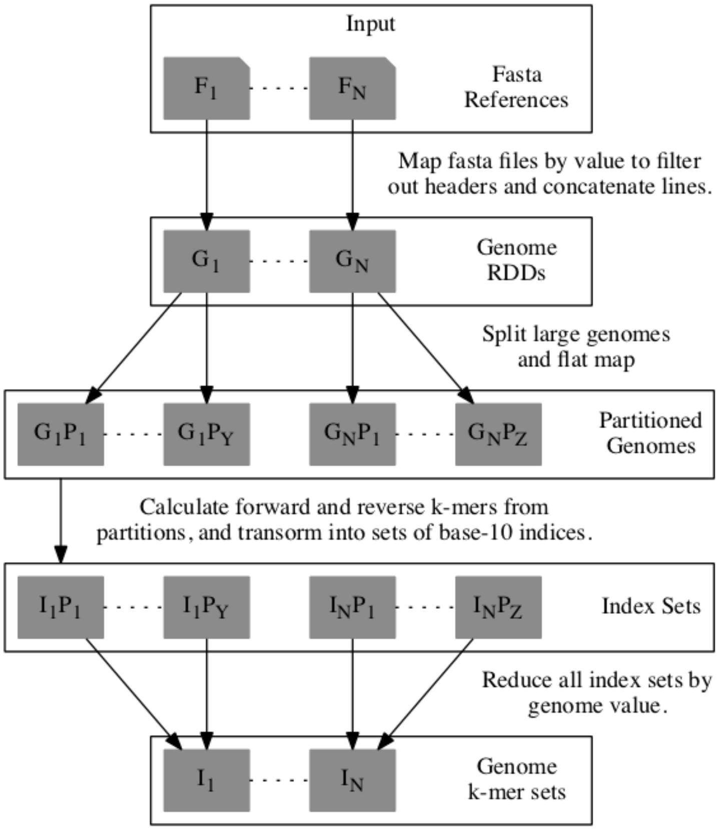
Input
F1
FN
Fasta References
Map fasta files by value to filter out headers and concatenate lines.
G1
GN
Genome RDDs
Split large genomes and flat map
G1P1
G1PY
GNP1
GNPZ
Partitioned Genomes
Calculate forward and reverse k-mers from partitions, and transorm into sets of base-10 indices.
I1P1
I1PY
INP1
INPZ
Index Sets
Reduce all index sets by genome value.
I1
IN
Genome k-mer sets

Fig. 8.2: PySpark workflow for reference k-mer counting.

8.1.5 Approach

8.1.5.1 Reference k-mer Counting After transferring a collection of Fasta references to the hadoop distributed file system (HDFS), all k-mers, for a specified k, are counted as laid out in Figure 8.2. First, all fasta files are filtered to remove header sequences and line breaks, and then concatenated into a contiguous sequence. For reference sets with large genomes, like eukaryotes, sequences can be split into sub-sequences with k-bp (k-mer) of overlap and repartitioned for evenly distributed processing.

k-mers for each section are then computed and immediately transformed into an index. The indices are calculated by treating each k-mer as a base-4 representation of a base-10 index, where {A:0, G:1, C:2, T:3}. We decided to use the base-4 index method as opposed to the more commonly used hash index after having two key collisions when counting 3-mers with 2^{10} valid keys. Not only does our indexing method avoid collisions, but it also runs in constant time and does not require a lookup table for the inverse.

Reads will be classified based on Jaccard Similarity, to index arrays are transformed into Python sets. The sets will not only allow for efficient unions and intersections during when calculating similarity and reducing, but also is a sparse representation of k-mer presence. Sparse data types are necessary when counting k-mers because a dense vector to keep track of all 20-mers requires 4TB of memory. We also experimented with sparse vector structures, but they were unnecessary since we never use the count, just the presence of a k-mer.

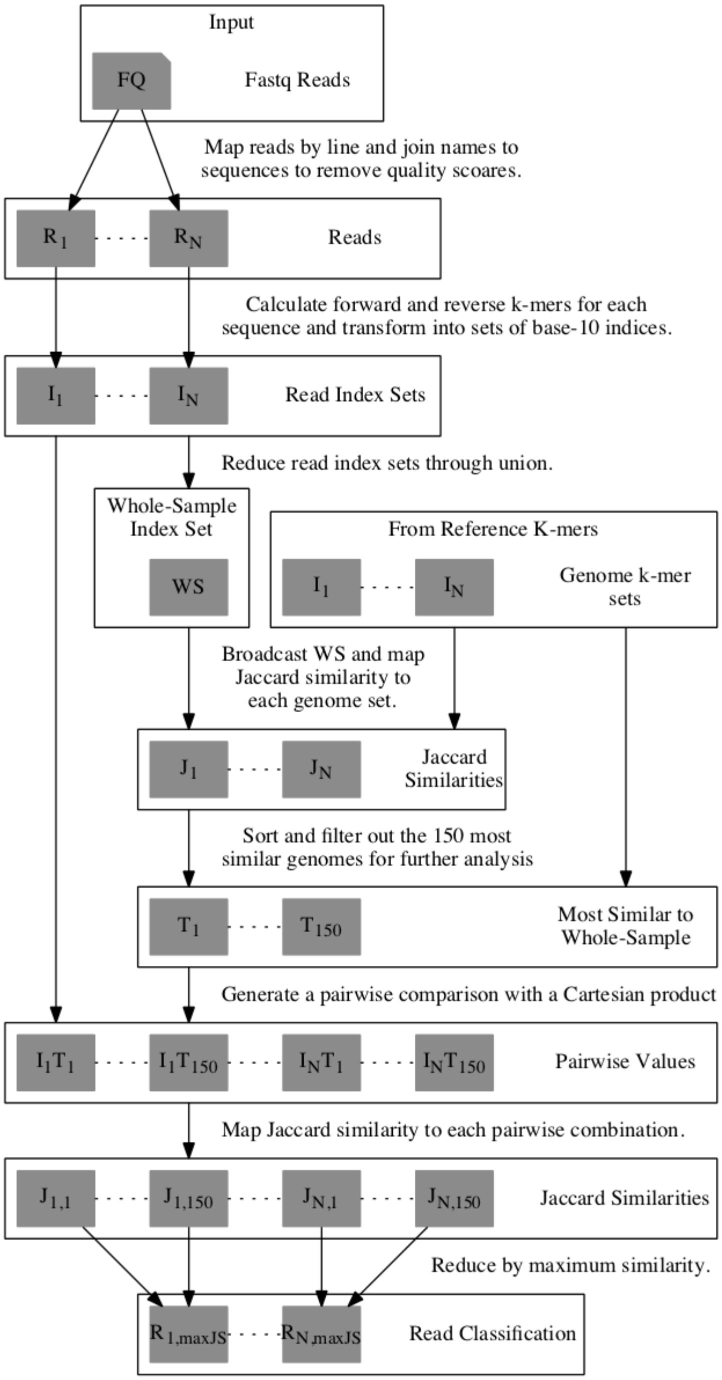
Input
FQ
Fastq Reads
Map reads by line and join names to sequences to remove quality scoares.
R1
RN
Reads
Calculate forward and reverse k-mers for each sequence and transform into sets of base-10 indices.
I1
IN
Read Index Sets
Reduce read index sets through union.
Whole-Sample Index Set
WS
From Reference K-mers
I1
IN
Genome k-mer sets
Broadcast WS and map Jaccard similarity to each genome set.
J1
JN
Jaccard Similarities
Sort and filter out the 150 most similar genomes for further analysis
T1
T150
Most Similar to Whole-Sample
Generate a pairwise comparison with a Cartesian product
I1T1
I1T150
INT1
INT150
Pairwise Values
Map Jaccard similarity to each pairwise combination.
J1,1
J1,150
JN,1
JN,150
Jaccard Similarities
Reduce by maximum similarity.
R1,maxJS
RN,maxJS
Read Classification

Fig. 8.3: PySpark workflow for classifying input reads.

8.1.6 Classifying Reads

Reads are then transformed into sets of k-mer indices using the same method for genomes and classified as shown in Figure 8.3. While developing this process, we experienced over-allocation errors from the executors after calculating read k-mers, no matter the number of partitions. After mapping the partitions of our data and counting the number of records in each and plotting the histogram (Figure 8.4), we discovered that a majority of the Resilient Distributed Dataset (RDD) partitions were being left empty. We are not sure if the low complexity read names all hashed to the same partitions or filtering and joining the reads caused the problem, but neither shuffling nor repartitioning had any effect on the distribution of the data. We finally fixed this by manually partitioning the reads with an integer index. This forced a uniform distribution of the data across the partitions, so no executors were overwhelmed later in the analysis.

Large quantities of reads and large reference databases leads to huge number of pairwise comparisons. To reduce the number of required comparisons, we filter our database down to 150 probable candidates. These candidates are determined by first creating a global k-mer set by reducing all input read sets with with a union. This global k-mer set is then compared to all 11,112 bacteria and virus genomes, and the 150 references with the highest similarity are kept. Then each of the read sets is compared to each of the 150 best-candidate references by mapping the Jaccard Similarity to the Cartesian product of the two RDDs and reducing each read by the maximum similarity as the final classification.

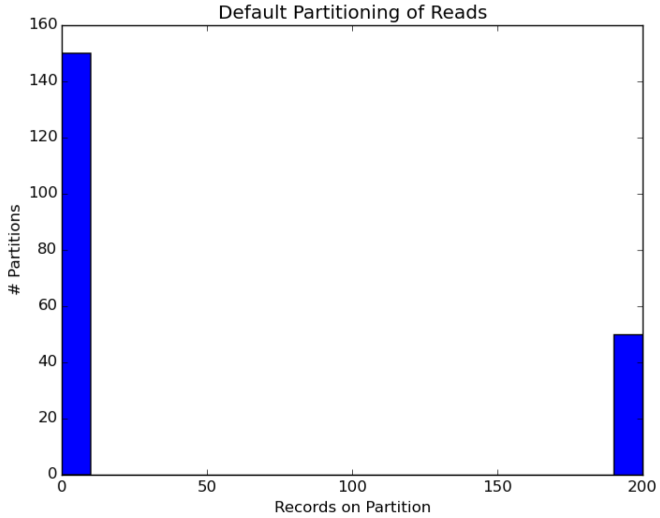

Fig. 8.4: Histogram illustrating the default partitioning of the reads RDD after k-mer calculation.

8.1.7 Methodology

We tested sparkmer on a 26 node (1 master, 25 workers) hadoop instance on the XSEDE system Wrangler. Wrangler is an ideal platform for Spark becuase each node has 4TB of EMC flash for the hadoop distributed file system, making Sparks ability to spill partitions to disk as efficient as possible. To validate sparkmer, we used the same 11,112 references (5,242 bacterial and 5,870 viral) used in the database for Kraken. Then, sparkmer was run using the same HiSeq_accuracy dataset created to test the accuracy of Kraken [84]. The HiSeq_accuracy dataset consists of 10,000 reads from 10 sources of origin:

- *A. hydrophila*
- *B. cereus*

- *B. fragilis*
- *M. abscessus*
- *P. fermentans*
- *R. sphaeroides*
- *S. aureus*
- *S. pneumoniae*
- *V. cholerae*
- *X. axonopodis*

Each of the 10 sources contributed 1,000 reads. Lastly, HiSeq_accuracy was converted to fastq format, a standard format for reads coming from a next-generation sequencer.

8.1.8 Results

No matter how dissimilar a read was to the 150 possible candidates it is compared to by sparkmer, an identity is always assigned. While this practice makes sense since each fragment of DNA had to originate from some organism, it does not ensure that the species of origin is present in the reference database. Kraken goes the opposite route and will label reads as unknown when there is an insufficient number of k-mers unique to a specific genus. To facilitate this unique requirement, Kraken has a default k-mer size of 31, keeping memory requirements high. The differences between these two methods can be obviously seen in the classification precision, which is the number of correctly-classified reads over the total number of classified reads, in Table 8.1. Kraken had a precision rate of 98% while sparkmer had 70%. These results may seem disparate, but Kraken left 20% (2,000) of its reads unclassified, even while using the extremely large 31-mers for analysis. Even though sparkmer classified all 10,000 reads, its sensitivity, or the number of correctly-classified reads of the total number of reads, was still lower than Krakenś, but it also used 15-mers for the analysis.

Tab. 8.1: Accuracy of read classification from sparkmer and Kraken.

	sparkmer	Kraken
Correct	6951	7760
Incorrect	3049	131
Unclassified	0	2109
Precision	69.5%	98.3%
Sensitivity	69.5%	77.6%

To test how well sparkmer scaled on our 24 worker nodes, we ran the HiSeq_Accuracy test with 60, 90, 120, and 150 executors. Sparkmer experienced strong scaling up until 150 executors, where the network traffic when reducing by key ended up being the bottleneck. Sparkmer's fastest runtime was 1 hour; classifying 167 reads per minute on average. Even though sparkmer could take advantage more processors than the 24 that Kraken ran on, Kraken was much faster and classified reads at a rate of 1422 reads per minute on Wrangler.

8.1.9 Conclusion

Based on the runtime analysis, PySpark is an ideal framework for counting k-mers, because sparkmer counted them across all 11,112 genomes in 3 minutes. Even

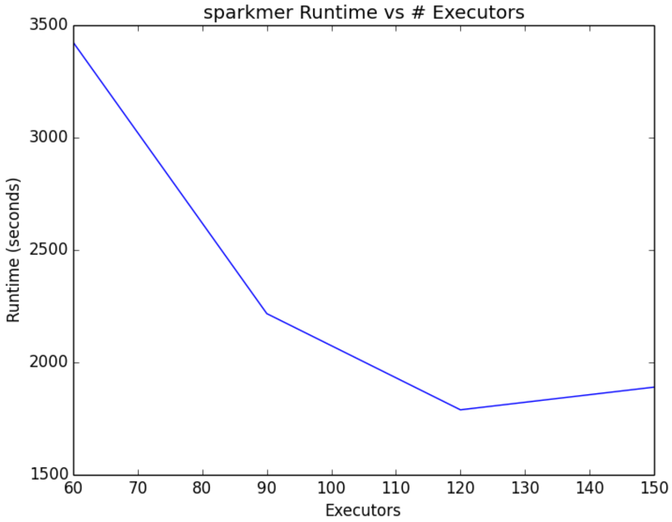

Fig. 8.5: Runtime of sparkmer on increasing numbers of executors.

though sparkmer was 10 times slower than Kraken for actual classification tasks, it still shows promise. Sparkmer was a naive approach to a difficult problem, and it was exciting to see it not only become feasible using the PySpark framework, but also scale without manually orchestrating communication and each task.

These initial results were promising, so we plan on exploring new ways to improve the performance of sparkmer. The final distance reduction by read name was one of the costliest because of the sheer number of keys to reduce by. The PySpark streaming API is still immature, but we hope to use is to process large files of reads in small windows so there are fewer keys to reduce by. This may even allow for the inclusion of all genomes in the final similarity computation. If not, we will continue using a filtered set of probable candidates to reduce the number of pairwise comparisons. However, we will improve on this step by collapsing extremely similar references to reduce the number of redundant genomes. Hopefully these changes not only increase the throughput of sparkmer in the future, but also increase the precision, so a commodity system running sparkmer can compete with the Kraken running on large-memory resources.

8.1.10 Availability

sparkmer is written in Python using the PySpark API and is available for download from `https://github.com/zyndagj/sparkmer`.

8.2 Repliscan: a tool for classifying replication timing regions

8.2.1 Abstract

8.2.1.1 Background Replication timing experiments that use label incorporation and high throughput sequencing produce peaked data similar to ChIP-Seq experiments. However, the differences in experimental design, coverage density, and possible results make traditional ChIP-Seq analysis methods inappropriate for use with replication timing.

8.2.1.2 Results To accurately detect and classify regions of replication across the genome, we present Repliscan. Repliscan robustly normalizes, automatically removes outlying and uninformative data points, and classifies Repli-seq signals into discrete combinations of replication signatures. The quality control steps and self-fitting methods make Repliscan generally applicable and more robust than previous methods that classify regions based on thresholds.

8.2.1.3 Conclusions Repliscan is simple and effective to use on organisms with different genome sizes. Even with analysis window sizes as small as 1 kilobase, reliable profiles can be generated with as little as 2.4x coverage.

8.2.2 Background

The most essential property of the cell is its ability to accurately duplicate its DNA and divide to produce two daughter cells [85]. The cell's replication cycle starts with G1 phase, in which molecules essential for cell division are produced, then proceeds to replicating DNA in S phase. After all DNA in the genome is duplicated, the cell continues to grow in G2 phase until it divides into two daughter cells at the end of Mitosis, or M phase, at which point it is ready to start the cell cycle again (Figure 8.6).

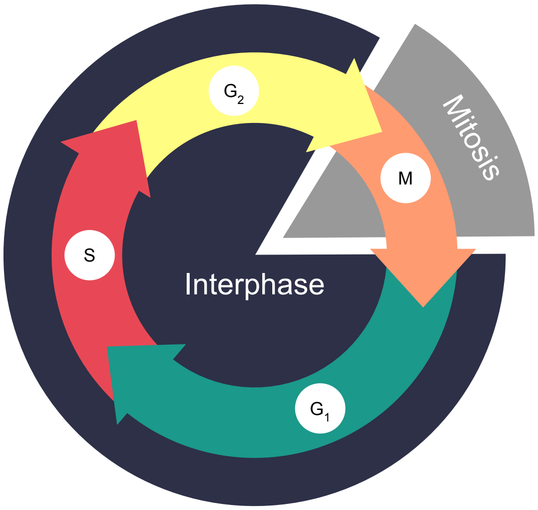

Fig. 8.6: Overview of the cell cycle. Cell division takes place in two stages: interphase and mitosis. Interphase is when a cell copies its genome in preparation to physically divide during mitosis. Interphase starts with cell growth and preparation for DNA synthesis in Gap (G1). After G1, DNA is replicated in regions during the Synthesis (S) phase. The cell then transitions into a second growth phase - Gap 2 (G2). When the cell has finished growing, the cell divides into two daughter cells in Mitosis (M).

To ensure accuracy and efficiency, S phase is complex and highly regulated. Instead of duplicating in a single zipping motion, reminiscent of transcription, DNA is synthesized in regions at distinct times in eukaryotes, initiating at multiple origins of replication [86]. This synthesis process takes place in a live cell, so replication mechanisms need to be coordinated with active transcription, chromatin configuration, and three-dimensional structure [87]. For example, early replication correlates with chromatin accessibility [88].

To better understand the coordinated program of DNA replication, two types of protocols have been developed to examine genome-wide replication profiles based on DNA sequencing data. One based on the time of replication, TimEx [89], [90], and the other based on incorporation of a labeled precursor into newly replicated DNA, Repli-seq [91]–[96]. Time of replication (TimEx) measures DNA coverage at sequential times in S-phase. The normalized early S-phase signal should be mostly 1x coverage, additively transitioning to 2x coverage in late S-phase. In contrast to this method, Repli-seq works by only sequencing newly replicated DNA. Theoretically, in a single cell, this means once a region is replicated, it should not appear in samples taken at later times, except in the case of allelic timing differences. Both methods have been shown to yield similar results [97], [98] for when and where genomic regions replicate, but each requires a distinct type of analysis. The methods described in this paper focuses on data produced by label incorporation (Repli-seq).

8.2.2.1 Data Description In continuation to our analysis of *A. thaliana* chromosome 4 in 2010 [99], we updated our laboratory protocol to be more stringent as described in Hansen *et al.* 2010[96], Bass *et al.* 2014[100], Bass *et al.* 2015[101], and Wear *et al.* 2016[102]. We increased the sensitivity of the labelling process by using 5-Ethynyl-2'-deoxyuridine (EdU), which does not require harsh denaturation of DNA, unlike 5-Bromo-2'-deoxyuridine (BrdU) used in previous work. A flow cytometer is then used to separate labeled from unlabeled nuclei, and to resolve labeled nuclei into different stages of S phase based on their DNA content. Next, DNA is extracted from sorted nuclei. The newly replicated DNA is immunoprecipitated and then sequenced using an Illumina sequencer. Previous protocols used microarrays for labeled DNA detection, which provided signal on probes at fixed intervals across a genome. Directly sequencing the immunoprecipitated DNA allows for a continuous display of replication activity across the genome.

Following the Repli-seq protocol, we created an exemplar *A. thaliana* dataset for development, with nuclei from: G1 (non-replicating control) and early, middle, and late S phase. While the amplification, fragmentation, and sequencing of next generation sequencing (NGS) libraries should be unbiased and random, physical factors affect the sequenceability of each region. To correct for these effects, we use the raw non-replicating DNA from the G1 control to normalize any sequenceability trends.

8.2.2.2 Introducing Repliscan In addition to our updated laboratory protocol for generically measuring DNA replication, we needed to improve the sensitivity and robustness of our analytical method. In previous work, log-ratios and aggressive smoothing were used to classify genomic regions by their time of replication. While this yielded results with high true positive rates, we found that this approach over-smoothed our deep coverage, next generation sequencing data. We created the Repliscan method to analyze generic, DNA sequence-based replication timing data without user-specified thresholds. Accepting any number of S-phase timepoints as input, Repliscan removes uninformative or outlying data, smooths replication peaks, and classifies regions of the genome by replication time.

8.2.3 Implementation

The analysis of the replication time data starts like any other DNA sequencing analysis, with quality control, mapping, and alignment filtering. Quality control consisted of removing contaminating 3' universal sequencing adapters from the paired reads, and trimming the 5' ends with quality scores below 20 with the program Trim Galore![103] version 0.3.7, which is designed to maintain read pairs. While it is obvious that low-quality regions need to be removed or masked because those base calls are untrustworthy, any contaminating sequences from adapters hinder the alignment process even more because they are always high-quality and may comprise a large part of the read. Therefore, reads in the output from Trim Galore! shorter than 40 base pairs were discarded, and resulting singletons (unpaired reads) were not included for alignment.

We then used BWA-MEM [104] version 0.7.12 with default parameters to align the quality-filtered reads to the TAIR10 *A. thaliana* reference genome [46]. After alignment, we filtered out any reads with multiple alignments using samtools[49] version 1.3. Removing these non-uniquely aligning reads is essential because they come from repetitive elements or other duplications in the genome that could replicate at different times, thereby confounding region classification into discrete replication times. After our stringent alignment requirements, fewer than 0.5% of our reads were identified as duplicates by samtools. We decided that removing the duplicates from our data was unnecessary due to the depth of our sequencing and localized nature of replication peaks. We also performed a correlation analysis of our samples and replicates, confirming their high level of similarity.

8.2.3.1 Windowing The DNA sequencing workflow leaves us with raw replication signals across a genome, which we must classify into distinct genomic regions and assign replication times. Our methods for this process build on methods from Lee *et al.*[99] and are illustrated in Figure 8.7.

At first glance, Repli-seq data appears similar to dense ChIP-seq data [105], when viewed in a genome browser (Figure 8.8). However, instead of highlighting a limited number of coverage peaks as sites of molecular interactions, replication timing data consists of coverage across the entire genome accented with extremely wide peaks corresponding to regions of replication initiation and subsequent spreading. This background coverage with subtle, broad increases in depth makes deep coverage essential to reduce sampling error when detecting statistically-relevant differences. Even though the cost of sequencing has plummeted since 2007, deep-coverage DNA sequencing is still expensive for higher eukaryotes.

Lee *et al.* defined putative replicons in *A. thaliana* and calculated the median length to be 107 kilobases [99]. To achieve greater signal depth in each replication timing sample, we transformed each BAM alignment file into 1 kilobase coverage windows using bedtools [106]. While this transformation slightly reduces the resolution of our analysis, Figure 8.8 shows that the proportion of sampling error to measured signal is greatly reduced with the increased coverage. The windows also put all changes in coverage on the same coordinate system, simplifying comparisons between samples

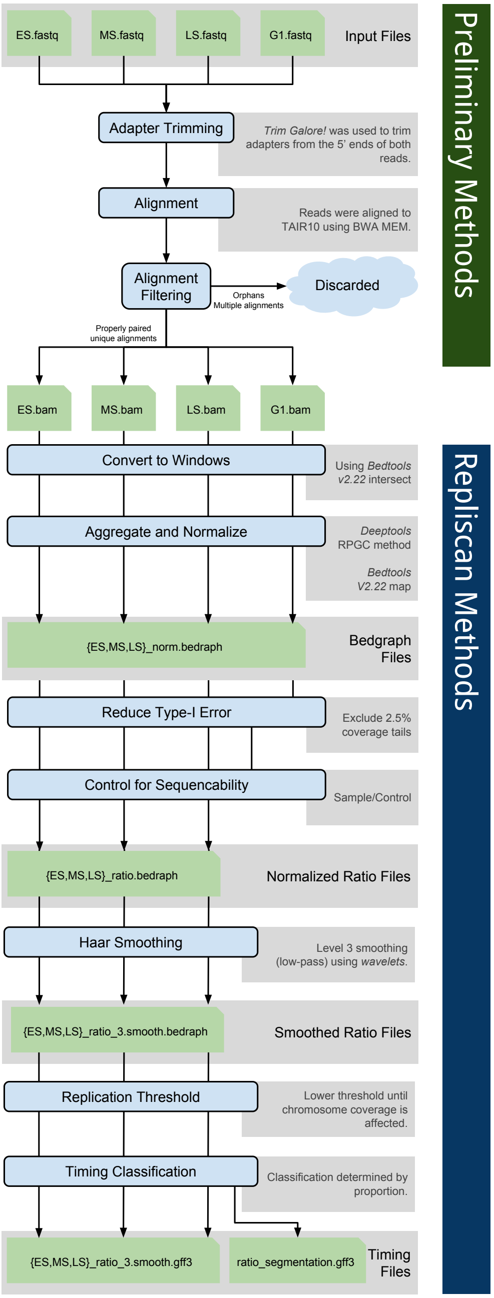

Fig. 8.7: Repliscan workflow. Diagram of the preliminary alignment and quality control methods at the top, and the Repliscan methods at the bottom.

and experiments.

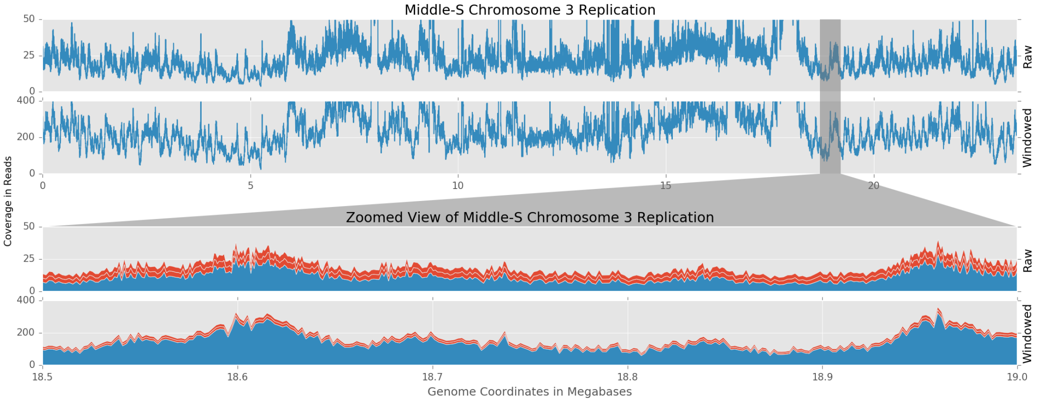

Fig. 8.8: Replication signal and sampling uncertainty. The top two graphs show raw and windowed replication signal across *A. thaliana* chromosome 3. The bottom two graphs show raw and windowed replications signals at 18.5-19.0 megabases from the top view as represented by the gray selection area. The red bars represent sampling uncertainty ($\sqrt{\lambda}$ for Poisson distributions).

We chose 1 kilobase windows because they not only reduce sampling error, but are also two orders of magnitude smaller than the expected *A. thaliana* replicons. Repliscan does not summarize information with sliding windows, so choosing a window size that is an order of magnitude smaller than the expected replicon size is important to approximately align to the actual replication borders. Our analysis will theoretically allow the detection of regions of replication as small as 1 kilobase; however such regions are unlikely to exist in cells subjected to realistic labeling protocols. Therefore, in the final timing classification, Repliscan will merge neighboring regions with similar properties into larger segments. The 1 kilobase resolution then helps to highlight transitions between such segments. In some circumstances, such as working with low coverage data, it may be advantageous to use a larger window size. However, to achieve the best results when adapting Repliscan to other species, we suggest the expected replicon size be factored into calculations that establish window size and sequencing depth.

8.2.3.2 Replicate Aggregation and Normalization

To further decrease sampling effects, and achieve consistent results between experiments, we used multiple biological replicates and adopted aggregation methods to either increase coverage or summarize replication signals using functions provided by “bedtools map” [106]. For experiments with low coverage, we pooled timing $t = 1..T$ replicates $r = 1..R$ together by summing coverage signal k across each window $i = 1..N$.

$$k_{it} = \sum_{r=1}^{R} k_{it_r} \tag{8.1}$$

When coverage was sufficient, we used the signal mean (or the more robust signal median) to clean up aberrant coverage. For these methods, replicates were first normalized for sequencing depth using sequence depth scaling [107]. This normalization step removed differences in sequencing depth between replicates by scaling each sample to an average depth of 1x.

$$k_{it} = \text{median}\left(\frac{N * k_{it_r}}{\sum_{i=1}^{N} k_{it_r}}\right) \tag{8.2}$$

After aggregation, the combined signals were normalized once more to scale any imbalances in replicate numbers back to 1x, prior to making comparisons between replication times.

$$k_{it} = \frac{N * k_{it}}{\sum_{i=1}^{N} k_{it}} \tag{8.3}$$

Our *A. thaliana* test data was relatively high coverage at 30x per bioreplicate, so we used the median function to generate a robust signal, instead of defaulting to sum.

8.2.3.3 Reducing Type I Error Repliscan aims to detect and highlight peaks of replication coverage, but some peaks may be too high and may in fact be false-positives caused by errors in the reference. For instance, if a repetitive element is present three times in the actual genome, but present only once in the reference sequence due to assembly error, all reads would align uniquely to the same location. If two of the actual elements replicate early and the third in middle S phase, the early peak would be twice as large and dominate the classification process. To reduce type I error arising from genomic repeats, we needed to detect and exclude these areas from the final classification because there is no way to resolve such duplication events without improving the reference genome.

The distribution of sequencing coverage is bounded on the left at zero, with very long, positive tails (Figure 8.9). Before we can detect any outliers we first need to transform the data to actually fit a probability distribution.

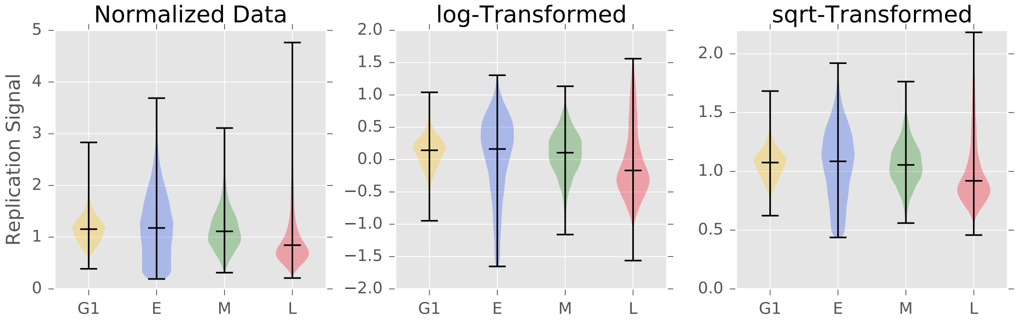

Fig. 8.9: Normalized and transformed replication signals. Violin plots showing how the normalized and aggregated *A. thaliana* chromosome 3 replication signals from G1, early (E), middle (M), and late (L) S-phase data was bounded from $[0, \infty)$. We separately experimented with with log transforms to make the distributions more normal-like, and square root transforms to stabilize the spread.

In Figure 8.9, we show that both the log and square root transformations stabilized the spread and skew. The log transformation extends the $(0,1)$ tail and shortens the $[1,\infty)$ tail, making the distribution more normal-like. The square root transform also shortens $[1,\infty)$ tail and spreads the $[0,1)$ tail, but not to the same extent, leaving the distribution skewed towards 0. While different, both transformations improve the fit of different probability distributions.

Normally, sequencing depth is modeled with a Poisson distribution because the integer counts are discrete[108], positive, and asymmetric. However, our aggregated and normalized data is continuous, positive, and asymmetric. To accurately model these sequencing values we use the Gamma distribution for highly-skewed data and the normal-like methods for symmetric data[109]. In all, we provide four combinations of methods to transform the data and detect outliers:

fitting a gamma distribution to the log transformed data,

$$\log(K_t) \sim \Gamma(\alpha_t, \beta_t) \equiv \text{Gamma}(\alpha_t, \beta_t) \tag{8.4}$$

fitting a gamma distribution to the square root transformed data,

$$\sqrt{K_t} \sim \Gamma(\alpha_t, \beta_t) \equiv \text{Gamma}(\alpha_t, \beta_t) \tag{8.5}$$

fitting a normal distribution to the log transformed data,

$$\log(K_t) \sim \mathcal{N}(\mu, \sigma^2) \equiv \text{Normal}(\mu, \sigma^2) \tag{8.6}$$

or calculating the whisker bounds (WB) of a boxplot from the log transformed data

$$X_t = log(K_t) \tag{8.7}$$

$$\mathrm{IQR}(X_t) = P_{75}(X_t) - P_{25}(X_t) \tag{8.8}$$

$$\mathrm{WB}(X_t) = \left[P_{25}(X_t) - 1.5 * \mathrm{IQR}(X_t), \quad P_{75}(X_t) + 1.5 * \mathrm{IQR}(X_t)\right], \tag{8.9}$$

$$\text{where } P \text{ is the percentile function.} \tag{8.10}$$

We use scipy[110] version 0.15.0 to fit all probability distributions to the actual coverage windows. Windows with coverage in the upper and lower 2.5% tails of the calculated probability distributions, or outliers when using whiskers, are considered unrepresentative and removed (Figure 8.10).

$$\log(k_{it}) = \begin{cases} 0 & P_{97.5}(\alpha_t, \beta_t) < X < P_{2.5}(\alpha_t, \beta_t) \\ k_{it} & \text{Otherwise} \end{cases} \tag{8.11}$$

For simple cases, or when the transformed data does not resemble a probability distribution, we also provide the option of a rank-based (percentile) cutoff. By default, this will remove the upper and lower 2.5% coverage values, but this value can also be customized by the user.

The outliers in the positive coverage tails that this method removes may comprise a significant amount of coverage, so we perform another round of normalization to return the sample to 1x coverage. Each of the five methods has its own strengths and computation complexity. Most coverage data can be accurately modeled with the normal distribution. For cases when the transformed coverage distributions are still skewed, we suggest using the gamma distributions. If for some reason, the coverage data is multimodal, the whisker or percentile cutoff methods will both remove outliers from the data. We recommend the whisker method over a percentile cutoff because the whiskers remove data from a derived distribution, while the percentile indiscriminately removes a percentage of the data.

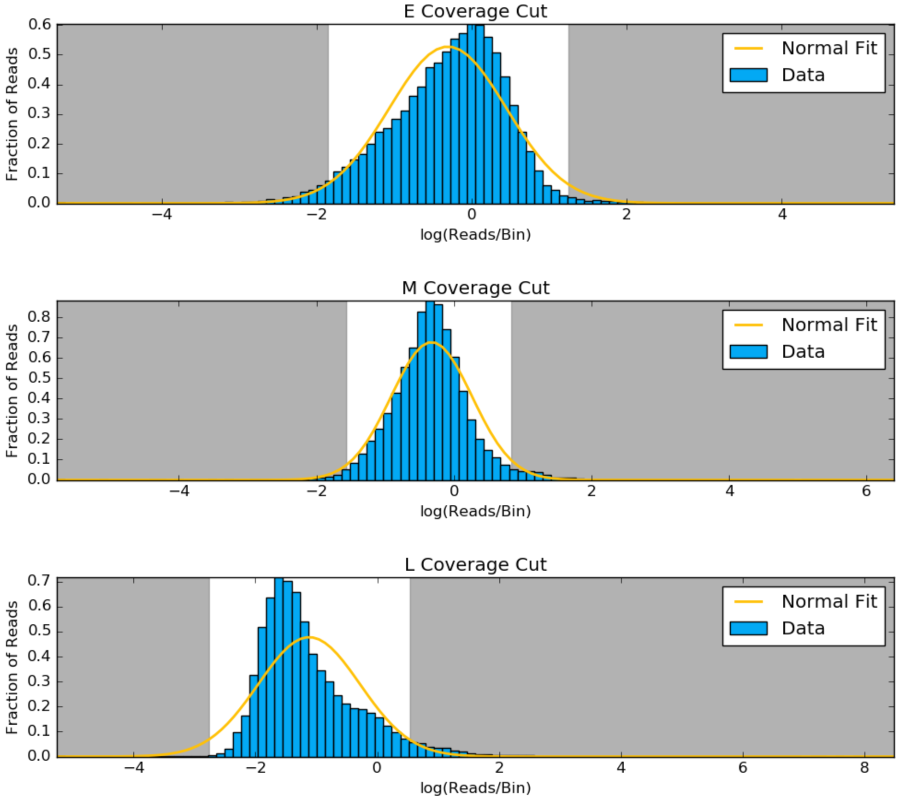

Fig. 8.10: Outlying coverage in chromosome 3. Based on the normal distribution fit (yellow) to the log transformed coverage distribution of early (E), middle (M), and late (L) S-phase data, windows that fall in the tails shaded in gray are removed from the analysis.

8.2.3.4 Normalize for Sequenceability Amplification, fragmentation, and shotgun sequencing DNA is a non-uniform random process. Coupled with imperfect alignment efficiency from repetitive regions and incomplete reference genomes, artificial peaks arising from differences in the efficiency with which specific genomic regions can be sequenced are easy to confuse with actual signal peaks. This does not have a significant impact on comparisons between samples, but makes it difficult to compare adjacent genomic regions. Our sequencing protocol included a sample of non-replicating G1 DNA to correct for this phenomenon.

In G1, the cell is growing in physical size but no DNA replication is taking place, so the copy number of each sequence in the genome is at the 2C level. Variations in sequenceability can thus be separated from variations in signal attributable to DNA replication. Dividing each of the S-phase samples by the G1 sample normalizes each of the windows by giving the ratio of treatment coverage over expected coverage.

$$r_t = \frac{k_t}{k_1}, \text{ where } k_1 \text{ is the control.} \tag{8.12}$$

To better illustrate this process, consider two replication coverage windows next to each other: the first one is accessible and easy to sequence, and therefore produces more fragments per unit input DNA than the second window, which is hard to sequence. The normalization step would lower the signal from the first window, dividing it by a big coverage number from G1. It would also raise the signal from the second window, which would be divided by a smaller G1 number, making the two windows more comparable and reducing background noise. We recommend that such a control be implemented in all DNA sequencing based experiments to detect replication timing, on the basis that a non-replicating G1 control is the best, and most uniform representation of the genome. However, in the event that a non-replicating G1 is not sequenced, all S-phase samples can be combined to synthesize a total-S control, or a total DNA control can be used.

8.2.3.5 Haar Wavelet Smoothing Data sampling is always affected by noise. Statistical noise can be accounted for and modeled with more sampling, more robust statistical methods, or by summarizing larger ranges of data. Adding replicates for additional statistical power is cost-prohibitive, especially for larger genomes. Instead, we adopted the Haar wavelet transform to summarize replication data as an orthonormal series generated by the Haar wavelet. Using Wavelets[111] version 1.0, we performed a maximum overlap discrete wavelet transform with the Haar wavelet using reflected boundaries and level 3 smoothing on a per-chromosome basis for each sample. Wavelet decomposition is designed to represent a signal as a collection of frequencies. Level 3 decomposition represents a signal as the upper 87.5% of frequencies. Smoothing works as a low-pass filter, where small and frequent changes are removed, while large and wide changes are preserved.

We specifically chose the Haar wavelet over other smoothing methods because it is a square function with discrete boundaries and thus resembles the signals we aim to detect. General smoothing methods like LOESS and moving average methods produce stabilized trends from data, but they work by summarizing subsets of the whole picture. These methods also leave behind artifacts. A moving average will change a square peak into a sawtooth pattern the size of the smoothing window and will be affected by a single point of noise. LOESS is designed to model trends in sliding subsets of the data, but each of the least-squares regression steps are vulnerable to noise as with the moving average. LOESS will also spread out peaks in our data because of our uniform window size (1 kilobase), and is designed to accurately model clusters of points. As demonstrated in Figure 8.11A with simulated data, the Haar wavelet accurately removes low-amplitude and high-frequency noise to reconstruct the original signal without artificially expanding the peaks of replication signal. Applying the moving average, LOESS, and Haar wavelet to actual *A. thaliana* data in Figure 8.11B shows that both the moving average and LOESS can capture large trends, but the Haar wavelet excels at highlighting subtle peaks in the data without under smoothing and requiring the user to choose the range they summarize on. Any proportion or range of the data is very different when choosing different window sizes. Haar only removes low-amplitude frequency trends from the wavelet transform.

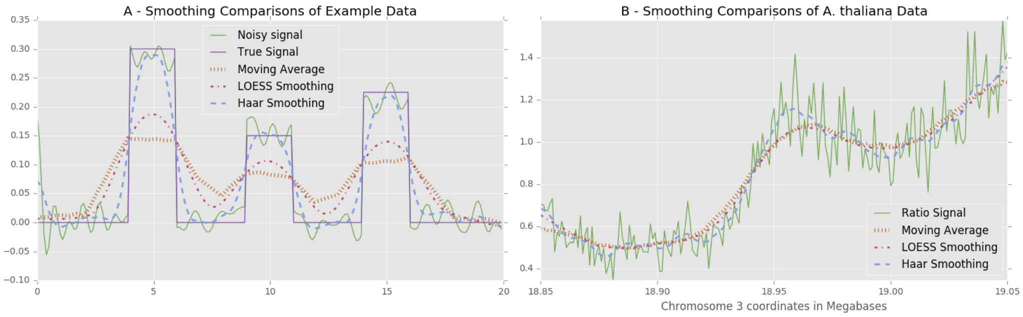

Fig. 8.11: Smoothing comparisons. **A** - Noise (green) is added to an original signal (purple), and then smoothed with a 4 unit (40 point) moving average (orange), a 5 unit (25% subset) LOESS (red), and a level 3 Haar wavelet (blue). Both the moving average and LOESS spread out the peaks and artificially lowered signal amplitudes, while the Haar wavelet keeps bounds and peak heights close to the original. **B** - The *A. thaliana* middle S-phase normalized signal (green), is smoothed with a moving average (orange), LOESS (red), and the level 3 Haar wavelet (blue) for comparison.

We experimented with several levels of decomposition with our data, and found that the low-frequency trends preserved with level 3 aligned to genes, transposable elements, and histone marks on each genome the best. If the window size is kept at the default of 1 kilobase, this decomposition level can be kept the same because the same frequencies are represented. If the window size is changed to accommodate different sequencing depths, we suggest that users experiment with different decomposition levels, because this essentially changes the sampling rate of the analysis.

8.2.3.6 Defining Replication The analysis to this point yields a smoothed ratio of normalized replication ratio signals $r_{c_w t}$ in windows ($w = 1..Y$) per chromosome ($c = 1..X$), with a range of $[0, \infty)$ that can be compared to each other, and leads to the question of which signals can be considered confidently as resulting from DNA replication. Lee *et al.*[99] originally considered array-based replication signals greater than the control as actively replicating in their investigation of *A. thaliana* as follows.

$$\text{replicating}_{ct}(w) = \begin{cases} 1, & \text{if } r_{c_w t} > 1 \\ 0, & \text{otherwise} \end{cases}, \text{ where } c_w = i \tag{8.13}$$

The Repliscan software allows users to adopt this threshold method, but we also include more robust methods to define replication. The simple threshold approach above is appropriate when considering replication as a ratio, but because all signals from the early, middle, and late S-phase samples represent labeled - and therefore, replicating - DNA, even signals that are less than the control must be considered as reflecting some level of replication activity. In other words, even though there may be noise in the data, all replication signals should be genuine because EdU is only incorporated into newly replicated DNA. Instead of simply choosing a smaller ratio threshold, we implemented a percentile cutoff based on the distribution of the ratios. By default, this method removes the lowest 2% of the values for a chromosome in a given sample.

$$\text{replicating}_{ct}(w) = \begin{cases} 1, & \text{if } r_{c_w t} > P_{0.02}(r_{ct}) \\ 0, & \text{otherwise} \end{cases} \tag{8.14}$$

While this method is a data-dependent means for establishing a cutoff, it was not considered ideal for an automatic analysis for two reasons. First, a cutoff is still being dictated, even if it is more robustly supported than in previous analyses. Second, this cutoff will always remove a flat percentage of the values, even if there is no evidence they are not high-quality data points. To improve on these deficiencies, we implemented a threshold for replication that depends on the information provided in addition to the data.

To maximize the fraction of a chromosome with valid replication signal (or information), we designed an optimization method that incorporates as much of each chromosome as possible by analyzing the rate that chromosome coverage changes with replication signal. Using data from all time points, coverage is defined as the fraction of windows with a signal greater than the threshold in at least one replication time.

$$\text{coverage}(T_c) = \frac{r_{c_w t} > T_c}{Y}, \text{ where } T_c \text{ is the threshold for chromosome } c \tag{8.15}$$

Our optimization process begins from the point of the largest absolute change in coverage (mT_c), and lowers the replication threshold (T_c) until the absolute chromosome fraction per sample/control coverage differential goes below 0.1, effectively leveling out. In rare cases where this process does not converge, the threshold is set to be the median of all chromosomes that do converge.

$$mT_c = \arg\max_{T_c}(|\text{coverage`}(T_c)|) \tag{8.16}$$

$$\hat{T}_c = \arg\max_{T_c<mT_c}(|\text{coverage`}(T_c)| < 0.1) \tag{8.17}$$

$$R_{c_w t} = \begin{cases} r_{c_w t}, & \text{if } r_{c_w t} > \hat{T}_c \\ 0, & \text{otherwise} \end{cases} \tag{8.18}$$

Such a search pattern circumvents any local optima in the coverage signal that may have stalled a gradient descent. That being said, we implemented the threshold to run on a per-chromosome basis to minimize the effect of any structural differences (Figure 8.12).

The end result is a method that includes as much of the genome and coverage information as possible, and prevents the use of small signals when they comprise a small portion of the chromosomes. Our method is generically applicable to experiments using the same Repli-seq protocol because the threshold is calculated from the data. A critical benefit is that users are not required to be masters of their data or this tool, and can instead focus on interpretation.

8.2.3.7 Classification/Segmentation Given a signal that can confidently be considered as arising from DNA replication, we are able to classify segments of the genome according to when in the cell cycle they are replicated. Suppose that in one of the windows in Chromosome 3, we have the following levels of replication in Table 8.2.

We already know from Figure 8.12 that any values below 0.92 in Chromosome 3 are not considered replicating, so the middle S-phase value would become 0 and we

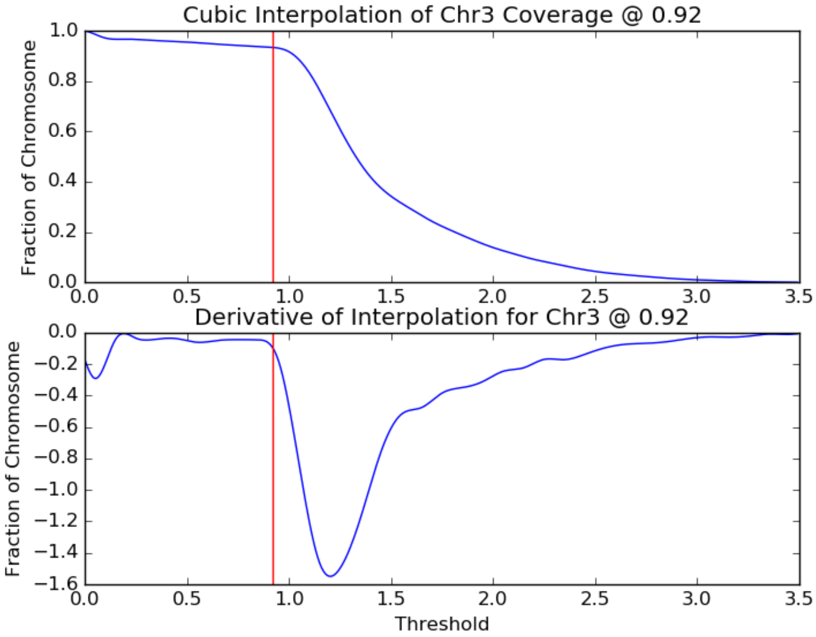

Fig. 8.12: Replication threshold from coverage. The upper plot shows how much of *A. thaliana* chromosome 3 will be kept for downstream analysis as a function of the signal threshold. The lower plot shows the chromosome coverage differential as a function of the threshold. The vertical red line in each plot marks the optimal threshold of 0.92.

Tab. 8.2: Example coverage values to demonstrate replication timing classification.

Time	Early	Middle	Late
Coverage	0.93	0.8	3.0
Replicating	0.93	0	3.0

would say this window replicates in both early and late S-phase. However, the late replication level is 3 times higher than that of early, which is just past the threshold for replication at 0.93. Instead of making another replication threshold, we implemented a general solution to compare values against each other using a proportion.

First, on a window-by-window basis, we take the infinity norm of all values, which means we divide all values by the maximum for that window position.

$$S_{ct}(w) = \frac{R_{c_w t}}{\|R_{c_w}\|_\infty} \tag{8.19}$$

This operation scales the largest value to 1 and the others to the range [0,1]. A time signal is then classified as predominantly replicating $C_c t(w)$ if the normalized value is greater than 0.5, or at least half the size of the largest signal for that window.

$$C_{ct}(w) = \begin{cases} 1, & \text{if } S_{ct}(w) > 0.5, \\ 0, & \text{otherwise} \end{cases} \tag{8.20}$$

The infinity-norm ensures that the largest value will always be classified as replicating, and this classification method allows for a window to be called strongly replicating at more than one time in S-phase (e.g. both early and late) when other signals are within 50% of the maximum value. Besides 0.5 being easy to test for, this creates an equally partitioned solution space in the form of an n-dimensional hypercube. In the case of our *A. thaliana* data, the space is a 3-dimensional cube with each dimension being one of the time points: early, middle, and late S-phase. The 0.5 partition then creates 8 equal-sized sub-cubes corresponding to each possible combination of times:

{Non-replicating, Early, Middle, and Late}

along with

{Early-Middle, Middle-Late, Early-Late, and Early-Middle-Late}

S-phase replication combinations.

8.2.4 Results and Discussion

8.2.4.1 Data To demonstrate the ability of our methods to adapt to different datasets, we ran our pipeline on the *A. thaliana* Col-0 cell culture data (PRJNA330547) that was used to develop these methods, and a separate similarly prepared *Z. mays* B73 replication timing dataset (PRJNA327875) also from our lab.

A.thaliana

The *A. thaliana* experiment was comprised of 3 early S bioreplicates, 3 middle S bioreplicates, 3 late S bioreplicates, and 1 G1 sample. Each bioreplicate was paired-end sequenced to 36x coverage. The unique and properly-paired alignment rate for each sample was approximately 85%, yielding a total of 30x viable replication data from each sample. Due to the high coverage, we decided to use 1 kilobase windows and merge bioreplicates with the median function for our analysis.

Z. mays

In the *Z. mays* experiment, there were 3 early S bioreplicates, 3 middle S bioreplicates, 2 late S bioreplicates, and 2 G1 technical replicates. Each bioreplicate was paired-end sequenced to about 5x coverage. While there were more reads than the *A. thaliana* experiment, the *Z. mays* genome is much larger, so coverage was lower. Using the B73 AGPv3 genome assembly, the unique and properly-paired alignment rate for each sample was approximately 99%, yielding a total of 5x viable replication data from each bioreplicate. Even though a larger analysis window could have been used, we decided to use the same 1 kilobase windows for this dataset, and deemed the summation of bioreplicates was necessary to achieve enough coverage to highlight peaks in the data.

8.2.4.2 Segmentation Overview Using 1 kilobase windows, median aggregation for *A. thaliana*, and sum aggregation for *Z. mays*, we used our default pipeline to classify the replication timing of our data. We generated Figure 8.13 to show the replication segmentation classification of Chromosome 3 in *A. thaliana* and Chromosome 10 in *Z. mays*.

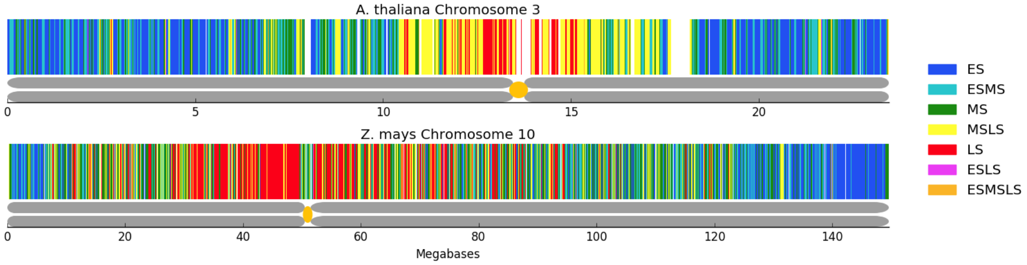

Fig. 8.13: Comparison of *A. thaliana* and *Z. mays* segmentation. Following the segmentation legend on the right, *A. thaliana* chromosome 3 (top) and *Z. mays* chromosome 10 (bottom) have been classified into segmentation regions by Repliscan. The large white regions in the *A. thaliana* figure are unclassified regions due to high or very low signal. Below each replication segmentation is a depiction of the chromosome, with the centromere location marked in yellow [112], [113].

In both instances, early replication is concentrated toward the ends of the chromosome arms, with middle and late replication becoming more prominent closer to the centromere and the highest concentration of late replicating sequences in the heterochromatin surrounding the centromere. These timing maps demonstrate that the method developed using the *A. thaliana* data was successfully applied to the lower coverage *Z. mays* data by simply choosing to aggregate replicates using the sum instead of the median.

8.2.4.3 Segment Composition and Size Instead of viewing the chromosomes as a whole, we can also get an idea of predominant replication times by looking at the proportional composition. Figure 8.14 shows that Early, Early-Middle, and Middle-Late S-phase replication makes up most of the segmentation profiles for *A. thaliana* Chromosome 3. About 6% of the chromosome is missing around the centromere and heterochromatic knob, which probably would have been classified in the Middle to Late times based on what we do see. In *Z. mays*, we see a more uniform distribution of Chromosome 10, which is 5-fold larger, across the replication segmentation classes. Lee *et al.*[99] previously hypothesised a two-stage replication program, but our results, which were generated using much shorter labeling times to capture much smaller increments of replication, show a more even spread (Figure 8.14).

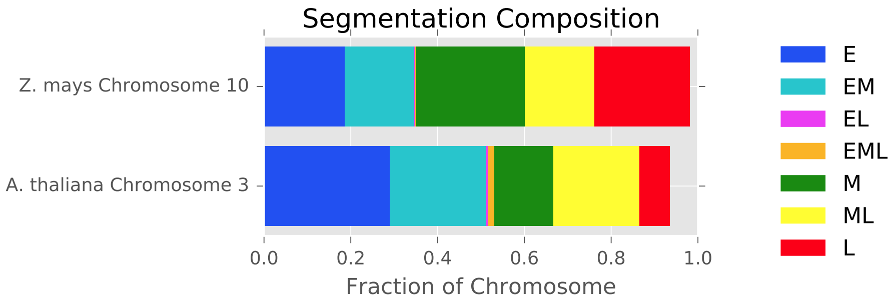

Fig. 8.14: Composition of replication segmentation. The segment composition shows that replication in *A. thaliana* is skewed towards early S replication, while *Z. mays* has an even distribution across early, middle, and late S. We can also see that the non-sequential early-late (EL) and early-middle-late (EML) classifications comprise a very small proportion of the classified segments in both cases.

The Early-Late and Early-Middle-Late comprise a small portion of the chromosomes in both organisms and could arise naturally in the data through allelic and cell population differences. Figure 8.15 shows a different summary of the segmentation breakdown, highlighting the segment size distribution with boxplots. Once again, Early-Late and Early-Middle-Late segments are distinct in that their lengths are small relative to the other timing categories.

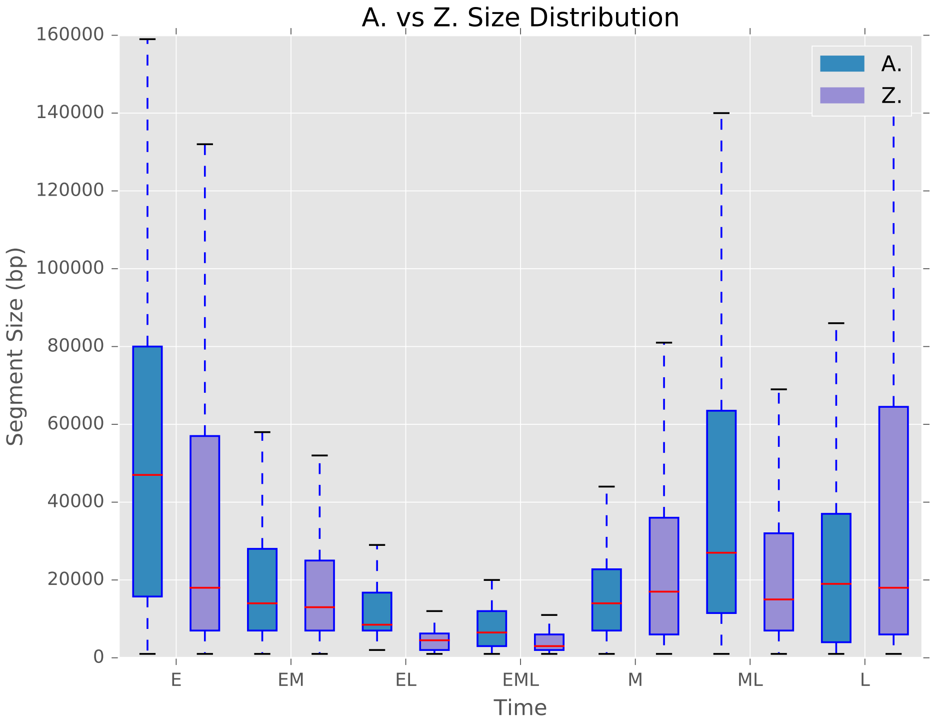

Fig. 8.15: Segment size distribution. Boxplots for every combination of replication time, illustrating the distribution of segment sizes. Early (E) and mid-late (ML) S were largest in *A. thaliana*, while early and late (L) were largest in *Z. mays*.

8.2.4.4 Downsampling and Stability of Results

The relatively small genome size of *A. thaliana* allowed us to obtain extremely deep sequencing coverage, which is currently cost-prohibitive for larger genomes. To estimate a minimum coverage requirement for our methods, we simulated experiments with lower coverage via downsampling. We first generated 3 technical replicates by randomly sorting the original alignment files. We removed reads from each of the replicates in 1% increments without replacement. Each of the 300 (100 x 3) simulated experiments were analyzed using both median and sum aggregation, and no (none), log gamma, square root gamma, normal, and whisker outlier removal. To account for differences arising from the sorting order, the final classification for each window was determined by majority across the 3 replicates. Classification ties were broken by treating the early, middle, and late time classification combination as a 2-bit binary number, and taking the median.

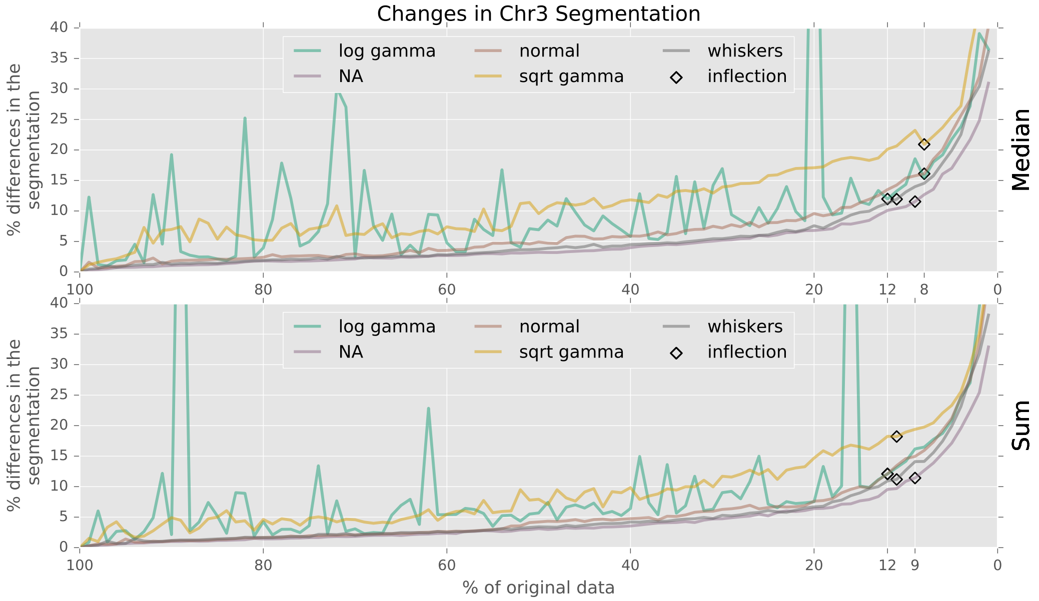

Fig. 8.16: Segmentation differences in downsampled data. After downsampling the *A. thaliana* data, the accuracy of median (top) and sum (bottom) aggregation, and outlier detection using log gamma, none (NA), normal, square root gamma, and whiskers. Inflection points in the differences are labeled with black diamonds.

After confirming that the segmentation profiles from all three 100% replicates were identical to our original segmentation, differences for each run type were calculated as percent Hamming distances from the 100% version. All differences were compounded and plotted as a fraction of the whole chromosome in Figure 8.16. The most obvious results are the spikes of differences in both the median and sum log transformed gamma runs when the iterative fitting function failed to converge (Figure 8.17).

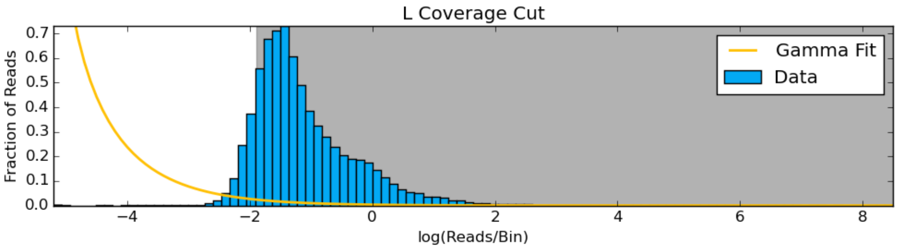

Fig. 8.17: Unconverged log gamma fit. Most of the data is removed when the iterative fitting function fails to converge with the log transformed gamma distribution. Instances like this produce the spikes of differences in Figure 8.16.

Shifting attention to the square root gamma experiments in Figure 8.16, we see that the fit function never fails to converge, but there is increased variability of results among each level of downsampling. All other probability functions are very stable between downsampling runs. We even see that summing the coverage to 90x provides no improvement over the median - even at low coverage levels. The inflection points show that the most stable method was aggregating replicates with the median operation and removing coverage by fitting a normal distribution to the log transformed data. Results from this method began to noticeably diverge when downsampled to 8%, or 2.4x coverage. This indicates 5x coverage for the commonly studied species *Z. mays* (2.3 gigabase genome[114]) is sufficient to calculate a replication profile, which is quite tractable for a laboratory of modest financial means.

8.2.4.5 General Application of Repliscan To demonstrate that Repliscan is generally applicable, we used it to analyze two published Repli-seq datasets: Human fibroblast data from Hansen *et al.* 2010[96] (GSM923444) and *D. melanogaster* data from Lubelsky *et al.* 2014[115] (PRJNA63463).

The Human fibroblast Repli-seq data contains samples from 6 fractions of S phase (G1b, S1, S2, S3, S4, and G2) with two replicates each providing an average depth of 0.02x coverage. Using the supplementary methods of Hansen *et al.*, we were able to reproduce their original tag density results. Reads from both replicates were first combined and then aligned to the human reference genome (hg19). After alignment, signals with more than 4 reads per 150 basepair window were removed. Lastly, a percent total coverage in 50 kilobase wide windows was calculated every 1 kilobase (Figure 8.18).

To analyze this data with Repliscan starting from the aligned reads, we first needed a sequencing control. Both G1b and G2 contain replicating DNA in this experiment, so we combined G1b, S1-4, and G2 to create a total-S (TS) control in the first line of the Repliscan input configuration. After crafting the configuration file, we ran Repliscan with a window size of 50 kilobases and aggregation through sum to match the methods of Hansen *et al.* Figure 8.18 compares the output of Repliscan against the reproduced results in a region from their original work. Given that there were 6 fractions of S-phase in the Repliscan input, there were ($2^6 - 1$) 63 possible classifications, but only 22 were present in the output segmentation. Repliscan presented temporally sensible results with replication initiating in G1b and spreading to G2 all while relying on the automatic tuning of Repliscan after matching the window size

(Figure 8.18). We compared our results from Repliscan to the "BJ-G1_segment" regions published by Hansen *et al.* in their Supplementary Table S4 using the accuracy statistical measure.

$$\text{accuracy} = (TP + TN)/(TP + FN + TN + FP) \tag{8.21}$$

Where TP is the number of G1bS1 Repliscan classifications that match "BJ-G1_segment", FN is the number of non-G1bS1 classifications that match "BJ-G1_segment", TN is the number of non-G1bS1 classifications that also do not match "BJ-G1_segment", and FP is the number of G1bS1 classifications that do not match "BJ-G1_segment." We found that our Repliscan reanalysis had an accuracy of 83% with the published "BJ-G1_segment" results.

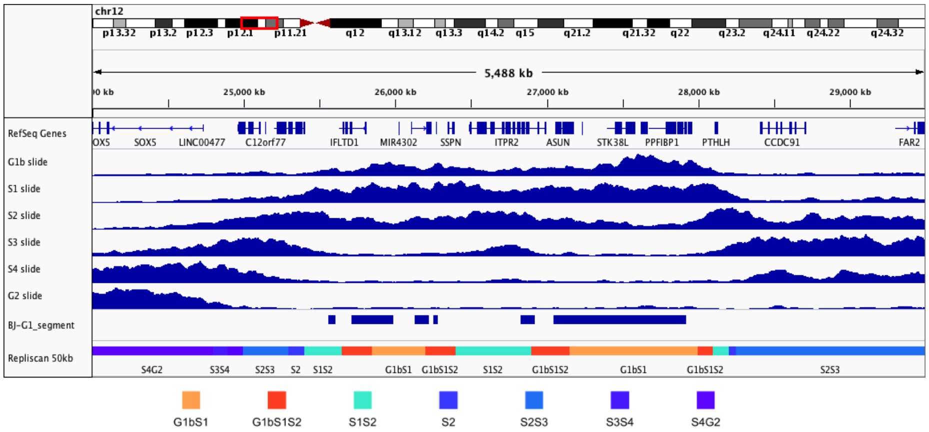

Fig. 8.18: Human fibroblast Repli-seq. 50 kilobase sliding window replication signals (blue) reproduced from Hansen *et al.*, published "BJ-G1_segment" regions, and 50 kilobase Repliscan results (bottom).

We also reproduced the original continuous replication profiles of Lubelsky *et al.* by processing the raw data as was done in the original paper. Replicates were combined from each fraction of S phase (Early, Early-Mid, Late-Mid, and Late) and aligned to the dm3 Release 5.12 genome. Unique alignments were kept and the RPKM was calculated in 10 kilobase windows along the genome. The RPKMs from the 4 samples were then weighted and combined to create a single replication signal from 0 to 1. The replication signal was then LOESS smoothed with a span of 200 kilobases (20 bins). This continuous signal was then classified as early replication when the value was less than or equal to 0.5, and late replication when above 0.5 (Figure 8.19).

Similar to the work by Hansen *et al.*, this experiment did not contain an non-replicating G1 control, so we combined all fractions into a total-S (TS) control. For inputting the raw data into Repliscan, we crafted two input configurations: one with Early (early, early-mid) and Late (mid-late, late) (2S) to match the discrete results of Lubelsky *et al.*, and another with Early, Early-Mid, Mid-Late, and Late classifications (4S) to highlight the classification capabilities of Repliscan. Coverage averaged around 4.4x, so we ran Repliscan with both (2S and 4S) input configurations, sum replicate aggregation, and 10 kilobase windows to match the original analysis (Figure 8.19).

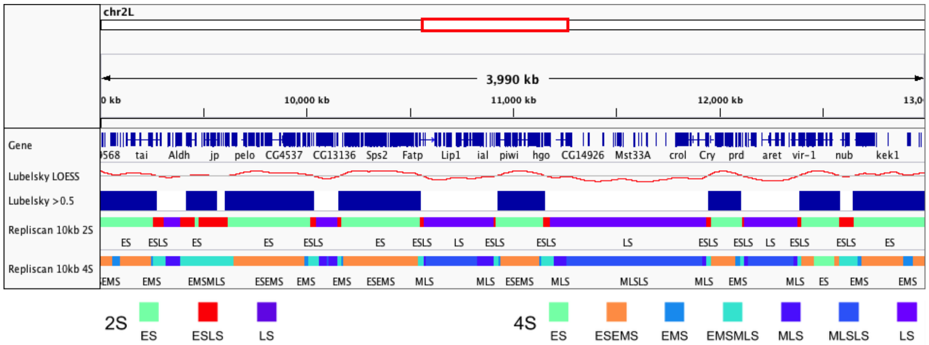

Fig. 8.19: *D. melanogaster* KC167 Repli-Seq. Reproduction of the LOESS smoothed continuous replication profile (Lubelsky LOESS), and the thresholded, discrete early (blue) and late timing domains (Lubelsky > 0.5) from original Lubelsky *et al.* study. Repliscan segmentation results with Early (Early, Early-Mid) and Late (Mid-Late, Late) replication (2S), and Early, Early-Mid, Mid-Late, and Late replication (4S) configuration with 10 kilobase windows.

The Repliscan configuration with two S-phase fractions (2S) highly resembled the thresholded continuous signal (Lubelsky > 0.5) with a statistical accuracy measure of 95%. When Repliscan was run to capture all 4 S-phase combinations, more information was revealed about the replication timeline. Looking at the two left-most late regions of "Lubelsky > 0.5" in Figure 8.19, we can see that the continuous signal rides along the 0.5 threshold, and Repliscan predicted a long region of EMS-EMLS with all four fractions of S taken into context, instead of detecting an initiation site in the center. This situation is a good example of the type of coarse grained calls that we are trying to avoid with Repliscan by allowing combinations of replication in our classifications. Our 4S results were also found to be highly similar with the discrete data, with a statistical accuracy of 78%.

8.2.5 Conclusions

Based on our results from running Repliscan on both *A. thaliana* and *Z. mays* data, we have demonstrated that our methods offer a robust means of analyzing data from replication timing experiments that use label incorporation. Although we argue that a non-replicating G1 control should be preferred for biological reasons, our analytical method can be used equally well with control datasets derived from synthetic total S phase pools or from total DNA. We have significantly improved on previous methods by incorporating non-destructive Haar smoothing, using optimization methods to define replication, and classification through signal proportion. When run using the same parameters but using data from different organisms, the methods automatically tuned their thresholds to adjust for differences in coverage. Downsampling our data showed our methods provided stable results at as little as 2.4x coverage and 1 kilobase analysis windows. Even lower coverages can be accommodated at lower resolution by using larger window sizes for the analysis. We also demonstrated that Repliscan can be used to classify replication regions in external Repli-seq data by applying it to both low-coverage Human and high-coverage *D. Melanogaster* experiments with 4 to 6 S-phase fractions and synthetic total-S controls. There is no current consensus pipeline for validation, so we compared the published results from the external datasets to those from Repliscan. We found that the Repliscan results were on average 85% identical to the original findings of these papers.

In-depth explorations of the replication programs in *A. thaliana* and maize will be published separately. We think these methods provide a path for greater understanding of the DNA replication program in plants, humans, and other higher organisms.

8.2.6 Availability

8.2.6.1 Software Project home page: `https://github.com/zyndagj/repliscan`

8.2.6.2 Data The datasets supporting the conclusions of this article are available in the NCBI Sequence Read Archive (SRA) BioProjects PRJNA330547, PRJNA327875, and PRJNA63463 and GEO dataset GSM923444. All reproduced Human and *D. Melanogaster* Repli-seq results can be generated and viewed as described in the Repliscan repository.

BIBLIOGRAPHY

[1] M. R. Amin, A. Yurovsky, Y. Tian, and S. Skiena, "Deepannotator: Genome annotation with deep learning," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2018, pp. 254–259.

[2] P. Ni, N. Huang, Z. Zhang, D.-P. Wang, F. Liang, Y. Miao, C.-L. Xiao, F. Luo, and J. Wang, "Deepsignal: Detecting dna methylation state from nanopore sequencing reads using deep-learning," *Bioinformatics*, vol. 35, no. 22, pp. 4586–4595, 2019.

[3] J. T. Simpson, R. E. Workman, P. Zuzarte, M. David, L. Dursi, and W. Timp, "Detecting dna cytosine methylation using nanopore sequencing," *Nature methods*, vol. 14, no. 4, p. 407, 2017.

[4] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[5] H. J. Jansen, M. Liem, S. A. Jong-Raadsen, S. Dufour, F.-A. Weltzien, W. Swinkels, A. Koelewijn, A. P. Palstra, B. Pelster, H. P. Spaink, *et al.*, "Rapid de novo assembly of the european eel genome from nanopore sequencing reads," *Scientific reports*, vol. 7, no. 1, pp. 1–13, 2017.

[6] C. Bleidorn, “Third generation sequencing: Technology and its potential impact on evolutionary biodiversity research,” *Systematics and biodiversity*, vol. 14, no. 1, pp. 1–8, 2016.

[7] NCBI, *Eukaryotic, prokaryotic, and virus genome reports*, Mar. 2019. [Online]. Available: `ftp://ftp.ncbi.nlm.nih.gov/genomes/GENOME_REPORTS/`.

[8] P. Chain, D. Grafham, R. Fulton, M. Fitzgerald, J Hostetler, D Muzny, J Ali, B Birren, D. Bruce, C Buhay, *et al.*, “Genome project standards in a new era of sequencing,” *Science*, vol. 326, no. 5950, pp. 236–237, 2009.

[9] M. Yandell and D. Ence, “A beginner’s guide to eukaryotic genome annotation,” *Nature Reviews Genetics*, vol. 13, no. 5, pp. 329–342, 2012.

[10] B. L. Cantarel, I. Korf, S. M. Robb, G. Parra, E. Ross, B. Moore, C. Holt, A. S. Alvarado, and M. Yandell, “Maker: An easy-to-use annotation pipeline designed for emerging model organism genomes,” *Genome research*, vol. 18, no. 1, pp. 188–196, 2008.

[11] F. Thibaud-Nissen, A. Souvorov, T. Murphy, M DiCuccio, and P Kitts, “Eukaryotic genome annotation pipeline,” *The NCBI Handbook*, vol. 2, 2013.

[12] B. L. Aken, S. Ayling, D. Barrell, L. Clarke, V. Curwen, S. Fairley, J. Fernandez Banet, K. Billis, C. García Girón, T. Hourlier, *et al.*, “The ensembl gene annotation system,” *Database*, vol. 2016, 2016.

[13] B. McClintock, “The origin and behavior of mutable loci in maize,” *Proceedings of the National Academy of Sciences*, vol. 36, no. 6, pp. 344–355, 1950.

[14] J. Wang, G. K.-S. Wong, P. Ni, Y. Han, X. Huang, J. Zhang, C. Ye, Y. Zhang, J. Hu, K. Zhang, *et al.*, “Reps: A sequence assembler that masks exact repeats identified from the shotgun data,” *Genome Research*, vol. 12, no. 5, pp. 824–831, 2002.

[15] A. Smit, R Hubley, and P Green, *Repeatmasker open-4.0. 2013–2015*, 2015.

[16] A. L. Price, N. C. Jones, and P. A. Pevzner, “De novo identification of repeat families in large genomes,” *Bioinformatics*, vol. 21, no. suppl_1, pp. i351–i358, 2005.

[17] J. A. Ågren and S. I. Wright, “Co-evolution between transposable elements and their hosts: A major factor in genome size evolution?” *Chromosome research*, vol. 19, no. 6, p. 777, 2011.

[18] R. C. Kennedy, M. F. Unger, S. Christley, F. H. Collins, and G. R. Madey, “An automated homology-based approach for identifying transposable elements,” *BMC bioinformatics*, vol. 12, no. 1, p. 130, 2011.

[19] B. Alberts, A. Johnson, J. Lewis, M. Raff, P. Walter, and K. Roberts, *Molecular Biology of the Cell*, 4th ed., ser. Molecular Biology of the Cell. Garland Science, 2002, ISBN: 9780815332183. [Online]. Available: `https://www.ncbi.nlm.nih.gov/books/NBK21054/`.

[20] E. M. McCarthy and J. F. McDonald, “Ltr_struc: A novel search and identification program for ltr retrotransposons,” *Bioinformatics*, vol. 19, no. 3, pp. 362–367, 2003.

[21] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, *et al.*, “Full-length

transcriptome assembly from rna-seq data without a reference genome," *Nature biotechnology*, vol. 29, no. 7, p. 644, 2011.

[22] C. Trapnell, A. Roberts, L. Goff, G. Pertea, D. Kim, D. R. Kelley, H. Pimentel, S. L. Salzberg, J. L. Rinn, and L. Pachter, "Differential gene and transcript expression analysis of rna-seq experiments with tophat and cufflinks," *Nature protocols*, vol. 7, no. 3, pp. 562–578, 2012.

[23] A. Hotz-Wagenblatt, T. Hankeln, P. Ernst, K.-H. Glatting, E. R. Schmidt, and S. Suhai, "Estannotator: A tool for high throughput est annotation," *Nucleic acids research*, vol. 31, no. 13, pp. 3716–3719, 2003.

[24] W. H. Majoros, M. Pertea, and S. L. Salzberg, "Tigrscan and glimmerhmm: Two open source ab initio eukaryotic gene-finders," *Bioinformatics*, vol. 20, no. 16, pp. 2878–2879, 2004.

[25] A. V. Lukashin and M. Borodovsky, "Genemark. hmm: New solutions for gene finding," *Nucleic acids research*, vol. 26, no. 4, pp. 1107–1115, 1998.

[26] M. Stanke, R. Steinkamp, S. Waack, and B. Morgenstern, "Augustus: A web server for gene finding in eukaryotes," *Nucleic acids research*, vol. 32, no. suppl_2, W309–W312, 2004.

[27] T. Flutre, E. Duprat, C. Feuillet, and H. Quesneville, "Considering transposable element diversification in de novo annotation approaches," *PloS one*, vol. 6, no. 1, 2011.

[28] A. F. Smit and R. Hubley, "Repeatmodeler open-1.0," 2008-2015.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[32] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, Austin, TX, vol. 4, 2010.

[33] J. Clauwaert, G. Menschaert, and W. Waegeman, "Deepribo: A neural network for precise gene annotation of prokaryotes by combining ribosome profiling signal and binding site patterns," *Nucleic acids research*, vol. 47, no. 6, e36–e36, 2019.

[34] G. Khodabandelou, J. Mozziconacci, and E. Routhier, "Genome functional annotation using deep convolutional neural network," *bioRxiv*, p. 330 308, 2018.

[35] M. H. da Cruz, P. T. Saito, A. R. Paschoal, and P. H. Bugatti, “Classification of transposable elements by convolutional neural networks,” in *International Conference on Artificial Intelligence and Soft Computing*, Springer, 2019, pp. 157–168.

[36] S. T. Hill, R. Kuintzle, A. Teegarden, E. Merrill III, P. Danaee, and D. A. Hendrix, “A deep recurrent neural network discovers complex biological rules to decipher rna protein-coding potential,” *Nucleic acids research*, vol. 46, no. 16, pp. 8105–8113, 2018.

[37] S. J. Cokus, S. Feng, X. Zhang, Z. Chen, B. Merriman, C. D. Haudenschild, S. Pradhan, S. F. Nelson, M. Pellegrini, and S. E. Jacobsen, “Shotgun bisulphite sequencing of the arabidopsis genome reveals dna methylation patterning,” *Nature*, vol. 452, no. 7184, pp. 215–219, 2008.

[38] N. Gilbert, S. Boyle, H. Fiegler, K. Woodfine, N. P. Carter, and W. A. Bickmore, “Chromatin architecture of the human genome: Gene-rich domains are enriched in open chromatin fibers,” *Cell*, vol. 118, no. 5, pp. 555–566, 2004.

[39] Z. Lippman, A.-V. Gendrel, M. Black, M. W. Vaughn, N. Dedhia, W. R. McCombie, K. Lavine, V. Mittal, B. May, K. D. Kasschau, *et al.*, “Role of transposable elements in heterochromatin and epigenetic control,” *Nature*, vol. 430, no. 6998, pp. 471–476, 2004.

[40] Q. Li, J. I. Gent, G. Zynda, J. Song, I. Makarevitch, C. D. Hirsch, C. N. Hirsch, R. K. Dawe, T. F. Madzima, K. M. McGinnis, *et al.*, “Rna-directed dna methylation enforces boundaries between heterochromatin and euchromatin in the maize genome,” *Proceedings of the National Academy of Sciences*, vol. 112, no. 47, pp. 14 728–14 733, 2015.

[41] L. Stein, “Generic feature format version 3,” *Sequence Ontology Project*, pp. 1–18, 2010.

[42] EMBL-EBI. (2020). Gene annotation in ensembl, [Online]. Available: `https://useast.ensembl.org/info/genome/genebuild/genome_annotation.html` (visited on 01/21/2020).

[43] D. M. Goodstein, S. Shu, R. Howson, R. Neupane, R. D. Hayes, J. Fazo, T. Mitros, W. Dirks, U. Hellsten, N. Putnam, *et al.*, “Phytozome: A comparative platform for green plant genomics,” *Nucleic acids research*, vol. 40, no. D1, pp. D1178–D1186, 2012.

[44] F. Chollet *et al.* (2015). Keras, [Online]. Available: `https://keras.io`.

[45] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.

[46] P. Lamesch, T. Z. Berardini, D. Li, D. Swarbreck, C. Wilks, R. Sasidharan, R. Muller, K. Dreher, D. L. Alexander, M. Garcia-Hernandez, *et al.*, “The arabidopsis information resource (tair): Improved gene annotation and new tools,” *Nucleic acids research*, vol. 40, no. D1, pp. D1202–D1210, 2012.

[47] M. K. Tello-Ruiz, S. Naithani, J. C. Stein, P. Gupta, M. Campbell, A. Olson, S. Wei, J. Preece, M. J. Geniza, Y. Jiao, *et al.*, “Gramene 2018: Unifying comparative genomics and pathway resources for plant research,” *Nucleic acids research*, vol. 46, no. D1, pp. D1181–D1189, 2018.

[48] C.-Y. Cheng, V. Krishnakumar, A. P. Chan, F. Thibaud-Nissen, S. Schobel, and C. D. Town, “Araport11: A complete reannotation of the arabidopsis

thaliana reference genome," *The Plant Journal*, vol. 89, no. 4, pp. 789–804, 2017.

[49] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[50] *Pysam*, `https://github.com/pysam-developers/pysam`, 2019.

[51] M. Frommer, L. E. McDonald, D. S. Millar, C. M. Collis, F. Watt, G. W. Grigg, P. L. Molloy, and C. L. Paul, "A genomic sequencing protocol that yields a positive display of 5-methylcytosine residues in individual dna strands.," *Proceedings of the National Academy of Sciences*, vol. 89, no. 5, pp. 1827–1831, 1992.

[52] G. J. Zynda, *Bsmapz*, `https://github.com/zyndagj/bsmapz`, 2019.

[53] Y. Xi and W. Li, "Bsmap: Whole genome bisulfite sequence mapping program," *BMC bioinformatics*, vol. 10, no. 1, p. 232, 2009.

[54] G. J. Zynda, *Meth5py*, `https://github.com/zyndagj/Meth5py`, 2019.

[55] A. Collette, *Python and HDF5*. O'Reilly, 2013.

[56] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[57] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[58] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[59] D. K. Seymour, D. Koenig, J. Hagmann, C. Becker, and D. Weigel, "Evolution of dna methylation patterns in the brassicaceae is driven by differences in genome organization," *PLoS genetics*, vol. 10, no. 11, 2014.

[60] J. Song, G. Zynda, S. Beck, N. M. Springer, and M. W. Vaughn, "Bisulfite sequence analyses using cyverse discovery environment: From mapping to dmrs," *Current protocols in plant biology*, vol. 1, no. 3, pp. 510–529, 2016.

[61] M. Burset and R. Guigo, "Evaluation of gene structure prediction programs," *genomics*, vol. 34, no. 3, pp. 353–367, 1996.

[62] G. J. Zynda, *Differannotate*, `https://github.com/zyndagj/differannotate`, 2019.

[63] M. El Baidouri, K. D. Kim, B. Abernathy, S. Arikit, F. Maumus, O. Panaud, B. C. Meyers, and S. A. Jackson, "A new approach for annotation of transposable elements using small rna mapping," *Nucleic acids research*, vol. 43, no. 13, e84–e84, 2015.

[64] *Gffcompare*, `https://github.com/gpertea/gffcompare`, version 0.11.5, 2019.

[65] D. S. Standage and V. P. Brendel, "Parseval: Parallel comparison and analysis of gene structure annotations," *BMC bioinformatics*, vol. 13, no. 1, p. 187, 2012.

[66] S. Seabold and J. Perktold, “Statsmodels: Econometric and statistical modeling with python,” in *9th Python in Science Conference*, 2010.

[67] R. Hubley, R. D. Finn, J. Clements, S. R. Eddy, T. A. Jones, W. Bao, A. F. Smit, and T. J. Wheeler, “The dfam database of repetitive dna families,” *Nucleic acids research*, vol. 44, no. D1, pp. D81–D89, 2016.

[68] *Autonomio talos*, `https://github.com/autonomio/talos`, 2019.

[69] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” *arXiv preprint arXiv:1807.05118*, 2018.

[70] M. L. Gonzalez-Garay, “Introduction to isoform sequencing using pacific biosciences technology (iso-seq),” in *Transcriptomics and Gene Regulation*, Springer, 2016, pp. 141–160.

[71] T. H. Kim and B. Ren, “Genome-wide analysis of protein-dna interactions,” *Annu. Rev. Genomics Hum. Genet.*, vol. 7, pp. 81–102, 2006.

[72] E. Stamatatos, N. Fakotakis, and G. Kokkinakis, “Text genre detection using common word frequencies,” in *Proceedings of the 18th conference on Computational linguistics-Volume 2*, Association for Computational Linguistics, 2000, pp. 808–814.

[73] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly, “Detecting spam web pages through content analysis,” in *Proceedings of the 15th international conference on World Wide Web*, ACM, 2006, pp. 83–92.

[74] F. Peng, D. Schuurmans, and S. Wang, "Augmenting naive bayes classifiers with statistical language models," *Information Retrieval*, vol. 7, no. 3-4, pp. 317–345, 2004.

[75] P. Melsted and B. V. Halldórsson, "Kmerstream: Streaming algorithms for k-mer abundance estimation," *Bioinformatics*, vol. 30, no. 24, pp. 3541–3547, 2014.

[76] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, vol. 18, no. 5, pp. 821–829, 2008.

[77] D. E. Wood and S. L. Salzberg, "Kraken: Ultrafast metagenomic sequence classification using exact alignments," *Genome Biol*, vol. 15, no. 3, R46, 2014.

[78] G. E. Sims, S.-R. Jun, G. A. Wu, and S.-H. Kim, "Alignment-free genome comparison with feature frequency profiles (ffp) and optimal resolutions," *Proceedings of the National Academy of Sciences*, vol. 106, no. 8, pp. 2677–2682, 2009.

[79] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[80] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[81] G. Marçais and C. Kingsford, “A fast, lock-free approach for efficient parallel counting of occurrences of k-mers,” *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011.

[82] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, *et al.*, “Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data,” *Nature methods*, vol. 10, no. 6, pp. 563–569, 2013.

[83] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson, “Adam: Genomics formats and processing patterns for cloud scale computing,” *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-207*, 2013.

[84] *Kraken accuracy data*, `https://ccb.jhu.edu/software/kraken/dl/accuracy.tgz`, Accessed: 2015-10-18.

[85] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*, 4th ed. New York: Garland Science, 2002, ISBN: 0815332181.

[86] R. Hand, “Eucaryotic dna: Organization of the genome for replication,” *Cell*, vol. 15, no. 2, pp. 317–325, 1978.

[87] N. Rhind and D. M. Gilbert, “Dna replication timing,” *Cold Spring Harbor perspectives in biology*, vol. 5, no. 8, a010132, 2013.

[88] R. S. Hansen, T. K. Canfield, M. M. Lamb, S. M. Gartler, and C. D. Laird, “Association of fragile x syndrome with delayed replication of the fmr1 gene,” *Cell*, vol. 73, no. 7, pp. 1403–1409, 1993.

[89] K. Woodfine, H. Fiegler, D. M. Beare, J. E. Collins, O. T. McCann, B. D. Young, S. Debernardi, R. Mott, I. Dunham, and N. P. Carter, "Replication timing of the human genome," *Human molecular genetics*, vol. 13, no. 2, pp. 191–202, 2004.

[90] K. Woodfine, D. M. Beare, K. Ichimura, S. Debernardi, A. J. Mungall, H. Fiegler, V. P. Collins, N. P. Carter, and I. Dunham, "Replication timing of human chromosome 6," *Cell Cycle*, vol. 4, no. 1, pp. 172–176, 2005.

[91] D. Schübeler, D. Scalzo, C. Kooperberg, B. van Steensel, J. Delrow, and M. Groudine, "Genome-wide dna replication profile for drosophila melanogaster: A link between transcription and replication timing," *Nature genetics*, vol. 32, no. 3, pp. 438–442, 2002.

[92] Y. Watanabe, A. Fujiyama, Y. Ichiba, M. Hattori, T. Yada, Y. Sakaki, and T. Ikemura, "Chromosome-wide assessment of replication timing for human chromosomes 11q and 21q: Disease-related genes in timing-switch regions," *Human Molecular Genetics*, vol. 11, no. 1, pp. 13–21, 2002.

[93] M. Hayashi, Y. Katou, T. Itoh, M. Tazumi, Y. Yamada, T. Takahashi, T. Nakagawa, K. Shirahige, and H. Masukata, "Genome-wide localization of pre-rc sites and identification of replication origins in fission yeast," *The EMBO journal*, vol. 26, no. 5, pp. 1327–1339, 2007.

[94] E. J. White, O. Emanuelsson, D. Scalzo, T. Royce, S. Kosak, E. J. Oakeley, S. Weissman, M. Gerstein, M. Groudine, M. Snyder, *et al.*, "Dna replication-timing analysis of human chromosome 22 at high resolution and different developmental states," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 51, pp. 17 771–17 776, 2004.

[95] D. M. MacAlpine, H. K. Rodríguez, and S. P. Bell, "Coordination of replication and transcription along a drosophila chromosome," *Genes & development*, vol. 18, no. 24, pp. 3094–3105, 2004.

[96] R. S. Hansen, S. Thomas, R. Sandstrom, T. K. Canfield, R. E. Thurman, M. Weaver, M. O. Dorschner, S. M. Gartler, and J. A. Stamatoyannopoulos, "Sequencing newly replicated dna reveals widespread plasticity in human replication timing," *Proceedings of the National Academy of Sciences*, vol. 107, no. 1, pp. 139–144, 2010.

[97] S. Farkash-Amar, D. Lipson, A. Polten, A. Goren, C. Helmstetter, Z. Yakhini, and I. Simon, "Global organization of replication time zones of the mouse genome," *Genome research*, vol. 18, no. 10, pp. 1562–1570, 2008.

[98] I. Hiratani, T. Ryba, M. Itoh, T. Yokochi, M. Schwaiger, C.-W. Chang, Y. Lyou, T. M. Townes, D. Schübeler, and D. M. Gilbert, "Global reorganization of replication domains during embryonic stem cell differentiation," *PLoS Biol*, vol. 6, no. 10, e245, 2008.

[99] T.-J. Lee, P. E. Pascuzzi, S. B. Settlage, R. W. Shultz, M. Tanurdzic, P. D. Rabinowicz, M. Menges, P. Zheng, D. Main, J. A. Murray, *et al.*, "Arabidopsis thaliana chromosome 4 replicates in two phases that correlate with chromatin state," *PLoS Genet*, vol. 6, no. 6, e1000982, 2010.

[100] H. W. Bass, E. E. Wear, T.-J. Lee, G. G. Hoffman, H. K. Gumber, G. C. Allen, W. F. Thompson, and L. Hanley-Bowdoin, "A maize root tip system to study dna replication programmes in somatic and endocycling nuclei during plant development," *Journal of experimental botany*, vol. 65, no. 10, pp. 2747–2756, 2014.

[101] H. W. Bass, G. G. Hoffman, T.-J. Lee, E. E. Wear, S. R. Joseph, G. C. Allen, L. Hanley-Bowdoin, and W. F. Thompson, "Defining multiple, distinct, and shared spatiotemporal patterns of dna replication and endoreduplication from 3d image analysis of developing maize (zea mays l.) root tip nuclei," *Plant molecular biology*, vol. 89, no. 4-5, pp. 339–351, 2015.

[102] E. E. Wear, L. Concia, A. M. Brooks, E. A. Markham, T.-J. Lee, G. C. Allen, W. F. Thompson, and L. Hanley-Bowdoin, "Isolation of plant nuclei at defined cell cycle stages using edu labeling and flow cytometry," *Plant Cell Division: Methods in Molecular Biology*, vol. 1370, pp. 69–86, 2016.

[103] F. Krueger, *Trim Galore!* [Online; accessed 2016-11], 2012–. [Online]. Available: `https://www.bioinformatics.babraham.ac.uk/projects/trim\_galore`.

[104] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[105] D. S. Johnson, A. Mortazavi, R. M. Myers, and B. Wold, "Genome-wide mapping of in vivo protein-dna interactions," *Science*, vol. 316, no. 5830, pp. 1497–1502, 2007.

[106] A. R. Quinlan and I. M. Hall, "Bedtools: A flexible suite of utilities for comparing genomic features," *Bioinformatics*, vol. 26, no. 6, pp. 841–842, 2010.

[107] A. Diaz, K. Park, D. A. Lim, J. S. Song, *et al.*, "Normalization, bias correction, and peak calling for chip-seq," *Stat Appl Genet Mol Biol*, vol. 11, no. 3, p. 9, 2012.

[108] J. C. Marioni, C. E. Mason, S. M. Mane, M. Stephens, and Y. Gilad, "Rna-seq: An assessment of technical reproducibility and comparison with gene expression arrays," *Genome research*, vol. 18, no. 9, pp. 1509–1517, 2008.

[109] B. Ding, L. Zheng, Y. Zhu, N. Li, H. Jia, R. Ai, A. Wildberg, and W. Wang, "Normalization and noise reduction for single cell rna-seq experiments," *Bioinformatics*, vol. 31, no. 13, pp. 2225–2227, 2015.

[110] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, [Online; accessed 2016-11-28], 2001–. [Online]. Available: `http://www.scipy.org/`.

[111] D. B. Percival and A. T. Walden, *Wavelet methods for time series analysis*. The Edinburgh Building, Cambridge CB2 2RU, UK: Cambridge university press, 2006, vol. 4.

[112] K. Nagaki, P. B. Talbert, C. X. Zhong, R. K. Dawe, S. Henikoff, and J. Jiang, "Chromatin immunoprecipitation reveals that the 180-bp satellite repeat is the key functional dna element of arabidopsis thaliana centromeres," *Genetics*, vol. 163, no. 3, pp. 1221–1225, 2003.

[113] H. Zhao, X. Zhu, K. Wang, J. I. Gent, W. Zhang, R. K. Dawe, and J. Jiang, "Gene expression and chromatin modifications associated with maize centromeres," *G3: Genes— Genomes— Genetics*, vol. 6, no. 1, pp. 183–192, 2016.

[114] P. S. Schnable, D. Ware, R. S. Fulton, J. C. Stein, F. Wei, S. Pasternak, C. Liang, J. Zhang, L. Fulton, T. A. Graves, *et al.*, "The b73 maize genome: Complexity, diversity, and dynamics," *science*, vol. 326, no. 5956, pp. 1112–1115, 2009.

[115] Y. Lubelsky, J. A. Prinz, L. DeNapoli, Y. Li, J. A. Belsky, and D. M. MacAlpine, "Dna replication and transcription programs respond to the same chromatin cues," *Genome research*, vol. 24, no. 7, pp. 1102–1114, 2014.

[116] S. N. Anderson, G. Zynda, J. Song, Z. Han, M. Vaughn, Q. Li, and N. M. Springer, "Subtle perturbations of the maize methylome reveal genes and transposons silenced by chromomethylase or rna-directed dna methylation pathways," *G3: Genes, Genomes, Genetics*, 2018. DOI: `10.1534/g3.118.200284`. eprint: `http://www.g3journal.org/content/early/2018/04/04/g3.118.200284.full.pdf`. [Online]. Available: `http://www.g3journal.org/content/early/2018/04/04/g3.118.200284`.

[117] L. Concia, A. M. Brooks, E. Wheeler, G. J. Zynda, E. E. Wear, C. LeBlanc, J. Song, T.-J. Lee, P. E. Pascuzzi, R. A. Martienssen, M. W. Vaughn, W. F. Thompson, and L. Hanley-Bowdoin, "Genome-wide analysis of the arabidopsis replication timing program," *Plant Physiology*, vol. 176, no. 3, pp. 2166–2185, 2018, ISSN: 0032-0889. DOI: `10.1104/pp.17.01537`. eprint: `http://www.plantphysiol.org/content/176/3/2166.full.pdf`. [Online]. Available: `http://www.plantphysiol.org/content/176/3/2166`.

[118] E. E. Wear, J. Song, G. Zynda, C. LeBlanc, T.-J. Lee, L. Mickelson-Young, L. Concia, P. Mulvaney, E. S. Szymanski, G. C. Allen, R. Martienssen, M. W. Vaughn, L. Hanley-Bowdoin, and W. Thompson, "Genomic analysis of the dna replication timing program during mitotic s phase in maize (zea mays l.) root tips," *The Plant Cell*, 2017, ISSN: 1040-4651. DOI: `10.1105/tpc.17.00037`. eprint: `http://www.plantcell.org/content/early/2017/08/25/tpc.17.`

00037.full.pdf. [Online]. Available: http://www.plantcell.org/content/early/2017/08/25/tpc.17.00037.

[119] G. J. Zynda, J. Song, L. Concia, E. E. Wear, L. Hanley-Bowdoin, W. F. Thompson, and M. W. Vaughn, "Repliscan: A tool for classifying replication timing regions," *BMC Bioinformatics*, vol. 18, no. 1, p. 362, 2017, ISSN: 1471-2105. DOI: 10.1186/s12859-017-1774-x. [Online]. Available: https://doi.org/10.1186/s12859-017-1774-x.

[120] C. Klijn, S. Durinck, E. W. Stawiski, P. M. Haverty, Z. Jiang, H. Liu, J. Degenhardt, O. Mayba, F. Gnad, J. Liu, G. Pau, J. Reeder, Y. Cao, K. Mukhyala, S. K. Selvaraj, M. Yu, G. J. Zynda, M. J. Brauer, T. D. Wu, R. C. Gentleman, G. Manning, R. L. Yauch, R. Bourgon, D. Stokoe, Z. Modrusan, R. M. Neve, F. J. de Sauvage, J. Settleman, S. Seshagiri, and Z. Zhang, "A comprehensive transcriptional portrait of human cancer cell lines," *Nature Biotechnology*, vol. 33, no. 3, pp. 306–312, 2015, ISSN: 1087-0156. [Online]. Available: http://dx.doi.org/10.1038/nbt.308010.1038/nbt.3080http://www.nature.com/nbt/journal/v33/n3/abs/nbt.3080.html{\#}supplementary-information.

[121] Q. Li, J. I. Gent, G. Zynda, J. Song, I. Makarevitch, C. D. Hirsch, C. N. Hirsch, R. K. Dawe, T. F. Madzima, K. M. McGinnis, D. Lisch, R. J. Schmitz, M. W. Vaughn, and N. M. Springer, "Rna-directed dna methylation enforces boundaries between heterochromatin and euchromatin in the maize genome," *Proceedings of the National Academy of Sciences*, vol. 112, no. 47, pp. 14 728–14 733, 2015. DOI: 10.1073/pnas.1514680112. eprint: http://www.pnas.

`org/content/112/47/14728.full.pdf`. [Online]. Available: `http://www.pnas.org/content/112/47/14728.abstract`.

[122] A. Kilaru, X. Cao, P. B. Dabbs, H.-J. Sung, M. M. Rahman, N. Thrower, G. Zynda, R. Podicheti, E. Ibarra-Laclette, L. Herrera-Estrella, K. Mockaitis, and J. B. Ohlrogge, "Oil biosynthesis in a basal angiosperm: Transcriptome analysis of persea americana mesocarp," *BMC Plant Biology*, vol. 15, no. 1, p. 203, 2015, ISSN: 1471-2229. DOI: `10.1186/s12870-015-0586-2`. [Online]. Available: `https://doi.org/10.1186/s12870-015-0586-2`.

[123] Q. Li, J. Song, P. T. West, G. Zynda, S. R. Eichten, M. W. Vaughn, and N. M. Springer, "Examining the causes and consequences of context-specific differential dna methylation in maize," *Plant Physiology*, 2015, ISSN: 0032-0889. DOI: `10.1104/pp.15.00052`. eprint: `http://www.plantphysiol.org/content/early/2015/04/13/pp.15.00052.full.pdf`. [Online]. Available: `http://www.plantphysiol.org/content/early/2015/04/13/pp.15.00052`.

[124] G. Zynda, N. Gaffney, M. Dalkilic, and M. Vaughn, "Feature frequency profiles for automatic sample identification using pyspark," in *Proceedings of the 5th Workshop on Python for High-Performance and Scientific Computing*, ACM, 2015, p. 5.

CURRICULUM VITAE

Gregory J. Zynda

Personal Data

E-Mail: gjzynda@iu.edu

Website: gregoryzynda.com

GitHub: https://github.com/zyndagj/

Education

April 2020	**Ph.D. in Informatics** *Indiana University* Concentration: Bioinformatics Minor: Computer Science
May 2010	**B.S. in Mathematics & Computer Science** *Rose-Hulman Institute of Technology* Minors in Computational Science and Japanese

Professional Experience

2014-present	**Texas Advanced Computing Center - Scientist** Supervisor: Matthew W. Vaughn Direct research on DNA replication and epigenetics. Development and support for SD2E, CyVerse, TACC, and XSEDE.
2010-2013	**The Center for Genomics and Bioinformatics - Analyst** Supervisors: Dr. Haixu Tang and Dr. Doug Rusch Served as a sequencing analyst for genome, transcriptome, epigenetic, and genome assembly projects at IU.

Summer 2012 **Genentech - Bioinformatics Intern**
Supervisor: Dr. Zhaoshi Jiang
Built and curated a comprehensive and dynamic pathogen database from public sources. Designed and implemented a parallel pipeline for detecting pathogen interactions in sequencing data.

Summer 2009 **Interdisciplinary Research Collaborative**
Rose-Hulman Institute of Technology
Supervisor: Dr. Allen Holder
Used linear programming to simulate a stable cell state and calculate the importance of metabolic processes in *E. coli*.

2009 **INFORMS - Web Developer**
Supervisor: Dr. Allen Holder
Built a redesigned version of the INFORMS Mathematical Programming Glossary as a customized MediaWiki to allow for easy editing, math fonts, and bulk browsing.

Activities

2015-2018 **CODE@TACC**
Instructed programming, electronics, robotics, and linux at TACC's summer camp for high school students.

2015-2018 **Shadow a Scientist**
Teach middle school students about computing and bioinformatics with hands-on activities.

Presentations

*Zynda, G.** 2019. Singularity and RGC. TACCSTER 2019. Austin, TX.

*Zynda, G.** 2018. Container ecosystem at TACC. SD2E. Seattle, WA.

*Zynda, G.** , Springer, N., Vaughn, MW. 2017. Utilizing DNA Methylation for Genome Annotation Through Deep Learning. Rocky 2017. Aspen, CO.

*Zynda, G.** , Springer, N., Vaughn, MW. 2016. Genome annotation through DNA methylation and a semi-supervised profile-HMM. Genome Informatics, Cambridge, UK.

*Zynda, G.** , Walling, D. 2016. Zika Microbiome Hub. Hackathon. Austin, TX.

*Zynda, G.** , Song, J., Proctor, WC, Vaughn, MW. 2016. Supporting complicated PacBio workflows with Agave. Workshop at PAG-XXV:CyVerse, San Diego, CA.

*Zynda, G.** , Gaffney, N., Dalkilic, M., Vaughn, MW. 2015. Feature frequency profiles for automatic sample identification using pyspark. 5th Workshop on Python for High-Performance and Scientific Computing, Austin, TX.

*Zynda, G.** , Song, J., Markham, E., Hanley-Bowdoin, L., Vaughn, MW. 2015. Separating DIP-Seq Signals Through Independent Component Analysis. Poster at PAG-XXIV, San Diego, CA.

*Zynda, G.** 2014. *De Novo* TE annotation with TEAM: TE Annotation from Methylation. Lightning talk at ISMB:HiTSeq 2014, Boston, MA.

*Zynda, G.** 2014. TEAM: TE Annotation from Methylation. Poster presented at RECOMB 2014, Pittsburgh, PA.

*Zynda, G.** , Tang, H. 2013. Methods to Detect Changes in Regions of Methylation. Presentation at The University of Toronto.

*Zynda, G.** , Jiang, Z. 2012. Pathogen Detection in Transcriptome of Cancer Samples. Poster at 2012 Genentech Intern Research Poster Session.

*Zynda, G.** , Lutz, K., Holder, A. 2009. The Mathematical E. coli. Poster at the 2009 Rose-Hulman Interdisciplinary Research Collaborative.

*Zynda, G.** , Krisenko M.*, Moseng M.*, Paine, D.*, Williamson R.* 2009. Tracing the Journey of Each Footstep. Poster at the 2009 Rose-Hulman Interdisciplinary Research Collaborative.

Zynda, G. , Williamson R.*, Campbell, B.*, Chappell, R. 2008. Schedule Planning System Using a Database. Presentation at the 2008 IN/IL American Society for Engineering Education conference.

* Denotes primary presenter(s).

Publications

S. N. Anderson, G. Zynda, J. Song, *et al.*, "Subtle perturbations of the maize methylome reveal genes and transposons silenced by chromomethylase or rna-directed dna methylation pathways," *G3: Genes, Genomes, Genetics*, 2018. DOI: `10.1534/g3.118.200284`. eprint: `http://www.g3journal.org/content/early/2018/04/`

04/g3.118.200284.full.pdf. [Online]. Available: http://www.g3journal.org/content/early/2018/04/04/g3.118.200284

L. Concia, A. M. Brooks, E. Wheeler, *et al.*, "Genome-wide analysis of the arabidopsis replication timing program," *Plant Physiology*, vol. 176, no. 3, pp. 2166–2185, 2018, ISSN: 0032-0889. DOI: 10.1104/pp.17.01537. eprint: http://www.plantphysiol.org/content/176/3/2166.full.pdf. [Online]. Available: http://www.plantphysiol.org/content/176/3/2166

E. E. Wear, J. Song, G. Zynda, *et al.*, "Genomic analysis of the dna replication timing program during mitotic s phase in maize (zea mays l.) root tips," *The Plant Cell*, 2017, ISSN: 1040-4651. DOI: 10.1105/tpc.17.00037. eprint: http://www.plantcell.org/content/early/2017/08/25/tpc.17.00037.full.pdf. [Online]. Available: http://www.plantcell.org/content/early/2017/08/25/tpc.17.00037

G. J. Zynda, J. Song, L. Concia, *et al.*, "Repliscan: A tool for classifying replication timing regions," *BMC Bioinformatics*, vol. 18, no. 1, p. 362, 2017, ISSN: 1471-2105. DOI: 10.1186/s12859-017-1774-x. [Online]. Available: https://doi.org/10.1186/s12859-017-1774-x

J. Song, G. Zynda, S. Beck, *et al.*, "Bisulfite sequence analyses using cyverse discovery environment: From mapping to dmrs," *Current protocols in plant biology*, vol. 1, no. 3, pp. 510–529, 2016

C. Klijn, S. Durinck, E. W. Stawiski, *et al.*, "A comprehensive transcriptional portrait of human cancer cell lines," *Nature Biotechnology*, vol. 33, no. 3, pp. 306–312, 2015, ISSN: 1087-0156. [Online]. Available: http://dx.doi.org/10.1038/nbt.308010.1038/nbt.3080http://www.nature.com/nbt/journal/v33/n3/abs/nbt.3080.html{\#}supplementary-information

Q. Li, J. I. Gent, G. Zynda, *et al.*, "Rna-directed dna methylation enforces boundaries between heterochromatin and euchromatin in the maize genome," *Proceedings of the National Academy of Sciences*, vol. 112, no. 47, pp. 14 728–14 733, 2015. DOI: 10.1073/pnas.1514680112. eprint: http://www.pnas.org/content/112/47/14728.full.pdf. [Online]. Available: http://www.pnas.org/content/112/47/14728.abstract

A. Kilaru, X. Cao, P. B. Dabbs, *et al.*, "Oil biosynthesis in a basal angiosperm: Transcriptome analysis of persea americana mesocarp," *BMC Plant Biology*, vol. 15, no. 1, p. 203, 2015, ISSN: 1471-2229. DOI: 10.1186/s12870-015-0586-2. [Online]. Available: https://doi.org/10.1186/s12870-015-0586-2

Q. Li, J. Song, P. T. West, *et al.*, "Examining the causes and consequences of context-specific differential dna methylation in maize," *Plant Physiology*, 2015, ISSN:

0032-0889. DOI: `10.1104/pp.15.00052`. eprint: `http://www.plantphysiol.org/content/early/2015/04/13/pp.15.00052.full.pdf`. [Online]. Available: `http://www.plantphysiol.org/content/early/2015/04/13/pp.15.00052`

G. Zynda, N. Gaffney, M. Dalkilic, *et al.*, "Feature frequency profiles for automatic sample identification using pyspark," in *Proceedings of the 5th Workshop on Python for High-Performance and Scientific Computing*, ACM, 2015, p. 5