SANDIA REPORT
SAND2021-12015
Printed Click to enter a date

Sandia National Laboratories

Data Science and Machine Learning for Genome Security

Stephen J. Verzi, Raga Krishnakumar, Daniel J Krofcheck, Drew Levin, and Kelly P Williams

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

This report describes research conducted to use data science and machine learning methods to distinguish targeted genome editing versus natural mutation and sequencer machine noise. Genome editing capabilities have been around for more than 20 years, and the efficiencies of these techniques has improved dramatically in the last 5+ years, notably with the rise of CRISPR-Cas technology. Whether or not a specific genome has been the target of an edit is concern for U.S. national security. The research detailed in this report provides first steps to address this concern. A large amount of data is necessary in our research, thus we invested considerable time collecting and processing it. We use an ensemble of decision tree and deep neural network machine learning methods as well as anomaly detection to detect genome edits given either whole exome or genome DNA reads. The edit detection results we obtained with our algorithms tested against samples held out during training of our methods are significantly better than random guessing, achieving high F1 and recall scores as well as with precision overall.

ACKNOWLEDGEMENTS

We wish to acknowledge the help and support of the Bioscience Research Foundation and IAT as well as from 08600 and 08700 management. We also want to thank Patrick D. Finley for his technical expertise and guidance in our work. He was instrumental in getting this effort off the ground and keeping it going early on. Callie Boskin worked many hours downloading and processing large datasets. She helped finalize and utilize our data processing pipeline and investigated many of our various data sources which led to the results presented herein. Adriana Morales Miranda and Daniel Carmody helped with early design and implementation of our machine learning algorithms. We thank Aundre Huynh, Meghan Othart, and Justin White for their astute project support and for keeping the us on track. Finally, we than Daniel J. Pless for providing technical quality review.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

EXECUTIVE SUMMARY

The current explosion of advances in gene editing technology and public accessibility to these techniques pose potential biosecurity threats. To accurately assess national security risk, threats from bioengineering must be clearly distinguishable from natural genome variation. This capability facilitates understanding of the consequences of such threats and provides options for mitigation and/or forensic investigation.

Current state-of-the-art methods measure edit efficacy and look for specific, known off-target effects, but there is a gap in the research community for detecting edits without prior information of the most likely regions in the genome where the edit effects will occur. Our research has focused on three specific classes algorithms, and this report documents that these methods have shown high potential for detecting edits. First, we applied decision-tree learners to many data sets and were able to identify subtle patterns and signatures that indicate an edit. Second, we used two complementary architectures of Deep Neural Networks learners both to 1) classify edit versus non-edit regions in next generation deep sequencing DNA reads and 2) learn grammatical patterns indicative of edit versus non-edit regions in DNA sequences. Third, novel Anomaly Detection algorithms were implemented to detect groups of anomalous bases in DNA sequence reads. While each of these methods has previously been shown to perform well in other domains, such as image processing and classification as well as natural language processing, they have not been applied to genomic edit detection and characterization.

We have acquired, processed, and utilized data from a large number of existing sources, from genome editing experiments, including those looking at off-target (edit) effects and those used to predict edit effects. We have designed and built our own data-processing pipeline for handling raw data file formats, providing genomic noise counts and ultimately features used in our machine learning algorithms. The results provided in this report show significant evidence of success in the challenging problem of detecting targeted edits in the human genome. The capabilities developed in this research provide decision support for national security community involved in assessing potential risks from biosecurity threats. Our research, documented in this report, demonstrates both successes and pitfalls of these techniques while providing avenues for future research to build upon what we have done.

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
1D	One-Dimensional
AMF	Adaptive Median Filtering (anomaly detection/filtering algorithm)
AMPLICON	(Product of amplification reaction, such as PCR or LCR)
ANOVA	Analysis of Variance
API	Application Programming Interface
BAM	Binary (version of SAM file format)
BERT	Bidirectional Encoder Representations from Transformers (neural network architecture)
bp	base position(s)
BWA	Burrows-Wheeler Aligner
BWA-MEM	BWA Maximal Effects Matches (alignment algorithm)
C4BPB	(Gene in chromosome 1: human)
Cas9	CRISPR-associated protein 9
CCR5	(Gene in chromosome 3: human)
CNN	Convolutional Neural Network
CRISPR	Clustered Regularly Interspaced Short Palindromic Repeats
CRISPR-Cas	CRISPR (system using) Cas (genes for adaptive mutation)
CV	Cross Validation
DNA	Deoxyribonucleic Acid
DNN	Deep Neural Network
DRYK1	(Gene in chromosome 21: human)
EMX1	(Gene in chromosome 2: human)
GRC38	Genome Reference Consortium 38
HDR	Homology-Directed Repair
HG38	Homo sapiens (human) Genome assembly GRC38
IAT	Investment Area Team
IGV	Integrated Genomics Viewer
LCR	Ligase Chain Reactions
LIF	Leaky Integrate-and-Fire (mathematical neuron model)
LINC00116	(Gene in chromosome 2: human)
MAD	Median Absolute Deviation from the median
MMEJ	Micro-homology Mediated End Joining
NCBI	National Center for Biotechnology Information

Abbreviation	Definition
NHEJ	Non-Homologous End Joining
PAM	Protospacer Adjacent Motif (2-6 base pair DNA sequence)
PC	Principal Component
PCA	Principal Component Analysis
PCR	Polymerase Chain Reactions
PHRED	(Per-base quality score in base calling)
PRAM	Parallel Random Access Machine
RF	Random Forest (machine learning algorithm)
RNA	Ribonucleic Acid
RoBERTa	Robustly Optimized BERT Pretraining Approach
SAM	Sequence Alignment/Map (file format)
sgRNA	single guide RNA
SMOTE	Synthetic Minority Over-sampling Technique
SORT1	(Sortilin gene in chromosome 1: human)
TAZ	(TAFFAZIN gene in chromosome X: human)
tSNE	t-distributed Stochastic Neighbor Embedding
ucb	upper confidence bound
WDR5	(Gene in chromosome 9: human)
WGS	Whole Genome Sequencing
WXS	Whole exome Sequencing
XGB	XGBoost (machine learning algorithm)

1. INTRODUCTION

The current explosion of advances in gene editing technology and public accessibility to these techniques poses potential biosecurity threats. To accurately assess national security risk, threats from bioengineering must be clearly distinguishable from natural genome variation. This capability would facilitate understanding of the consequences of such threats as well as allow mitigation and/or forensic investigation.

The most recently well-studied system for genome editing is the CRISPR system, which is an adaptation of the endogenous immune system of bacteria, developed to guard against attack from viruses [Pickar-Oliver & Gersbach, 2019; Uddin et al., 2020; Nidhi et al., 2021]. Briefly, a small-guide RNA (sgRNA) is used as a beacon to guide the CRISPR machinery to a specific location in a molecule of DNA (or RNA). This machinery usually consists of at least one enzyme that possesses a cleavage activity making a break in the nucleic acid (or the removal of a single base), which in the case of the adapted systems, can be repaired by host-directed mechanisms [Du et al., 2018]. The most common such enzyme is Cas9 from S. pyogenes, which is an RNA-directed DNA endonuclease [Jiang & Doudna, 2017; Whinn et al., 2019; Yourik et al., 2019]. Many CRISPR enzymes are directed both by their sgRNA but also by the presence of what is known as a PAM sequence (protospacer-adjacent motif) located downstream of the target region [Gleditzsch et al., 2019; Walton et al., 2021]. For the purposes of this work, we focus on the detection of edits on DNA genomes as opposed to RNA molecules, but we believe that some of the techniques we describe in this report could be used to detected similar mutations in RNAs, when that technology evolves to the same extent as the DNA-targeting CRISPRs [Kannan et al., 2021].

CRISPR technology has a wide range of applications in healthcare, agriculture and biomanufacturing (among other areas), which accounts for the significant allocation of resources and efforts towards furthering research in this field [Adli, 2018; Ding et al., 2020; Hirakawa et al., 2020; Zhu et al., 2020]. This invariably brings up the issue of biosafety and dual-use with such technologies, highlighting the need to simultaneously develop methods to track and diagnose nefarious use of CRISPR tools [Braddick & Ramarohetra, 2020; El-Mounadi et al., 2020]. To this end, we chose to focus on evaluating how machine learning tools could be used to determine whether a genome (or other nucleic acid molecule) had been edited using CRISPR technology.

The following figure shows how the natural DNA damage repair processes of non-homologous end-joining (NHEJ) and homology directed repair (HDR) can be stimulated by site-specific DNA cleavage implemented by CRISPR-Cas9 and used to promote genome editing [Brandsma & Gent, 2012; Du et al., 2018; West & Gronvall, 2020].

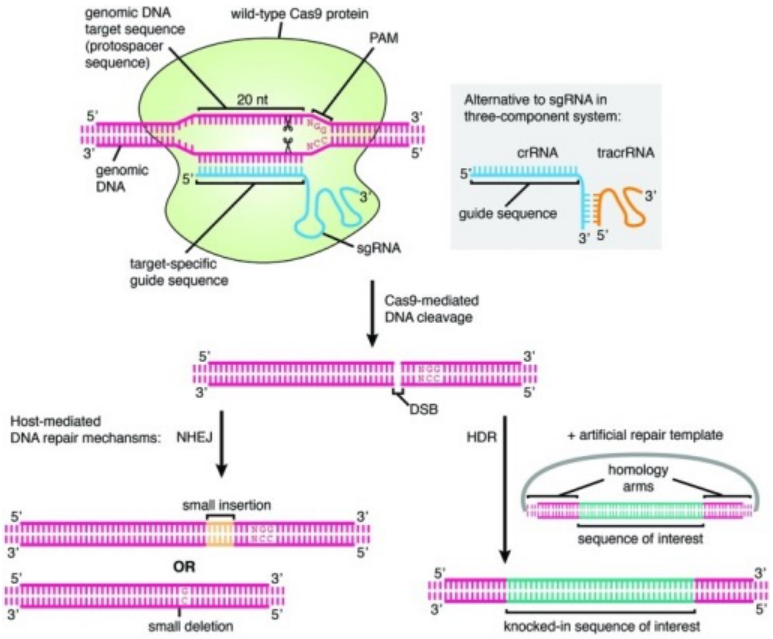

Figure 1-1. Illustration genome editing (Image from Genome Research Limited).

The genome editing illustrated in Figure 1-1, can also be described by the following process:

- Site-specific DNA cleavage by CRISPR-Cas9 alone can be repaired by NHEJ, leading to indels at the cleavage site. At least some of these indels may have consequences for gene expression that may be the desired result of the genome editing attempt.
- If a DNA template is introduced together with the CRISPR-Cas9, more controlled mutations can be achieved through HDR. Here the template DNA has homology to the two flanks of the cleavage site, and the desired mutation is encoded in between. The desired mutation may be a deletion, an insertion of an entire gene, or as small as a single point mutation in an existing gene.

Our research, described in this report, uses data science methods to detect the use of this editing process upon a genome. First, in order to understand and characterize the difference between statistics of base composition with respect to the genome around the edit target, we have designed and implemented our own bioinformatics software tools for counting genomic noise in sequence data. Second, the left and right junctions adjacent to the cut (and potential insertion) contain unique sequence signals, specifically in their noise characterization. Third, we show how to leverage sequence learning, by using deep learning neural networks (such as 1D convolutional neural networks, also called CNNs, and transformer networks [Goodfelow et al., 2016; Devlin et al., 2018]) and decision-tree learning (such as random forests and XGBoost [Chen & Guestrin, 2016]) trained to recognize (at least part of) the normal genome sequence as well as anomaly detection to detect anomalous sequence regions and ensembles of these methods. Principles that may be exploited in editing detection are: 1) some chromosomes in some cells of the targeted tissue may escape editing, i.e., penetrance of editing may be incomplete. 2) Indels indicative of NHEJ may result as side

reactions when attempting to edit by HDR. Both of these effects can be detected and measured using deep sequencing.

1.1. Machine Learning (ML) Methods

In our research we use several machine learning methods, including decision-tree learning, deep neural networks, and anomaly detection algorithms as well as ensembles of these methods. A decision-tree learner is built by constructing a tree of decisions (see Figure 1-2a) whereby the path from the root to any leaf node forms a composite decision such as in classification (i.e., edit versus no-edit). At each node in the path, including the root but excluding the leaf, feature values are used in a local decision (i.e., between edit versus no-edit), and the composite decision is formed during learning to best approximate data used during training. A deep neural network (DNN) is a layered architecture of nodes, where each node computes a (typically non-linear) function of its inputs to pass onto the next layer (see Figure 1-2b). In a DNN, the mapping from inputs to target outputs is learned during training, where the weights connecting nodes from each layer to the next are adjusted using backpropagation.

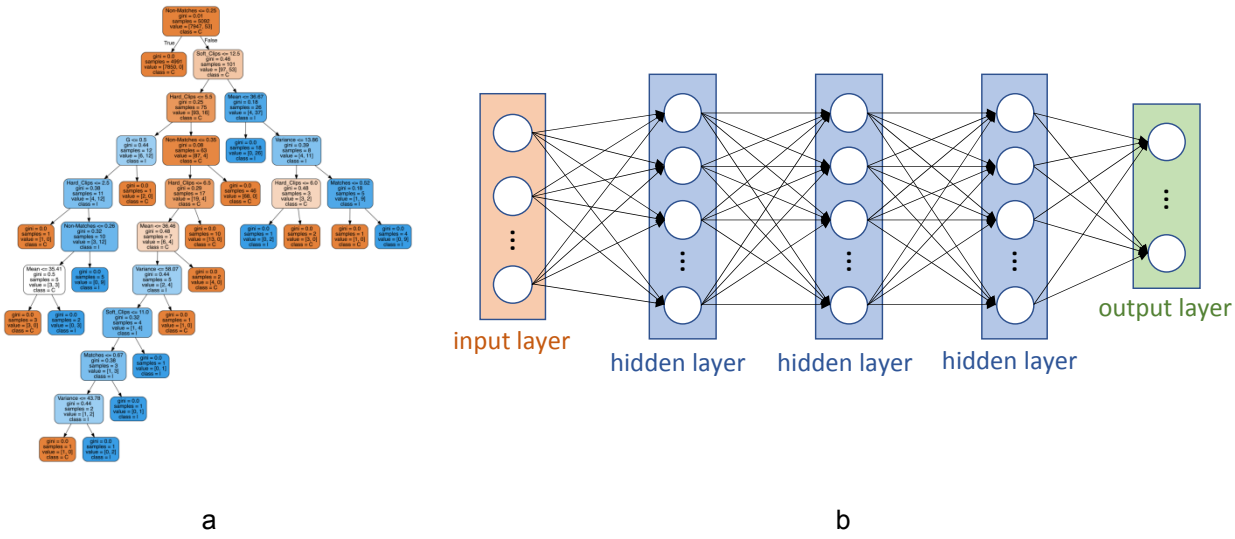

Figure 1-2. Example machine learning models: a) decision-tree and b) deep neural network.

Backpropagation learning works by first computing error (also called "loss") at the output layer between feed-forward propagation through the network (from the input layer all the way to the output layer) and desired target outputs. Then, this error is propagated backward from each layer (h_j^l) to the previous layer (h_i^{l-1}) through an assignment of credit for composition of the error computed at h_j, due the weighted connections from the previous layer h_i^{l-1}, using a loss function and application of gradient descent. This kind of learning operates by adjusting the weights connecting the layers (i.e., between h_j and h_i^{l-1}, for all i) to reduce the error (at h_j).

We employ two kinds of DNNs in our methods: 1) convolutional neural networks (CNNs) [Fukushima, 1980; Homma et al., 1987; Goodfellow et al., 2016] and 2) transformer networks, such as the BERT (Bidirectional Encoder Representations from Transformers) architecture [Devlin et al., 2018]. Convolutional neural networks are useful in learning hierarchically organized spatial relationships, such as objects in images built from successively smaller elements all the way down to very small groups of pixels just downstream from the input layer (where the image is presented as input). Transformer networks are useful in learning sequential relationships from successive inputs, which allows the networks to learn natural language sequences.

Anomaly detection is typically simpler in construction than decision-tree and DNN learning, and thus such algorithms are designed to find and output outliers in the data that are processed through the detection algorithm. An anomaly detector can be formed from a simple threshold applied to a given feature, whereby one side of the threshold indicates anomalous data. Anomaly detectors can also be more complex taking into account multiple features as well as spatial or sequential data relationships.

1.2. Ensembles

Ensemble learners leverage the aggregation of multiple lightweight (also called "weak" – defined as slightly better than random guessing) learners to form a composite learner with greater performance (i.e., no longer "weak"). In our research, we implemented several types of ensemble learners. The random forest and XGBoost are themselves ensemble learners (i.e., each is composed of multiple decision trees used in aggregate for the result that is ultimately output), but we have also implemented ensembles of deep neural networks as well as ensembles of anomaly detectors. In an ensemble, each method augments overall performance by maintaining relative orthogonality to the others (in the ensemble).

We expected that ensembles of learners would be beneficial in the domain of genome edit detection due to the fact that genome edits can vary significantly from experiment to experiment (see principal component analysis in the Results section). To address this variance, we employ multiple learners in aggregate to provide both coverage (across the variance space) and statistically significant accuracy in voting across the ensemble.

2. RELATIONSHIP TO PREVIOUS RESEARCH

Genome editing, using methods such as CRISPR-Cas9 [Jinek et al., 2012; Cong et al., 2013; Mali et al., 2013; Wang et al., 2018], continues to be developed over the last several years with applications in several areas including, drug development in medicine, understanding genetic variation in biology, and more productive food and biofuels in biotechnology [Hsu et al., 2014]. Two important issues with CRISPR editing are how effective the editing is at the target site as well as if there are any off-target sites, and how much edit activity happens there. Several researchers have investigated off-target effects from CRISPR [Li et al., 2013; Cho et al., 2014; Chuai et al., 2018; Anzalone et al., 2020], and the general conclusion is that there is very little off-target activity with many standard single guide RNA (sgRNA) assays and high specificity enzymes, such as Cas9. A number of recent papers describe predictive models for CRISPR edit effect in the genome [van Overbeek et al., 2016; Allen et al., 2018; Chuai et al., 2018; Chakrabarti et al., 2019; Chen et al., 2019; Arbab et al., 2020].

Use of machine learning, in particular deep neural networks (DNNs), is the subject of many recent papers trying to understand genome structure, the function of coding regions and calling of genomic indels[1] and mutations [Chuai et al., 2018; Cai et al., 2019; Chen et al., 2019; Arbab et al., 2020]. This research does not specifically attempt to find genome edits in deep sequence read data. Our research described in this report addresses this last gap mentioned.

[1] The term indel refers to either a small insertion or deletion in the genome sequence.

3. METHODS

Our contribution, described in this manuscript, is to provide CRISPR edit detection tools. Our edit detection uses raw sequence read data, and predicts whether or not any CRISPR edits exist in the data as well as a measure for the effectiveness of the edit, given the sequence read data. To provide CRISPR edit detection, we use ensembles of machine learning and anomaly detection algorithms, including random forests, deep neural networks and spiking neural networks.

Our results show that we can reliably detect edit signatures in genomic noise (see our initial explorations in Visualizing CRISPR Experiments in Appendix A and full results in the Results section below).

In this section we will describe each of our edit detection methods, including decision tree methods, deep neural network models and anomaly detection algorithms.

3.1. Decision-tree Learners

Decision tree-based ensemble methods such as Random Forest (RF) and XGBoost (XGB) have often been favored in biological classification problems due to their ability to cope with complex relationships between features that are often not fully independent. In addition, the models tend to be reasonably interpretable, which helps with assigning biological meaning to the outcomes [Boulesteix et al., 2012; Siddiquee & Tasnim, 2018; Chang et al., 2019; Chen et al., 2020; do Nascimento et al., 2020; Gao et al., 2020; Vanhaeren et al., 2020]. We therefore wanted to evaluate both Random Forest and XGBoost models for identification of genomic regions cut by Cas9 and repaired by non-homologous end-joining (NHEJ) [Yeh et al., 2019].

We started by defining how we would assign our training, validation and test data sets. We developed a method by which a window surrounding a pre-defined edit-positive would be randomly sampled such that at least 50% of the edited region would fall into the window. Negative regions were chosen randomly from data sets with no pre-defied edits. Window sizes were optimized for each of the models, and that process will be described in the next paragraph. For training of each model, we used a balanced data set, with an equal number of positive and negative windows (10,000 each) – we left out one experimental data set to serve as the test set, and generated the positive windows from the remainder of the edit-positive data.

While certain data sets (such as [Chakrabarti et al., 2018]) have a significant number of data points that we could have used for training, we noticed that pulling a disproportionate amount of our data from a single publication, even across different edited sites, created a bias in the training set (see results for details). This is likely due to site- and lab-specific batch effects that manifest in an unknown, latent way in the data, and are extremely difficult to account for. We therefore wanted to keep the sources of our positive data relatively balanced, without one publication or data source making up the majority of the samples. However, since all our positive windows were generated using repetitive sampling of a handful of edited regions, we could potentially run into the problem of overfitting our models. To solve this, we decided to generate synthetic positive data based on the existing positives using SMOTE [Chawla et al., 2002]. We demonstrate that correlations between features and class (i.e., edit or no edit) are not significantly affected by SMOTE, suggesting that the generated data stays true to distribution of the existing positive data (see results).

We used the genome counter to generate features for machine learning for each type of alignment call made by BWA-MEM (matches, insertions, deletions, mutations, soft clips, hard clips) as follows:

- **Average fraction of alignment calls**: In any given window, the average fraction of coverage per base.
- **Standard deviations of alignment calls**: The standard deviation of the individual base fractions.
- **Crosses**: Given a percentile threshold (optimized per model, see description later), count the number of times the signal within the window crosses that threshold.
- **Peaks**: Using scipy's 'find_peaks' module [Virtanen et al., 2020], count the number of signal peaks within a window.

3.1.1. Random Forest (RF)

The RandomForestClassifier function of scikit-learn [Virtanen et al., 2020] was used to train a Random Forest model. BayesianOptimization from GPyOpt [González et al., 2014; González et al., 2016] was used to tune hyperparameters and a subset of feature parameters as follows (in Table 1):

Table 1. Random forest hyperparameter value ranges for optimization.

Feature/hyperparameter	Range of Values
Window size for data collection	50 – 500
Percentile threshold for 'cross' feature	10 – 90
Neighbors to consider for SMOTE	2 – 20
Maximum depth	1 – 150
Number of estimators	10 – 1000
Minimum number of samples per split	2 – 10

The ucb (upper confidence bound) function was used as the acquisition function, and we used the default kappa value of 2.56, which balances exploitation of known regions of the hyperparameter space with exploration of unknown regions (the upper confidence bound is the 99th percentile with a kappa of 2.56). Training was done on 75% of the windowed data, and validation was performed on the remaining 25% (data was split so that positives and negatives were evenly represented in each group). For each iteration of optimization, a 10-fold cross-validation (CV) was performed, and an average F1 score was obtained for all 10 CVs, for a total of 20 iterations (all optimizations converged after 20 iterations, see results for examples). These average F1 scores were used to determine the optimal features and hyperparameters for model training.

We found that when directly compared with XGBoost, the Random Forest method on average performed worse with respect to precision and sensitivity (see results section for details), so we decided to proceed with XGBoost.

3.1.2. XGBoost

The XGBClasifier function from xgboost [Chen & Guestrin, 2016] was used to train a XGB model. Bayesian optimization was used in a similar manner to the RF model to optimize the following features/hyperparameters (in Table 2):

Table 2. XGBoost hyperparameter value ranges for optimization.

Feature/hyperparameter	Range of Values
Window size for data collection	50 – 500
Percentile threshold for 'cross' feature	10 – 90
Neighbors to consider for SMOTE	2 – 20
Maximum depth	1 – 150
Gamma	0 – 1
Learning rate	0 – 1
Number of estimators	10 – 1000

Training, cross-validation and validation was done as described for the RF model.

3.2. Deep Neural Network Models

Neural network models are a directed set of arithmetic operations defining a function that converts an input of fixed shape to a desired output. Through supervised learning, where a model's output is compared to known ground truth, the arithmetic operations can be adjusted through gradient descent to gradually improve the model's ability to predict outcomes for previously unseen examples. Here we define and train deep convolutional neural networks (CNN) to examine a 500-length window of one of our Genome Counter objects and predict whether or not there is a CRISPR edit within that window.

A Genome Counter object has a depth of 10 features (four for the reference genome, six for the counts of the different alignment types), and therefore our CNN will take in input arrays of shape 10-by-500. Each input will be associated with a binary label marking whether or not the input contains an edit, and the network will be trained through supervised learning to predict whether or not an edit has occurred in previously unseen examples. We will evaluate our performance based on accuracy metrics of how the model makes its predictions on this validation set. We use the PyTorch [Paszke et al., 2019] deep learning library to train and evaluate our CNNs.

3.2.1. Data Iterator

The Data Iterator is responsible for dynamically generating both training and validation data for the CNN. The iterator will return batches of data with appropriate labels upon each request. It is up to

the user to define the number of training examples per batch as well as the ratio of positive to negative examples.

The size of a human genome precludes the use of a complete dataset for model training. Conversely, the relatively small number of positive examples of true CRISPR edits creates an extreme class imbalance for binary prediction. To best alleviate these issues, we randomly down-sample from the negative (non-edited) class and augment examples from the positive class. We maintain a 50/50 split between our training set and our validation set, only training on half of both our positive and negative examples. To maintain balance, we use a stratified split such that data from different sources, chromosomes, and sequence types are equally represented in each.

To select a negative input, we select a valid window from all possible unedited regions uniformly at random from all Genome Counter object files in our possession. To do so, we first randomly select one of 85 files containing fully sequenced chromosomes. We then traverse the file's corresponding block index (generated during the Pre-Processing step defined above) and select a random valid location of width 500, again uniformly at random. This location is then converted from our Genome Counter format to an PyTorch array.

Positive training inputs do not need to be sampled randomly as we know the location of the edit in each example. Our positive examples derive from two different types of sequencing. The data from Chakrabarti and colleagues [Chakrabarti et al., 2018] are amplicon sequences of known edit locations. There are 975 examples (prior to the train/validation split) of this type, and each has a larger number of reads at the edit location. Conversely, the rest of our data consists of only 11 whole exome or whole genome sequences of a full chromosome that include a single known edit location. In each case, we have the ability to place the exact edit location anywhere within the 500-length window we wish to pass to the CNN. Thus, for each positive training example, we randomly shift the edit location it within the window to augment our dataset as much as possible.

For each positive input, we first select a single example at random. Because of the large imbalance between the amplicon and the whole exome and genome sequencing positive examples, we select the whole sequences at a 30x inflated rate, though this still results in a larger proportion of amplicon data overall.

The length of the human genome means that there is a near infinite number of negative examples available for our training loop. Thus, we must instead place a limit on the number of negative examples that are included in a single passthrough of the data. In defining this limit, we also indirectly set the positive-to-negative class imbalance ratio. We must balance the trade-off of knowing that our prediction problem is imbalanced in the extreme with the challenge that class imbalance provides to training a machine learning model. In the end, we select a class ratio of 1:32 (positive to negative) and define a full training epoch as containing 65,536 negative examples and 2,048 positive examples.

3.2.2. Neural Network Architecture

We train a CNN to predict the presence of an edit in a 500-length window of a Genome Counter sequence. The convolutional neural network consists of three one-dimensional convolutional layers followed by two dense hidden layers that result in a single binary prediction. Each 1D convolutional

block in the network is followed by a max pooling layer of width 5 and a ReLU activation function. The kernel width of the convolutional layers was varied from 3 to 11 to produce an ensemble prediction (see the Results section). Each convolutional layer includes appropriate zero padding to balance the output shape. Each dense layer is also passed through a ReLU activation function. The full layout of the CNN is as follows:

1. a convolutional layer with input depth 10 and output depth 30,
2. a convolutional layer with input depth 30 and output depth 100,
3. a convolutional layer with input depth 100 and output depth 200,
4. a reshaping layer to flatten the remaining output,
5. a dense layer of width 100,
6. a dense layer of width 10, and
7. a final dense layer with a single output and a sigmoid activation function for binary prediction.

All layers include a bias term, and all are regularized using an L2 norm. The neural network operates over batches of 2D tensors supplied by the data iterator, each with a shape of 10-by-500 to match the PyTorch dimension ordering for 1D convolutions.

3.2.3. Tuning

Hyperparameters for the CNN were evaluated using the Sherpa tuning Python library [Hertel et al., 2020]. Due to the large number of training runs required, the number of unique inputs per epoch were reduced from 67,584 to 8,446 and the total number of epochs were reduced from 100 to 50, resulting in a 16-fold speed-up per run. We used Sherpa's Bayesian Optimization package with a total number of trials of 50 and an objective of minimizing the validation loss. The hyperparameters evaluated are listed in the following table.

Table 3. Hyperparameter ranges and values evaluated and chosen ensemble model.

Hyperparameter	Values	Chosen Value
1D-CNN kernel width	3, 5, 7, 9, 11	Ensemble of all values
Batch normalization	Yes or No	No
Dropout rate	Between 0 and 0.3	0
Learning rate	Between 0.00001 and 0.001	0.0001
L2 regularization	Between 0.00001 and 0.001	0.00001

The kernel width showed minimal sensitivity. Because it was our intention to use an ensemble of independent CNNs to make a final prediction, it made sense for each separate CNN to use a separate kernel size (the same size across all layers for each CNN). Thus, our final prediction model is an ensemble of five CNNs, trained on separate randomized datasets, each with a unique kernel width.

3.2.4. Training

We trained each of the five CNNs using the full dataset parameters defined in the Data Iterators subsection. The neural networks are built and trained using the PyTorch deep learning library. We use the Adam optimizer for backpropagation and use binary cross-entropy as our loss function. Each network is trained for 100 epochs (each epoch consisting of 65,536 negative examples and 2,048 positive examples) with a batch size of 256. We use a class weighting of 32 for the positive examples, corresponding to their inverse frequency when compared to the negative examples. All other training decisions are described in the previous hyperparameter tuning section.

3.3. Transformer Methods

In an effort to increase the number of orthogonal approaches applied to the edit detection task, we developed a transformer-based detection architecture with genome counter language encoding. While transformers have been applied to a host of data modalities, transformers have most widely been applied to sequences and language [Devlin et al., 2018], and learn through an attention mechanism that seeks to build contextual understanding. This initial embedding process is typically unsupervised, and results in a set of deep embeddings referred to as a language model. Language models are then exploited through the addition of subsequent, shallow neural networks to perform tasks such as binary classification, regression, and generation.

We leveraged a transformer model zoo curated by Huggingface [Wolf et al., 2019], which has quickly become a leading standard as both a collection of pre-trained language and classification models, as well as the maintainers of a Python library and API for interacting with official and community hosted models (https://www.huggingface.co/). While our entire workflow is designed to be extensible and accept a large variety of language and classification models to be combined in an ensemble framework, our methods and results presented here make use of a single state of the art transformer model (RoBERTa [Liu et al., 2019]), and subsequent binary classification head, shown in Figure 3-1. We chose RoBERTa due to its tokenizing approach and recent success associated with similar genomic classification tasks.

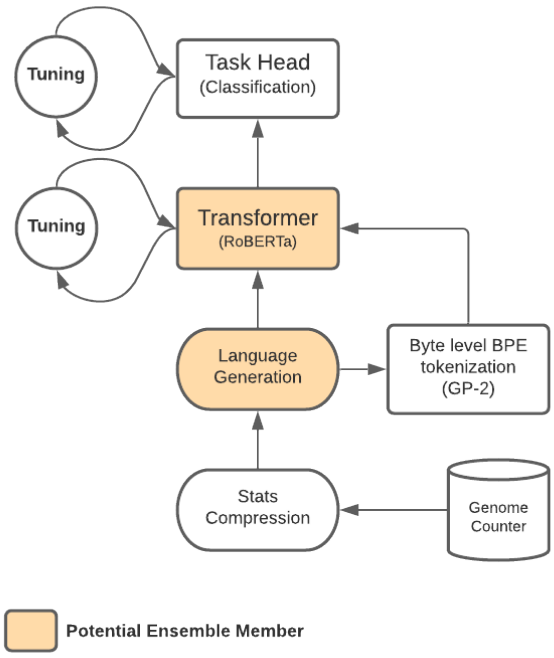

Figure 3-1. Transformer architecture for edit detection.

Figure 3-1 walks through the transformer workflow, and indicates algorithmic components that are fungible with respect to their selection from the model zoo. Given our desire to implement this transformer solution in concert with the CNN classification methods described above, we leveraged the workflow described in the Data Iterator subsection in the Deep Neural Network Models section to build training and validation data sets. Generally, the workflow involved the following steps.

1. Gather all the genome counter data to develop a language corpus, for use in training the language model.
2. Encode genome counter statistics using a statistics-to-text mapping function, to embed the genome counter statistics into the human reference genome base sequence.
3. Tokenize the new genome counter language corpus.
4. Train the language model.
5. Train the classification head.

Given that the genome counter outputs are composed of statistical representations of the likelihood of various events for each base, as well as a one-hot-encoded representation of the human reference genome base, we sought to transform these statistics into a continuous representation of the reference human genome and statistical likelihoods, as a text-based representation. The inspiration for this approach was based simply on the fact that transformers learn the context of words and substrings at the level of the sentence (characterizing subjects and predicates for instance), as well as at the level of the phrase and paragraph (deeper sequence context). We hoped that by encoding the statistics as additional characters in the reference genome, we would be able to anchor the attention mechanism of the transformer architecture to regions of the genome where edits are both possible and more likely to occur (depicted in Figure 3-2).

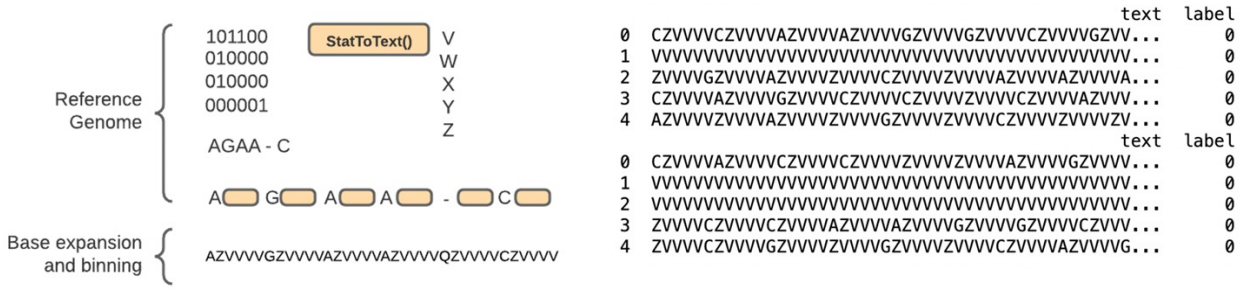

Figure 3-2. Data encoding for use in our transformer model.

Figure 3-2 shows a graphical representation of our genome base expansion and statistics encoding, using a simple statistic-to-text function to bin float ranges to unique characters. The substring tokenization then, hopefully, would learn that the first, second, third etc. character following the reference genome base mapped to matches, mismatches, etc. as generated by the genome counter. The resulting base expansion for the reference genome became the input to the language model. This then was also used to generate balanced train/test splits for training the classification head.

3.4. Anomaly Detection Algorithms

In our research we explored two flavors of anomaly detection algorithms. The first was based upon the edit classes defined by Chakrabarti and colleagues [Chakrabarti et al., 2018], which we refer to by name as the 'Chakrabarti-inspired Anomaly Detector'. In their work, these authors defined three classes of edits: precise (commonest indel frequency is greater than 0.5 but less than or equal to 1), medium (commonest indel frequency is greater than 0.25 but less than or equal to 0.5) and imprecise (commonest indel frequency is greater than 0 but less than or equal to 0.25). In our work, since we are trying to detect the edit site instead of starting with is location, we cannot use Chakrabarti and colleagues classifications exactly, but we can search for noise frequencies at each base position in the genome that fall in each of these three categories: 1) greater than or equal to 0.5 but less than 1; 2) greater than or equal to 0.25 but less than 0.5; or 3) greater than 0 but less than 0.25. We also added a fourth category (labeled class '0') where the noise frequency is exactly 1.

The second kind of anomaly detection we used was based upon the spiking adaptive median-filtering (AMF) algorithm [Verzi et al., 2018] modified for single dimension data, which we refer to by name as the 'Spiking AMF Anomaly Detector'. Detailed descriptions of both types of anomaly detection algorithms as well as a couple of variations on these algorithms are described in a soon to be published manuscript by the authors of this SAND report [Verzi et al., 2021].

4. DATA ACQUIRED AND DATA PROCESSING

In this section, we describe the datasets we have gathered, and how we have processed them for using in our machine learning algorithms. We start with the datasets next.

4.1. Data Sources

In our search for relevant datasets, we looked for data that contained successful edits for use in machine learning algorithms. These data include amplicon (also spelled as AMPLICON), whole exome sequence (WXS) and whole genome sequence (WGS) data. Amplicon data is detected using a target sequence, which is appropriate for genome edit experiments which successful edits as well as in verifying potential unintended off-target sites that can closely match the edit guide sequence (e.g., sgRNA). Whole exome sequence is used to detect all protein-coding regions in the DNA, which is the definition of exome. The breadth of coverage, across the genomic DNA, of WXS data is much greater than amplicon data, and the depth of coverage is much less. Protein-coding regions, or exons, in the human genome consist of about 1% of the base pairs. Whole genome sequencing attempts to detect the entire genome including all coding and non-coding regions in the DNA. Therefore, the breadth of WGS data is much greater than WXS data. The (file) size for WGS data can be considerably larger than either WXS or AMPLICON data.

The most common datasets available are AMPLICON sequences, and WGS datasets are the least common. It is likely that in application of our methods that either WXS or WGS data would be the most commonly available, thus we have focused on these datatypes in our research.

4.1.1. Cho Dataset

Cho and colleagues [Cho et al., 2014] provide data for two edit target sites, CCR5 in chromosome 3 and C4BPB in chromosome 1, as well as several off-target sites throughout these and other chromosomes, varying by 1 to 5 nucleotides in the CRISPR-Cas9 sgRNAs (consisting of a 20-base guide sequence and a proto-adjacent motif, also called a PAM, of 3 bases) used. Their datasets are provided in both AMPLICON and WXS formats. The AMPLICON data show a clear edit signature, allowing us to design and train our algorithms with visualizable effects, and the WXS data provide excellent test platform for our algorithms' performances. One disadvantage of the Cho dataset is that the authors conducted many experiments spread across 11 sequence files (7 AMPLICON and 4 WXS), without explicit use of barcoding. Thus, we had to mine their data to see which experiments were contained in which files, see Figure 4-1.

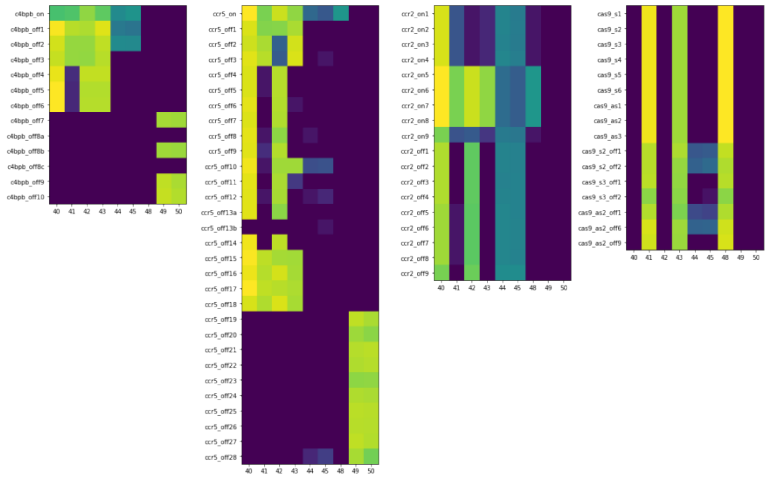

Figure 4-1. Heatmap showing coverage of data for experiments (y-axis) across 9 of 11 data files (x-axis) for 4 sets of experiments (at C4BPB, CCR5, CCR2 and Cas9 nickase targets).

4.1.2. Lin Dataset

The second data set we gathered came from the Doudna Lab, Lin and colleagues (Lin et al., 2014), provides examples of homology directed repair (HDR) edits at two target sites (EMX1 and DRYK1). Unfortunately, although barcodes were used, we could not decipher the complete barcodes for each experiment in each file. Also, this data was provided only in AMPLICON format, making it less desirable for training our algorithms (due to the fact that targeted reads make it easier for machine learning algorithms to detect the edits) and not appropriate for testing either. Thus, we were not able to use this dataset up to this point.

4.1.3. Wang Dataset

Wang and colleagues [Wang, et al., 2018] have data for 90 different target edit sites across 1144 samples. Unfortunately, all the data is collected using only AMPLICON sequencing, which makes it hard to use for either training or testing. Thus, we were not able to use this dataset up to this point either.

4.1.4. Guo Dataset

Guo and colleagues [Guo, et al., 2018] experimented with mice and human cells in their CRISPR-Cas9 genome edit experiments. The mouse data includes both first and second generation (parents and children), which is beyond the scope of our current research, but would be very interesting to return to in future work. These authors also have human, and overall they experimented upon 71 different target sites. Unfortunately, all their data consists of AMPLICON reads. When working with this dataset, we determined that we could process their AMPLICON data by removing duplicate reads, which are prevalent in such data. In this way, we proceeded to use all further AMPLICON data that we gathered, and it gives a path to return to use previously collected AMPLICON data in the future.

4.1.5. Veres Dataset

Veres and colleagues [Veres, et al., 2014] collected whole genome sequence (WGS) reads for 2 target edit sites (SORT1 and LINC00116) using CRISPR-Cas9 as well as with the Talen Cas. Although WGS data is very large, this data provides excellent samples for our learning algorithms.

4.1.6. Church (Yang) Dataset

Another WGS dataset from the Church Lab comes from Yang and colleagues [Yang, et al., 2014]. This data covers a single target edit site (TAZ) using CRISPR-Cas9. In this report, this is referred to as the Church dataset.

4.1.7. Chakrabarti Dataset

Chakrabarti and colleagues [Chakrabarti et al., 2019] conducted an extensive study to provide edit target efficiency measures for many different CRISPR-Cas9 sgRNA guide sequences. Their research is very close to our research, except that these authors are trying to measure edit efficacy as opposed to detecting edits, which is the goal of our research. The Chakrabarti dataset provides AMPLICON data for more than 1,000 target edit sites across 450 genes. Since, at this point, we had determined a solution for utilizing AMPLICON data (see the Guo dataset described previously), we were able to use many Chakrabarti samples. Note that Chakrabarti also included many samples from the van Overbeek dataset, described next.

4.1.8. van Overbeek Dataset

van Overbeek and colleagues [van Overbeek et al., 2016] research various metrics for measuring CRISPR-Cas9 edit efficiency, which was later extended by Chakrabarti and colleagues (described previously). The van Overbeek dataset includes 223 target edit sites in AMPLICON format. These authors also studied microhomology mediated end joining (MMEJ) as opposed to non-homologous end joining (NHEJ), which although it is beyond the scope of our current research would provide interesting future work.

4.1.9. Chuai DeepCRISPR Model

Chuai and colleagues [Chuai et al., 2018] also research edit efficacy prediction and optimization using deep learning, via their DeepCRISPR model, which provided inspiration for our work. These authors did not use sequence reads for prediction, but rather they used many (more than 15,000) sgRNAs, that were used in existing experiments, such as Cho and colleagues [Cho et al., 2014] and van Overbeek and colleagues [van Overbeek et al., 2016] as well as many others, but also included many synthetically generated variants from these existing sgRNAs.

4.2. Data Processing Pipeline

In this section, we describe our data processing pipeline, from data acquisition through quality trimming, alignment, deduplication and sorting/indexing, for use in our genome noise counter (described in the next section).

Data sets (SRR#s) were downloaded from the sequence read archive (SRA) [https://www.ncbi.nlm.nih.gov] as fastq files. Trimmomatic [Bolger et al., 2014] was used to triage

the data for poor quality reads. The following conditions were used: "LEADING:3 TRAILING:3 SLIDINGWINDOW:4:30 MINLEN:36". In short, this means that reads had to be at least 36 bp long, and sliding windows of 4bp along the read were checked for Q-score > 30. If the ends of the read have regions of a low Q-score, they were trimmed. If the reads had an average Q-score of less than 30, they were eliminated. BWA-MEM [Li, 2013] was used (standard mapping settings, and using -M to mark short reads as secondary) to align data sets to hg38. Following this, samtools [Li et al., 2009] was used for conversion in BAM format, and for sorting and indexing files. Samtools was also used to mark and remove duplicate reads to eliminate bias from PCR duplication.

4.3. Genome Noise Counter

Early on in our research our goal was to create a genome counter that could catalog the different types of deviations from normal in our alignment, as well as record the original reference sequence. Specifically, we wanted to follow the instances of mutations, insertions and deletions (indels), and soft and hard clips (indicating partial matches). In order to use this information found in the aligned BAM files to create features to use in our machine learning algorithms, we designed our own lightweight (sparse matrix) implementation of genomic noise counting, including matches, mismatches, insertions, deletions and clips.

The Genome Counter is based on a dictionary of keys sparse matrix implementation. Rather than store the entire matrix, the object instead only explicitly stores locations that have non-zero values. These locations are divided into blocks of a fixed width, and each block is stored in a hash table (also referred to as a dictionary in Python) for rapid access. The Genome Counter supports three main operations, listed below.

1. **Initialization** – The counter object is initialized with a fixed length, but no memory is allocated to store the actual array. Instead, an empty dictionary is created that will hold references to filled in blocks of non-zero data. The object also has a set block width and data type. Our default block width was length 32, and the data type would be either 8-bit or 32-bit unsigned integers, depending on the amount of data that was expected.
2. **Ingestion** – Once initialized, the counter object will accept new data. The object is designed to act as a two-dimensional array, and values can be set by indexing into specific locations. When a value is set, the object first checks to see if a block has already been allocated at that position. If not, a new block is created, and values are updated accordingly. It is possible for new data to span multiple block widths, in which case multiple blocks are created and added to the main lookup dictionary. The counter object maintains knowledge of its own length, and it will reject values that fall outside of its boundaries.
3. **Reference** – Once created, it is a simple matter to look up values stored within the object. A value stored at a specific location must reside within the block responsible for covering that location. The block index can be obtained by integer dividing the true index by the counter's block width. If the block index is a valid key of the lookup dictionary, the value can be retrieved from that block. If the index is not a valid key, it can be assumed that the corresponding region of the counter consists of only zeros. It is possible to reference a range of values and that range can span multiple blocks. If so, each block can be dealt with separately, after which all the results are concatenated into a single regular array representation.

Because we are concerned with a specific use case of a genome array consisting of a very long first dimension and relatively short second dimension, the sparse array only implements block indexing across the first array. For our purposes, we use a specific ordering of the second dimension to store values corresponding to the specific characteristics of the sequences we are interested in. Namely, the ordering of the second dimension are as follows:

1. the reference genome – at each location a numerical value is stored corresponding to the reference nucleotide at that location: 0 – empty or unknown, 1 – A, 2 – C, 3 – T, 4 – G, and 5 – N (a variable location),
2. the number of matches at that location,
3. the number of insertions,
4. the number of deletions,
5. the number of mismatches,
6. soft clips, and
7. hard clips.

This encoding results in a sparse two-dimensional array where the first dimension is the length of the chromosome, and the second dimension is length seven.

In all cases, the Genome Counter object is handled as a regular array once initialized. All sparse matrix special operations are performed behind the scenes and the user does not have to perform any special operations. The effect of the sparse implementation can be very pronounced for genome data. In many cases, the sparse array implementation can compress memory usage requirements by over 100-fold. This is an essential prerequisite when dealing with up to hundreds of genome sequence files, each with a file size of 5 GB or more. This sparse matrix data type is leveraged in our anomaly detection methods.

4.4. Measure Alignment

We use the Genome Counter object to store the results of the larger genome sequence alignments from BWA-MEM stored as BAM files. To do so, we traverse the entire alignment and transfer its information into the counter. We use the Python library pysam (https://github.com/pysam-developers/pysam, a Python wrapper around the samtools application) to iterate over every individual aligned read within any given BAM file. For each read, we examine its full sequence and given its location, we increment the count of each match, insertion, deletion, mismatch, soft clip, and hard clip in the corresponding location in the counter object. Finally, we use the MD tag of the read to reconstruct the reference genome nucleotide sequence at the location of the read. We record the reference genome in the counter as well, as described in the previous section.

4.5. Genome Pre-Processing

Our machine learning pipeline operates over sets of individual chromosomes, yet a full chromosome consists mostly of non-coding regions. Because we are specifically interested in detecting edits of functional regions, we limit our search to known gene coding locations. Specifically, we want to apply our algorithms to only the exons within a chromosome. Because exons make up such a small proportion of the full genome, it is efficient to pre-process each of our Genome Counter objects so that we can limit the regions our algorithms must look.

For each counter object, we define a set of criteria for valid regions, and then traverse the entire object to generate a complementary index file that contains the indices of these valid regions. There are three main criteria for inclusion, listed below.

1. **Minimum Coverage**: We require each location to have a minimum number of aligned reads to give us confidence in having a representative sample. As per our data processing pipeline, these reads have already been clipped based on their PHRED scores. Our default setting for minimum coverage is five reads, though this value can be set to any positive value.
2. **Gene Coding Region**: All valid locations must exist within a valid gene coding exon. We ignore introns and other non-coding regions to better focus on edit locations that could have functional effect on protein creation.
3. **Maximum Gap**: Many regions of the aligned genome have small gaps of invalid region surrounded by large contiguous areas of valid coverage. In these cases, it makes sense not to break these regions up at each gap. Instead, the gaps are allowed to exist within the rest of the valid data, encoded as all zeros in the Genome Counter object. The maximum gap can be customized during this pre-processing step. For our purposes, we chose a maximum gap size of 250, which corresponds to half the width of the 500-length window we use for our analyses.

We fully traverse every counter object and track the locations of valid windows. These locations are stored in a separate block index file to allow for rapid indexing into valid locations without having to traverse an entire chromosome each time we want to find a specific valid window.

5. RESULTS

In this section, we document our genome edit detection results using decision tree learners (random forest and XGBoost), deep neural network models (deep CNNs and transformers) and anomaly detection methods. Note that in some cases, these methods were applied using different features or different views of the same data (i.e., different window lengths) or using different parameterizations, as described above, but we also compare them to each other using the same datasets.

5.1. Tree-based Methods

For each edited region, we labeled the 100 bp (base positions) around the edit site as an edited region (or positive), since our smallest window size possible is 50 bp. Everything else in each file was labeled as non-edited or negative. Note that number of neighbors for SMOTE was set at 10 for these initial correlation calculations, but would later be tuned to optimize models. We first wanted to determine the correlation between our features and these labels, so we used Pearson correlation to determine the average correlation across 10 iterations of sampling (Figure 5-1). Specifically, for positive windows, we used all the negative data and single positive data sets one at a time to determine correlations, and for negative data sets, we changed the labels of individual negative data sets to 'positive' one at a time to determine correlations with features.

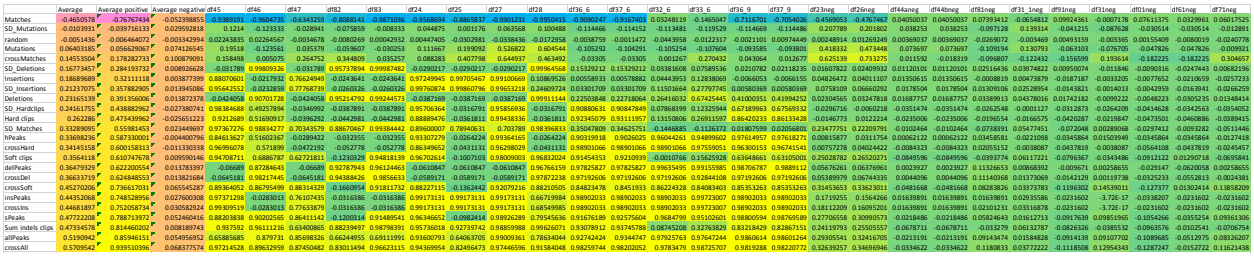

Figure 5-1. Average (Pearson) correlation across 10 randomly generated samples for features (rows) versus edit or non-edit labels (columns) used in training.

We find that while the negative data on average does not correlate with any specific feature, that is not the case with the positive data. Some features, such as fraction of matches, number of peaks of total signal, and how many times the total signal crosses a threshold (set at 50% for the purpose of this initial correlation), seem uniformly correlated with edited regions across the samples we tested. However, of note was the fact that most of the other features show varied correlation with edited regions depending on the sample. This suggests that simple linear relationships between features and edit sites may not be sufficient to capture the entire breadth of edited regions, prompting us to turn to tree-based methods. Before that, however, we wanted to verify that our SMOTE transformation was not changing the relationship of the features to the classification, so we checked the correlation of the entire data set with the various features before and after SMOTE (Figure 5-2 and Figure 5-3). We find that overall, the relationships between the features and the labels are maintained even after transformation with SMOTE, suggesting that this is a valid way to augment our data for machine learning.

	Pre-smote	Post-smote
Matches	-0.334572	-0.570796
random	-0.005705	-0.004884
crossMatches	0.004259	-0.014383
delPeaks	0.050335	0.181153
crossSoft	0.057115	0.191615
crossDel	0.058821	0.229701
insPeaks	0.065569	0.325902
crossIns	0.065626	0.319157
allPeaks	0.078111	0.256226
sPeaks	0.081481	0.287515
crossAll	0.140162	0.424111
Soft clips	0.197254	0.469467
SD_Softclips	0.211422	0.539551
SD_Matches	0.232588	0.573208
SD_Insertions	0.250619	0.46122
hPeaks	0.252463	0.479862
Insertions	0.267338	0.451866
crossHard	0.27584	0.494565
SD_Deletions	0.293409	0.424185
Deletions	0.323052	0.371885
Sum indels cl	0.357617	0.57898
SD_Hardclips	0.389561	0.509483
Hard clips	0.400161	0.508797

Figure 5-2. Feature importance values both with and without SMOTE.

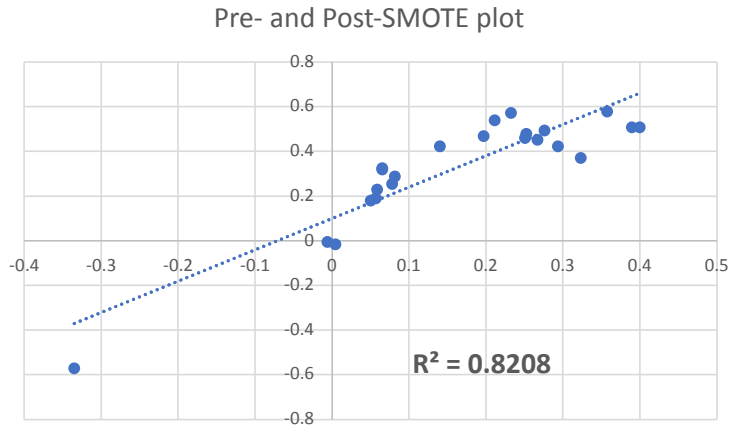

Figure 5-3. Correlation plot for Pearson coefficients both with and without SMOTE.

Initially we proceeded to train both a RF and XGB (short for XGBoost) model with a small subset of the data to directly compare the performance of the two. We chose to use two data sets as tests: Cho 45 and Chakrabarti 36_6. For each, we trained both a RF and XGB, leaving out the test set and training on the rest of the data. Hyperparameters were optimized as outlined in the methods section

for ten iterations. Each iteration had a 10-fold cross validation, and the average F1 score from these cross-validations for Cho 45 is shown in Figure 5-4. The average F1 scores suggested that the hyperparameter tuning did not significantly change the performance of the models, and that in any case, by 10 iterations the maximum score obtained with the acquisition function had converged successfully. As a side note, we could increase the kappa value of the acquisition function to increase exploration into unknown hyperparameter space to see whether that changes the rate of convergence of the average F1 scores, but since the scores are quite high to begin with, we did not deem this necessary.

CV	XGB	RF
1	0.9756	0.9982
2	0.9774	0.9982
3	0.9737	0.9826
4	0.9765	0.9982
5	0.9759	0.9982
6	0.9765	0.9466
7	0.9744	0.9982
8	0.9755	0.8698
9	0.9756	0.997
10	0.9788	0.9436

Figure 5-4. Average F1 score from 10-fold cross-validation study for Cho 45 dataset (left out for testing).

We then ran the trained RF and XGB models on their respective test sets (see Figure 5-5 for results), and we observed a number of things. First, the two data sets showed very different results in terms of precision and recall, with the Chakrabarti 36_6 edits having better recall, and the data set on the whole having much better precision. This confirmed the conclusions from our correlation analysis that the data sets were likely to behave quite differently, and that we likely needed to further fine tune the features to capture different types of edits. Second, we see that the XGB models outperform the RF models significantly. While it is known that XGB is more prone to overfitting than RF [Quora.com, 2015], we think this is unlikely in this case because each positive data set is very different, and our data is inherently noisy, making it unlikely that the exact patterns shown in the test data set exist in the training set (this will be further demonstrated at a later point when we visualize the data using dimensionality reduction techniques). Also, as has been observed by others in the literature, limiting the iterations of Bayesian Optimization of hyperparameters can address the issue of overfitting in XGB, which is one reason we keep the iterations at no more than 10 [Makarova et al., 2021]. We therefore chose to proceed with just the XGB model, and examine whether we could improve its performance by revisiting the feature space.

Test set	Algorithm	Accuracty	Precision	Recall	Specificity	F1
Cho45	RF	0.995	0.5	0.1	0.9995	0.167
Cho45	XGB	0.995	0.444	0.4	0.998	0.421
Chak36_6	RF	0.97	0.962	0.51	0.999	0.667
Chak36_6	XGB	0.985	0.93	1	0.981	0.964

Figure 5-5. Test performance of RF and XGB models for Cho 45 and Chak 36_6 datasets (left out for testing).

In order to better characterize the relationships and differences between the various data sets, we decided to use dimensionality reduction techniques to visualize and categorize the data. First, we performed a principal component analysis (PCA) [Pearson, 1901] using the features described above (Methods section). We performed the analysis 5 times on positive and negative data sets (chosen stochastically as described in the Methods), and show the results with a standard deviation of these 5 trials (Figure 5-6). Over 97% of the variation in the data is explained by the top 10 PCs, and almost 60% of the variation is explained by the first two PCs. We looked closer at these first two PCs to determine which features contributed most to each of these axes. We see an almost inverse relationship between the contributions in PC1 and PC2, suggesting that they are truly representing two categories of behavior involving different features, further highlighting the complexity of the relationships between features and class.

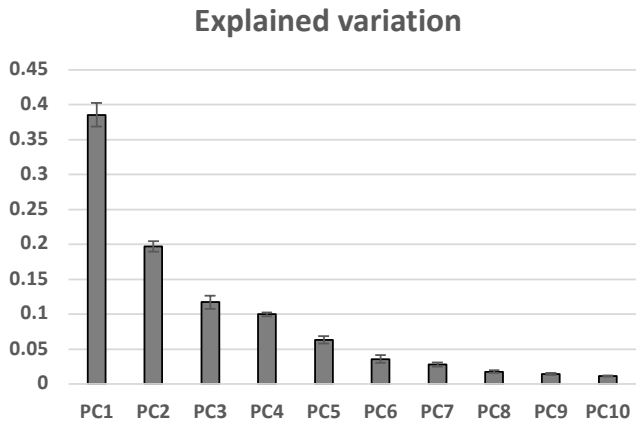

Figure 5-6. Principal components (x-axis) plotted with variance explained (y-axis).

We fed the top 10 PCs into t-distributed Stochastic Neighbor Embedding (tSNE) so we could visualize the data sets that we were working with. For each data set we plotted 20 random samples (see Methods for how the positive and negative data was sampled) (Figure 5-7). We tried shifting the perplexity parameter of tSNE from 5 – 50 and found that a perplexity of 30 provided the best resolution of our data (not shown). We see that the negative data (pastel colors) tends to cluster in two areas: one is a circular compact cluster, and the second is a longer spread-out cluster. On the other hand, the positive data sets are spread out pretty widely on the 2-dimensional tSNE projection, with individual experiments clustering together, but each data set almost in its own space.

This matches our original correlation analysis which suggested that every data set had some variation, and it emphasizes the need for an unbiased data augmentation method such as SMOTE to sample the positive data space.

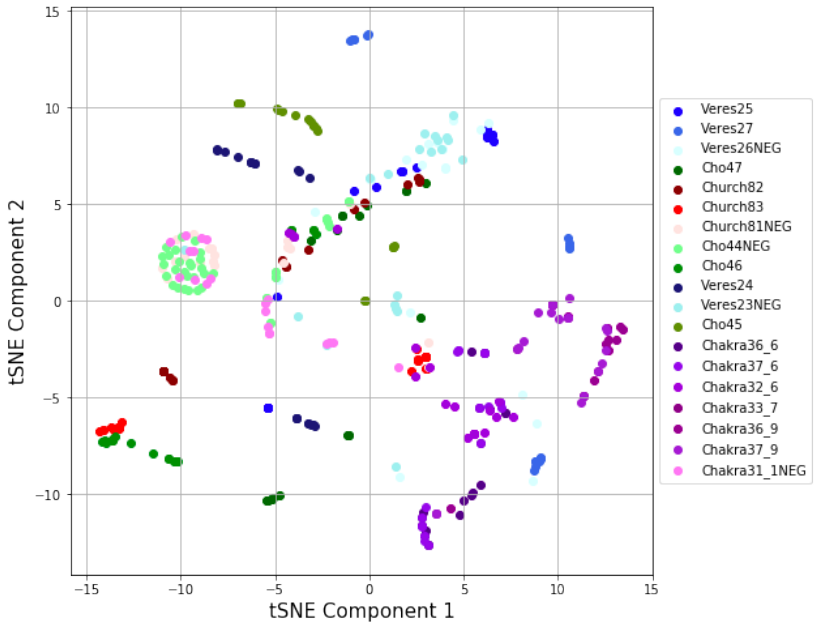

Figure 5-7. Two-dimensional tSNE visualization of first two principal components for both positive (edit) and negative (non-edit) datasets.

We next used DBSCAN [Ester et al., 1996] to cluster the 2D tSNE-projected samples, and required for each cluster to have a minimum of 4 points (failing which the points were categorized as being in no cluster). We adjusted the epsilon value (i.e., radius of search) until most of the negative samples fell in a single cluster, which ended up being an epsilon value of 10 (Figure 5-8). We noticed that while most of the negatives fell within one cluster (cluster 10) some were found outside this cluster, suggesting that based on the features we were using, these regions that were behaving as positives would were being classified as negative (we are calling them 'erratic negatives'). This could account for our algorithms being trained to think that anything similar to those erratic negatives were also negatives, thereby reducing the overall recall of our model. To remedy this issue, we sought to identify which features were responsible for these erratic negatives. We looked at the distribution of values of each feature associated with genuine negative ('good') versus erratic negatives ('bad), and we found that three features stood out as having significantly different values (ANOVA $p<0.01$) between the good and bad categories (Figure 5-9). These were mutations, standard deviation of mutations and cross matches (i.e., how many times the signal for matches crosses a given threshold). This suggested to us that these three features were misleading our algorithms in terms of what a negative sample should look like, and therefore should be removed, which is what we did moving forward.

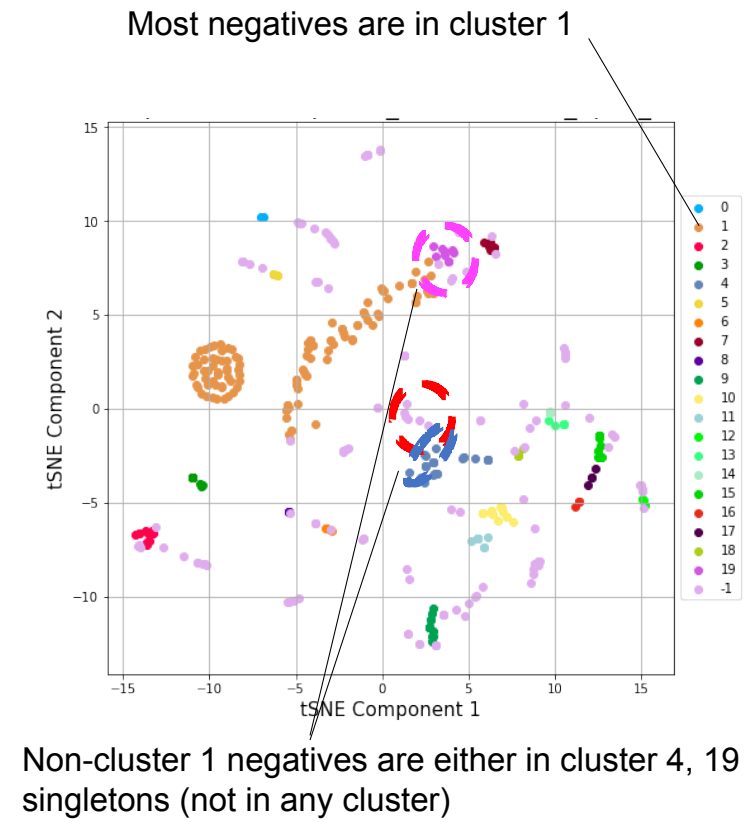

Figure 5-8. DBSCAN clustering visualized using tSNE for first two principal components with both positive (edit) and negative (non-edit) datasets.

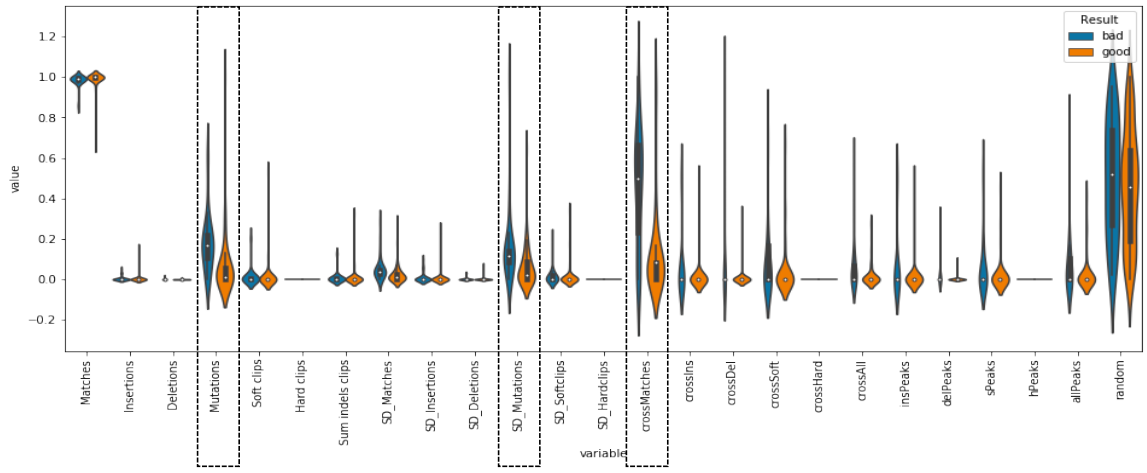

Figure 5-9. Distribution of feature values using ANOVA for both good (useful in classification) and bad (not useful in classification) features.

Having identified the pertinent features for classification of edits using our models, we then proceeded to tune multiple XGB models, each one with one positive data set left out (for cross-

validation). In order to diversify the negative data sets, we added a subset of exome data from the 1000 genomes project [The 1000 Genomes Project Consortium, 2015]. We recorded various metrics from the models (accuracy, precision, recall, specificity and F1 score) for each left-out data set with the corresponding model (Figure 5-10). We observe that the majority of the tests run on the various models show very high scores across the board. In a couple of the data sets (Veres 25 and Church 82), we fail to identify any of the edited windows (something that we will investigate in a later section when we examine entire chromosomes), and in one of the Chakrabarti data sets (32_6), we only identify about half the edited windows (which would still result in an identified edit, however). Overall, the average accuracy of our models is 91%, with recall at 77% (meaning 77% of edited windows are successfully identified) and specificity at >99% (suggesting that very few negative regions are identified as edited).

Test set	Accuracy	Precision	Recall	Specificity	f1
Church83	1	1	1	1	1
Cho45	1	1	1	1	1
Veres25	0.6659	0	0	0.9989	0
Veres24	1	1	1	1	1
Cho46	1	1	1	1	1
Cho47	0.8993	0.9956	0.7517	0.9978	0.8567
Church82	0.5	0	0	1	0
Veres27	0.98518519	0.98863636	0.96666667	0.99444444	0.97752809
Veres28	1	1	1	1	1
Chak36_6	0.99667774	0.99667774	1	0.99888889	0.99833611
Chak37_6	1	1	1	1	1
Chak32_6	0.88666667	0.98809524	0.55333333	0.99777778	0.70940171
Chak33_6	1	1	1	1	1
	Accuracy	Precision	Recall	Specificity	f1
Average	0.9179792	0.84376995	0.79013077	0.99906239	0.81092045

Figure 5-10. Accuracy, precision, recall, specificity and F1 score for leave-one-out study.

We compared the feature importance derived from each individual model. There is one feature that stands out as having the most significant importance, and that is the sum of indels and clips. The error bar represents the standard deviation of these importance values across our XGB models. This intuitively makes sense, since the majority of CRISPR edits generated by NHEJ cause indels and/or larger errors that result in reads being clipped during alignment [Song et al., 2021]. We notice however, that other features (including for example, the number of peaks of signal in deletions or soft clips, amongst others) also play a role in the XGB models, and the large error bars actually highlight the importance of keeping all these features – each individual model weights the features slightly differently that an ensemble solution of all the XGB models might be the optimal solution. Further, even though the sum of insertions and deletions looks like it might alone be sufficient to classify these samples, we can revisit our correlation analysis to see that this is in fact not the case (Figure 5-2), suggesting again that even though the importance of other features is not as high, they are necessary for successful classification by the XGB models.

While our current method for selecting positive and negative regions yielded promising results with our XGB models, in reality the sequencing data we would need to analyze for edits is likely to be far more unbalanced, with the vast majority of a genome being edit-free. We therefore applied our model to a so-called chromosome walk, where we analyzed an entire chromosome windows (the size of which is determined by the optimization of the corresponding model – for example, if we were walking a chromosome from the Cho 45 data, we would use the model that was trained with Cho 45 left out, and we would use the window size that had been determined by that Bayesian Optimizer). We chose 6 data sets from 3 different publications for analysis of whole chromosomes, and the results are shown in Figure 5-11. For the Cho and Church data sets, we see 0.1-0.4% of the windows being flagged by our model as potentially edited, while for the two Veres data sets, it's an order of magnitude higher (4-6%). Of note, this is not a percentage of the whole chromosome, but rather a percentage of the windows with sufficient depth of data (i.e., coverage of at least 5 reads across the entire window). This is consistent with the observation that the Veres data sets have a higher level of baseline mutation compared with the other data sets (this will be further discussed shortly). We also see that for the two data sets with zero recall from our previous test, Church 82 and Veres 25 (Figure 5-10), we are still unable to find the edits, as expected, whereas for all other data sets, the actual edited region is one of the windows that gets flagged, though still a small fraction of the total number of flagged windows. One potential work-around would be to look for contiguously flagged windows to determine the likelihood of an edit. The disadvantage of this approach is that it would disfavor precise edits that only fall within a single window, so we chose not to pursue it for now (although it could certainly be an area of future research).

Paper	Data set	Chromosom	Fraction of windows that are positive	Fraction of positives that are real	Edit found?
Cho	45	3	0.0022	0.0078947	YES
	46	3	0.0032	0.0057034	YES
Church	82	X	0.0018	0.0000000	NO
	83	X	0.0034	0.0063291	YES
Veres	24	1	0.0573	0.0000053	YES
	25	1	0.0466	0.0000000	NO

Figure 5-11. Results from chromosome walk for each of 6 datasets (each left out separately).

To further examine the chromosome walk data, we visualized the data using tSNE to reduce the dimensionality of the data and project it in 2 dimensions (Figure 5-12). We noticed some recurring patterns in the 2D projection to follow-up on. Specifically, we see three distinct shapes/conformations that the data adopt and fall into. All six data sets have a region that is linear or curvilinear, and another region that is scattered. The Cho (A, B) and Church (E, F) data sets also have a third set of points that are in a circular cluster.

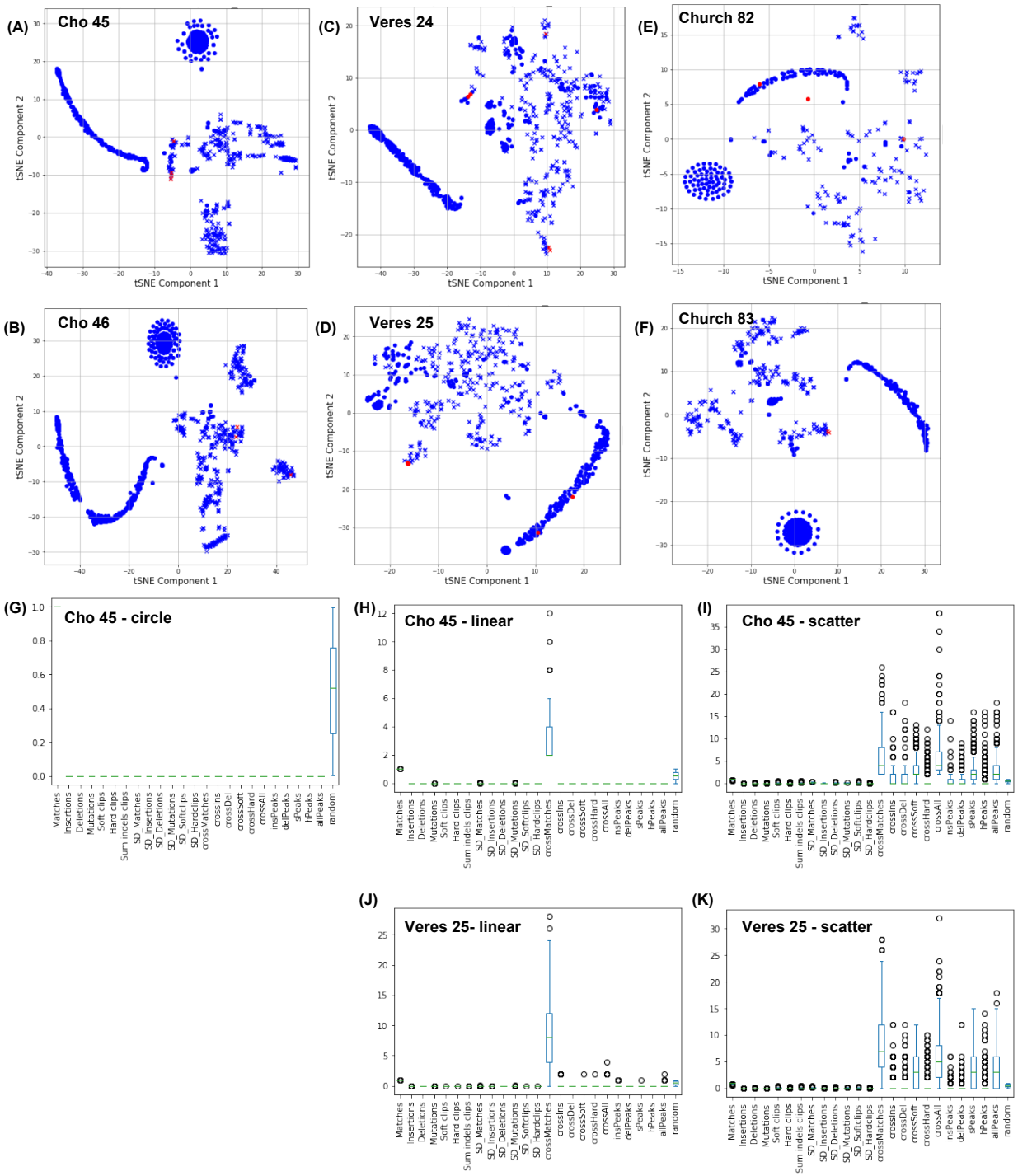

Figure 5-12. (A)-(F) DBSCAN clustering visualized using tSNE for first two principal components across 6 datasets in single chromosome walk and (G)-(K) scatter plots of ranges of feature values across the same chromosome.

We decided to separate the data into groups based on these three conformations, and look at the various features that we use for our ML models (for illustration purposes, we focused on two data sets). In particular, we looked at the range of values for each variable to see which of the features

might be contributing to these groupings, and whether that could give us insight as to why we may not be identifying certain edits. The Cho 45 data set shows us that the sequencing data that make up the points in the circle are perfectly matched to the reference (only the random variable varies, as expected). On the other hand, from both the Veres and Cho data sets, we see that the points in the linear/curvilinear section (H and J) show more baseline variation among a lot of the variables, with a specific spike in the 'crossmatches' variable – suggesting that the fraction that maps is more variable across these regions. Finally, the points in the scatter (I and K) show a lot more variability among many of the features. This region is where most of the true edits and edits called by the XGB models lie. Given that there is not a lot differentiating the false positives from the true positives in this area, we conclude that as they stand, our models could not reasonably be expected to differentiate between points within this scatter. Our models do however reduce the genomic space significantly, ranging anywhere from 0.1 – 6 % of eligible windows (i.e., windows with a certain amount of coverage). Our approach may not be ideally suited for data sets that have a higher level of baseline mutation, but even in those cases, we can still identify edited regions in a subset of cases.

5.2. Convolutional Neural Networks

Five convolutional neural networks wither kernel sizes of 3, 5, 7, 9, and 11, were trained on a full set of data provided by the Data Iterator over 100 epochs. The networks were trained on half of the available positive examples of both the amplicon and whole sequence data and then evaluated on the other half. The networks were then evaluated individually and as a single ensemble.

5.2.1. Training Loss

Training versus testing loss for the five convolutional neural networks are shown in Figure 5-13.

Figure 5-13. Loss plots of the individual CNN training runs. Training set loss shows a consistent decrease and validation sets all progress to a minimum around epoch 80-90. The noisy results of the validation training curves are likely due to the non-deterministic.

Each CNN model was trained on data provided by the Data Iterator, using the Adam optimizer and binary cross-entropy loss to attempt to minimize loss on the held-out validation set. While training losses show a relatively smooth decrease, the validation losses have numerous outliers as they also gradually decrease during training. Validation loss seems to bottom out around epoch 80, suggesting further training is unnecessary. Once complete, each model selects as its final version the configuration leading to the best validation performance (lowest loss value) over the course of training. The end result is five separate models each with a unique kernel size. The variance of the performance on the validation loss suggests that each individual CNN may provide somewhat noisy predictions on individual data, therefore we look to use all five in concert as a single ensemble predictor to minimize the effects of this random variation.

5.2.2. Individual Results

Each model in isolation performs very well. All five models demonstrate a very high Receiver Operator Characteristic Area Under the Curve (ROC AUC) as well as a high average precision given the class imbalance of the data. The results do show a consistent but small trend of improvement as the model's kernel width increases from 3 to 11, suggesting that the extra free model parameters are able to pick up and learn meaningful information (see Figure 5-14).

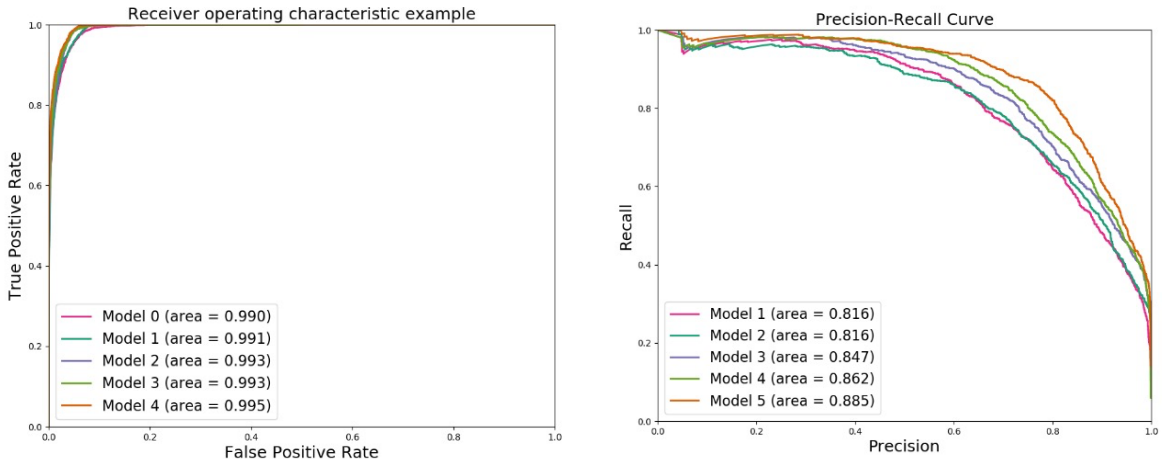

Figure 5-14. Receiver Operator Characteristic and Precision Recall Curves for each of the five individual CNN models. AUC scores range from 0.990 to 0.995 and average precision scores range from 0.816 to 0.885.

The AUC scores for Models 1 through 5 respectively are: 0.990, 0.991, 0.993, 0.993, and 0.995. The average precision scores for Models 1 through 5 are: 0.816, 0.816, 0.847, 0.862, and 0.885.

The models demonstrate different calibrations (not shown). Some models showing a strong capability to identify the negative class with high confidence, but struggle to clearly label the positive

class examples. Conversely other models are better able to predict the positive class with high confidence, but struggle more with the negative examples. These differences do not necessarily affect the final result as the final binary prediction is determined by the comparison to a chosen threshold value, not by the raw model outputs. The differences do suggest that the five models may be amenable to being combined into a single ensemble.

5.2.3. Ensemble Results

Combining all five individual models into a single ensemble provides a single aggregated prediction for each input. To create a single binary prediction, the raw scores of each network was averaged and this average was evaluated against a binary threshold of 0.9.

We can thus evaluate the performance of the ensemble as its own single model. The ensemble model was evaluated on the full validation set which consisted of a positive to negative class imbalance ratio of 32 to 1. The aggregate performance metrics are as follows: an accuracy of 0.987, precision of 0.862, recall of 0.688, and a F1 score of 0.765.

The confusion matrix breaks down the full classification ability of the ensemble predictor, revealing a true negative rate of 0.966, a true positive rate of 0.0209, a false positive rate of 0.00334 and a false negative rate of 0.00944 (note that negative examples exist at a frequency of 0.969 while positive examples exist at a rate of 0.03125 in our training and validation sets), shown in Figure 5-15.

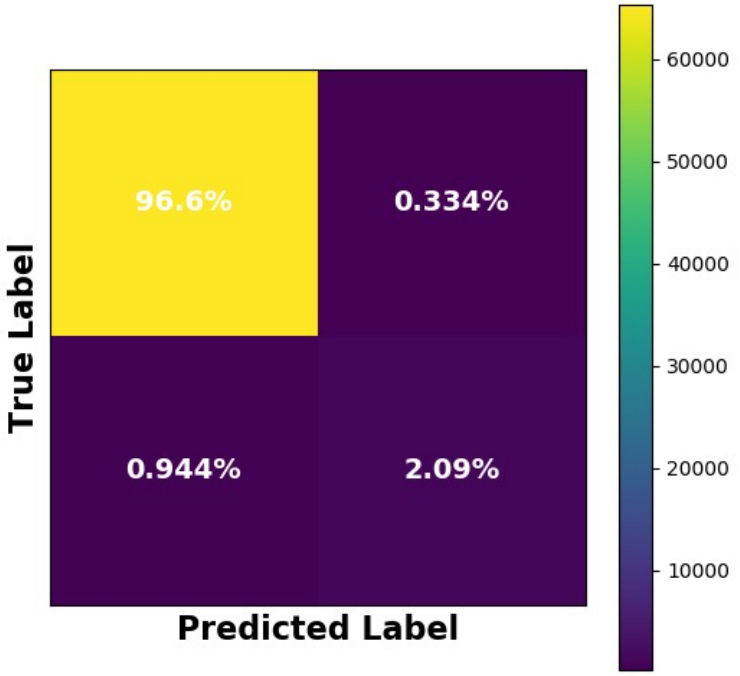

Figure 5-15. CNN Ensemble Confusion Matrix. The model has low rates of both false positive and false negatives, though these values are partially affected by our 32:1 negative to positive class ratio.

We further evaluate the performance of our ensemble predictor by examining its receiver operator characteristic (ROC) curve and its precision recall curve. Each curve evaluates the trade-offs the model offers between setting a low positive threshold for high recall versus a high threshold for high precision. An ideal result maximizes the area under the curve to as close to a value of one as possible. Here, we see an area under the ROC curve (AUC) of 0.994 and the area under the precision recall curve (average precision) of 0.872, shown in Figure 5-16.

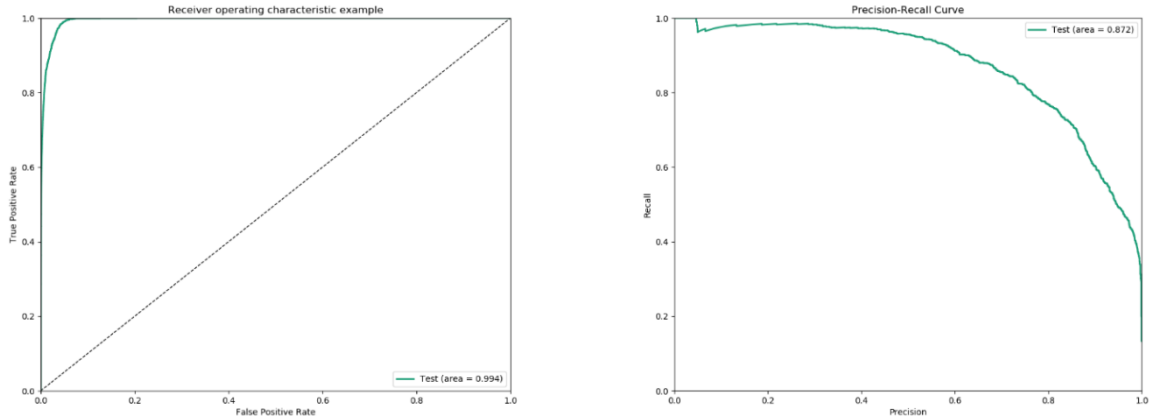

Figure 5-16. Receiver Operator Characteristic and Precision Recall Curves for the Ensemble Predictor. The ensemble model achieves an ROC area under the curve of 0.994 and an average precision of 0.872.

We then look at the ensemble's performance across each specific input type to better understand how it is making its predictions. We examine the results in the form of the distributions of scores applied to each class's validation examples, see Figure 5-17 and Figure 5-18. We further split the positive examples into those that come from whole sequencing and those that come from the Chakrabarti amplicon dataset, see Figure 5-19.

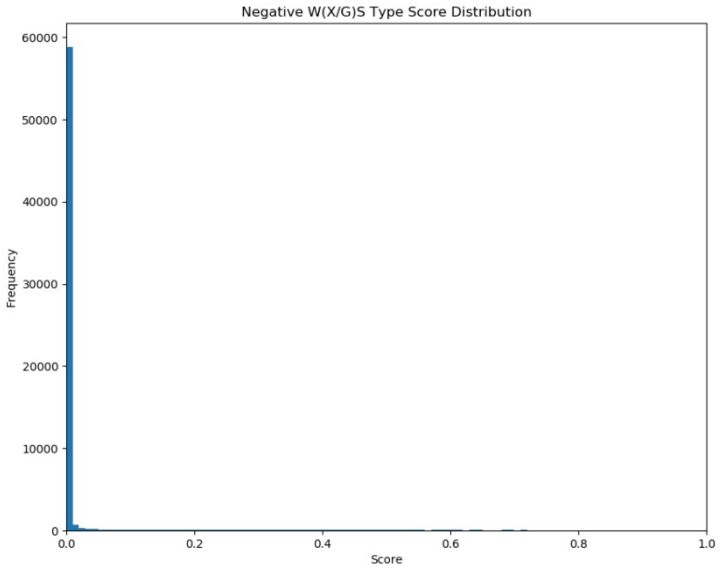

Figure 5-17. CNN ensemble score distribution for negative samples. An extremely majority of examples are properly given a minimal score, with a barely noticeable number given any score

above 0.05. This result is encouraging as a strong ability to reject false positives is necessary for such a large and imbalanced dataset.

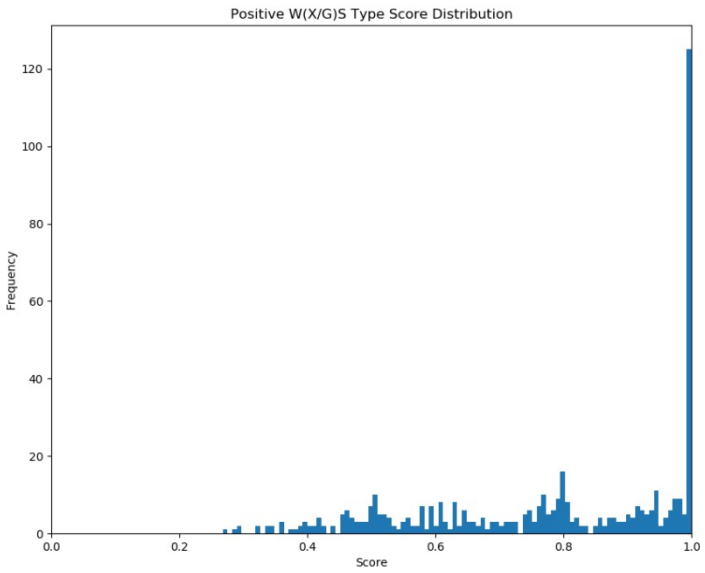

Figure 5-18. CNN score distribution for whole sequenced positive examples. The noisiness of this distribution suggests that the model has difficulty identifying the proper edit signatures of this type. Note that there are only 5 true examples in this dataset (augmented by moving them within the 500-length window). The low scores may suggest the ensemble model's inability to identify edits near the edges of the input window.

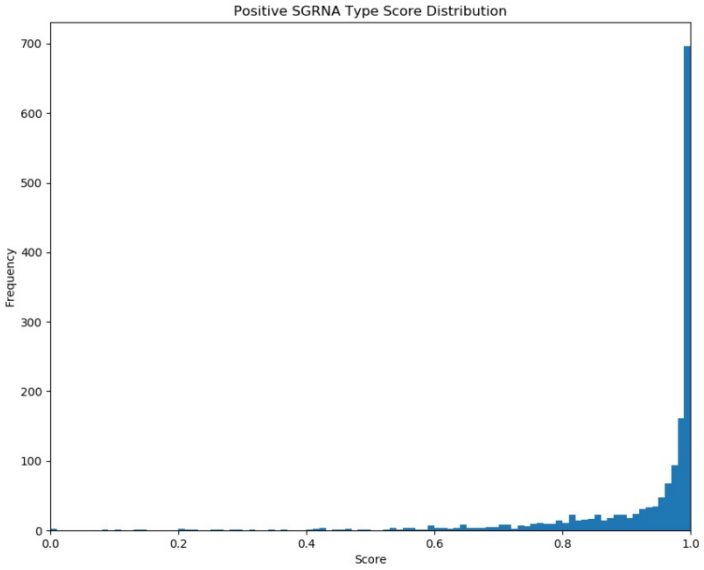

Figure 5-19. CNN score distribution for Chakrabarti amplicon positive examples. The ensemble model performs better on the positive amplicon data than the whole sequence data, which may likely be due to the significantly large number of unique training examples of this class. Though noisier than the negative dataset, an extreme number of positive examples rest above a threshold of 0.5, suggesting very strong predictive performance of this type.

The scores for the negative examples are extremely left-skewed, showing a strong ability of the ensemble model to correctly identify non-edit locations appropriately. This corroborates the low false positive rate seen in the confusion matrix. Both positive classes show a higher level of disagreement, though both appropriately demonstrate a strong right-skewed distribution. Because the negative class examples are so strongly left-skewed, it makes sense to set a very low binary prediction threshold to maximize overall prediction performance and further minimize false negatives.

We can also combine the five individual models as voting members (rather than averaging). To do so, we calibrate the sensitivity of each model to provide a consistent recall value. For our purposes, we choose a binary threshold for each model such that the expected recall is 0.995, which corresponds to a false negative rate of one in 200. We can then have each model make an individual prediction, and combine those predictions using a majority voting scheme to obtain a final single binary prediction value for each input. The final ensemble voting model has an accuracy score of 0.932, an F1-Score of 0.471, a precision of 0.308, and a recall of 0.998. The confusion matrix (see Figure 5-20) shows that the voting ensemble is more aggressive in classifying positive examples than the averaged ensemble and this evidence is confirmed by its precision value of 0.998 (a false negative rate of 1 in 500). This trade-off results in a lower overall accuracy and precision score, but ensure that potential editing sites won't be missed by the classification algorithm.

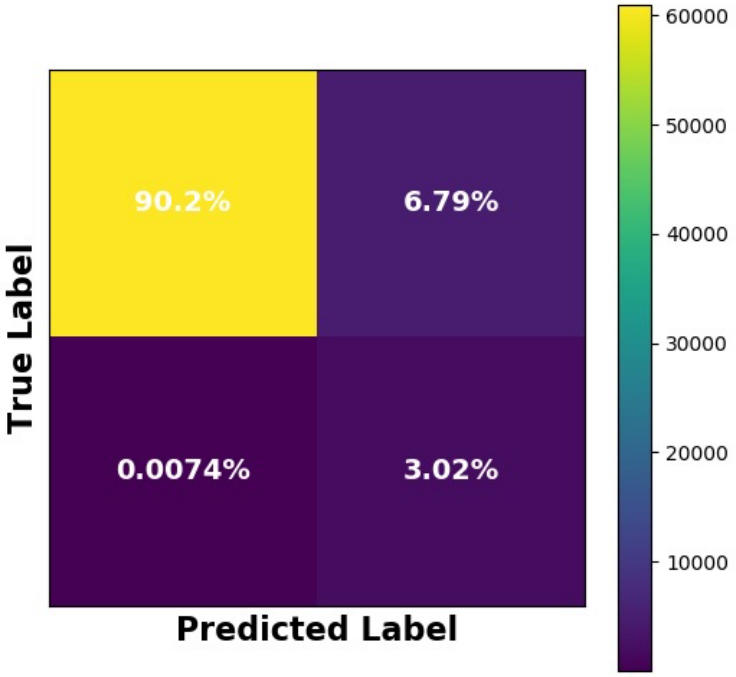

Figure 5-20. CNN Voting Ensemble Confusion Matrix. The voting ensemble has a higher false positive rate which is a result of tuning the individual models' thresholds to be sensitive to possible edit sites. The configuration reduces false negative predictions to a rate of 1 of every 500 true positives.

5.2.4. Across a Chromosome

We finally apply the voting ensemble across full chromosomes to evaluate the total number of potential edit locations identified by the model. The large size of a full chromosome can make edit identification efforts prohibitive if a model identifies more sites than can be possibly confirmed by a human subject matter expert. To evaluate the total positive rate, we apply our model across full chromosomes, pre-filtering out regions as defined by the previously described pre-processing step. This leaves only regions that belong to gene-coding exomes that have sufficient coverage (minimum of five sequence reads) to give our model a reliable signal.

We traverse the chromosome in steps of length 250, half the width of our evaluation window. This allows the model to avoid situations where an edit overlaps the edge of the window, but it does double the number of steps required to complete the full evaluation. Thus, it is also possible for a single location of interest to be flagged twice by this approach. We evaluate a number of unique chromosomes from different datasets to test the sensitivity of the final voting ensemble. The results are shown in the table below (see Table 4).

Table 4. Total number of possible edit sites identified in full chromosome by the voting ensemble model. Model sensitivity is set to achieve a 0.998 recall.

Data Source	Data ID	Chromosome	Number of Positives / Total Locations
1,000 Genomes	SRR233071	1	556 / 3,098
Cho	DRR014244	3	7,058 / 10,408
Church	SRR1583555	1	1, 303 / 73,852
Veres	SRR1217719	1	3,996 / 70,545

The overall number of positive sites identified is high, but possibly tractable, though the results for the Cho sample are concerning. Note again that these values are obtained with a model set to a extremely high sensitivity, such that its recall is 0.998 (missing one positive out of every 500). These total positive location numbers could be lowered with a higher tolerance for false negatives.

5.3. Transformer Architecture

Our transformer architecture also performs well, even with limited data. Figure 5-21 shows the language model training performance, for a 90/10 train/test split on the language data (expanded genomic sequence) samples.

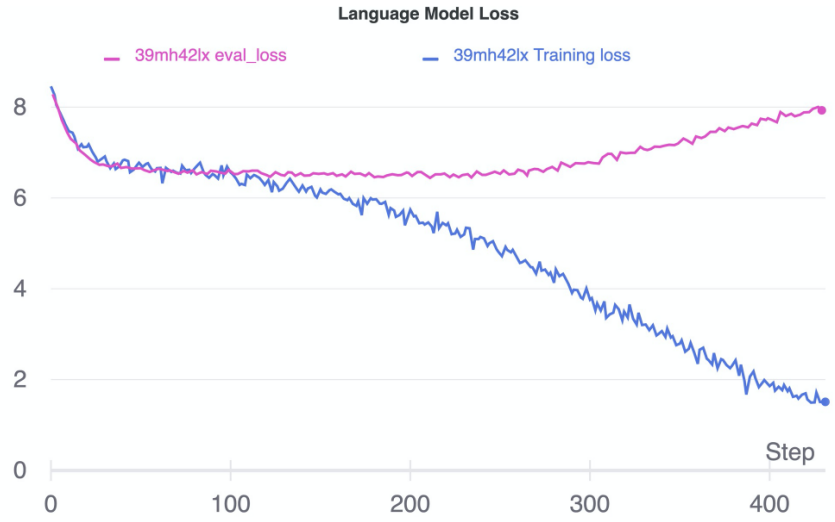

Figure 5-21. Training performance for our transformer architecture.

Most likely due to relatively small training corpus, the model quickly overfits – indicating a relatively shallow contextual representation in the data. Given that our synthetic language's grammar is a series

of simple words, composed of a reference genome base followed by characters that map to a statistical value, this makes sense. We are, at most, learning the sequence of the human genome, or at least the portions that are represented in our training data. The best performing language model (validation loss) was used to train the classification head for the binary prediction task. Binary prediction using transformers is notoriously hyperparameter sensitive. We ran several large scale hyperparameter tuning sweeps to determine the subset of hyperparameters most likely to contribute to model performance. shown in Figure 5-22.

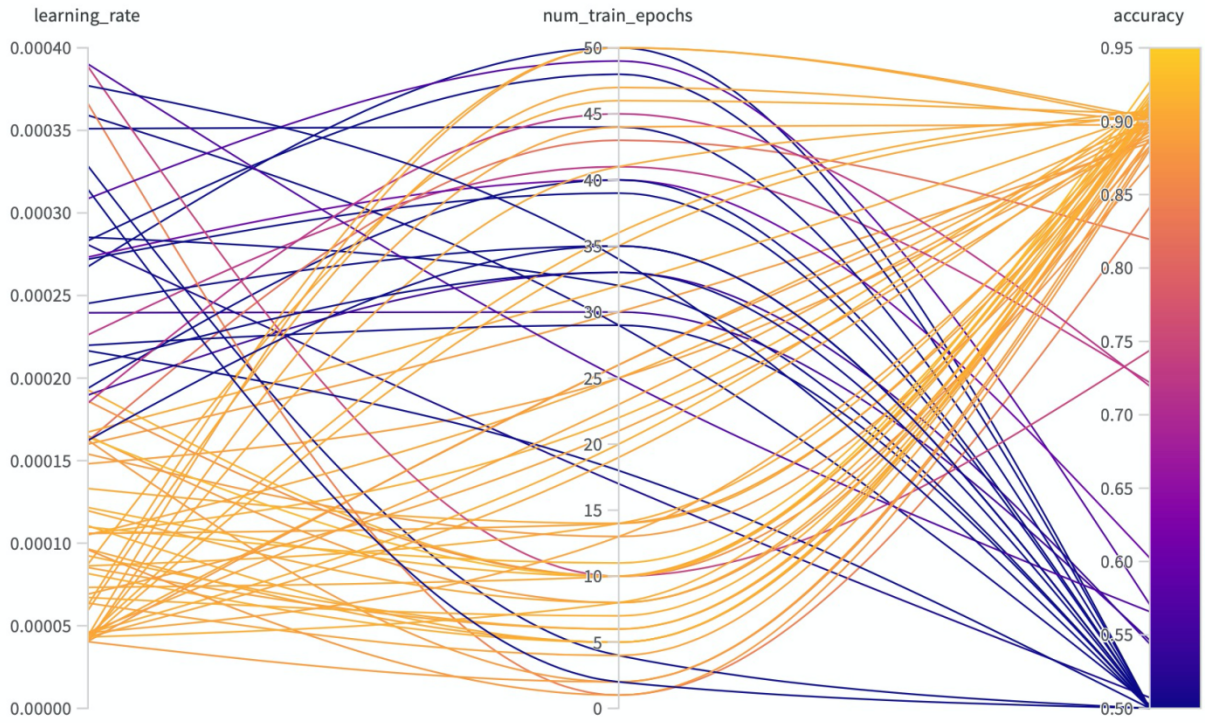

Figure 5-22. Effect of learning rate and number of epochs on training accuracy.

While these takeaways may not translate to the same task with different base architectures, learning rate and the number of training epochs has the largest independent and conditional influence on binary classification accuracy. We optimized their combination to maximize model performance for this task, but stress that the specific values used for learning rate and number of training epochs in this case will likely not translate to adjacent model architectures, with results shown in the following table.

Table 5. Transformer edit versus no-edit classification results.

TP	FP	TN	FN	AuROC	AuPRC	MCC
191	19	186	14	0.97	0.97	0.84

Table 5 shows the initial results from the trained classification head, with true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) being used to calculate the Matthew's correlation coefficient (MCC). Area under the receiver-operator curve (AuROC) and area under the precision-recall curve (AuPRC) of 0.97 indicates a high likelihood of good performance on binary classification tasks for genomic edit prediction using this approach. Decreasing FP and FN will likely require a larger language training and binary classification corpus, specifically for the task of binary prediction. Transformer architectures perform best, like most deep learning models, with large amounts of training data. In this case, given that the datasets were heavily bias with true negatives, the size of the true positive samples constrained the amount of data the model could be trained with. However, we want to stress that these are very initial and cursory results – there is great potential to improve and refine the process, as well as see its integration with anomaly detection, tree-based, and other Deep Neural Network approaches described here. These initial results show promise for what is a novel embedding approach to condense a large number of genomic sequences and embed their statistical deviations into a reference genome. The resulting language of genomic variability may serve as a salient feature for prediction and binary classification tasks using transformer architectures.

5.4. Anomaly Detection

Our anomaly detection algorithms are easier to implement and apply than the other methods described in this report, but they are also not as generalizable or as accurate.

5.4.1. Chakrabarti-inspired Anomaly Detector

Our Chakrabarti-inspired anomaly detector detects four classes of potential edit anomalies, corresponding the three precision classes from Chakrabarti and colleagues [Chakrabarti et al., 2018] plus an additional class for perfect noise anomalies (i.e., those base positions which for some reason mismatch entirely with the reference genome alignment). Note that this anomaly detector, in accordance with the edit classes identified by Chakrabarti and colleagues, only uses indels in detecting anomalous base positions, and only those base positions with 10 or more reads are considered for anomalies. An example of this anomaly detector applied to the Cho dataset [Cho et al., 2014] at the CCR5 edit target site is shown in Figure 5-23.

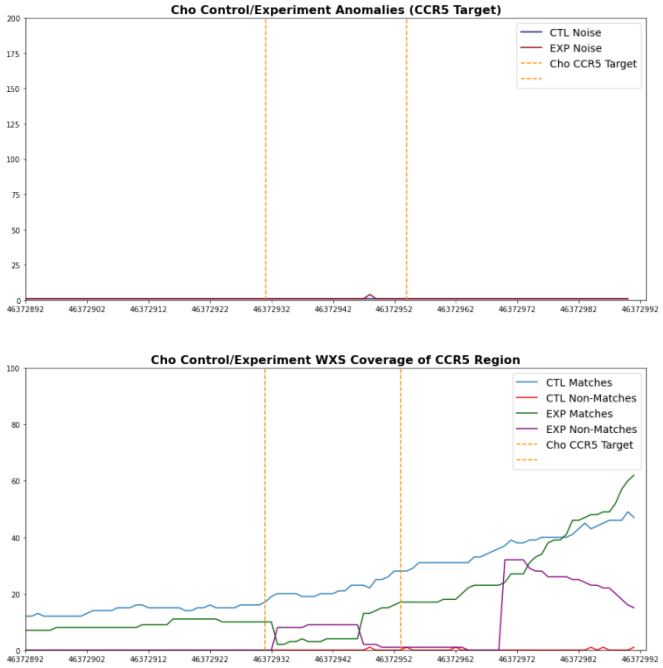

Figure 5-23. Chakrabarti-inspired anomalies detected (upper plot) and matches versus non-matches (bottom plot) at the CCR5 target site in chromosome 3 in [Cho et al., 2014].

Note that the anomaly detector does detect an anomaly at base position 46,372,948 (with respect to reference hg38), but the noise count is very low (4 counts out of a coverage of 15 reads – corresponding to a medium edit class), thus it is a very small peak in the upper plot in Figure 5-21. Note that this upper plot is formed by using the anomaly detector to mask out the non-anomaly noise values per base, shown in the bottom plot (in Figure 5-23). This masking is used in the upper plots in the next several figures, too.

Looking at another edit site in chromosome 1 (the C4BPB target) from Cho and colleagues [Cho et al., 2014], we can see a more visibly distinct anomaly detection in Figure 5-24.

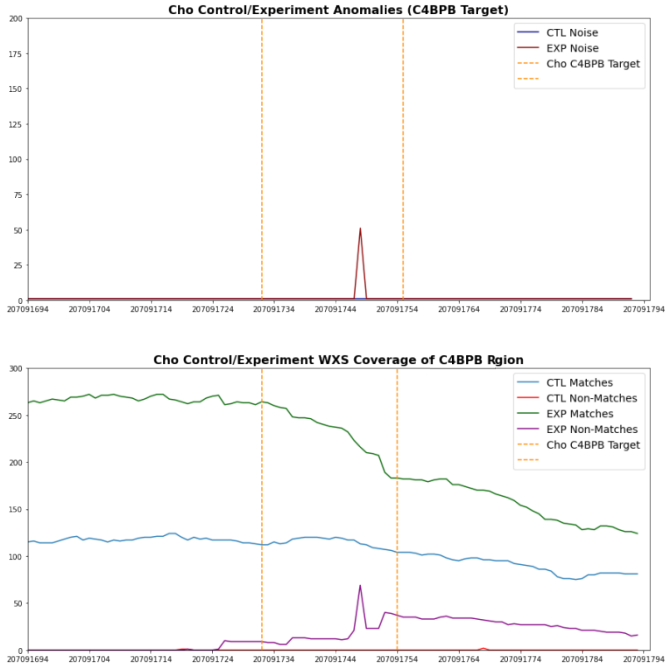

Figure 5-24. Chakrabarti-inspired anomalies detected (upper plot) and matches versus non-matches (bottom plot) at the C4BPB target site in chromosome 1 in [Cho et al., 2014].

Here the detector classifies this anomaly as a class 1 (or precise) edit, although it only is precise at the single base position (207,091,748).

We also extended this anomaly detector to include all the genome noise counter categories of noise, specifically including mismatches and clips (both soft and hard), for detecting anomalies, shown in Figure 5-25 and Figure 5-26.

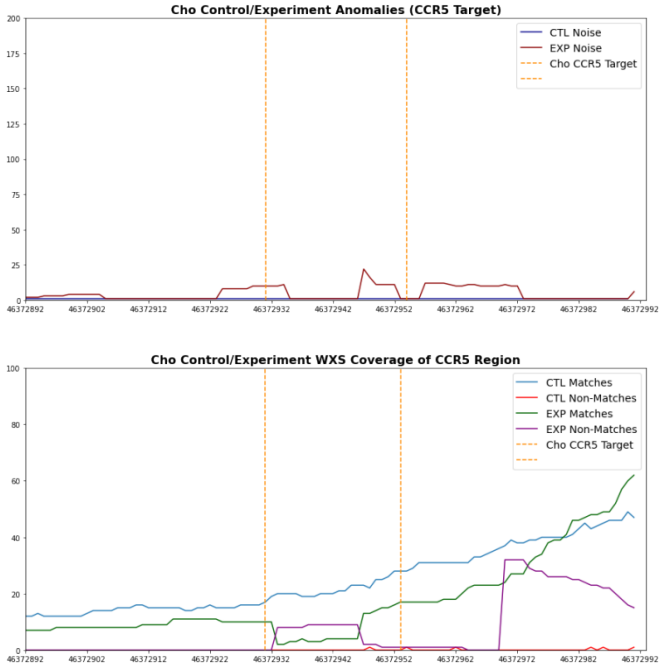

Figure 5-25. Chakrabarti-inspired anomalies detected (upper plot) using all non-matches (all the genome noise counter noise types) as well as matches versus non-matches (bottom plot) at the CCR5 target site in chromosome 3 in [Cho et al., 2014].

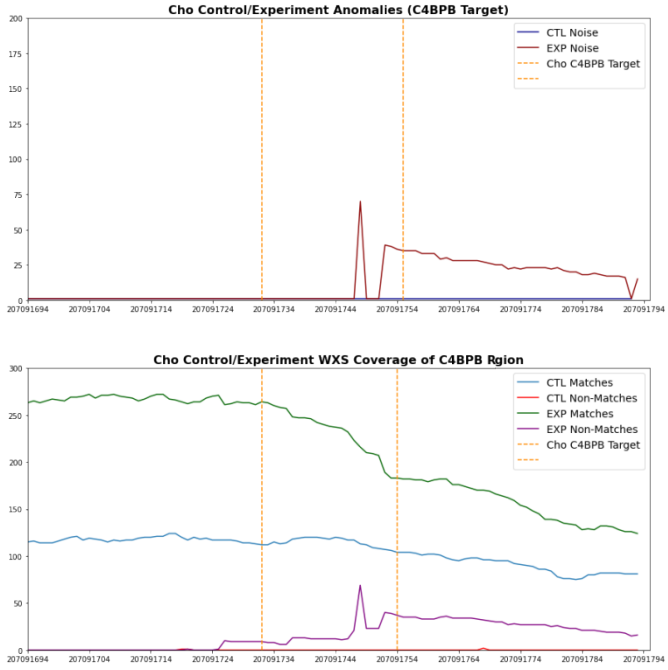

Figure 5-26. Chakrabarti-inspired anomalies detected (upper plot) using all non-matches (all the genome noise counter noise types) as well as matches versus non-matches (bottom plot) at the C4BPB target site in chromosome 1 in [Cho et al., 2014].

This extension of the Chakrabarti-inspired anomaly detector does pick up more anomalies in the region of the edit targets, but it also detects more false positives. Across the entire chromosome 3 (where the CCR5 edit site is located), this anomaly detector detects a total of 2,972 anomalies using only indels and 4,957 using all the genome noise counter noise types. Across the entire chromosome 1 (where the C4BPB edit site is located), this anomaly detector detects a total of 7,889 anomalies using only indels and 10,672 using all the genome noise counter noise types. In the human genome, chromosomes 3 and 1 have 198,295,559 and 248,956,422 base pairs each, but since the data sources we are using (from [Cho et al., 2014]) are WXS reads, only about 1% of these base pairs will have non-zero read coverage. Thus, the positive detection rates are 2972/1,982,955.59 = 0.0015 (indels only) and 7,889/1,982955.59 = 0.0040 (all genome noise counter noise types) as well as 4,957/2,489,564.22 = 0.0020 (indels only) and 10,672/2,489,564.22 = 0.0043 (all genome noise counter noise types) for chromosomes 3 and 1, respectively.

5.4.2. Spiking AMF Anomaly Detector

Our spiking AMF anomaly detector filters out base positions where there are fewer than 5 total read counts (versus 10 for the Chakrabarti-inspired anomaly detector), and it is looking for clustered groups of anomalies versus single anomalous bases. The spiking AMF anomaly detector can detect edit anomalies in several ways. First, it can detect strong transitions from no noise to significant noise present in data output from our genome noise counter, see Figure 5-27.

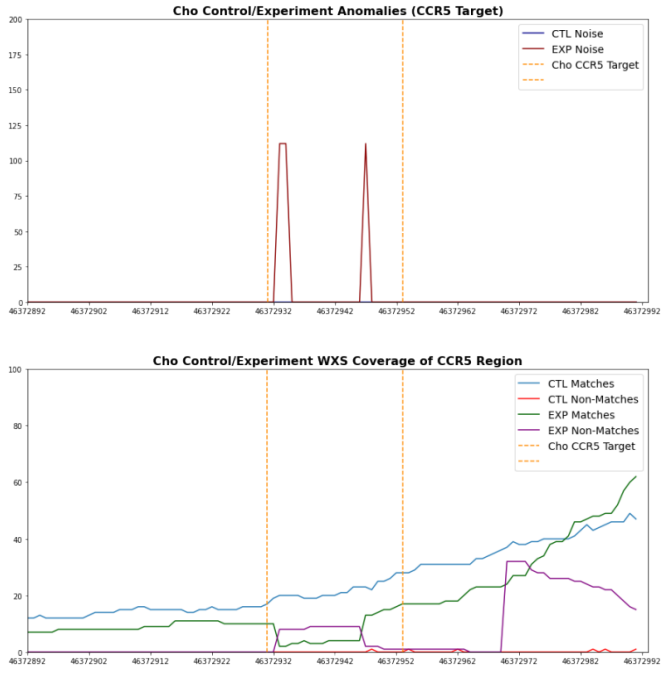

Figure 5-27. Spiking AMF anomalies detected at the CCR5 target site in chromosome 3 in [Cho et al., 2014].

In the upper plot in Figure 5-27, the spiking AMF anomaly detector has detected three anomalous bases, two at the left and one at the right in the sgRNA (23-mer) region used by Cho and colleagues

for their edit experiment in the CCR5 gene [Cho et al., 2014]. Note that there are no anomalies detected in the control data (the blue line in the upper plot is at 0). This plot is obtained by using the anomalies detected as a mask for plotting the (non-match) noise coverage at each base position. The bottom plot in Figure 5-27 shows the actual coverage (both matches and non-matches or noise) at the site of the edit with both control data (blue and red lines) as well as experiment data (green and purple).

The spiking AMF anomaly detector can also detect significant noise signals that surpass the median value of the surrounding base positions, see Figure 5-28.

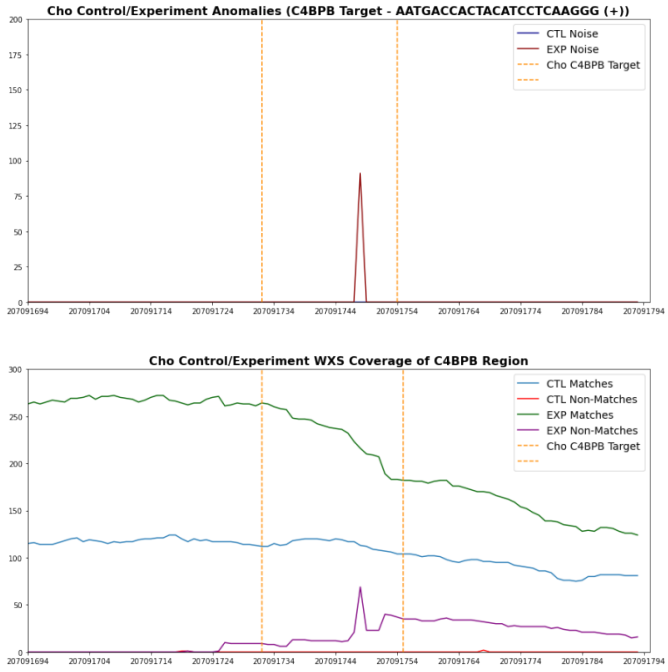

Figure 5-28. Spiking AMF anomalies detected at the C4BPB target site in chromosome 1 in [Cho et al., 2014].

When the spiking AMF algorithm is applied to an entire chromosome, 3 (for the CCR5 edit target site) and 1 (for the C4BPB edit target site), it detects a total of 1,403 and 3,461 (potential) anomalies, respectively. These would correspond to positive detection rates of 1,403/1,982,955.59 = 0.0007 and 3,461/2,489,564.22 = 0.0014, respectively. This anomaly detection works well at known edit sites, but it also does detect noise signals that are not (necessarily) edit sites (both in the control and experiment data).

In order to compare the anomaly detection with the other machine learning methods, we need to be able to normalize these numbers. First, we need to define which base positions are involved in the edit. Chakrabarti and colleagues showed that each of the four base positions at: -5, -4, -3, and -2, relative to the PAM influence about 98% of the predictive capability of their model, thus we also use these 4 positions to identify the edit target site. In each of the edits described above (one in chromosome 1 and the other in chromosome 3), then there would be 4 bases marked as edits across

the whole chromosome. Our anomaly detector picks up 1 of each of these. Therefore, we can estimate accuracy as (1 + 2,489,564 – 4)/2,489,564 = 0.999999 for chromosome 1 and (1 + 1,982,956 – 4)/1,982,956 = 0.999998. These number show why accuracy is not a good metric for this problem. Precision would be estimated as 1/(1 + 3461) = 0.000289 and 1/(1 + 1,403) = 0.000712, and Recall would be 1/(1 + 3) = 0.25 and 1/(1+3) = 0.25, for chromosomes 1 and 3 respectively. F1 score can be estimated as 1/(1 + ½(3461 + 3)) = 0.000577 and 1/(1 + ½(1403 + 3)) = 0.001420.

More complete details of these anomaly detection algorithms and extensions are available in a forthcoming manuscript by the authors of this report [Verzi et al., 2021].

The following table lists results from our methods applied to edit detection on test datasets.

Table 6. Method results for edit detection.

Method		Accuracy	Precision	Recall	F1 Score
XGBoost	average[2]	0.918	0.844	0.790	0.811
CNN ensemble	average[3]	0.987	0.862	0.688	0.765
	voting[4]	0.932	0.308	0.998	0.471
RoBERTa		0.920	0.910	0.932	0.920
Anomaly Detector	average[5]	1.000	0.000	0.25	0.004

[2] Average over leave-one-out study for 13 datasets.
[3] Average over 5 ensemble members.
[4] Majority voting across 5 ensemble members.
[5] Average across two chromosome walks.

6. CONCLUSIONS

Our goal was to use DNA sequencing data to infer the locations of genomic edits, and through various machine learning and anomaly detection methods we have achieved this goal to a large extent. We learned that correlating different types of alignment mismatches with edited regions of the genome could give us an idea of how to identify edits, but that the linear relationships alone were not sufficient for the identification. Turning to machine learning methods allowed us to examine more complex relationships between these mis-alignments and regions edited by Cas9 nucleases. We chose to examine a number of methods including neural networks, anomaly detection, transformers and tree-based methods, and we demonstrate the usefulness and potential limitations of these different methods, leaving open the door to an ensemble method that incorporates a number of these techniques to make the final call on whether a region is edited.

One of the biggest challenges of this project ended up being data collection. We had to balance collecting a large amount of data with obtaining data sets that we thought reasonably represented a significant portion of the likely diversity of NHEJ sequencing data. This not only refers to the biological diversity of Cas9 cutting (which is an important consideration) but also lab-, reagent-, equipment- and personnel-specific technical variability, which is always a concern when combining experimental data generated out of different studies [Čuklina et al., 2020]. An important aspect of this variability includes penetrance of the edit, which is heavily influence by both the intracellular environment (biological variables) as well as the success rate of introducing CRISPR components into cells (technical variables). This meant that ideally, we wouldn't want a large fraction of our data to be from one paper or experimental setup (otherwise, we would have used the entire data set from Chakrabarti et al to train our models, for example). To address this, we collected data from 7 different studies, and used SMOTE [Chawla et al., 2002] to augment our data without biasing it towards any single study. We believe that this approach allows for equal representation of multiple studies with minimal bias (with the caveat that not all possible edits may be represented by the studies we chose).

In comparing Random Forests with XGBoost, we found that even though both models had high cross-validation F1 scores, XGBoost almost uniformly performed better on test data sets. There are a couple of reasons why this might be. XGBoost prunes trees based on similarity scores, or in other words, if the difference between the similarity score of a node and its children are minimal, that node is considered non-useful and the tree is pruned at that node, thereby greatly reducing the risk of overfitting. In a problem where individual edited regions should not be considered to be representative of the population as a whole, overfitting can become a problem, and it is possible that XGBoost does a better job of avoiding this. In a similar vein, because XGBoost involves iterative optimization through gradient descent (to minimize a loss function), the original hyperparameters are only applied to the initial iteration of the model, and subsequent iterations adjust themselves based on the results of the gradient descent. On the other hand, for the Random Forest, the initial hyperparameters are applied to the whole forest, potentially exacerbating any overfitting issues [Gupta, 2021].

In terms of how long our method would take to make a call given a sequencing data set, it depends both on the size of the genome (i.e., how big is it?) and the depth of the sequencing data. If we translate this to an entire genome, it would make the time much more significant. It is worth noting,

however, that this model can easily be run on subsets of the genome (e.g., chromosomes, or even smaller increments) in parallel, meaning that an entire genome could be run in as long as it takes to analyze the longest chromosome. Our model currently runs on 10 cores in parallel (we found that increasing this did not significantly improve performance, data not shown), a 23-chromosome genome could be run on a compute cluster with 230 cores in parallel.

Our XGBoost models can identify a majority of the edits that we have using leave-out experiments, suggesting that for the most part, the features that we selected are appropriate for the task at hand. In the cases where we couldn't identify an edit, closer examination of the data using tSNE suggests that this is due to the similarity of the edited regions to unedited genome that exhibits a certain level of 'noise' with respect to alignment. One simple explanation for this is that there is natural variation in human genomes, and that a natural variant may look like an edit to the model [Armstrong et al., 2018; Valenzuela et al., 2018]. One distinction is that natural variants (in the absence of aneuploidy) will have either 50% or 100% penetrance in a given cell (i.e., one or both alleles will be altered). The caveat of using this to screen natural variants is that (i) a CRISPR edit may also be either 50% or even 100% penetrant (indeed there are now predictive modeling methods that can predict the editing outcomes of sgRNAs [Naert et al., 2020], allowing for selection of sgRNAs with high penetrance), or (ii) there may be a mix of cells and tissue that have acquired somatic mutations that are not related to genome editing activity, in which case the penetrance of the misalignment could be anything. We therefore draw attention to the fact that the goal of our models is not necessarily to identify the only edited region in a genome (which is likely an unrealistic goal with current technology), but rather to flag specific regions of the genome for follow-up as potentially edited. This allows for researchers and genome security specialists to focus their attention on a small subset of genomic regions.

A limitation of the tree-based methods is their inability to consider the longitudinal nature of DNA and genomic encodings. The sequential nature of nucleotide to amino acid translation suggests that the ordering of the individual bases can play an important role in the function of an edit. Further, because edit signatures can span across a contiguous group of nucleotides, it is important to evaluate models that can take that information into account. Convolutional neural networks and transformer models are deep learning neural network architectures designed specifically to be able to discover and understand patterns with spatial structure. In this research, we apply these techniques to best exploit this inherent structure to genomic structure.

Both the CNN ensemble models and the transformer models perform exceptionally well. Both demonstrate a high ROC AUC (0.995 for the CNN ensemble and 0.97 for the transformer) while balancing their false positive rate (an average precision of 0.872 for the CNN ensemble and 0.97 for the transformer). Limiting the rate of false positives is a critical need for our prediction models due to the prohibitive length of the human genome and the class imbalance of our dataset. Our two deep learning models demonstrate excellent performance, even when set to high sensitivities to avoid false negatives.

In order to narrow down the regions that have been flagged as edited even further, future research in this area needs to incorporate information on the likelihood of a Cas9 edit based on other features (such as DNA accessibility in that given cell type), as well as the functional outcomes of the edit (for example, is it in a known coding or regulatory region, and what are the downstream effects of

mutating that region in the given cell type?). As the literature has shown, these factors are critically important in determining whether an Cas9 edit is likely in a given region of the genome [Jensen et al., 2017; Yarrington et al., 2018; Liu et al., 2019; Liscovitch-Brauer et al., 2021], and we expect that incorporating this information downstream of our existing models will allow for further refinement of our method, and ultimately, identification of a larger fraction of edits in a timely manner.

In addition to NHEJ, which was the focus of our work, there are a number of other ways in which CRISPR proteins can modify the genome. For example, Cas9 can use homologous recombination to alter DNA using a given template [Sansbury et al., 2019; Yang et al., 2020]. In addition, there are a number of precise base- and prime-editors that can change single nucleotides [Hirakawa et al., 2020; Kantor et al., 2020]. Given that the presence of single base differences between sequencing data and reference genomes is not uncommon (both due to natural variation and due to technical glitches during amplification and sequencing [The 1000 Genomes Project Consortium, 2010; Ma et al., 2019]), In these cases, the additional information of context and functional outcomes discussed above become increasingly important to filter out noise and focus on real areas of threat.

REFERENCES

[1] Adli M. (2018). The CRISPR tool kit for genome editing and beyond. Nature communications, 9(1), 1911. https://doi.org/10.1038/s41467-018-04252-2

[2] Allen, F., Crepaldi, L., Alsinet, C., Strong, A. J., Kleshchevnikov, V., De Angeli, P., Páleníková, P., Khodak, A., Kiselev, V., Kosicki, M., Bassett, A. R., Harding, H., Galanty, Y., Muñoz-Martínez, F., Metzakopian, E., Jackson, S. P., & Parts, L. (2018). Predicting the mutations generated by repair of Cas9-induced double-strand breaks. Nature biotechnology, 10.1038/nbt.4317. Advance online publication. https://doi.org/10.1038/nbt.4317

[3] Anzalone, A. V., Koblan, L. W., & Liu, D. R. (2020). Genome editing with CRISPR-Cas nucleases, base editors, transposases and prime editors. Nature biotechnology, 38(7), 824–844. https://doi.org/10.1038/s41587-020-0561-9

[4] Arbab, M., Shen, M. W., Mok, B., Wilson, C., Matuszek, Ż., Cassa, C. A., & Liu, D. R. (2020). Determinants of Base Editing Outcomes from Target Library Analysis and Machine Learning. Cell, 182(2), 463–480.e30. https://doi.org/10.1016/j.cell.2020.05.037

[5] Armstrong, J., Fiddes, I. T., Diekhans, M., & Paten, B. (2019). Whole-Genome Alignment and Comparative Annotation. Annual review of animal biosciences, 7, 41–64. https://doi.org/10.1146/annurev-animal-020518-115005

[6] Atlas, L., Homma, T., & Marks, R. (1987). An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification. NIPS. https://proceedings.neurips.cc/paper/1987/hash/98f13708210194c475687be6106a3b84-Abstract.html

[7] Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics (Oxford, England), 30(15), 2114–2120. https://doi.org/10.1093/bioinformatics/btu170

[8] Braddick, D., Ramarohetra, R.F. (2020). Chapter 21 - Emergent challenges for CRISPR: biosafety, biosecurity, patenting, and regulatory issues, In Editor(s): Vijai Singh, Pawan K. Dhar, Genome Engineering via CRISPR-Cas9 System, 281-307. https://doi.org/10.1016/B978-0-12-818140-9.00021-0

[9] Brandsma, I., & Gent, D. C. (2012). Pathway choice in DNA double strand break repair: observations of a balancing act. Genome integrity, 3(1), 9. https://doi.org/10.1186/2041-9414-3-9

[10] Cai, L., Wu, Y. & Gao, J. DeepSV: accurate calling of genomic deletions from high-throughput sequencing data using deep convolutional neural network. BMC Bioinformatics 20, 665 (2019). https://doi.org/10.1186/s12859-019-3299-y

[11] Chakrabarti, A. M., Henser-Brownhill, T., Monserrat, J., Poetsch, A. R., Luscombe, N. M., & Scaffidi, P. (2019). Target-Specific Precision of CRISPR-Mediated Genome Editing. Molecular cell, 73(4), 699–713.e6. https://doi.org/10.1016/j.molcel.2018.11.031

[12] Chang, W., Liu, Y., Xiao, Y., Yuan, X., Xu, X., Zhang, S., & Zhou, S. (2019). A Machine-Learning-Based Prediction Method for Hypertension Outcomes Based on Medical Data. Diagnostics (Basel, Switzerland), 9(4), 178. https://doi.org/10.3390/diagnostics9040178

[13] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785

[14]Chen, W., McKenna, A., Schreiber, J., Haeussler, M., Yin, Y., Agarwal, V., Noble, W.S., & Shendure, J. (2019). Massively parallel profiling and predictive modeling of the outcomes of CRISPR/Cas9-mediated double-strand break repair, Nucleic Acids Research, 47(15), 7989–8003. https://doi.org/10.1093/nar/gkz487

[15]Cho, S. W., Kim, S., Kim, Y., Kweon, J., Kim, H. S., Bae, S., & Kim, J. S. (2014). Analysis of off-target effects of CRISPR/Cas-derived RNA-guided endonucleases and nickases. Genome research, 24(1), 132–141. https://doi.org/10.1101/gr.162339.113

[16]Chuai, G., Ma, H., Yan, J., Chen, M., Hong, N., Xue, D., Zhou, C., Zhu, C., Chen, K., Duan, B., Gu, F., Qu, S., Huang, D., Wei, J., & Liu, Q. (2018). DeepCRISPR: optimized CRISPR guide RNA design by deep learning. Genome biology, 19(1), 80. https://doi.org/10.1186/s13059-018-1459-4

[17]Cong, L., Ran, F. A., Cox, D., Lin, S., Barretto, R., Habib, N., Hsu, P. D., Wu, X., Jiang, W., Marraffini, L. A., & Zhang, F. (2013). Multiplex genome engineering using CRISPR/Cas systems. Science (New York, N.Y.), 339(6121), 819–823. https://doi.org/10.1126/science.1231143

[18]Čuklina, J., Pedrioli, P., & Aebersold, R. (2020). Review of Batch Effects Prevention, Diagnostics, and Correction Approaches. Methods in molecular biology (Clifton, N.J.), 2051, 373–387. https://doi.org/10.1007/978-1-4939-9744-2_16

[19]Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805. https://arxiv.org/abs/1810.04805

[20]Ding, W., Zhang, Y., & Shi, S. (2020). Development and Application of CRISPR/Cas in Microbial Biotechnology. Frontiers in bioengineering and biotechnology, 8, 711. https://doi.org/10.3389/fbioe.2020.00711

[21]do Nascimento, P. M., Medeiros, I. G., Falcão, R. M., Stransky, B., & de Souza, J. (2020). A decision tree to improve identification of pathogenic mutations in clinical practice. BMC medical informatics and decision making, 20(1), 52. https://doi.org/10.1186/s12911-020-1060-0

[22]Du, J., Yin, N., Xie, T., Zheng, Y., Xia, N., Shang, J., Chen, F., Zhang, H., Yu, J., & Liu, F. (2018). Quantitative assessment of HR and NHEJ activities via CRISPR/Cas9-induced oligodeoxynucleotide-mediated DSB repair. DNA repair, 70, 67–71. https://doi.org/10.1016/j.dnarep.2018.09.002

[23]El-Mounadi, K., Morales-Floriano, M. L., & Garcia-Ruiz, H. (2020). Principles, Applications, and Biosafety of Plant Genome Editing Using CRISPR-Cas9. Frontiers in plant science, 11, 56. https://doi.org/10.3389/fpls.2020.00056

[24]Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics, 36[4], 193-202. https://doi.org/10.1016/0893-6080(88)90014-7

[25]Gleditzsch, D., Pausch, P., Müller-Esparza, H., Özcan, A., Guo, X., Bange, G., & Randau, L. (2019). PAM identification by CRISPR-Cas effector complexes: diversified mechanisms and structures. RNA biology, 16(4), 504–517. https://doi.org/10.1080/15476286.2018.1504546

[26]González, J., Longworth, J., James, D., Lawrence, N. (2014). Bayesian Optimization for synthetic gene design. The Neural Information Processing Systems (NIPS'14), Workshop in Bayesian Optimization. arXiv:1506.01627. https://arxiv.org/abs/1505.01627

[27]Gonzalez, J., Osborne, M. & Lawrence, N.D. (2016). GLASSES: Relieving The Myopia Of Bayesian Optimisation. Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, in Proceedings of Machine Learning Research, 51, 790-799. Available from https://proceedings.mlr.press/v51/gonzalez16b.html

[28]Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org

[29]Guo, T., Feng, Y. L., Xiao, J. J., Liu, Q., Sun, X. N., Xiang, J. F., Kong, N., Liu, S. C., Chen, G. Q., Wang, Y., Dong, M. M., Cai, Z., Lin, H., Cai, X. J., & Xie, A. Y. (2018). Harnessing accurate non-homologous end joining for efficient precise deletion in CRISPR/Cas9-mediated genome editing. Genome biology, 19(1), 170. https://doi.org/10.1186/s13059-018-1518-x

[30]Gupta, A. (2021). XGBoost vs Random Forest, https://medium.com/geekculture/xgboost-versus-random-forest-898e42870f30.

[31]Hertel, L., Collado, J., Sadowski, P., Ott, J., & Baldi, P. (2020). Sherpa: Robust hyperparameter optimization for machine learning, SoftwareX, 12, 100591. https://doi.org/10.1016/j.softx.2020.100591

[32]Hirakawa, M. P., Krishnakumar, R., Timlin, J. A., Carney, J. P., & Butler, K. S. (2020). Gene editing and CRISPR in the clinic: current and future perspectives. Bioscience reports, 40(4), BSR20200127. https://doi.org/10.1042/BSR20200127

[33]Hsu, P. D., Lander, E. S., & Zhang, F. (2014). Development and applications of CRISPR-Cas9 for genome engineering. Cell, 157(6), 1262–1278. https://doi.org/10.1016/j.cell.2014.05.010

[34]Jensen, K. T., Fløe, L., Petersen, T. S., Huang, J., Xu, F., Bolund, L., Luo, Y., & Lin, L. (2017). Chromatin accessibility and guide sequence secondary structure affect CRISPR-Cas9 gene editing efficiency. FEBS letters, 591(13), 1892–1901. https://doi.org/10.1002/1873-3468.12707

[35]Jiang, F., & Doudna, J. A. (2017). CRISPR-Cas9 Structures and Mechanisms. Annual review of biophysics, 46, 505–529. https://doi.org/10.1146/annurev-biophys-062215-010822

[36]Jinek, M., Chylinski, K., Fonfara, I., Hauer, M., Doudna, J. A., & Charpentier, E. (2012). A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity. Science (New York, N.Y.), 337(6096), 816–821. https://doi.org/10.1126/science.1225829

[37]Kannan, S., Altae-Tran, H., Jin, X., Madigan, V. J., Oshiro, R., Makarova, K. S., Koonin, E. V., & Zhang, F. (2021). Compact RNA editors with small Cas13 proteins. Nature biotechnology, 10.1038/s41587-021-01030-2. Advance online publication. https://doi.org/10.1038/s41587-021-01030-2

[38]Kantor, A., McClements, M. E., & MacLaren, R. E. (2020). CRISPR-Cas9 DNA Base-Editing and Prime-Editing. International journal of molecular sciences, 21(17), 6240. https://doi.org/10.3390/ijms21176240

[39]Li, D., Qiu, Z., Shao, Y., Chen, Y., Guan, Y., Liu, M., Li, Y., Gao, N., Wang, L., Lu, X., Zhao, Y., & Liu, M. (2013). Heritable gene targeting in the mouse and rat using a CRISPR-Cas system. Nature biotechnology, 31(8), 681–683. https://doi.org/10.1038/nbt.2661

[40]Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv: Genomics.

[41]Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup (2009). The Sequence

Alignment/Map format and SAMtools. Bioinformatics (Oxford, England), 25(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

[42]Lin, S., Staahl, B. T., Alla, R. K., & Doudna, J. A. (2014). Enhanced homology-directed human genome engineering by controlled timing of CRISPR/Cas9 delivery. eLife, 3, e04766. https://doi.org/10.7554/eLife.04766

[43]Liscovitch-Brauer, N., Montalbano, A., Deng, J., Méndez-Mancilla, A., Wessels, H. H., Moss, N. G., Kung, C. Y., Sookdeo, A., Guo, X., Geller, E., Jaini, S., Smibert, P., & Sanjana, N. E. (2021). Profiling the genetic determinants of chromatin accessibility with scalable single-cell CRISPR screens. Nature biotechnology, 10.1038/s41587-021-00902-x. Advance online publication. https://doi.org/10.1038/s41587-021-00902-x

[44]Liu, G., Yin, K., Zhang, Q., Gao, C., & Qiu, J. L. (2019). Modulating chromatin accessibility by transactivation and targeting proximal dsgRNAs enhances Cas9 editing efficiency in vivo. Genome biology, 20(1), 145. https://doi.org/10.1186/s13059-019-1762-8

[45]Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692. https://arxiv.org/abs/1907.11692

[46]Ma, X., Shao, Y., Tian, L., Flasch, D. A., Mulder, H. L., Edmonson, M. N., Liu, Y., Chen, X., Newman, S., Nakitandwe, J., Li, Y., Li, B., Shen, S., Wang, Z., Shurtleff, S., Robison, L. L., Levy, S., Easton, J., & Zhang, J. (2019). Analysis of error profiles in deep next-generation sequencing data. Genome biology, 20(1), 50. https://doi.org/10.1186/s13059-019-1659-6

[47]Mali, P., Yang, L., Esvelt, K. M., Aach, J., Guell, M., DiCarlo, J. E., Norville, J. E., & Church, G. M. (2013). RNA-guided human genome engineering via Cas9. Science (New York, N.Y.), 339(6121), 823–826. https://doi.org/10.1126/science.1232033

[48]Naert, T., Tulkens, D., Edwards, N. A., Carron, M., Shaidani, N. I., Wlizla, M., Boel, A., Demuynck, S., Horb, M. E., Coucke, P., Willaert, A., Zorn, A. M., & Vleminckx, K. (2020). Maximizing CRISPR/Cas9 phenotype penetrance applying predictive modeling of editing outcomes in Xenopus and zebrafish embryos. Scientific reports, 10(1), 14662. https://doi.org/10.1038/s41598-020-71412-0

[49]Nidhi, S., Anand, U., Oleksak, P., Tripathi, P., Lal, J. A., Thomas, G., Kuca, K., & Tripathi, V. (2021). Novel CRISPR-Cas Systems: An Updated Review of the Current Achievements, Applications, and Future Research Perspectives. International journal of molecular sciences, 22(7), 3327. https://doi.org/10.3390/ijms22073327

[50]Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035.

[51]Pearson, Karl F.R.S. (1901) LIII. On lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2:11, 559-572, DOI: 10.1080/14786440109462720

[52]Pickar-Oliver, A., & Gersbach, C. A. (2019). The next generation of CRISPR-Cas technologies and applications. Nature reviews. Molecular cell biology, 20(8), 490–507. https://doi.org/10.1038/s41580-019-0131-5

[53]Quora.com. (2015). What are the advantages/disadvantages of using Gradient Boosting over Random Forests? [online] Available: https://www.quora.com/What-are-the-advantages-disadvantages-of-using-Gradient-Boosting-over-Random-Forests

[54]Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., & Mesirov, J. P. (2011). Integrative genomics viewer. Nature biotechnology, 29(1), 24–26. https://doi.org/10.1038/nbt.1754

[55]Sansbury, B. M., Hewes, A. M., & Kmiec, E. B. (2019). Understanding the diversity of genetic outcomes from CRISPR-Cas generated homology-directed repair. Communications biology, 2, 458. https://doi.org/10.1038/s42003-019-0705-y

[56]Song, B., Yang, S., Hwang, G. H., Yu, J., & Bae, S. (2021). Analysis of NHEJ-Based DNA Repair after CRISPR-Mediated DNA Cleavage. International journal of molecular sciences, 22(12), 6397. https://doi.org/10.3390/ijms22126397

[57]The 1000 Genomes Project Consortium. (2010). The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature 467, 1061-1073. The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature, 526, 68-74. https://doi.org/10.1038/nature09534

[58]The 1000 Genomes Project Consortium. (2015). A global reference for human genetic variation. Nature, 526, 68–74. https://doi.org/10.1038/nature15393

[59]Uddin, F., Rudin, C. M., & Sen, T. (2020). CRISPR Gene Therapy: Applications, Limitations, and Implications for the Future. Frontiers in oncology, 10, 1387. https://doi.org/10.3389/fonc.2020.01387

[60]Valenzuela, D., Norri, T., Välimäki, N., Pitkänen, E., & Mäkinen, V. (2018). Towards pan-genome read alignment to improve variation calling. BMC genomics, 19(Suppl 2), 87. https://doi.org/10.1186/s12864-018-4465-8

[61]van Overbeek, M., Capurso, D., Carter, M. M., Thompson, M. S., Frias, E., Russ, C., Reece-Hoyes, J. S., Nye, C., Gradia, S., Vidal, B., Zheng, J., Hoffman, G. R., Fuller, C. K., & May, A. P. (2016). DNA Repair Profiling Reveals Nonrandom Outcomes at Cas9-Mediated Breaks. Molecular cell, 63(4), 633–646. https://doi.org/10.1016/j.molcel.2016.06.037

[62]Veres, A., Gosis, B. S., Ding, Q., Collins, R., Ragavendran, A., Brand, H., Erdin, S., Cowan, C. A., Talkowski, M. E., & Musunuru, K. (2014). Low incidence of off-target mutations in individual CRISPR-Cas9 and TALEN targeted human stem cell clones detected by whole-genome sequencing. Cell stem cell, 15(1), 27–30. https://doi.org/10.1016/j.stem.2014.04.020

[63]Verzi S.J., Vineyard C.M., & Aimone J.B. (2018) Neural-Inspired Anomaly Detection. In: Morales A., Gershenson C., Braha D., Minai A., Bar-Yam Y. (eds) Unifying Themes in Complex Systems IX. ICCS 2018. Springer Proceedings in Complexity. Springer, Cham. https://doi.org/10.1007/978-3-319-96661-8_21

[64]Verzi, S.J., Krishnakumar, R., Levin, D., Krofcheck, D.J., & Williams, K.P. (2021). Sequence-based Anomaly Detection. Manuscript in preparation.

[65]Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors. (2020). SciPy 1.0: fundamental algorithms for

scientific computing in Python. Nat Methods 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2

[66]Walton, R.T., Hsu, J.Y., Joung, J.K., & Kleinstiver, B.P. (2021). Scalable characterization of the PAM requirements of CRISPR–Cas enzymes using HT-PAMDA. Nature Protocols, 16, 1511–1547 (2021). https://doi.org/10.1038/s41596-020-00465-2

[67]Wang, Y., Liu, K.I., Sutrisnoh, N.A.B., Srinivasan, H., Zhang, J., Li, J., Zhang, F., Lalith, C.R.J., Xing, H., Shanmugam, R., Foo, J.N., Yeo, H.T., Ooi, K.H., Bleckwehl, T., Par, Y.Y.R., Lee, S.M., Ismail, N.N.B., Sanwari, N.A.B., Lee, S.T.V., Lew, J., & Tan, M.H. (2018). Systematic evaluation of CRISPR-Cas systems reveals design principles for genome editing in human cells. Genome biology, 19(1), 62. https://doi.org/10.1186/s13059-018-1445-x

[68]West, R. M., & Gronvall, G. K. (2020). CRISPR Cautions: Biosecurity Implications of Gene Editing. Perspectives in biology and medicine, 63(1), 73–92. https://doi.org/10.1353/pbm.2020.0006

[69]Whinn, K.S., van Oijen, A.M., & Ghodke, H. (2019). Spy-ing on Cas9: Single-molecule tools reveal the enzymology of Cas9. Current Opinion in Biomedical Engineering, 12, 25-33. https://doi.org/10.1016/j.cobme.2019.08.013

[70]Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J. Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., & Rush, A.M. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing, arXiv:190.03771. https://arxiv.org/abs/1910.03771

[71]Yang, H., Ren, S., Yu, S., Pan, H., Li, T., Ge, S., Zhang, J., & Xia, N. (2020). Methods Favoring Homology-Directed Repair Choice in Response to CRISPR/Cas9 Induced-Double Strand Breaks. International journal of molecular sciences, 21(18), 6461. https://doi.org/10.3390/ijms21186461

[72]Yang, L., Grishin, D., Wang, G., Aach, J., Zhang, C.Z., Chari, R., Homsy, J., Cai, X., Zhao, Y., Fan, J.B., Seidman, C., Seidman, J., Pu, W., & Church, G. (2014). Targeted and genome-wide sequencing reveal single nucleotide variations impacting specificity of Cas9 in human stem cells. Nature Communications, 5, 5507. https://doi.org/10.1038/ncomms6507

[73]Yarrington, R.M., Verma, S., Schwartz, S., Trautman, J.K., & Carroll, D. (2018). Nucleosomes inhibit target cleavage by CRISPR-Cas9 in vivo. Proceedings of the National Academy of Sciences Sep 2018, 115 (38) 9351-9358; DOI: 10.1073/pnas.1810062115

[74]Yeh, C. D., Richardson, C. D., & Corn, J. E. (2019). Advances in genome editing through control of DNA repair pathways. Nature cell biology, 21(12), 1468–1478. https://doi.org/10.1038/s41556-019-0425-z

[75]Yourik, P., Fuchs, R. T., Mabuchi, M., Curcuru, J. L., & Robb, G. B. (2019). Staphylococcus aureus Cas9 is a multiple-turnover enzyme. RNA (New York, N.Y.), 25(1), 35–44. https://doi.org/10.1261/rna.067355.118

[76]Zhu, H., Li, C., & Gao, C. (2020). Applications of CRISPR-Cas in agriculture and plant biotechnology. Nature reviews. Molecular cell biology, 21(11), 661–677. https://doi.org/10.1038/s41580-020-00288-9

APPENDIX A. VISUALIZING CRISPR EXPERIMENTS

A.1. In the data files

One of the most important first steps in handling CRISPR data is to understand what is in (or not in) each file. Many recent researchers use barcodes, but earlier data do not always have them. We have built visualization tools to help with this process, see Figure A-1.

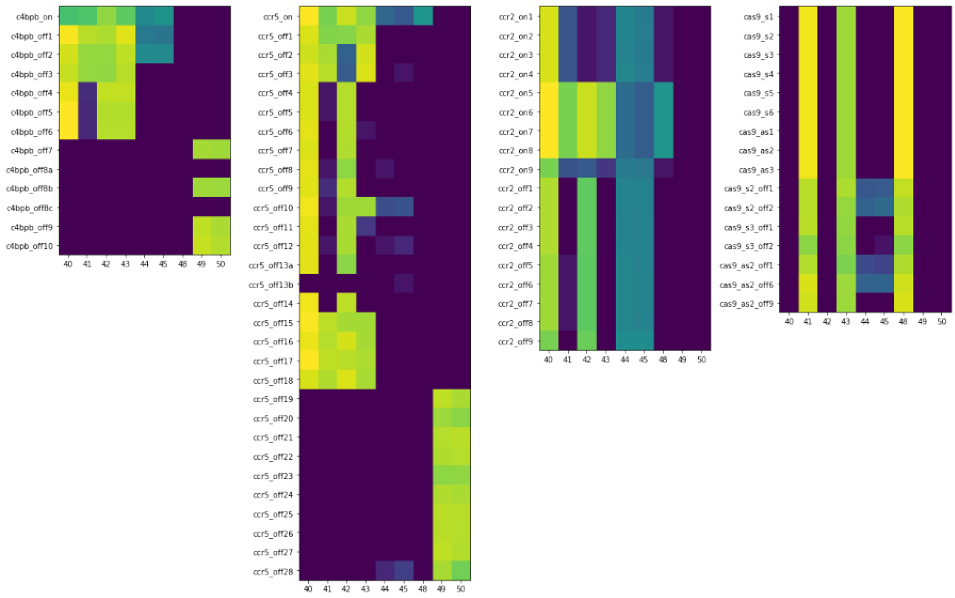

Figure A-1. Heatmap visualization of CRISPR experiments (c4bpb*, ccr2*, ccr5* and cas9*) across data files (40-50) for Cho and colleagues [Cho et al., 2014].

In Figure A-1, experimental target and off-target sites are listed along the y-axes, and the data files (available from NCBI) are listed along the x-axes. The lighter the color, the more data (number of reads) is present in each file for particular experimental sites.

A.2. In Sashimi Plots Using IGV

A next step in the evolution of our research was to see if we could detect editing at a known edit site using existing tools. Figure A-2 shows a series of Sashimi plots for three different datasets we have explored, [Cho et al., 2014; Lin et al., 2014; Wang et al., 2018], from IGV (https://software.broadinstitute.org/software/igv/) [Robinson et al., 2011].

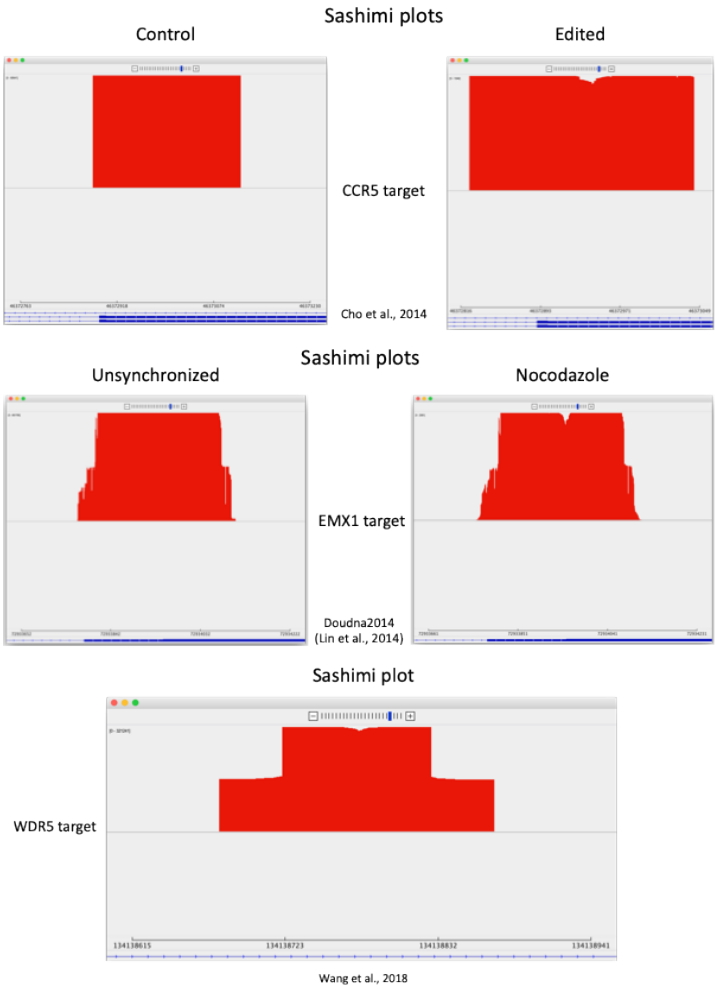

Figure A-2. Visualization of CRISPR edit target (and control) sites for three different experiments [Cho et al., 2014; Lin et al., 2014; Wang et al., 2018].

In Figure A-2, the CRISPR edit site are clearly visible as "notches" (missing coverage) in the Sashimi plots in IGV. Note that both the Cho and Lin experiments included control data, shown on the left, but the Wang experiments did not include control data. All three of these are from AMPLICON sequence reads, where the Wang experiment has overlapping AMPLICON data spanning the edit site, and the Lin experiment has significant edge effects.

A.3. In Noise Profile

Since our research explored ways of using data science and machine learning to help automate the process of edit detection, we then looked specifically at the "notches" which are non-matches versus the matches determined when each read in an experiment dataset is aligned with a reference. For our work use HG38 [cite] as the alignment reference for human sequence data. Figure A-3 shows a visualization of an aggregation of insertion and deletion (indel) counts across datasets for both the Wang [Wang et al., 2018] and Cho [Cho et al., 2014] experiments.

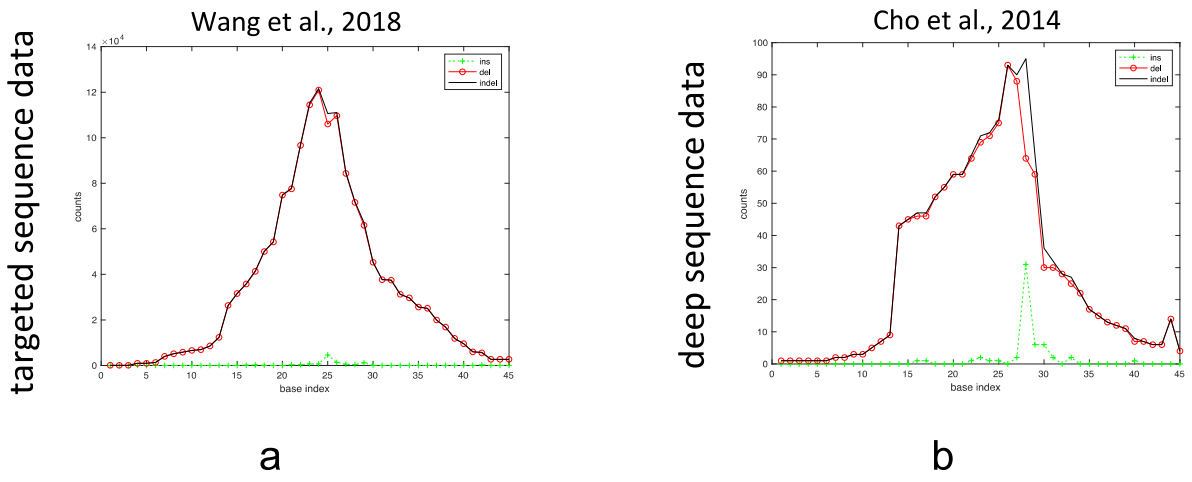

Figure A-3. Visualization of noise peaks at edit sites in two datasets, a) [Wang et al., 2018] and b) [Cho et al., 2014].

We were able to identify a clear pattern of insertions and deletions (indels) at target sites for CRISPR-Cas9 editing in the data analyzed early on (see Figure A-3). This identification and its associated visualization provided confidence and motivation for successive algorithms for edit detection.

In successive research, we developed and improved out data processing pipeline for noise characterization of sample reads, including both control and edit sample reads. We developed and implemented a sparse-data algorithm for characterizing noise in sample reads, which is much faster and uses less memory than a brute force method. We explored several windowed statistical processing methods and data smoothing techniques for analysis/visualization. We acquired data from the 1000 Genomes project [The 1000 Genomes Project Consortium, 2010; The 1000 Genomes Project Consortium, 2015] (https://www.internationalgenome.org) for control comparison and visualization using our noise characterization results. Our analysis here (see Figure A-4) show that deletions dominate, and the noise we see in the control data comes mostly from non-indel mismatches, referred to here as single nucleotide polymorphisms (SNPs), which suggest an important area of continued research.

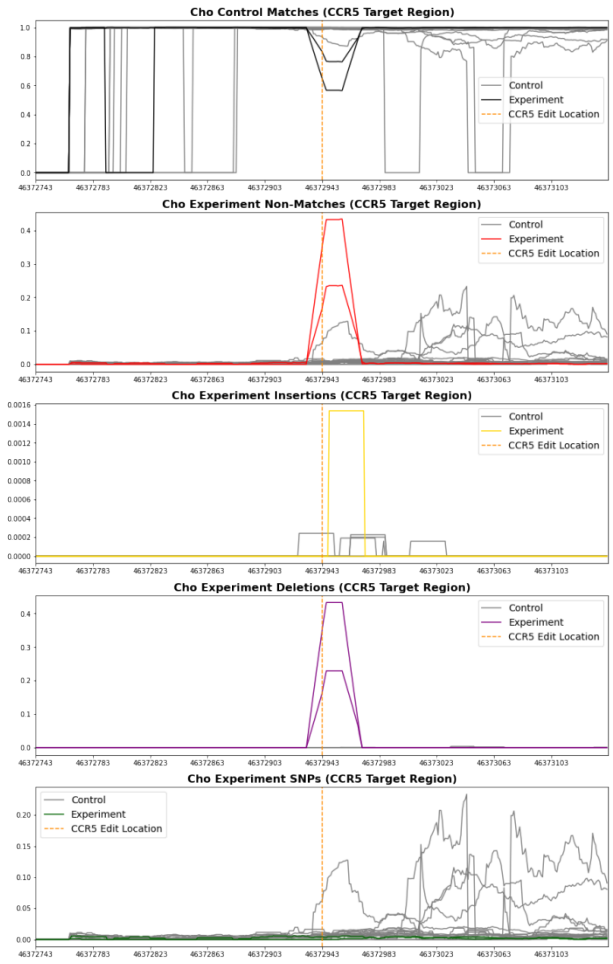

Figure A-4. Visualization of separation of noise at CCR5 edit site from [Cho et al., 2014].

As we expanded our research, we added counting of clips, both hard and soft, to our genomic noise counter (see Genome Noise Counter section). Clips can be important indicators of edit sites, where either the left or right side of the edit is repaired with another sequence, leading to either an inability to align the added sequence (indicated by soft clips) or multiple alignments for each port of the fused reads (indicated by hard clips). Note that in this latter case (hard clips), a homology driven repair (HDR) could have been the cause, especially if the knock-in sequence was provided intentionally during editing, which results in more than one read with multiple, overlapping alignments.

A.4. Visualization of Machine Error

There are many different (possible) kinds of machine error, but one that is easily visualized can occur in AMPLICON datasets, see Figure A-5.

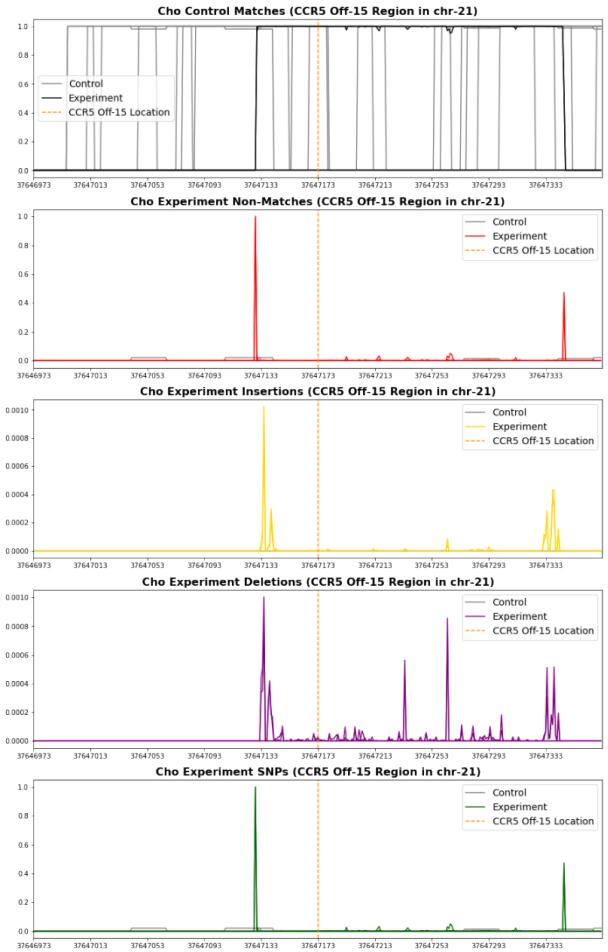

Figure A-5. Visualization of separation of noise at CCR5 Off-15 off-target site from [Cho et al., 2014].

In Figure A-5, we see a separation of signals from our genome counter (listed from top to bottom: matches, total non-matches, insertions, deletions, and mismatches (SNPs)), at an off-target site named CCR5 Off-15 by Cho and colleagues [Cho et al., 2014]. In the image across all of the non-match categories (below the top plot), there are significant noise signatures at the ends of the AMPLICON reads. Note that the region in Figure A-5 contains two sets of overlapping AMPLICON reads, where the right set of reads has a left end just left of center, and the left set of reads has a right end on the far right of the plots. We were able to deal with this kind of noise using quality trimming and filtering based upon quality scores.

Another kind of machine error can occur when certain nucleotides or groups of nucleotides have a greater propensity for manifesting noise, see Figure A-6.

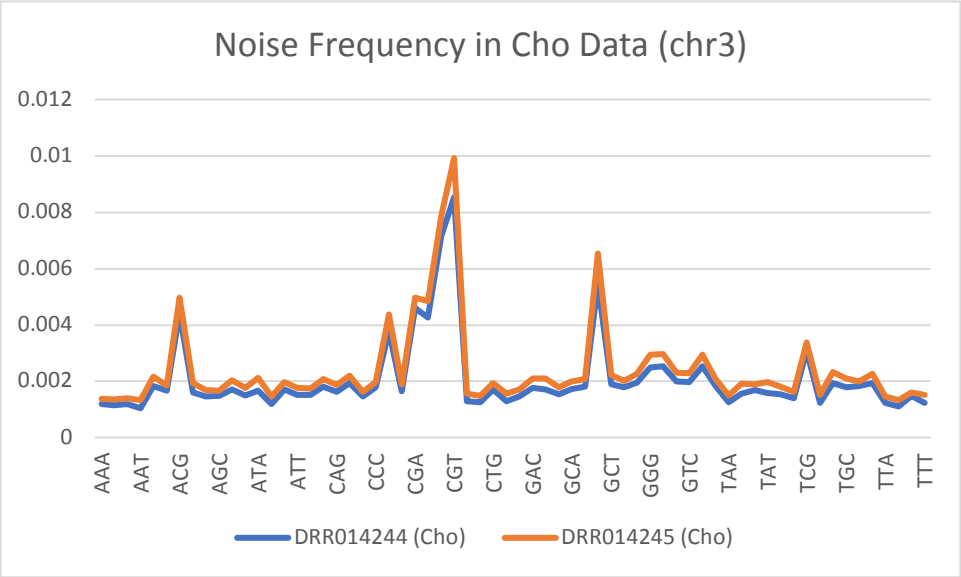

Figure A-6. Frequency for mismatch in chromosome 3 for each possible trimer across Cho datasets DRR014244 (no edit) and DRR014245 (edit) [Cho et al., 2014].

Figure A-6 shows the frequency for mismatch in trimers, or sequences of three bases, across all possible 3 base nucleotide sequences. Note that these are not amino acid sequences, but all possible trimers in scanning across an entire chromosome (chromosome 3 in this case). We employed data science and machine learning algorithms to handle this noise, especially since it is similar across both non-edit (negative) and edit (positive) samples.

A.5. Visualizing Natural Mutation

One form of natural mutation can be seen as a perfect mismatch from a reference genome at a single base position, see Figure A-7.

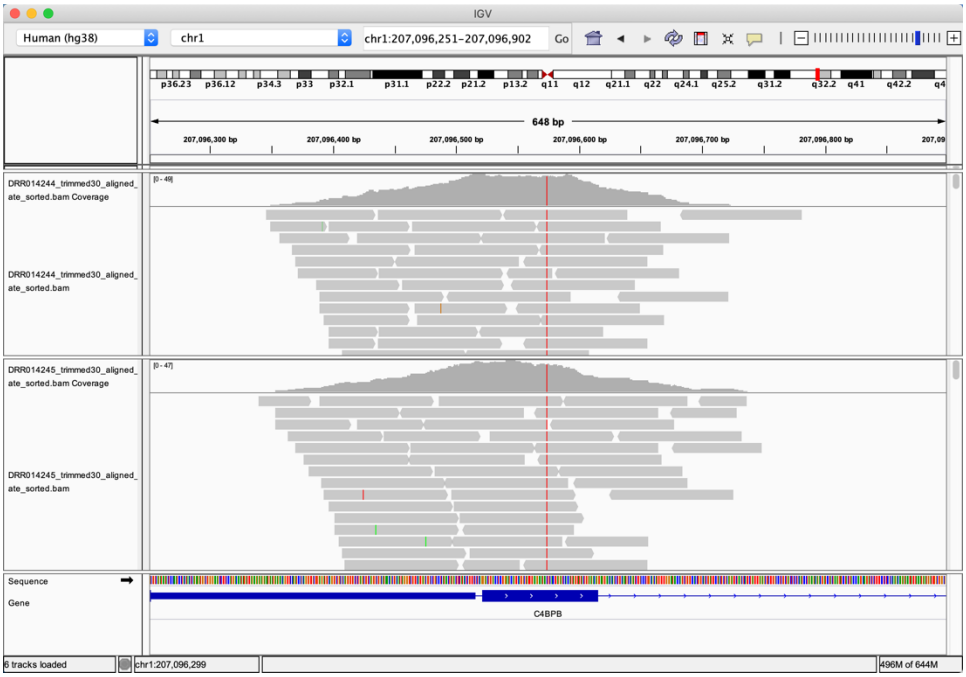

Figure A-7. View of area in chromosome 1, using IGV, with Cho datasets DRR014244 and DRR014245 [Cho et al., 2014].

The red line visible near the center of Figure A-7 (common in the same position in both datasets) shows a distinctive mismatch in both non-edit and edit sample files, which is most likely due to a natural mutation and difference from the reference (hg38) used for alignment. We employed both filtering (in anomaly detection) and machine learning to handle natural mutation in distinguishing it from targeted edits.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	01977	sanddocs@sandia.gov

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.