Analysis of Deep Learning Models and Hurdle Frameworks for CRISPR-Cas9 Cleavage Rate Prediction

Candidate no. 1063862

Word count: 8002 via Overleaf's word counter

Part C - Mathematics and Computer Science

Trinity 2025

Abstract

This thesis introduces a novel application of the hurdle model architecture to predict CRISPR-Cas9 cleavage rates under conditions of extreme class imbalance. Traditional machine learning models often fail to generalize on synthetic datasets in which over 90% of targets exhibit zero cleavage. However, by separating the prediction of cleavage occurrence (a classification task) from the prediction of cleavage magnitude (a regression task), the hurdle model effectively addresses the zero-inflated nature of the data to yield substantial performance improvements. We evaluate a range of deep learning architectures, including FNNs, CNNs, LSTMs, as well as conventional tree-based models, employed as both classifiers and regressors within the hurdle framework. The most effective configuration combines an LSTM classifier with a Gradient Boosting Regressor, achieving a Spearman rank correlation of 0.881. A stacking ensemble of 25 such models further enhances performance to 0.893. An ablation study highlights the critical role of thresholding in the hurdle model, while SHAP analysis identifies the individual models that most influence the ensemble's predictions.

Contents

1 Introduction

The CRISPR-Cas9 system was first derived from studying the adaptive system of bacteria such as E. coli in 1987. Originally functioning as a microbial defense mechanism against viral infections, this system has been repurposed as a tool for precise gene editing [1]. The CRISPR-Cas9 system relies on the Cas9 protein, guided by a single-guide RNA (sgRNA) to locate and cleave specific target DNA sequences complementary to the sgRNA. The cleavage rate, defined as the frequency or efficiency with which Cas9 cuts its intended DNA target, is a crucial metric since it directly influences both the specificity and the efficacy of gene editing.

Accurately predicting the cleavage rate for any given sgRNA and target DNA pair would revolutionize genome editing by enabling the design of highly efficient and precise sgRNA for any target DNA. For example, researchers could easily optimize the on-target editing efficiency rate for therapeutic or research applications, ensuring robust gene modifications with minimal experimental variability. Furthermore, by identifying sgRNA with minimal off-target cleavage activity, we can enhance the safety of CRISPR based medical therapies, by reducing the risk of mutations.

Experimental methods for measuring the cleavage rate do exist. For example, GUIDE-Seq [2] identifies double-stranded breaks by tagging, amplifying and sequencing them, with the number of sequence reads at each site providing an estimate of the cleavage frequency. However these experimental methods are expensive, labor-intensive, and time consuming, which makes large-scale data collection impractical in many cases. Hence, researchers are now primarily looking to use machine learning models, trained on the existing experimental data to predict the cleavage rate, which is faster, cheaper, and more scalable.

1.1 Current Research

Research in traditional machine learning approaches, such as those using logistic regression, support vector machines (SVMs), and random forests, has had success in predicting cleavage rates. For example, the SVM model in CRISPRpred(SEQ) [3] managed to achieve results close to those of deep learning models by using extensive feature engineering. However, the main drawback is that the performance of these models plateaus as the number of data points grow, since those models struggle to capture all the complex sequence patterns present in the data.

Alternatively, researchers are using deep learning models that can automatically learn the complex sequence patterns which influence cleavage activity. For example, Convolutional Neural Networks (CNNs) such as DeepSpCas9 [4] have proven particularly effective by scanning DNA sequences with multiple convolutional layers to identify local sequence motifs and position-dependent features critical for cleavage activity.

Similarly, Recurrent Neural Networks (RNNs) such as DeepHF [5], excel at modeling sequential dependencies in sgRNA-DNA interactions, capturing long-range patterns that affect binding and cutting efficiency. These deep learning models consistently outperform traditional machine learning methods by eliminating the need for manual feature engineering while achieving superior predictive accuracy.

Researchers are constantly developing increasingly complex models, in hopes of further increasing the state-of-the-art predictive accuracy. Hybrid models such as CRISPR-Net [6], which integrate CNNs with gradient boosted trees, and attention-based models like CrisprDNT [7], which employ self-attention mechanisms, exemplify the recent efforts to enhance predictive performance through architectural innovation. Furthermore, current model have tried to incorporate biologically relevent features such as epigenetic data, to better reflect the environment in which cleavage activity occurs, as in this paper [8]. As these computational approaches continue to evolve, they promise to deliver more accurate predictions, leading to more reliable tools for CRISPR-based genome engineering applications.

1.2 Contribution

Despite the effectiveness of both the traditional and deep learning models, a problem that remains underexplored is their performance on datasets with zero inflation, which is a situation where the majority of data points have a cleavage rate of exactly zero. We will see this problem in the expanded dataset used in this thesis, where we make the assumption that all our synthetic sgRNA-target DNA pairs have a cleavage rate of 0.

This assumption is based on the premise that modern detection techniques, such as GUIDE-Seq, are sensitive enough to capture even weak off-target activity. Hence, any sgRNA–target DNA pairs which are not detected in these datasets are likely to represent biologically inactive or irrelevant interactions. We do note that this assumption may overlook rare but real low-frequency cleavage events that escaped detection.

Previously mentioned models in the literature such as DeepSpCas9 and DeepHF demonstrate a high predictive accuracy on the benchmark datasets, which often contain balanced, curated data. However it turns out that both traditional and deep learning models struggle to handle imbalanced datasets with zero inflated distributions, because the models tend to overfit to the abundant zeros and fail to capture the sparse but important non-zero cleavage events, resulting in a much lower performance. There have been attempts to deal with imbalanced data sets, such as using oversampling or undersampling techniques [9], but I have not found a machine learning model in the literature that is explicitly designed to handle the extreme distribution imbalances which occur as a result of large-scale synthetic data generation.

To address the gap in the literature where existing models struggle with extreme class imbalance caused by zero-inflated distributions, this thesis makes the following key contributions:

- **Application of hurdle models** - Introduces a hurdle model architecture with a probability threshold to explicitly handle zero-inflated datasets, a currently unexplored method in CRISPR-Cas9 cleavage rate prediction.

- **Comparison of deep learning architectures** - Trains, evaluates, and compares the performance of 5 different machine learning models for both classification and regression tasks, to assess their suitability as components for the hurdle model.

- **Innovative combinatorial ensemble** - Develops a unique exhaustive framework that evaluates all 25 possible pairwise combinations of 5 classifiers and 5 regressors as hurdle models. This systematic exploration results in the creation of a much greater variety of predictive models, allowing the identification of the most effective classifier–regressor pairings.

- **Stacked ensemble approach** - Proposes and evaluates a stacked ensemble model trained off the prediction of all 25 hurdle models, showing that ensemble methods can further improve predictive performance over individual models.

2 Models

2.1 Hurdle Model Architecture for Zero-Inflated Data

The hurdle model, first introduced in 1971 [10], is a statistical model designed for data with an excess of zero outcomes. Since 91.8% of our expanded dataset is synthetic with a cleavage rate of 0, the Hurdle model is appropriate in our case. We can formally define the Hurdle model as follows:

Let $Y \in [0, 8]$ represent the cleavage rate, and let X denote the set of input features which are the encoded sgRNA and target DNA sequences in our case. The hurdle model assumes a two-part process for modeling the distribution of Y as follows:

$$P(Y \mid X) = \begin{cases} \theta(X), & \text{if } Y = 0 \\ (1 - \theta(X)) \cdot p(Y \mid Y > 0, X), & \text{if } Y > 0 \end{cases}$$

where $\theta(X)$ is the probability of observing cleavage rate 0 given features X, and $p(Y \mid Y > 0, X)$ is the conditional probability density (or mass) function of Y given that $Y > 0$. This density is modeled using a truncated Gaussian distribution, which is an appropriate assumption when $Y > 0$, due to the Box-Cox transformation which we will apply to our data when preprocessing (see section 3). We can now calculate the expected value of the cleavage rate given X as follows:

$$\begin{aligned} \mathbb{E}[Y \mid X] &= P(Y > 0 \mid X) \cdot \mathbb{E}[Y \mid Y > 0, X] + P(Y = 0 \mid X) \cdot \mathbb{E}[Y \mid Y = 0, X] \\ &= P(Y > 0 \mid X) \cdot \mathbb{E}[Y \mid Y > 0, X] \qquad (1) \end{aligned}$$

Here we use the fact that $\mathbb{E}[Y \mid Y = 0, X] = 0$ to cancel out the equation, since by definition, if $Y = 0$, then its conditional distribution is simply 0.

If we are able to find functions f and g where $f(X) = P(Y > 0 \mid X)$ and $g(X) = \mathbb{E}[Y \mid Y > 0, X]$, then we would be able to find the expected cleavage rate using $\mathbb{E}[Y \mid X] = f(X)g(X)$.

Now with our input features X, we can use supervised machine learning techniques to find the component models f and g. Since f is the probability of a non-zero cleavage event taking values between 0 and 1, we can model it using a binary classification model, where the target label is 1 if $Y > 0$ and 0 otherwise. Similarly, since g is the expected magnitude of the cleavage rate conditioned on being non-zero, we can model it using a regression model but trained on only the subset of data where Y $>$ 0.

Then the hurdle model's final prediction can be formulated as the product of the predictions of two trainable models: a classification model f and a regression model g. Hence, we can train a variety of regression and classification models, and then use a combinatorial approach to find the pair of models which combine to give the hurdle model the best possible performance.

2.2 Deep Neural Network Architectures

- **FNN model** - We first build a simple Feedforward Neural Network (FNN), which will process our sequence globally. It does so by first flattening our input matrix, then passing the result into two fully connected layers: the first with 128 units and the second with 64 units. Each dense layer is followed by a ReLU activation function to introduce non-linearity. The final output layer consists of a single neuron that produces the predicted value. This model is designed to act as a baseline, allowing us to assess whether the more complex deep learning models below can provide meaningful improvements by taking into account local patterns or long-term dependencies.

- **CNN model** - We then attempt to build a Convolutional Neural Network (CNN), which has been taken and adapted from the following paper [11]. The idea is that by encoding our input as a 7×23 input matrix (see section 3 for the exact sequence encoding), the CNN can process it just like a 7×23 image. The hope is that our CNN architecture is able to extract diverse local patterns from the input sequences to make its predictions.

The input first passes through three convolutional layers with different kernel sizes (1, 3 and 5), each using ReLU activations. The different kernel sizes are used to capture the different length patterns in the DNA sequence. For example, a kernel size of 1 could capture individual nucleotide effects, while a kernel size of 5 could detect more complex interactions. This way, we are able to learn both the short and the long range dependencies in the sequence which may influence the cleavage rate.

This output is then passed through batch normalization, which stabilizes and accelerates training. Next, a max-pooling layer reduces the sequence dimension to retain only the most important features. These features are passed into two fully connected layers. Finally, a dropout layer with dropout rate 0.2 is applied before the final output to prevent overfitting by randomly zeroing out a portion of the learned weights during training.

- **RNN model** - Now we construct an RNN model ourselves, by adapting common RNN architectures. More specifically, we use an LSTM [12] which is a specific type of RNN designed to remember long-term dependencies. This is achieved through the design of the LSTM layers, which use the tanh activation internally and includes built-in forget gates to manage these long-range dependencies. Our encoding of our input, a 7×23 input matrix means the RNN can treat the sequence as a series of 23 different time steps, each with 7 features. The hope is that the LSTM can extract patterns in how earlier sequence elements can influence later ones to make its predictions.

  The model begins with two stacked LSTM layers, each with 64 hidden units. Importantly, we require the first layer to return the hidden state at each time step (not just the final one), which allows the next LSTM layer to analyse patterns across all time steps in the output. For example, we hope that the first LSTM might capture local nucleotide interactions while the second LSTM might learn how these interactions combine or influence each other across the full sequence to affect cleavage rate. Each LSTM layer is followed by a normalization layer, which stabilizes and accelerates training.

Similarly to the CNN, this output is passed through two fully connected layers followed by a dropout layer with droupout rate 0.15, and then projected onto the final neuron. However, at the second fully connected layer, I've incorporated a residual skip connection, where the input to the layer is added to its output before passing forward. This idea is borrowed from ResNet architectures [13], and has been shown to improve learning efficiency and convergence by allowing deeper layers to refine earlier representations instead of learning entirely new transformations.

All three models above were implemented with Tensorflow and are designed to be used for both regression and classification tasks. For regression, the final output neuron produces a scalar value corresponding to the predicted cleavage rate. For classification, a sigmoid activation function is added to the final neuron, transforming the output into a probability between 0 and 1, indicating the likelihood of a non-zero cleavage event.

2.3 Model Training Configuration

We base our training configurations primarily on industry-standard best practices, although we do require the use of a custom classification loss function to meet the unique challenges of our zero-inflated dataset. Through preliminary experiments, I verified that the following configurations consistently resulted in a stable convergence and strong performance for all the deep learning models in this paper.

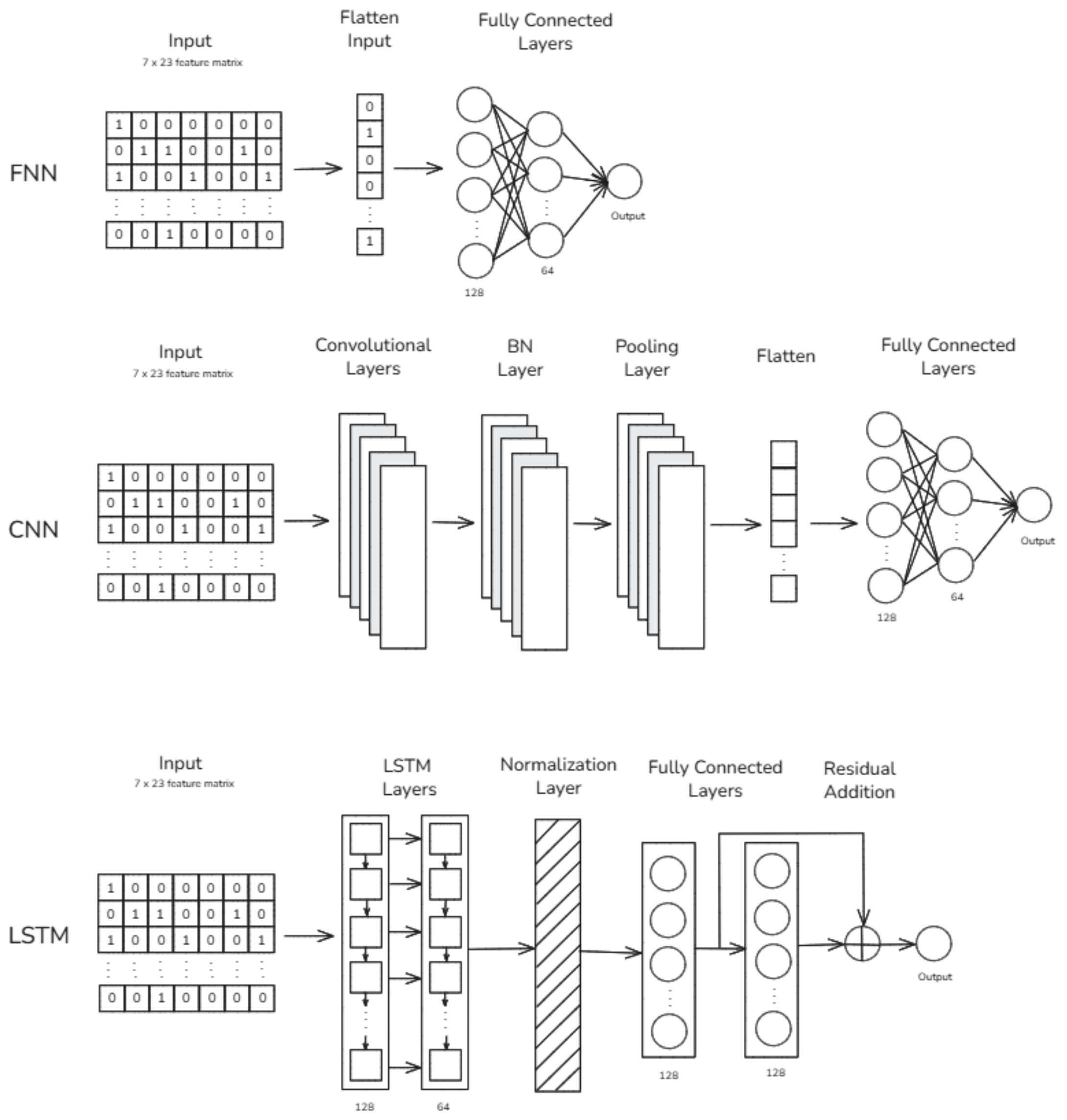


Figure 1: Architectures of the three deep learning models used in this thesis. All models take a 7×23 sequence encoding matrix as input. The FNN flattens the input and passes it through fully connected layers. The CNN applies convolutional and pooling layers to capture local patterns before flattening and classification. The LSTM model processes the input sequentially, followed by normalization, dense layers, and residual addition to capture long-range dependencies.

- **Batch size** - We use a batch size of 32, which provides enough samples to compute a stable gradient estimate.
- **Optimizer** - We use the Adam optimizer [14] with the default settings, since it's efficient, requires minimal tuning and converges quickly.

- **Learning rate** - We use a learning rate of 10^{-3}, the default setting for Tensorflow.
- **Number of epochs** - We set the maximum number of epochs as 20, since our preliminary experiments showed that model performance consistently plateaued before reaching 20 epochs for all models. For performance reasons, we implemented an early stopping mechanism (see 3), so this maximum is almost never reached.
- **Loss functions** - We used Mean Squared error loss (MSE) for regression and focal loss for classification. We will describe our choice of loss functions in more detail below.

In our code, we shared configurations such as loss functions, optimizers, and early stopping logic across all our deep learning models. This minimized code duplication, made it easy to swap in new variants, and ensured consistent comparisons between the architectures.

2.4 Loss Functions Selection

In our deep learning models, the choice of the loss function depends on whether the model is used for regression or classification.

- **Regression** - We choose to use the MSE loss, which takes the square of the difference between the predicted value $\hat{y}$ and the actual value y. Mathematically:
$$\text{MSE}_{\text{total}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$
This is a standard choice for regression and is appropriate here because, after the Box-Cox transformation, the non-zero cleavage rates approximate a Gaussian distribution, making MSE optimal under the assumption of normally distributed errors.

- **Classification** - We choose to use the focal loss [15] for classification between the predicted probability $\hat{y}$ and the actual value y. This loss function addresses the class imbalance in classification tasks by down-weighting well-classified samples to improve the performance on the minority class. This loss is especially suitable in our case due to our biased dataset where over 90% of our cleavage rates are 0. Mathematically, the loss is defined by:

$$\mathrm{FL}_{\text{total}} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{FL}(y_i, \hat{y}_i)$$

where:

$$\mathrm{FL}(y_i, \hat{y}_i) = \begin{cases} -\alpha(1-\hat{y}_i)^{\gamma}\log(\hat{y}_i) & \text{if } y_i = 1 \\ -(1-\alpha)\hat{y}_i^{\gamma}\log(1-\hat{y}_i) & \text{if } y_i = 0 \end{cases}$$

In this formulation, $\alpha \in [0,1]$ is a weighting factor that balances the importance of the positive vs the negative examples and $\gamma \geq 0$ is the focusing parameter that reduces the relative loss for the well-classified examples (i.e., those with high confidence predictions). We will choose values $\gamma = 3$ (since it has been shown empirically to work well in similar class-imbalanced settings) and $\alpha = 0.9$, since this increases the influence of the minority class of datapoints with a positive cleavage rate.

2.5 Traditional Machine Learning Models

In addition to our deep-learning models, we trained and evaluated a set of traditional machine learning algorithms using scikit-learn. Note that for all of these models, the 7×23 input matrix is flattened into a 1 dimensional feature vector prior to training.

- **Random Forest Regressor** - An ensemble method of regression which constructs multiple decision trees during training, each trained on a different subset of the data. We used 100 trees, no maximum depth, and the MSE as the split criterion. During testing, each tree predicts a numerical value for the cleavage rate. Then the result is obtained by calculating the average of the numerical prediction of all the trees.

- **Gradient Boosting Regressor** - An ensemble method that instead builds decision trees sequentially, where at each step we fit a new tree to correct the residual errors made from the prior trees. It does so via gradient descent by minimizing the MSE loss at each step. We configured 300 estimators, a learning rate of 0.1 and a maximum depth of 5 for each tree.

- **Random Forest Classifier** - Similar to the random forest regressor, except that each tree votes on the class label. We used 100 trees, no maximum depth and the Gini impurity as the split criterion. During testing, each tree predicts whether the cleavage rate is positive or not. The model then outputs the proportion of positive predictions as the probability score.

- **Gradient Boosting Classifier** - Similar to the gradient boosting regressor, but instead trained to minimize the logistic loss via gradient descent. We use the same configuration as the regression model: 200 estimators, a learning rate of 0.1, and a maximum tree depth of 5.

In addition to the models described above, several other traditional machine learning algorithms were considered but excluded after initial testing. For example, linear and logistic regression models, even with regularization, underperformed due to their inability to capture the complex relationships present in biological sequence data, leading to their exclusion from the final model selection.

Support vector machines (SVMs), including the Support Vector Regressor (SVR) and Support Vector Classifier (SVC), were also initially considered due to their past success in CRISPR prediction tasks such as CRISPRpred(SEQ). However, these models scale poorly with dataset size, particularly during inference. Since our synthetic dataset contains over 300,000 examples, the evaluation times for SVMs became impractically long. In our use case with the hurdle model, each trained classifier must be evaluated alongside every regressor, resulting in each classifier being evaluated up to five times. The same applies to every trained regressor. This multiplicative computational burden makes SVM based approaches prohibitively expensive for our pipeline, also ultimately leading to their exclusion from the final model selection.

Table 1: Summary of the Traditional and Deep Learning Models for Regression vs. Classification Tasks used in this thesis

Regression Models	**Classification Models**
Random Forest Regressor	Random Forest Classifier
Gradient Boosting Regressor	Gradient Boosting Classifier
FNN Regressor	FNN Classifier
CNN Regressor	CNN Classifier
LSTM Regressor	LSTM Classifier

3 Training and Evaluation

3.1 Data Processing Pipeline

Before we train, our models, we perform the following steps to prepare our dataset:

1. **Data Acquisition** - The data source for this thesis is taken from the crisprSQL database [16]. This database consists of experimental data from 17 different CRISPR-Cas9 studies and includes 25,632 different guide-target pairs from humans and rodents, of which 25,554 contain cleavage frequency data. I chose to use this dataset to obtain the data required to train our models, since it provides one of the most comprehensive collections of CRISPR-Cas9 data available while being easy to download and use. We discard all guide–target pairs in this dataset which lack a measured cleavage rate, retaining only the valid data points for training.

2. **Box-Cox transformation** - If we plot a histogram of the cleavage rate frequency distribution (see figure 2), we see a large positive skew, as more than 24,000 of our datapoints have a cleavage rate between 0 and 0.1, which forms a very imbalanced dataset. Hence, we first apply a Box-Cox transformation [17] to the data. This optimizes the skewed distribution to approximately a Gaussian distribution with mean around 0 and standard deviation around 2. This step is important because many machine learning algorithms, particularly those using mean squared error (MSE) loss, assume that target values are approximately Gaussian-distributed.

3. **Clipping and shifting** - The cleavage rate is then constrained to the range of $[-4, 4]$, which prevents extreme outliers from skewing the analysis while still capturing over 99% of the expected variance. Next, we apply a uniform shift of +4 to all values, ensuring that cleavage rates of 0 in the original data remain unchanged, mainly to improve interpretability.

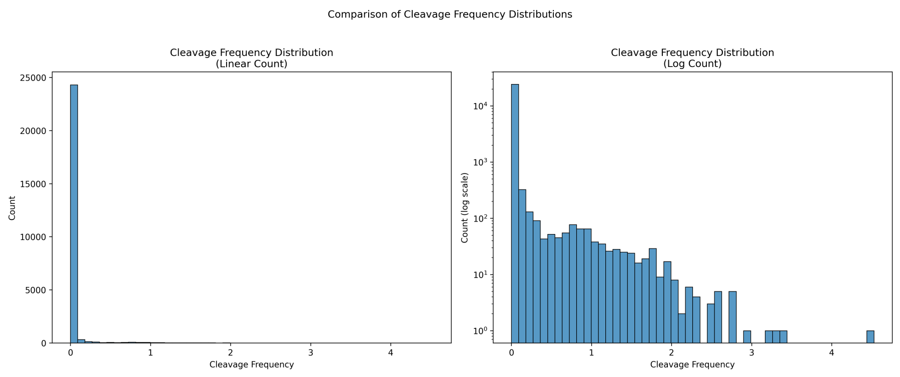


Figure 2: A plot of the cleavage rate frequency distribution on a linear scale (left) and a log scale (right), highlighting the extreme zero inflation in the dataset.

4. **Synthetic data generation** - To augment our dataset with synthetic sequences, we employ BATMis [18] which is an efficient tool for aligning short DNA reads to a reference sequence while allowing up to k-mismatches. In our case, we use $k = 5$, which means that we consider alignments with up to 5 mismatches between the sgRNA and the target DNA. This allows us to generate another 284,510 guide-target pairs. We can set the cleavage rate of each one to 0, making the assumption that these guide-target pairs would have been detected in the original experiments if they were biologically active. Altogether, this totals to 310,064 datapoints.

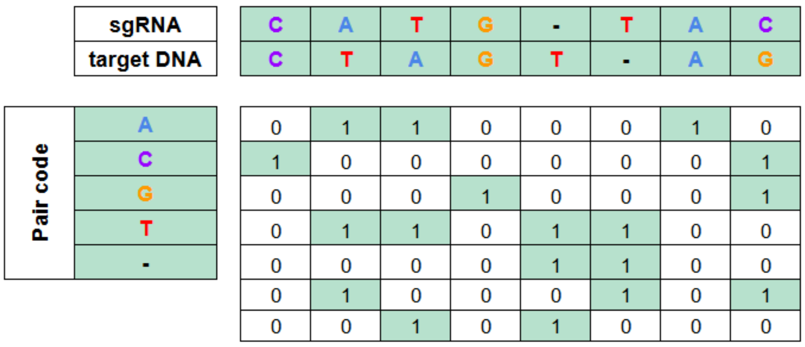
5. **Sequence encoding** - We apply the method of data encoding from CRISPR-Net [6], where each sgRNA-target pair is encoded as a 7x23 matrix. Here is the process of how this works:

   (a) Nucleotides of both sgRNA and target DNA are mapped to a 5-bit one-hot vector as follows: A->[1,0,0,0,0], C->[0,1,0,0,0], G->[0,0,1,0,0], T->[0,0,0,1,0], and missing nucleotides (indels) are mapped to [0,0,0,0,1].

   (b) Apply a logical OR to the one-hot encodings of the sgRNA and target DNA sequences channel-wise so that each position now reflects whether a given nucleotide appears in either sequence.

   (c) Capture directional information in two additional channels by assigning each nucleotide and indel an integer rank, and then comparing these ranks. If the sgRNA base's rank is higher than the target DNA's, it returns [1,0]. If lower, [0,1]. If equal (a perfect match), [0,0].

Compared to the 10-bit encoding where the two 5-bit sequences are concatenated channel wise, the 7-bit encoding is much faster to train since it reduces the number of model parameters. Furthermore, this method of encoding is still injective, which guarantees that there is no ambiguity as two distinct sequence pairs are never mapped to the same feature vector, so all the relevant features are preserved.



sgRNA	C	A	T	G	-	T	A	C
target DNA	C	T	A	G	T	-	A	G

Pair code								
A	0	1	1	0	0	0	1	0
C	1	0	0	0	0	0	0	1
G	0	0	0	1	0	0	0	1
T	0	1	1	0	1	1	0	0
-	0	0	0	0	1	1	0	0
	0	1	0	0	0	1	0	1
	0	0	1	0	1	0	0	0

Figure 3: A diagram showing sgRNA-target pairs as 7×23 matrices using a 5-bit one-hot scheme (A/C/G/T/indel), applying a logical OR between sequences plus two directional channels comparing nucleotide ranks. This compact 7-bit encoding preserves injectivity while being more parameter-efficient than 10-bit concatenation, maintaining all sequence features including mismatches and indels.

6. **Data Transformation** - For training the regression model, a subset of the data containing only non-zero cleavage rates is created. This truncated dataset enables the regression model to focus solely on the points where cleavage activity is predicted. For training the classification model, a binary dataset is created, where the target label is 1 if cleavage activity is nonzero and zero otherwise. This enables the classification model to focus on only distinguishing between the sgRNA pairs with and without cleavage activity.

7. **Train/validation/test split** - Before training, all our datasets are randomly

split into 80% training set and 20% test set, where the test set is left strictly for final evaluation to assess model generalization on unseen data. It is important here that we used stratified sampling, to preserve the original class distribution in both the training and test sets, preventing any imbalance that could skew the model's performance. The 80% training portion is then further divided into an 80% train subset and a 20% validation subset, with the validation set used exclusively during development for hyperparameter tuning and model selection, to ensure that no information from the test set leaks into the iterative training process.

3.2 Model Training and Optimization

We first iterate over all the available classification models, training each model in turn on the binary dataset using the given model configurations (i.e. focal loss, Adam optimizer, etc.). The hyperparameters are then tuned via grid search on the validation set, with selection based on the maximal F1 score (more details below). For simplicity, we will keep the same hyperparameters between each classification model and its corresponding regression model, and only optimize them for the classification model. While it is theoretically possible that separately tuning the regression model might result in further performance improvements, our experiments in section 4 show that the overall performance of the hurdle model is dominated by the classifier component, suggesting that additional gains would likely be minimal.

We then iterate over all the available regression models, training each model in turn on the truncated (non-zero) dataset using the model configurations outlined in section 2 (MSE loss, Adam optimizer, etc.). No further hyperparameter training is required here. This two-stage process results in five trained models per task: regression (predicting cleavage magnitude) and classification (predicting cleavage occurrence).

Note that while training our deep learning models, we use early stopping, which is a regularization technique that prevents overfitting by terminating training when validation performance plateaus. In our implementation, we use a custom early stopping mechanism that tracks task-specific metrics. More specifically, we track the Spearman correlation for regression and the F1 score for classification (more details below). We set a patience of 3, which means that if no improvement of these metrics is seen for 3 consecutive epochs, training halts before reaching the maximum 20 epochs. By halting training before overfitting occurs, early stopping helps both improve model generalization to unseen data and also reduce unnecessary computation, resulting in more efficient training overall.

3.3 Model Evaluation and Performance Metrics

After training, we want to assess the performance of each regressor and classifier on the test data by computing an evaluation metric. In this section, I will introduce the Spearman rank r_s for regression performance, the F1 score for threshold-dependent classification performance, and finally the PR AUC for evaluating performance across all possible classification thresholds.

- **Regression** - The standard practice for CRISPR-Cas9 regression models is to evaluate performance using the Spearman rank correlation coefficient r_s, which measures the strength and direction of the monotonic relationship between two variables.

  Mathematically given datapoints $X = (x_1, \ldots x_n)$ and $Y = (y_1, \ldots, y_n)$, the Spearman Rank is computed by:

  $$r_s = \frac{\mathrm{Cov}(R_X, R_Y)}{\sigma_{R_X}\sigma_{R_Y}}$$

  where $R_X = (R(x_1), \ldots, R(x_n))$, $R_Y = (R(y_1), \ldots, R(y_n))$ are the ranks of X and Y respectively, $\mathrm{Cov}(R_X, R_Y)$ is the covariance between rank variables, and σ_{R_X} and σ_{R_Y} are the standard deviations of the ranks.

Since we can assume that no two cleavage rates are equal for the non-zero cleavage rates the regression models are trained on, there are no tied ranks in our data, so we can use a more computationally efficient formula given by:

$$r_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n\left(n^2-1\right)}, \quad d_i = R(x_i) - R(y_i).$$

where n is the number of datapoints.

- **Classification** - Evaluating a classification model relies on forming a confusion matrix. This is a table that visualizes the performance of a classification model by comparing its predicted values to the true values. In our binary cleavage rate classification example, the confusion matrix has the following values:

  1. True positive (TP) - Correctly predicted cleavage rates > 0
  2. False positive (FP) - Incorrectly predicted cleavage rates > 0
  3. True negative (TN) - Correctly predicted cleavage rates $= 0$
  4. False negative (FN) - Incorrectly predicted cleavage rates $= 0$

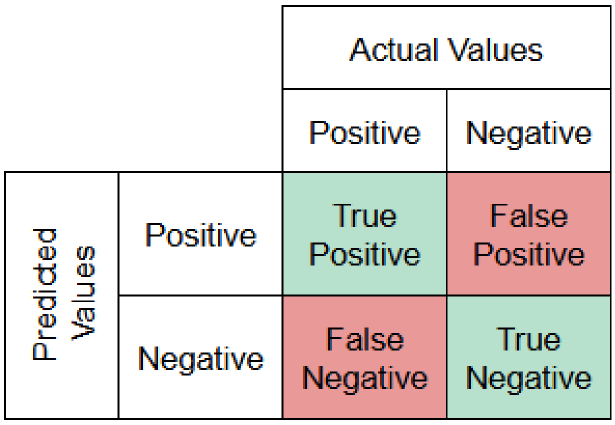


Figure 4: A diagram of a confusion matrix.

The precision, which measures the proportion of predicted positives that are actually positive, is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

The recall, which measures the proportion of actual positives correctly identified, is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

All classification models have a precision-recall tradeoff, since making the model more inclined to choose positives tends to increase recall (catches more true events) but also decreases precision (increased false positives). Hence, we can use the F_1 score, defined by:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

to provide a single metric that balances both concerns, which we will use as one of the overall metrics of the model's performance.

An alternate metric we could use is the area under the precision-recall curve (PR AUC). To calculate this, we threshold our model's predicted probabilities at various levels to generate different binary classification outcomes, and for each threshold, the corresponding precision and recall values are computed by comparing the predicted classes to the true labels. These precision recall pairs are then used to plot the precision-recall curve, and the area under this curve is calculated (usually using the trapezium rule) to obtain the PR AUC score.

While the F1 score captures the precision and recall at the specific threshold we are using (0.5), the PR AUC summarizes the precision-recall trade-off across all thresholds, providing a broader view of model behavior. Using both metrics allow us to make a more complete evaluation.

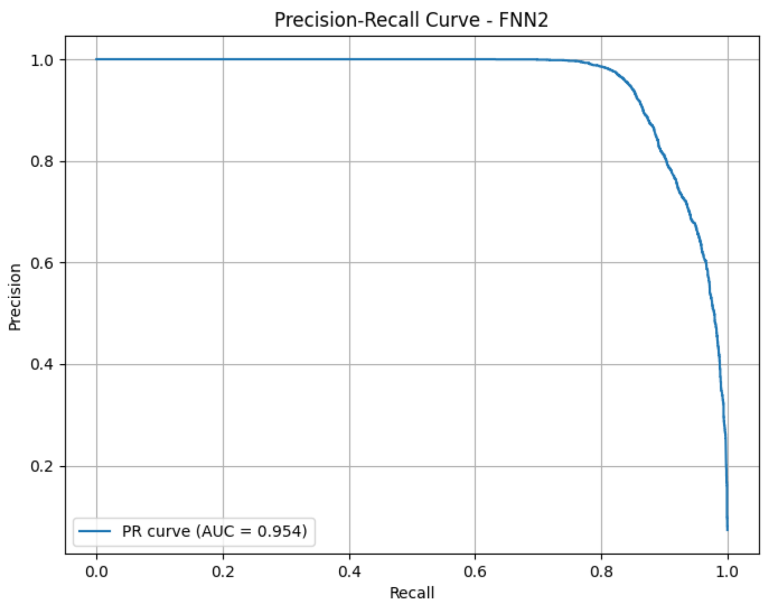


Figure 5: Precision-recall curve for the FNN classifier on the test set. The area under the curve (PR AUC) is 0.954, indicating a strong performance in identifying positive cleavage events despite the severe class imbalance. Precision remains high across a wide range of recall values, demonstrating the model's reliability in detecting true positives with minimal false positives.

Note that we decided not to use the commonly reported classification metric ROC AUC, because this metric can be misleading in highly imbalanced datasets like ours, where the majority of cleavage rates are zero. In this setting, a classifier can achieve a deceptively high ROC AUC by correctly identifying true negatives, even if it fails to detect any positive cleavage events.

3.4 Hurdle Model Implementation and Thresholding

Recall from Section 2 that the hurdle model is defined by combining two components: a classification model f and a regression model g, and cleavage rates are predicted by multiplying the results of the two models together. Since we have already trained five distinct models for each component (five classifiers and five regressors), we can form 25 distinct hurdle models pairing each classifier with each regressor. This process requires no additional training. We can then evaluate all 25 models on the validation data using the Spearman Rank r_s, select the top-performing model (highest r_s) and report its performance on the test data.

The biggest problem with our approach is that the model g is not trained on the subset of data where $Y = 0$, meaning it has a very limited ability to predict cleavage rates for those cases. For example, a problem arises when f outputs a low but non-zero probability while g predicts an unrealistically high cleavage rate, leading to a dubious final prediction. This can happen since g is only trained on positive examples and extrapolates poorly for cases where f has a low confidence that the cleavage rate is positive.

Hence, our solution is to introduce a threshold hyperparameter M in our classification model f so that if $f(X) < M$, then we override the prediction as 0. This ensures that only high-confidence predictions that the cleavage rate is positive from f contribute to the regression output. We chose the threshold $M = 0.4$ based on our tests on the validation dataset.

This threshold parameter is essential for our hurdle model to work well, as confirmed by our ablation study in section 4. In fact, without this threshold hyperparameter, our hurdle model performs no better than a standard CNN model. I suspect that this is because a standard CNN with enough parameters is already able to approximate hurdle-like behavior implicitly by learning a flexible decision boundary, but it struggles to explicitly enforce a sharp confidence threshold like M due to the continuous nature of its output probabilities.

3.5 Stacking Ensemble Methodology

We designed our individual models to capture different patterns in the data. For example, our CNNs can identify local patterns and our RNNs excel at detecting long-range sequential dependencies. By combining all 25 hurdle models through stacking, we hope to leverage each model's strengths and mitigate their individual weaknesses, resulting in an ensemble model which surpasses the performance of

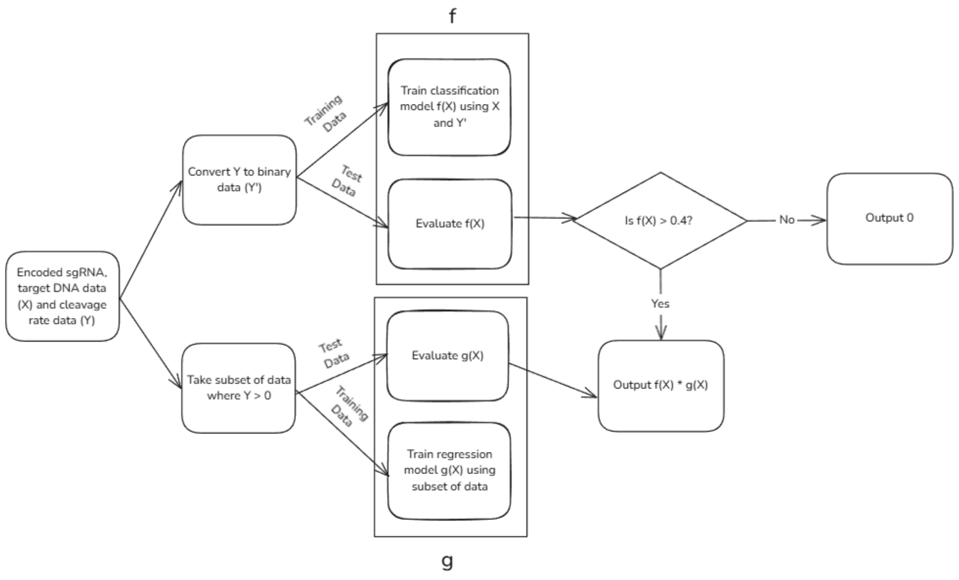

f
Train classification model f(X) using X and Y'
Training Data
Test Data
Convert Y to binary data (Y')
Evaluate f(X)
Is f(X) > 0.4?
No
Output 0
Yes
Encoded sgRNA, target DNA data (X) and cleavage rate data (Y)
Test Data
Training Data
Take subset of data where Y > 0
Evaluate g(X)
Output f(X) * g(X)
Train regression model g(X) using subset of data
g

Figure 6: Diagram of the hurdle model architecture used in this thesis. The input consists of encoded sgRNA and target DNA sequence data X along with cleavage rate data Y. A binary label Y' is created to train a classifier $f(X)$ to predict whether cleavage occurs. A separate regression model $g(X)$ is trained on the subset of data where $Y > 0$ to predict the magnitude of cleavage. During inference, if the classifier confidence $f(X)$ exceeds a threshold (0.4), the final output is computed as $f(X) \cdot g(X)$; otherwise, the output is set to zero.

any individual base model. The ensemble model operates as follows:

1. **Base-Model Predictions** - We first modify the code for the hurdle model so that each of the 25 hurdle models now output a predicted cleavage rate for every guide–target pair in the validation and test sets. Now, save these predictions in separate files. Note that we also need to include the true cleavage rates for the validation set.

2. **Ensemble Model Training** - Assemble a new feature matrix using the saved validation data, where each column is the prediction of one base model and each row corresponds to a guide–target pair. The target vector is the true cleavage rate. Given the continued zero-inflation in the aggregated data, a hurdle model remains the natural choice for our ensemble framework.

   - **Random Forest Classifier** - This component uses 100 trees with a maximum depth of 5. We train it to classify whether the cleavage rate is greater than zero by assigning a label of 1 to positive cleavage rates and 0 otherwise.

- **Gradient Boosting Regressor** - We train this model to predict the cleavage rate on the subset of data where the true cleavage rate is greater than zero. This component uses 100 trees with a learning rate of 0.1, a maximum depth of 3, and a subsample rate of 0.8.

3. **Final Prediction** - Using the saved test data, we can take the 25 predicted cleavage rates from the base hurdle models as the input feature matrix. Then the trained stacking hurdle model can predict the cleavage rate by combining the classifier and regressor outputs using the same method as described above. A Spearman Rank test can then be used to evaluate this model's performance.

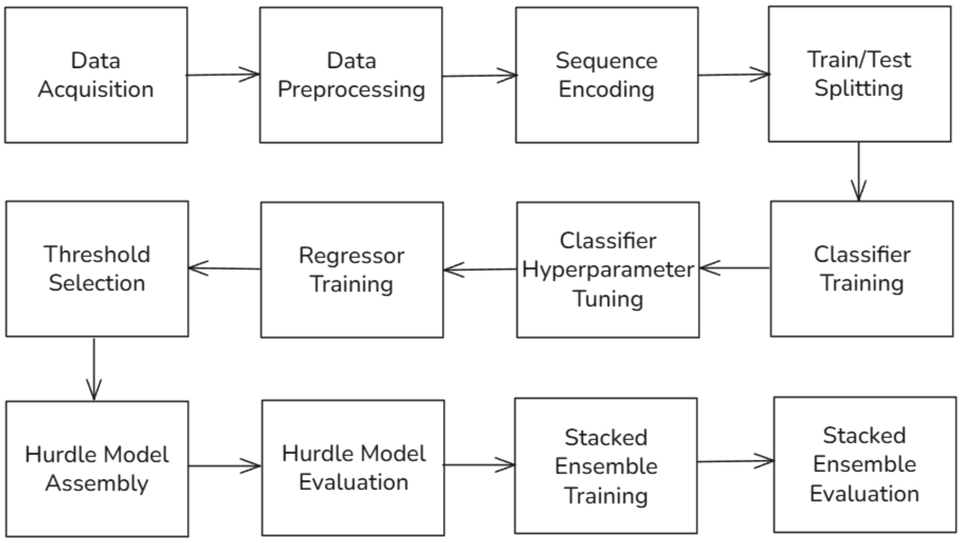


Figure 7: Overview of the training and evaluation pipeline used in this thesis. The process begins with data acquisition and preprocessing, followed by sequence encoding and stratified train/test splitting. Classifier and regressor models are trained independently, with classifier hyperparameters tuned and a threshold selected to form the hurdle model. The assembled hurdle models are then evaluated and used to train a stacked ensemble, which is subsequently evaluated on the test set.

4 Results

4.1 Component Analysis: Classification Stage Performance

We first show a summary of the performance of the individual classification models that make up the hurdle model. This step ensures that each component of the hurdle model performs well for themselves before integration, as errors in the classification stage now would propagate and distort the final hurdle model results later.

Table 2: Comparative Performance of Classification Models on Key Metrics

Model	**Precision**	**Recall**	**F1 Score**	**PR AUC**
Random Forest Classifier	0.9429	0.7912	0.8604	0.9297
Gradient Boosting Classifier	0.9612	0.7930	0.8690	0.9316
CNN Classifier	0.9770	0.7983	0.8787	0.9467
LSTM Classifier	0.9631	0.8350	0.8945	0.9508
FNN Classifier	0.9431	0.8503	0.8943	0.9537

From the data in table 2, we see that the LSTM classifier and the FNN classifier have the best overall performance, since both models achieve very high F1 scores (0.8945 for LSTM and 0.8943 for FNN), indicating a strong balance between precision and recall. However the difference between them is that the FNN classifier outperforms in recall (0.8503 vs 0.8350), suggesting it manages to identify more true positives, whereas the LSTM classifier outperforms in precision (0.9631 vs 0.9431), suggesting that it makes fewer false positive predictions. This implies that the LSTM classifier is more conservative but more precise, whereas the FNN classifier may detect more actual cleavage events at the expense of a few more false alarms. Both models outperform others in PR AUC as well, showing that they maintain strong precision-recall trade-offs across different classification thresholds.

The CNN classifier is a solid third choice. Despite its lower F1 score of 0.8787, it has the highest precision of all the models at 0.9770, suggesting that it is very selective and rarely makes any false positive predictions. Conversely, traditional machine learning models such as the Random Forest Classifier and the Gradient

Boosting Classifier clearly show a weaker performance, as their F1 scores (0.8604 and 0.8690, respectively) and PR AUC scores (0.9297 and 0.9316, respectively) are noticeably lower, which illustrates their limitations in capturing complex patterns within the dataset.

4.2 Component Analysis: Regression Stage Performance

Similarly we show a summary of the performance of the individual regression models which make up the hurdle model.

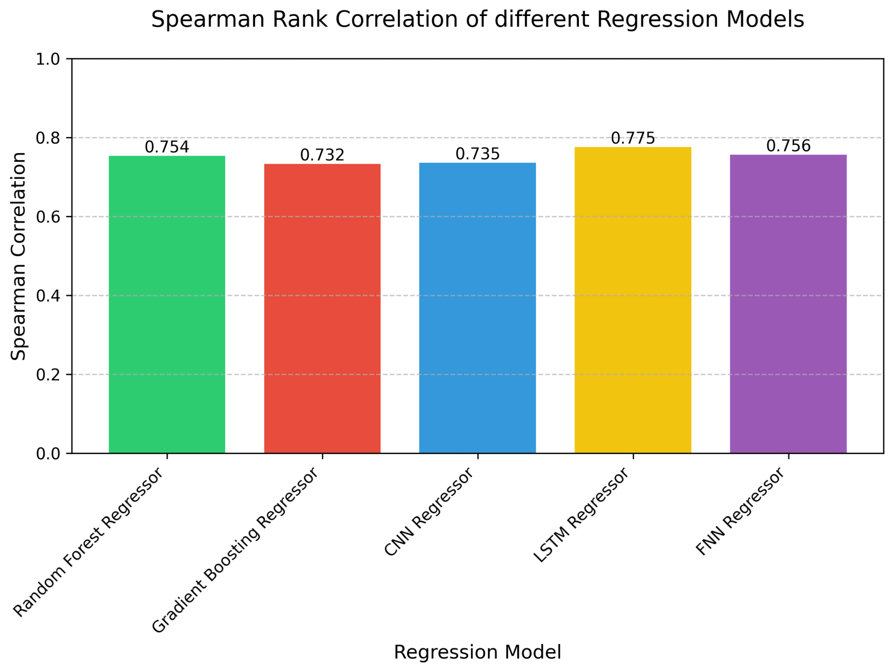


Figure 8: Spearman rank correlation of different regression models on the test set. The LSTM regressor achieved the highest correlation (0.775), followed by the FNN (0.756) and Random Forest (0.754) regressors. CNN and Gradient Boosting regressors performed slightly lower, at 0.735 and 0.732 respectively.

We can rank the regression models based on their performance with the Spearman rank correlation, using the results shown in figure 8. The LSTM Regressor ranks the highest with a Spearman correlation of 0.775, indicating the strongest ability to capture the correct order of cleavage rates. It is followed by the FNN Regressor (0.756), the Random Forest Regressor (0.754), the CNN Regressor (0.735), and finally the Gradient Boosting Regressor (0.732).

An observation is that all our Spearman correlation values fall within a relatively narrow range (0.732 to 0.775), suggesting that all the models perform similarly. We also see that the Random Forest Regressor actually slightly outperforms the CNN Regressor, which is surprising given the expectation that deep learning models are able to capture more complex data patterns. I suspect that both these observations are due to the fact that the regression was conducted on a subset of the data containing only non-zero cleavage rates, which is less than 10% of the total dataset. With such limited data, deep models like CNNs and FNNs may struggle to generalize effectively, while traditional models like Random Forest can remain more robust with fewer training examples. Furthermore, the limited data reduces the performance gap between simpler and more complex models, as none of them have enough training examples to fully leverage their respective strengths, resulting in Spearman correlation values that cluster within a narrow range.

4.3 Performance Limitations of Non-Hurdle Approaches

To contextualize the performance improvements made by the full hurdle model, we first examine the industry standard baseline. In the following paper, which analyses how epigenetic descriptors affect CRISPR-Cas9 off-target activity [8], the authors trained an XGBoost model and a CNN model, both trained using the same crisprSQL dataset (extended with synthetic data) as this thesis. Their results reveal the limited predictive power of their models. Despite using extra epigenetic descriptors, their XGBoost model only managed to achieve a Spearman correlation of 0.419 and the CNN model was not much better, achieving a Spearman correlation of 0.424.

Since there are no other studies in the literature that have evaluated Spearman rank prediction on the crisprSQL dataset for us to compare to, we conducted our own baseline experiments by training regression models directly on the full dataset. Our preliminary results shown in figure 9 align with the findings above. All our models except the Random Forest Regressor achieve a very similar Spearman

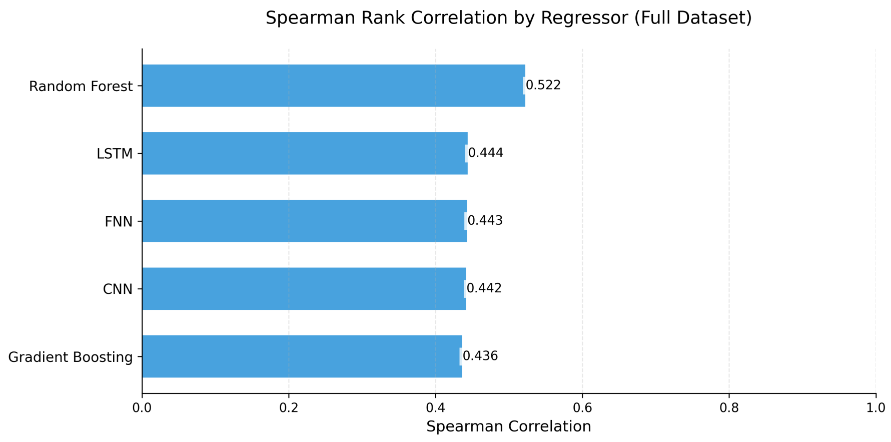


Figure 9: Spearman rank correlation of regression models trained on the full dataset without using a hurdle architecture. The Random Forest Regressor achieved the highest correlation (0.522), while deep learning models such as LSTM, FNN, and CNN performed similarly, at around 0.44.

correlation between 0.436 and 0.444. The Random Forest Regressor achieves a slightly higher Spearman correlation of 0.522, which still falls short of the predictive accuracy needed for robust experimental applications.

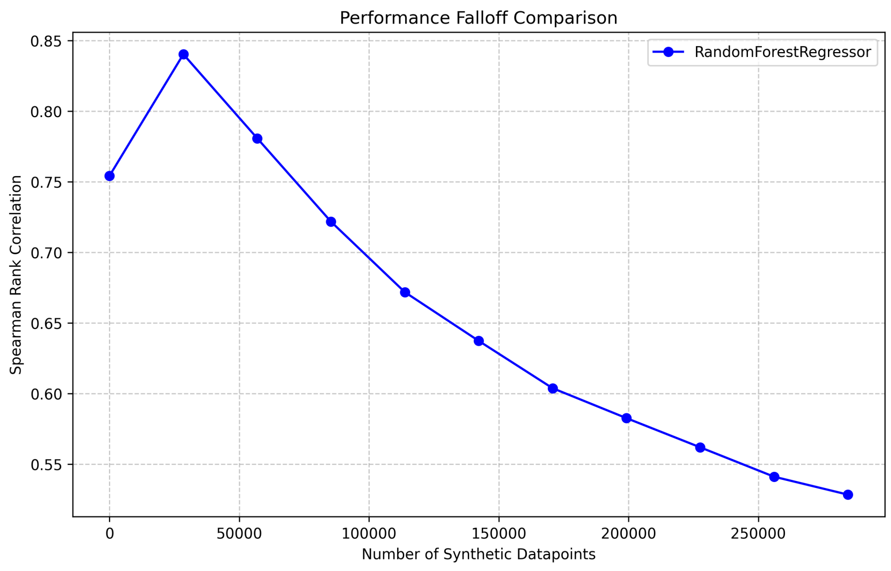


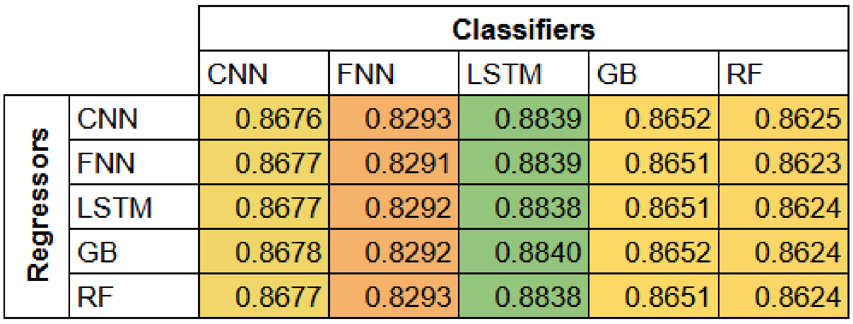
Figure 10: Performance falloff of a Random Forest Regressor as the number of synthetic datapoints increases. While initial additions of synthetic data improve performance, the model's accuracy declines steadily beyond a certain point due to the overwhelming number of zero-labeled synthetic examples.

Figure 10 further illustrates the performance falloff of these models as synthetic data is incrementally added to the original 25,554 experimental data points. Initially, the Spearman Rank rises slightly from 0.7542 with no synthetic data points to 0.8403 with 28,451 synthetic data points (52.7% synthetic data), likely due to a better feature representation. However, beyond this point, performance steadily declines as more synthetic data is added, with the Spearman Rank dropping to 0.522 when all 284,510 synthetic datapoints with zero cleavage rate are included (91.8% synthetic data).

These results collectively suggest that traditional architectures, whether tree-based or deep learning, face fundamental limitations when trained on zero-inflated cleavage rate data, which motivates our hurdle-based approach. By separating the detection of cleavage activity (classification stage) from efficiency prediction (regression stage), we can address the core data imbalance issue that constrains existing methods.

4.4 Hurdle Model Performance

We will now evaluate the performance of our hurdle models, summarized in figure 11, from which we can then select the best hurdle model configuration.



		Classifiers				
		CNN	FNN	LSTM	GB	RF
Regressors	CNN	0.8676	0.8293	0.8839	0.8652	0.8625
	FNN	0.8677	0.8291	0.8839	0.8651	0.8623
	LSTM	0.8677	0.8292	0.8838	0.8651	0.8624
	GB	0.8678	0.8292	0.8840	0.8652	0.8624
	RF	0.8677	0.8293	0.8838	0.8651	0.8624

Figure 11: Spearman rank correlation values for all 25 combinations of classifier and regressor models in the hurdle architecture. Rows indicate the regressor used and columns the classifier. The best-performing combination is the Gradient Boosting regressor with the LSTM classifier, achieving a correlation of 0.8840.

There are two important observations that we can make from the data.

1. **Significant improvement in performance** - All 25 hurdle models signifi-

cantly outperformed the baseline regressors used on their own. While the best standalone regressor (Random Forest Regressor) achieved a Spearman correlation of only 0.522, all hurdle model combinations achieve correlations above 0.82, with the best combination reaching 0.884, an outstanding improvement of 0.362 from the standalone models.

2. **Classification component drives performance** - We see that the choice of classification model has a much greater effect on final performance than the choice of regression model. For example, for any fixed classifier, the variation in performance across different regressors is minimal (on the order of 0.0002), whereas changing the classifier leads to much larger shifts in Spearman correlation (up to 0.055).

   This is likely due to the fact that classification models are responsible for determining the predictions of the entire dataset, whereas regression models only make a prediction on the small subset of data where the classifier outputs a value larger than the hurdle model threshold. This also means that improvements in our classification models translate more directly to the improved overall ranking performance, justifying our decision to keep the same hyperparameters between our classification models and regression models and optimize them only in the classification models.

To assess the robustness of these results, we repeated the training and evaluation process with multiple random seeds. We found that the LSTM models and the tree based models (Random Forest and Gradient Boosting), had the most stable performance, with variations in Spearman correlation of less than 0.05 between runs. By contrast, FNN and CNN models showed greater variability in performance. I suspect that this may be due to poor hyperparameter choices. While the chosen hyperparameters appeared to maximize performance during initial testing, they may have led to instability across different random initializations.

We note that the results in Figure 11 only reflect the performance on the validation set, from which we selected the best hurdle model (which was the model consisting of the LSTM classifier and the Gradient Boosting regressor). However, when testing this model using a test set, we still obtained a high Spearman Rank of 0.881, suggesting that the model generalizes well to unseen data and maintains a strong predictive performance.

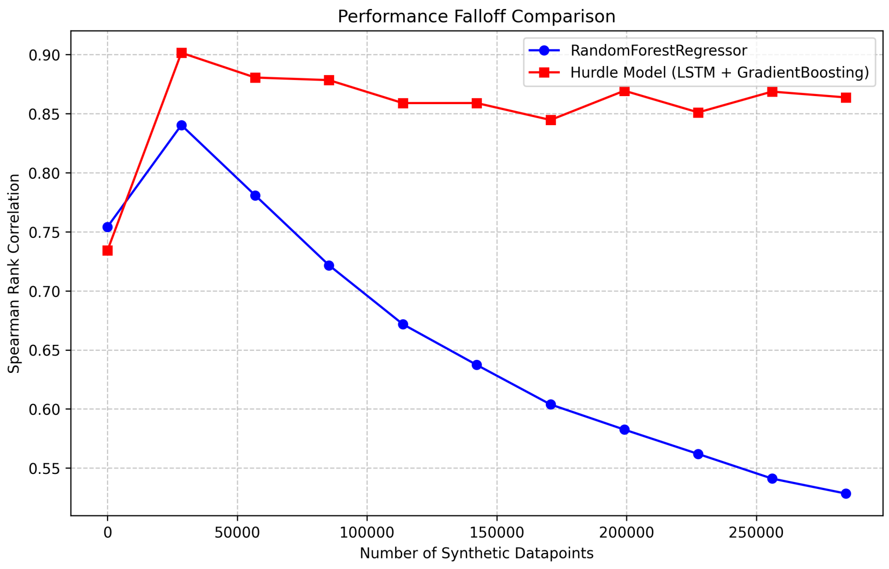


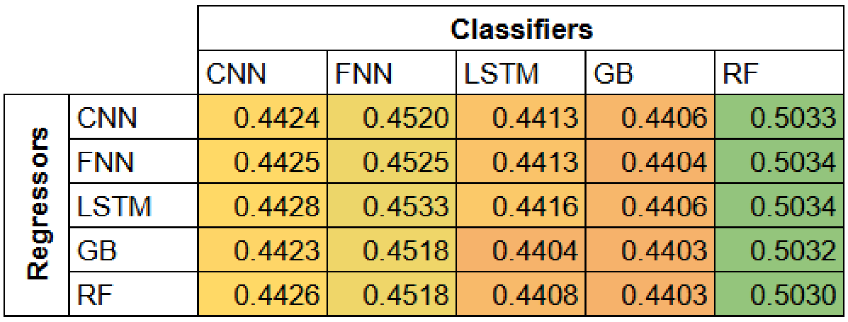
Figure 12: Comparison of performance falloff between a standard Random Forest Regressor and the best-performing hurdle model (LSTM classifier + Gradient Boosting regressor) as the number of synthetic datapoints increases. While the Random Forest model's performance steadily declines due to increasing class imbalance, the hurdle model remains robust, consistently achieving high Spearman rank correlations above 0.84.

To produce the results in figure 12, we evaluated both models on a series of datasets with increasing amounts of synthetic data. For each dataset, we retrained the Random Forest Regressor and the selected Hurdle model, and then measured their performance on the same held-out test set.

At zero synthetic data points, the hurdle model actually underperforms the standalone Random Forest Regressor (0.734 vs. 0.754). However, as we introduce synthetic datapoints, the hurdle model quickly pulls ahead. For example, by 28,451 points its Spearman score is already at 0.902 (versus 0.840 for Random Forest), and this lead continues to grow. After around 120,000 synthetic datapoints, the performance of the hurdle model seems to stabilize in the 0.86-0.88 range, even reaching as high as 0.869 at the higher data scales. This trend suggests that the hurdle framework is not only more effective in utilizing synthetic data but also potentially infinitely scalable.

4.5 Ablation Study: Impact of Thresholding in the Hurdle Model

We make a small tangent here to better understand the contribution of the thresholding mechanism in our hurdle model, by conducted an ablation study in which the threshold step was removed. This effectively means that the final prediction is computed as the product of the classifier and regressor outputs for all inputs, regardless of the classifier's confidence. This modification eliminates the decision boundary that distinguishes between zero and non-zero predictions, allowing all data points to pass through to the regression component.



		Classifiers				
		CNN	FNN	LSTM	GB	RF
Regressors	CNN	0.4424	0.4520	0.4413	0.4406	0.5033
	FNN	0.4425	0.4525	0.4413	0.4404	0.5034
	LSTM	0.4428	0.4533	0.4416	0.4406	0.5034
	GB	0.4423	0.4518	0.4404	0.4403	0.5032
	RF	0.4426	0.4518	0.4408	0.4403	0.5030

Figure 13: Spearman rank correlations of all classifier–regressor combinations in a hurdle model without thresholding. All inputs were passed to the regression model regardless of classifier confidence. Performance is uniformly lower than the thresholded hurdle model

The results of this modified hurdle model are summarized in figure 13. The best-performing combination (Random Forest Classifier + LSTM Regressor) achieved a Spearman correlation of 0.5034. While this is slightly better than the standalone regressors trained on the full dataset (best: 0.522 for Random Forest, see figure 9), it falls significantly short of the full thresholded hurdle model, where the performance reached as high as 0.8840 (see figure 11).

These results demonstrate that the thresholding step is essential for maximizing performance. Without it, the classifier's output acts only as a soft weight, and low-confidence predictions still influence the final score, introducing noise. The threshold provides a hard cutoff that filters out unreliable predictions, ensuring that only samples likely to have non-zero cleavage are passed to the regressor, critical for avoiding the dilution of regression quality.

4.6 Stacking Ensemble Model Evaluation

After training all 25 hurdle models and then training the stacking ensemble which combines them all, we obtain a training set correlation of 0.8998 and a test set correlation of 0.8930. While the ensemble model shows only a modest performance gain (0.012 improvement over the best individual model), its primary advantage is its enhanced robustness and consistency across different random initializations. This stability stems from the ensemble's ability to reduce prediction variance by combining outputs from multiple diverse architectures. However, this comes at a substantial computational cost, since the stacking ensemble model requires nearly an order of magnitude more training time than the individual models.

To better understand how the ensemble makes use of its component models, we turn to SHAP (SHapley Additive exPlanations) [19], which is a method used to explain how machine learning models make their predictions by assigning each input a score that quantifies how much it contributes to the final output. This allows us to interpret the ensemble's decision process and identify which individual hurdle models contribute most to its output.

In this next section I will denote each hurdle model as their classifier and regressor pair so that it is easier to identify the contributions of the individual components within the ensemble. There are 2 main observations we can make using the SHAP plot:

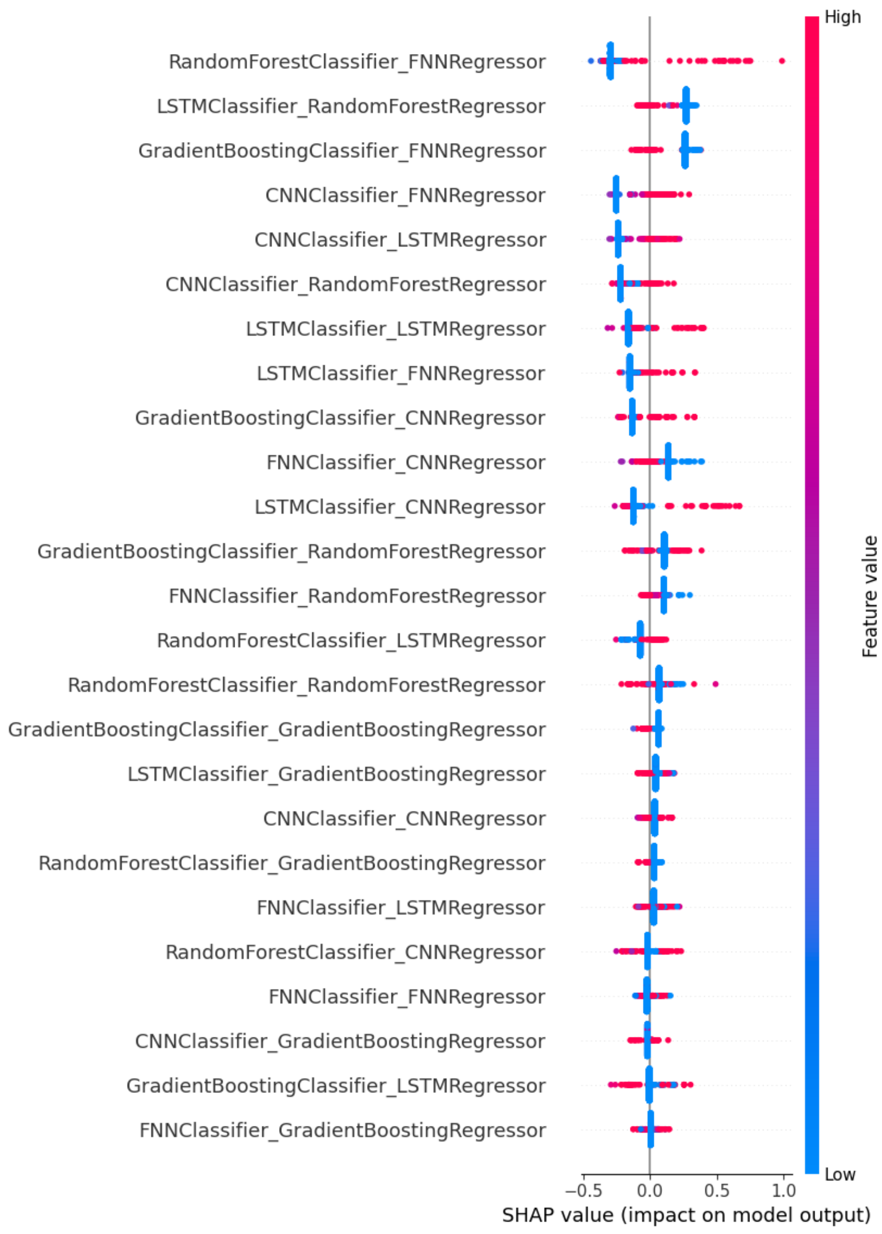


Figure 14: SHAP summary plot showing the contribution of each individual hurdle model to the stacking ensemble's output. Each row corresponds to a unique classifier–regressor pair, and each dot represents a SHAP value for a test example. Higher absolute SHAP values indicate greater influence on the final ensemble prediction.

1. The most influential model pairs (RandomForestClassifier with FNNRegressor, LSTMClassifier with RandomForestRegressor, GradientBoostingClassifier with FNNRegressor) show the highest mean absolute SHAP values. Their high individual outputs (denoted by the red dots) consistently correlate with positive SHAP values, indicating that when these models predict strongly,

they contribute positively to the ensemble's final cleavage rate prediction.

2. There are many combinations which contribute very little. Pairs such as FNNClassifier with LSTMRegressor or CNNClassifier with CNNRegressor have SHAP values clustered closely around zero. This suggests that these models have limited influence on the predictions of the ensemble and could potentially be removed to simplify the model without a noticeable loss in accuracy.

Observation 2 is the most important, as it suggests that training only the most influential models (those with high SHAP values) could achieve comparable predictive performance while significantly reducing computational costs and training time. Indeed, by modifying our method to only train the top five most influential models, we obtain a new test set correlation of 0.8945, which is actually outperforms the full ensemble while requiring a much shorter training time.

5 Conclusion

In this thesis, we presented a new application of the hurdle model framework to the problem of CRISPR-Cas9 cleavage rate prediction in order to address the challenge of extreme class imbalance introduced by large-scale synthetic data. By separating the tasks of cleavage occurrence prediction and cleavage magnitude estimation, the hurdle model provides an effective modeling strategy for zero-inflated datasets.

We implemented and evaluated a wide range of both traditional and deep learning architectures, including FNNs, CNNs, and LSTMs, in both classification and regression settings. Our results show that hurdle models significantly outperform standalone regression models across all configurations, with the best-performing combination (LSTM classifier and Gradient Boosting Regressor) achieving a Spearman rank correlation of 0.881. Furthermore, a stacked ensemble built from the 5 most influential hurdle models achieved an even higher test correlation of 0.8945, demonstrating the advantages of combining diverse model architectures.

5.1 Reflections

The development of a successful model for cleavage rate prediction in a highly zero-inflated dataset consisted of many challenges, most of which came from the fact that most of the available models which are well-suited to general machine learning tasks struggled to generalize on this synthetic dataset.

I enjoyed exploring a wide variety of approaches, including traditional machine learning architectures such as random forests and SVMs and deep learning architectures such as CNN and RNNs. I also experimented with large ensemble methods such as bagging, boosting and stacking models to combine my approaches. However, these approaches constantly fell short, as my models would consistently overfit to the zero class and fail to detect the rare but important non-zero cases.

However, I managed to start making progress after discovering the hurdle model, with the idea of separating the classification and regression components of the model. This structure, combined with probability thresholding, helped overcome the zero-inflation problem and greatly improved the overall performance.

Importantly, the insights gained from the earlier failed attemps were crucial in shaping the final approach. For example, I managed to use elements from previously tested models to help create the different versions of the Hurdle model. Furthermore, in the final stages, I managed to apply stacking ensemble techniques (which I experimented with earlier) across the different hurdle versions to further refine performance. This development process has taught me the value of persistence and learning from previous experiments, even when they are not immediately successful.

5.2 Future Work

While this thesis has demonstrated the effectiveness of the hurdle model in handling zero-inflated CRISPR off-target cleavage data, several promising directions remain for improving both its architecture and generalizability.

Firstly, future improvements could involve adding more base classification and regression models, since the models used in this thesis represent only a fraction of the possible machine learning models which we could have used. For example, among the traditional machine learning models, we could include SVMs, XGBoost, Catboost and many more. For deep learning, bi-directional RNNs and transformer-based architectures, such as BERT can also be included. These models could capture potential sequence dependencies missed by the other models, improving performance when they are all combined together in the ensemble model.

Secondly, notice in figure 10 that when the regression model is trained on a small number of synthetic datapoints, it's Spearman rank performance actually increases. Hence, subsequent studies can try to add a limited number of synthetic zero-cleavage examples when training the regression component, to see if this will improve the overall performance of the full model. Furthermore, we can test whether these synthetic data benefits are consistent across different model architectures.

Finally, future research should adapt the hurdle model to include any other biologically relevant features. Beyond guide-target sequences, other useful features include epigenetic markers (e.g. DNA methylation and histone modifications), chromatin accessibility and local sequence context (e.g. GC content, sequence complexity, and nucleosome positioning). Using these features could enhance the performance of the hurdle model in circumstances where sequence alone may not fully explain the cleavage rate.

Appendices

References

[1] M. Jinek, K. Chylinski, I. Fonfara, M. Hauer, J. A. Doudna, and E. Charpentier, "A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity," en, *Science*, vol. 337, no. 6096, pp. 816–821, Aug. 2012. [Online]. Available: `https://doi.org/10.1126/science.1225829`.

[2] L. J. Zhu, M. Lawrence, A. Gupta, *et al.*, "GUIDEseq: A bioconductor package to analyze GUIDE-Seq datasets for CRISPR-Cas nucleases," en, *BMC Genomics*, vol. 18, no. 1, Dec. 2017. [Online]. Available: `https://doi.org/10.1186/s12864-017-3746-y`.

[3] A. H. Muhammad Rafid, M. Toufikuzzaman, M. S. Rahman, and M. S. Rahman, "CRISPRpred(SEQ): A sequence-based method for sgRNA on target activity prediction using traditional machine learning," en, *BMC Bioinformatics*, vol. 21, no. 1, p. 223, Jun. 2020. [Online]. Available: `https://doi.org/10.1186/s12859-020-3531-9`.

[4] H. K. Kim, Y. Kim, S. Lee, *et al.*, "SpCas9 activity prediction by DeepSpCas9, a deep learning-based model with high generalization performance," en, *Sci. Adv.*, vol. 5, no. 11, eaax9249, Nov. 2019. [Online]. Available: `10.1126/sciadv.aax9249`.

[5] D. Wang, C. Zhang, B. Wang, *et al.*, "Optimized CRISPR guide RNA design for two high-fidelity cas9 variants by deep learning," en, *Nat. Commun.*, vol. 10, no. 1, p. 4284, Sep. 2019.

[6] J. Lin, Z. Zhang, S. Zhang, J. Chen, and K.-C. Wong, "CRISPR-net: A recurrent convolutional network quantifies CRISPR off-target activities with mismatches and indels," en, *Adv. Sci. (Weinh.)*, vol. 7, no. 13, p. 1 903 562, Jul. 2020. [Online]. Available: `https://doi.org/10.1002/advs.201903562`.

[7] Z. Guan and Z. Jiang, “Transformer-based anti-noise models for CRISPR-Cas9 off-target activities prediction,” en, *Brief. Bioinform.*, vol. 24, no. 3, May 2023. [Online]. Available: https://doi.org/10.1093/bib/bbad127.

[8] J. K. Mak, F. Störtz, and P. Minary, “Comprehensive computational analysis of epigenetic descriptors affecting CRISPR-Cas9 off-target activity,” en, *BMC Genomics*, vol. 23, no. 1, p. 805, Dec. 2022. [Online]. Available: https://doi.org/10.1186/s12864-022-09012-7.

[9] Z. Guan and Z. Jiang, “A systematic method for solving data imbalance in CRISPR off-target prediction tasks,” en, *Comput. Biol. Med.*, vol. 178, no. 108781, p. 108 781, Aug. 2024. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2024.108781.

[10] J. G. Cragg, “Some statistical models for limited dependent variables with application to the demand for durable goods,” *Econometrica*, vol. 39, no. 5, p. 829, Sep. 1971. [Online]. Available: https://doi.org/10.2307/1909582.

[11] J. Lin and K.-C. Wong, “Off-target predictions in CRISPR-Cas9 gene editing using deep learning,” en, *Bioinformatics*, vol. 34, no. 17, pp. i656–i663, Sep. 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty554.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” en, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015. eprint: 1512.03385 (cs.CV). [Online]. Available: https://doi.org/10.48550/arXiv.1512.03385.

[14] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. eprint: 1412.6980 (cs.LG). [Online]. Available: https://doi.org/10.48550/arXiv.1412.6980.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017. eprint: 1708.02002 (cs.CV). [Online]. Available: https://doi.org/10.48550/arXiv.1708.02002.

[16] F. Störtz and P. Minary, "crisprSQL: A novel database platform for CRISPR/Cas off-target cleavage assays," en, *Nucleic Acids Res.*, vol. 49, no. D1, pp. D855–D861, Jan. 2021. [Online]. Available: https://doi.org/10.1093/nar/gkaa885.

[17] R. M. Sakia, "The box-cox transformation technique: A review," *Statistician*, vol. 41, no. 2, p. 169, 1992. [Online]. Available: https://doi.org/10.2307/2348250.

[18] C. Tennakoon, R. W. Purbojati, and W.-K. Sung, "BatMis: A fast algorithm for k-mismatch mapping," en, *Bioinformatics*, vol. 28, no. 16, pp. 2122–2128, Aug. 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts339.

[19] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017. eprint: 1705.07874 (cs.AI). [Online]. Available: https://doi.org/10.48550/arXiv.1705.07874.