Performance and Interpretability of Graph Convolutional Networks for the prediction of COVID-19 Vaccine mRNA degradation

Veer Shah
Institute for Computational and Mathematical Engineering
Stanford University
veer@stanford.edu

Ziwei Chen
Department of Computer Science
Stanford University
ziwei75@stanford.edu

Ishan Shah
Department of Statistics
Stanford University
ijshah@stanford.edu

Collin Schlager
Department of Computer Science
Stanford University
schlager@stanford.edu

Leore Bensabath
Institute for Computational and Mathematical Engineering
Stanford University
leoreb@stanford.edu

December 29, 2020

Abstract

mRNA vaccines have emerged as a critical technology in the global effort to develop a COVID-19 vaccine. However, the chemical properties of RNA pose a challenge to their utility as a vaccine candidate: the molecules are prone to degradation and prove challenging to effectively distribute. Furthermore, little is known about the degradation properties of individual RNA bases in a molecule. For our project, we sought to investigate whether deep learning could predict RNA degradation from an RNA sequence. We investigated a variety of neural architectures, from convolutional neural networks (CNNs) to graph convolutional neural networks (GCNs). We also explored the interpretability of these models. GCNs that computed over the structural graph of the mRNA molecule outperform CNN-based models during test time by a large margin. This suggests that modeling RNA molecules using graphs and performing convolution directly on graph representations are critical in understanding molecule degradation mechanisms. We also analyzed the behavior of our model using in silico mutagenesis (ICM) and found that our model pays local attention when predicting a given position's reactivity, and exhibits interesting behavior on neighboring bases in the sequence.

1 Introduction

The COVID-19 pandemic has had, and continues to have, an immense impact on the lives of people around the world. The global crisis has brought about an unprecedented effort to develop, approve of, and distribute a vaccine against the novel virus in record time. A process that would normally take between 10-15 years is being accelerated to a timeline of a year, thanks to extraordinary support and collaboration across industry, academia, and governments across the globe. The accelerated effort is for good reason, too. At the time of writing, the global death toll of COVID-19 is at a staggering 1.36 million deaths and climbing, not including excess morbidity from the pandemic itself. Furthermore, the number of new cases each day is currently around 620,000 [1]. The time-sensitive need to develop a vaccine hardly needs additional motivation.

One of the most prominent technologies among the leading vaccine candidates is the mRNA vaccine [2]. Indeed, the two leading candidates (as of this writing) are mRNA based. Both candidates, separately developed by Pfizer/BioNTech and Moderna, have reported their vaccines to be $\geq 90\%$ effective [3]. In contrast to traditional vaccines, which comprise inactivated or attenuated components of the pathogen itself, the mRNA vaccine provides the template for the cellular

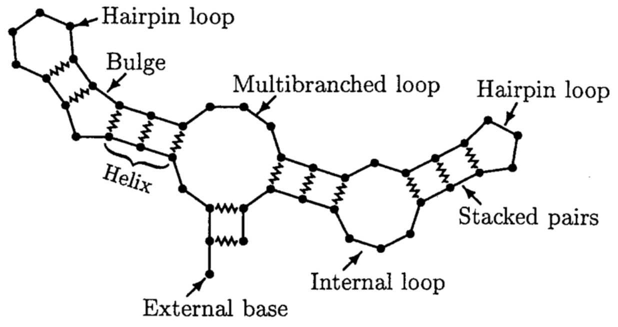


Figure 1: A diagram of different RNA loop structures. Covalent bonds are indicated by straight line segments, while H-bond base pairing is indicated by jagged lines. Image credit: [11].

synthesis of a viral component. Development of these RNA sequences is comparatively less costly and time consuming, because it avoids the challenging purification process of proteins [4].

However, mRNA vaccines are a novel technology and face unique challenges as a result of their chemical structure. In particular, mRNA molecules are known to spontaneously degrade over time [5]. In a laboratory setting, RNA is generally stored in specialized freezers kept at -80°C [6]. From a logistical perspective, this temperature preference poses a substantial hurdle to the successful administration of an RNA-based vaccine: such freezers are not readily available and degradation of a single base could render the vaccine useless. Since the latest effectiveness announcements from Pfizer/BioNTech and Moderna, the logistical challenges of distributing their vaccines at these requisite temperatures have been of great interest and concern [7]. Relatively little is known about the tendencies for specific bases to degrade [8]. Motivated by this problem, we sought to investigate whether deep neural network architectures could predict base-resolution degradation in RNA.

2 Task description and data construction

In September 2020, the Das Lab of Stanford Biochemistry and Eterna partnered to sponsor a Kaggle competition focused on the problem of RNA degradation [8]. We used their published dataset of 3029 RNA sequences annotated with base-wise information relevant to degradation. The competition features an additional 3005 RNA sequences reserved for a private test set.

Each sequence in the training set comprises 107 bases. The data include base identities (A, G, U, C) as well as secondary structure information indicating which bases are paired to each other. This pairing is denoted by a string of opening and closing parentheses, where matching pairs indicate paired bases at those indices. Additionally, the data provide a prediction of the type of RNA loop structure in base resolution. These categories (e.g. bulge, hairpin loop, paired stem, etc.) indicate the local characteristics of the sequence structure as predicted by an algorithm – bpRNA from Vienna RNAfold 2 [9]. Figure 1 shows prototypical examples of these loop structures [10].

Input Label	Example
sequence	GGAAAAGCUCUAAUAACAGGAGA
structure	.....((((((.......)))).))..
predicted_loop_type	EEEEESSSSSHHHHHHHSSSSBSSX

Table 1: Input data and their examples.

The dataset's labels are a set of reactivity and degradation values experimentally measured in different conditions at each base. The reactivity was measured using SHAPE-Seq and features how structurally flexible the nucleotide is [12]. The degradation rates were measured using MAP-Seq under four different conditions and feature the likelihood of degradation in each condition [13]. The dataset includes 5 metrics of reactivity and degradation (listed below in Table 2). However, only the first three metrics are evaluated in the competition, so we focused our attention on those values: `reactivity`, `deg_Mg_ph10`, and `deg_Mg_50C`.

Thus, the goal is to develop a multi-task network that takes the RNA sequence information as input and produces three predictions at each base: `reactivity`, `deg_Mg_pH10`, and `deg_Mg_50C`. Model performance is evaluated on two

Output Label	Description
reactivity	General reactivity score.
deg_Mg_pH10	Likelihood of degradation after incubation at high pH with magnesium.
deg_Mg_50C	Likelihood of degradation at high temperature with magnesium.
deg_pH10	Likelihood of degradation after incubation at high pH (pH 10).
deg_50C	Likelihood of degradation at high temperature (50 deg C)

Table 2: Reactivity labels and their descriptions. The first three metrics are the predicted outputs of our models, while the last two are not evaluated.

held-out test sets, including a public test set and a private test set (defined by the original Kaggle competition). The evaluation is based on MCRMSE (mean columnwise root mean squared error):

$$MCRMSE = \frac{1}{N_t} \sum_{j=1}^{N_t} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_{ij} - \hat{y}_{ij})^2}$$

where N_t is the number of columns/tasks (i.e. `reactivity`, `deg_Mg_pH10` and `deg_Mg_50C`) and (y_{ij}, $\hat{y}_{ij}$) are the ground-truth and predicted values for reactivity type and RNA sequence at a specific base, respectively.

3 Method

3.1 Baseline Models

We first implemented a simple baseline model that would give us a lower bound on performance. It consisted of a simple CNN architecture with two 1D convolution layers followed by two fully connected layers. This model inputted encoded features of a small window size (21) around each base in the sequence, and predicted the 3 outputs: `reactivity`, `deg_Mg_pH10`, and `deg_Mg_50C`.

We note that the performance of this first baseline model is sensitive to the selected window size. Additionally, the model's prediction is limited to a local window of neighboring bases along the primary sequence. We designed a second baseline model, where the sequence and structure of the entire RNA molecule is encoded as the input to the CNN. In this way, the model could leverage global sequencing information in its prediction. The sequence, loop type, and base pairing of each base in the RNA sequence are one-hot encoded and concatenated together as the final input. The inputs are passed to three convolution layers, which applies average pooling in windows of 3. All layers apply batch normalization, rectified linear units, and dropout. The final output from the CNN is passed to a linear layer to predict the RNA degradation rates at each base.

3.2 Graph Convolutional Network (GCNs)

As we noted above, our 1D convolution-based baseline only aggregates information from neighboring bases along the primary sequence of the molecule. However, as depicted in Figure 1, mRNA molecules are 3D structures, wherein bases loop back on one another to form bonding interactions with linearly distant bases. To reflect a more realistic 3D structure, the RNA molecules can be represented as graphs, where nodes contain information of each base and edges represent bases that are adjacent or paired by bonding interactions [14]. Traditional CNNs cannot operate directly on graphs due to their irregular structure. However, a generalized form of the CNN, the graph convolutional network (GCN) was developed for this explicit purpose [15]. Thus, the GCN is an attractive architecture for inference on RNA structures and, indeed, it has been used a few times in the literature [14].

In our implementation, the sequence, loop type, and base pairing information are used to generate the embedding for each node after passing the integer encoded input to a embedding layer (see Figure 2). The edges are represented using adjacency matrices calculated from the secondary structure. We used a type of GCN architecture called GraphSAGE. GraphSAGE [16] is an instance of GCNs developed for representational learning. Instead of learning node embeddings directly, GraphSAGE learns the aggregator function and computes the node embeddings by applying the aggregator function to the neighboring nodes. In our application, we trained GraphSAGE in a supervised fashion. The node embeddings were extracted from the GCN and fed to another neural networks to make the final prediction. Thus, the aggregator functions are trained jointly with the RNA degradation prediction (see Figure 2).

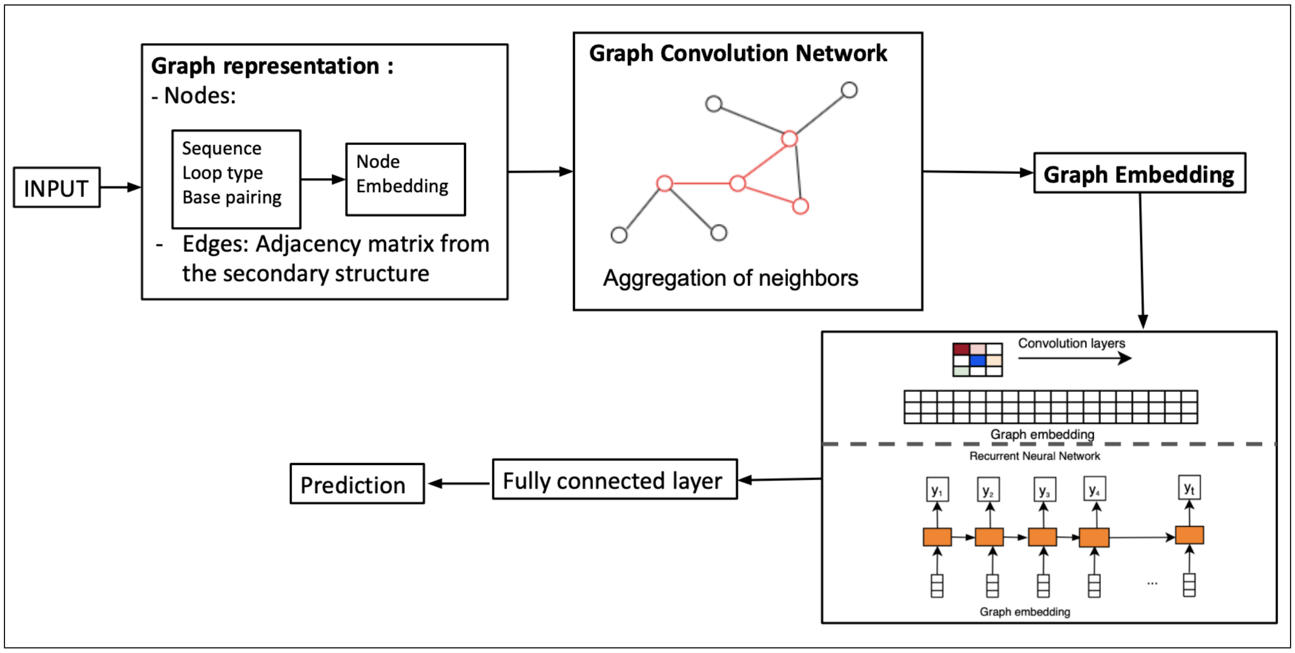


Figure 2: Overview of model architecture, from input sequence to reactivity prediction. Node embeddings are a representation of the sequence and structure of each base. A GCN is used to compute a graph embedding for each node. The node embeddings are then passed through a GRU (or CNN) and a fully-connected layer to make the final prediction of reactivity.

3.2.1 Aggregate functions comparison

There are various forms of aggregator functions in GraphSAGE. Here, we experimented with three different aggregator functions (mean, convolution, and LSTM [17]) and compared their performance in RNA degradation prediction. The mean aggregator takes the average of neighboring nodes and concatenates it with the original node embedding. The LSTM aggregator feeds the embeddings of neighboring nodes sequentially to the LSTM and concatenates the final output with the original node embedding. For LSTM and mean aggregators, the node embeddings after concatenation are fed into a linear layer to reduce it to the original dimension. The convolution operation aggregates over the neighboring nodes (including the node itself) and then passes the result to a linear layer for the final embedding. When comparing different aggregate functions, we extracted the node embeddings from the GCN and then passed it to the Gated Recurrent Units (GRU) [18], which is a variant of the Recurrent Neural Network (RNN). The output from the GRU is fed into a fully connected layer to make the final prediction.

3.2.2 Model architecture comparison

The basic model architecture is shown in Figure 2. As discussed before, the graph embeddings generated by GCNs can be fed into a GRU, and we refer to this architecture as the GCN_GRU architecture. As an alternative, we also experimented with using CNNs on top of the graph embeddings and passed the output from the CNN to a fully connected layer to make the prediction. This model is referred to as the GCN_CNN architecture, and we tried CNN architectures with and without residual connections [19] between layers. In addition to varying the model components, we also experimented on the number of GCN layers (K). The number of GCN layers, K, affects how the node embedding is generated. For instance, when $K = 2$, the node embedding is generated by aggregating all neighbors that are at most two edges apart. Because this a multi-task learning problem, we also experimented on weighted loss, assigning higher weights for tasks that are explicitly evaluated during the competition.

The competition authors note that measurement error increases for bases at the end of the RNA sequence due to technological challenges, so only the measurements of the first 68 bases are reported. Thus, for all of the models, the predictions are truncated to the first 68 bases to calculate the loss, which is the mean squared error (MSE) between the predictions and ground truth. All of the models were trained using batch gradient descent with a learning rate adapted via ADAM [20]. They were also trained with 5-fold cross validation and the prediction was the average of all folds. The loss curve of the GCN_GRU architecture during training and validation is shown in Figure 3. The gap between the

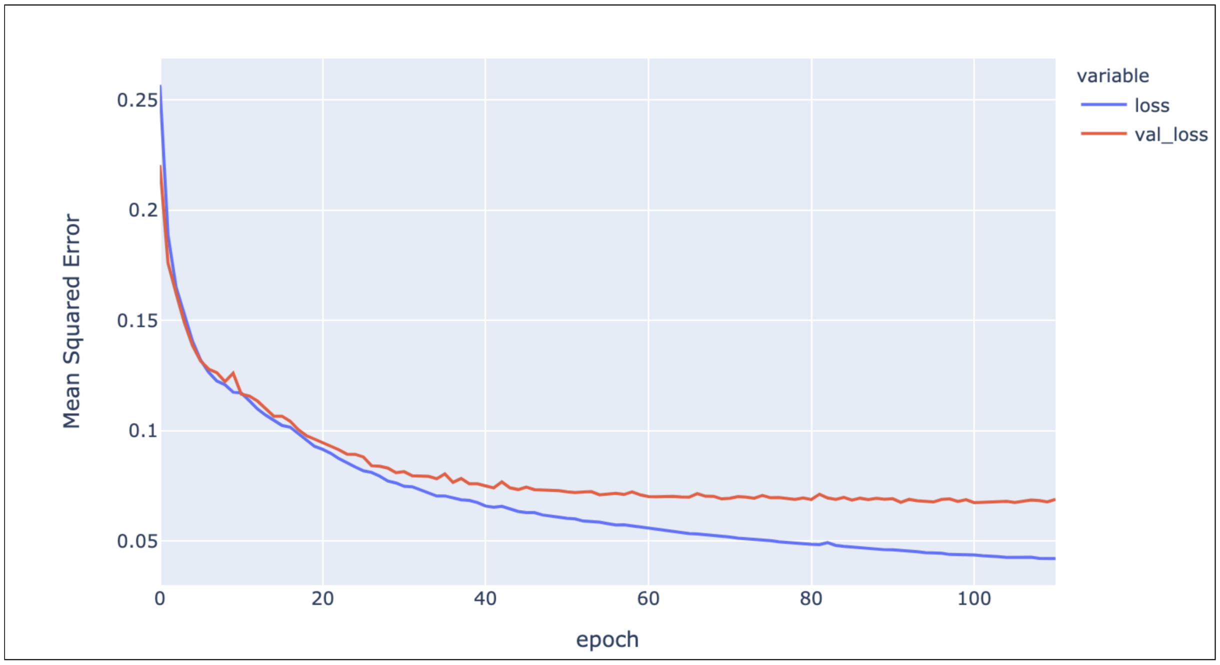


Figure 3: Loss curve for the GCN_GRU architecture during training and validation

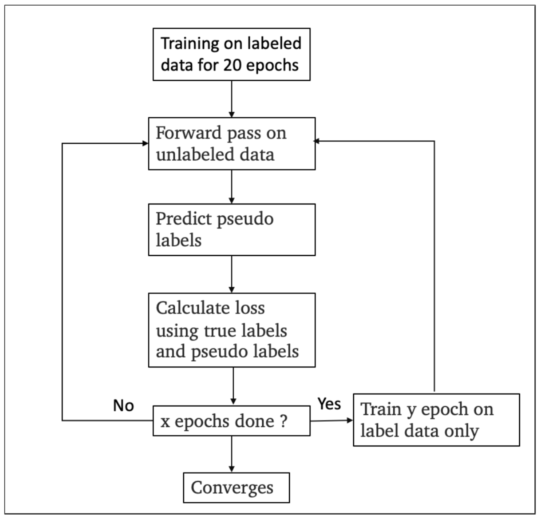

Training on labeled data for 20 epochs
Forward pass on unlabeled data
Predict pseudo labels
Calculate loss using true labels and pseudo labels
No
x epochs done ?
Yes
Train y epoch on label data only
Converges

Figure 4: Pseudo labeling algorithm

training loss and validation loss does not indicate over-fitting because we stopped the model when the validation loss did not decrease for a certain number of epochs. Stopping the model earlier led to worse performance during test time.

3.2.3 Pretrain node embedding

The size of the training set is relatively small. Only 2096 examples are used in training after filtering measurements with a low signal-to-noise ratio. Therefore, we also experimented on pretraining the node embedding in an auto-encoder [21] fashion using the sequences in the training set as well as the 3000 sequences in the test set. The node embeddings from the GCN layer were fed into a fully-connected network to reconstruct the original sequences and structures (i.e, base pairing and loop types). The MSE (Mean Squared Error) between the input and the reconstruction was used as the loss function for the auto-encoder. After the auto-encoder converged, the node embeddings were extracted to initialize the node embeddings in the GCN_GRU architecture.

3.2.4 Data augmentation with pseudo labelling

Given the limited training data, we tried to use pseudo labeling [22] to leverage additional, non-labeled data for training. Pseudo labeling is a data augmentation technique that uses unlabeled data along with labeled data during the training – it is a semi-supervised learning algorithm. The process is shown in Figure 4. We first train the network on the labeled data for 20 epochs. Then we introduce the unlabeled data by mixing training on unlabeled batches and labeled batches.

We used 2 types of unlabeled data for pseudo labeling: First we used the end of the sequences (bases 68 to 107) of the training set. After training the network on labeled data for 20 epochs, we start using unlabeled data: For each epoch, we compute the loss both on the labeled part and on the unlabeled part using the pseudo labels. The pseudo labels are computed by a forward pass of the unlabeled data in the network at the state gotten from the previous epoch. This procedure is repeated for every epoch after the 20th. The exception is that for every one out of 10 epochs, we go back to training only on labeled data.

Second, we used the private test set sequences as unlabeled data. After training the network on labeled data for 20 epochs, we trained the model for 2 epochs on unlabeled data only, followed by 5 epochs on labeled data. Here, we used a "save and load" method to prevent the pseudo labeling from deteriorating the model performances too much: at the end of each pseudo labeling section, we compare the current model with the model right before the pseudo labeling. If the current loss is higher than the loss before pseudo labeling by 0.04, we would not accept the current model and would reload the weights from the model before the pseudo labeling section.

4 Results

We evaluated our models on two different test sets: a public test set and a private test set (as defined by the Kaggle competition). The length of the RNA sequence in the public test is 107 bp and the length of the RNA sequence in the private test is 130 bp. The measurement of RNA degradation rates does not cover the last 39 bases in the sequence due to technological challenges. Thus, the length of the prediction is 68 and 91 for public and private test respectively. Because the test scores are MCRMSE (mean columnwise root mean squared error) between predictions and labels, a lower score represents better model performance.

The results of the two baseline models are shown in Table 3. Compared to the first baseline model that only focuses on local structure, the second baseline model outperforms it by a large margin. This is within our expectation since the second model contains the structure and sequence information over the entire RNA molecule.

	Public test score	Private test score
Baseline1	0.3798	0.4727
Baseline2 (CNN over the entire sequence)	0.30424	0.41348

Table 3: Public and private test scores for baseline model

The test results for GCNs with different aggregator functions are shown in Table 4. The ability of GCNs to perform convolution on data with irregular structures helps to better capture and leverage the structure information of RNA molecules: neighboring bases in secondary structure (not just primary structure) can have immediate impact on the predicted reactivity. As expected, all GCN models outperform our CNN-based baseline. The mean aggregator function achieves the best performance during test time. Thus, the mean aggregator was chosen in comparing different model architectures.

Aggregate Function	Public test score	Private test score
Mean	**0.26614**	**0.38571**
Conv	0.27173	0.38989
LSTM	0.28126	0.39904

Table 4: Public and private test scores for GCNs with different aggregate functions

The test results for GCNs with different model architectures are summarized in Table 5. Replacing the GRU simply with a CNN does not boost our model performance. After adding residue connections between convolutional layers, the performance of the model is comparable but still slightly worse than the original GCN_GRU architectures. One of the possible explanation for the performance of the GCN_CNN model is that the GCN is a generalized form of CNNs. So, GCN and CNN will have similar operations. On the contrary, combining GCNs with GRUs, which operate in a recurrent fashion, will bring more diversity to the model architecture and better capture sequential data.

Model Architecture	Public test score	Private test score
GCN_GRU	0.26614	0.38571
GCN_CNN	0.27275	0.39280
GCN_CNN (with residue connection)	0.26729	0.38822
GCN_GRU (weighted loss)	0.26514	0.38494
GCN_GRU (pretrained embedding)	**0.26401**	**0.38490**

Table 5: Public and private test scores for GCNs with different model architectures

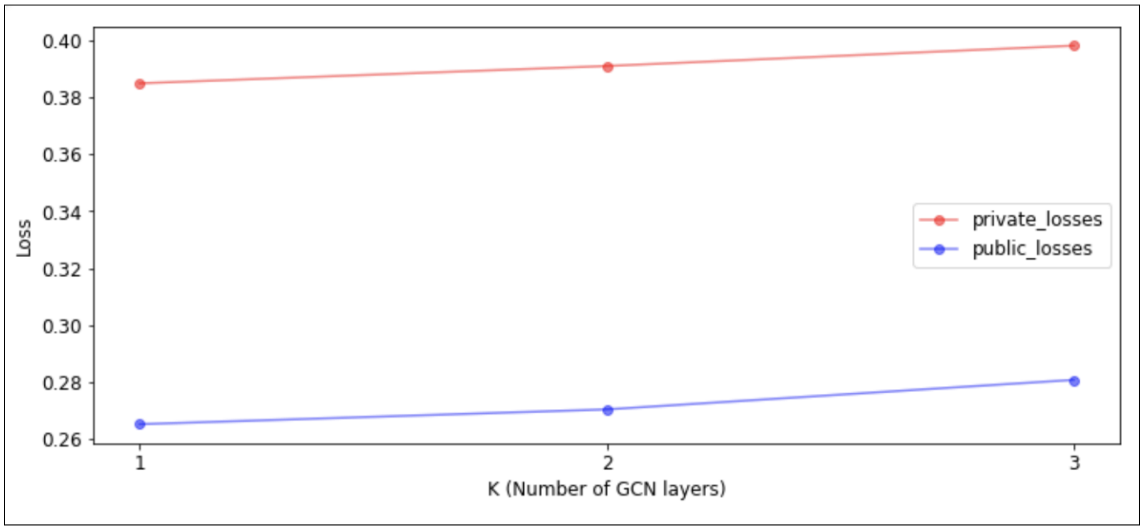


Figure 5: public and test losses for the model with different number of gcn layers

The test results for models with different number of GCN layers (K) is plotted in Figure 5. As the number of GCN layers increases, the test loss also increases. We hypothesize that this deterioration in performance is caused by the small size of graph. The graph representation of RNA molecules only contains roughly 100 nodes. When the number of GCN layers increases, the node embedding starts to capture more global information in favor of local information. As a result, the model's performance in predicting the degradation rate at the *base resolution* deteriorates.

The test results for using weighted loss and pretrained node embeddings are shown in Table 5. As expected, assigning higher weights for tasks that are evaluated during test time improved the model performance. Pretraining the node embedding in an auto-encoder fashion also improves the model on both the public test set and the private test set. However, the improvement of the performance on the private test set is marginal.

The test results for pseudo labeling on the GCN_GRU architecture are shown in Table 6. We used the GCN_GRU model with the weighted loss but without pretained node embeddings as the architecture for pseudo labeling. We see that both the end of the sequences and the private test set improve the model performances. However, the improvement of pseudo labeling using the private test set is smaller than that of pseudo labeling using the training set. This effect can be explained by the fact that the sequences in the training set follow a distinct distribution compared to the sequences in the private test set. We discuss this observed effect in detail in Section 6.1, below. This difference in the distribution could make training the model using pseudo labels on the private test set less stable, making it more challenging for the model to converge to an optimal solution.

	Public test score	Private test score
GCN_GRU (before pseudo labeling)	0.26514	0.38494
Train set - bases 68 to 107	**0.26345**	**0.38113**
Private test set - bases 1 to 91	0.26434	0.38368

Table 6: Public and private test scores for GCN_GRU achitecture using pseudo labeling

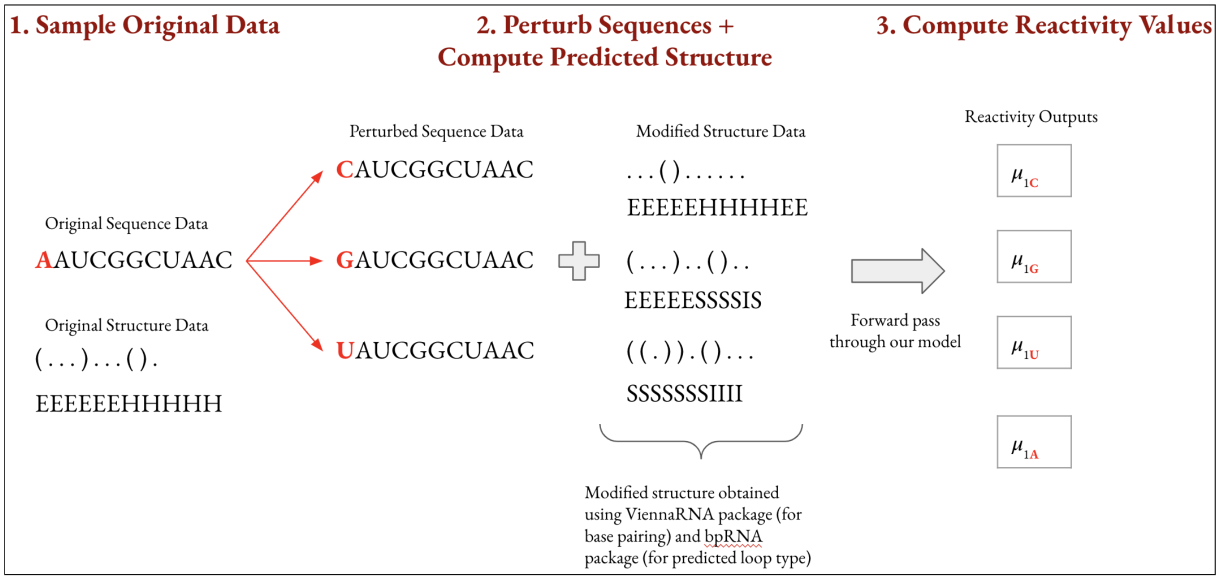


Figure 6: Inference Workflow. Note that $\mu_{B,N}$ is a vector of reactivity outputs of length 107 where B is the position of the base value that was perturbed (1 in the example above) and N is the base value it was perturbed to.

5 Model Interpretation

Synthesizing our model performance in terms of public and private test scores helps us quantify whether our model will accurately assess the degradation rates of real mRNA sequences in practice. However, as with many black box methods, we are also interested in understanding the behavior and interpretability of our model.

With relatively small data, we can implement statistical in silico mutagenesis (ICM) to probe the behavior of our model. Given the computational costs of running a forward pass over many mutated inputs, we restrict our analysis to a sample of the sequences and perturb the sequences at intervals of 5 rather than at every base. In implementing ICM, we perturb the input data at every 5 positions in the sequence, and measure the output predictions for each of the 5 different measures of reactivity and degradation. Figure 6 portrays an example of this process for a perturbation on the first nucleotide of a given sequence.

In **Step 1**, we select a sample of our original sequence and structure data to work with. For the following analyses, we have sampled 250 examples from the 2000 examples in the original public training dataset.

In **Step 2**, we perturb the bases in the sequences: at every 5 base positions (0, 5, 10, etc.), we change the base value to each of the other three nitrogenous bases that are possible in an RNA sequence. In our example above, since base position 0 is "A", we change this base to each of "C", "G", and "U". With the perturbed sequence, the original secondary structure is no longer valid: perturbing even a single base can have ripple effects on the overall structure of the molecule [23]. Thus, we must compute the secondary structure of our new, perturbed sequence. We utilize the `RNA.fold` function from the ViennaRNA 2.0 package [9] to extract the predicted base pairing for the perturbed sequence. Then, we feed both the perturbed sequence and the predicted base pair data to `bpRNA`, which predicts the loop type of each base [24]. We repeat **Step 2** for each of the 250 sequences in our sample, so that we have complete input data for each of the perturbed sequences.

In **Step 3**, we take these modified sequences and their secondary structure annotations and feed them into our GCN_GRU model. Our model outputs all 5 predicted reactivity and degradation values for each modified sequence (although, recall that the eventual evaluation is computed only using the first 3 reactivity and degradation values). We can now analyze these predicted values to better understand the behavior of our model.

5.1 Inferential Analyses

In order to capture the behavior of the model, we can probe the model from several different angles, with the goal of understanding where the model sees important signals for its prediction task, and how bases at different positions can affect the model's attention. We break down our inference tasks into 3 buckets: **Point to Point Analysis**, **Point to Sequence Analysis**, and **Sequence to Point Analysis**. In each analysis, we have restricted our focus to a single reactivity metric (`reactivity`) from the overall five available. We note that each of these analyses can be extended to the other reactivity metrics as well.

5.1.1 Point to Point Analysis

In **Point to Point Analysis**, we aim to understand the effect the original base at position *B* has on its own reactivity. The effect is normalized by the predicted reactivity values that position *B* could have in all of its perturbations.

Once we have run our ICM, we have a L x 4 x S x 5 x L multi-dimensional array, where L is the sequence length (107) and S is the number of sampled sequences (250). In order to understand base position *B*'s effect on its own reactivity, we must compute the normalized original reactivity, which is the original reactivity normalized by the reactivities of all perturbed reactivities at position *B*. To put it more concretely, we compute the following mean:

$$\theta_B = \frac{1}{4}\sum_N \mu_{B,N}^{(B)}$$

where μ is the vector of reactivity values, the *B* in superscript is the index of μ, the *B* in the subscript is the base position we have perturbed, and N is modified base value we perturb to. We can then normalize the original reactivity at position *B* as follows:

$$\delta_{B,O} = \mu_{B,O}^{(B)} - \theta_B$$

where O is the original reactivity value at position *B*. We display a sampling of the sequence reactivity profiles produced using this method below. Note that the other sequences displayed similar behavior to the sequences displayed here.

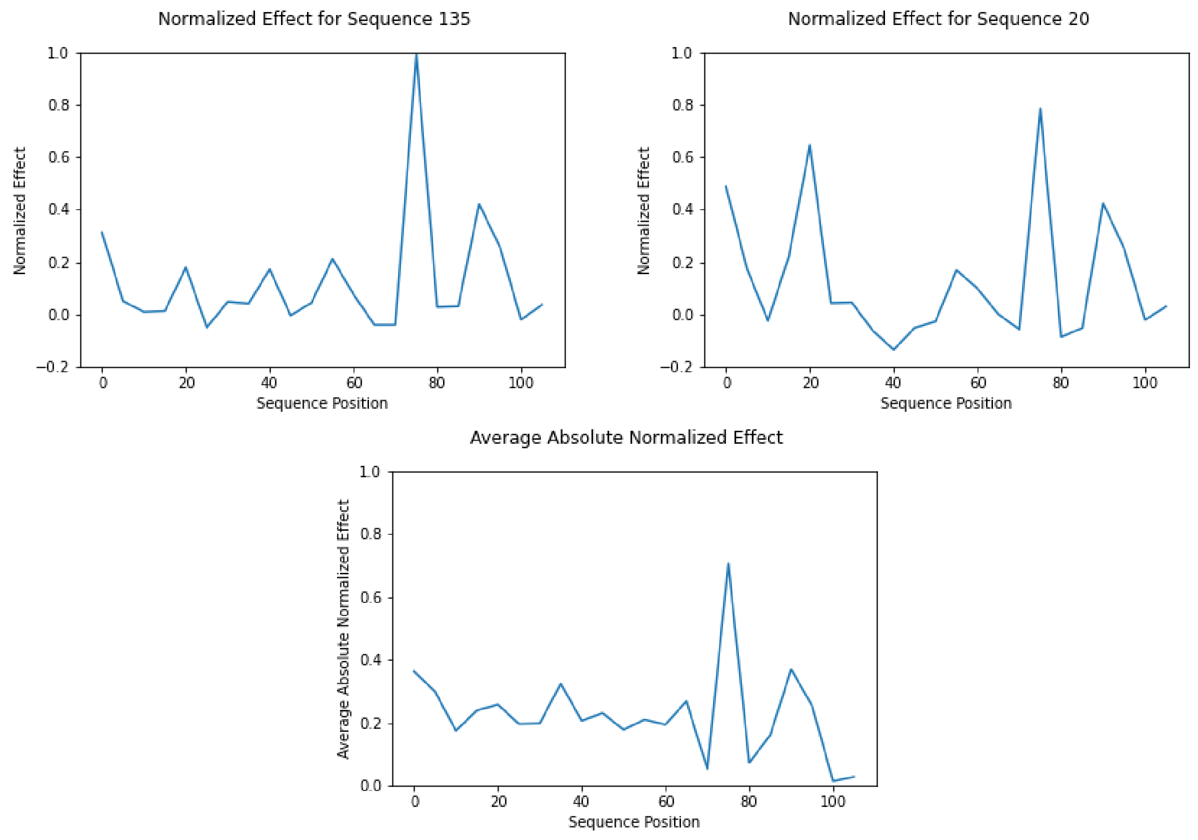


Figure 7: Normalized Effect Plots for Randomly Selected Sequences and Averaged over all Sequences

We can see from the plots in Figure 7 that each sequence expresses noticeably different normalized effects across their bases. Effects for sequence 135, for example, oscillate between slightly below 0 to 0.4 for most of its positions, but position 75 expresses a large effect size, at around 1. On the other hand, there are a few base positions in sequence 20 that have noticeable effect size relative to 0, with position 20 and position 75 producing the most pronounced effects.

Interestingly, both sequences' profiles revealed a large effect size at 75. To investigate this behavior, we averaged each sequence's absolute normalized effect, the result of which is displayed in the "Average Absolute Normalized Effect" plot (bottom of Figure 7). Indeed, the average absolute effect confirms this pattern – the position 75 seems to have an outsized impact on its own reactivity value relative to other bases, whose averaged effect mellows below 0.4. Another noticeable pattern is the behavior between the 85th and 100th positions. In most of our sequences, position 80 has a low effect; position 85 has a slightly higher effect than does position 80; position 90 has a much larger effect than does position 85; position 95's effect drops compared to position 90; position 100's effect drops to nearly 0; and finally position 105 has a very slightly higher effect size than position 100.

Recall that in our training data, we do not actually have reactivity values for base positions higher than 68. Thus, the patterns that we see for position 75 and above may simply be an artifact of our model, and of our data construction. Upon further examination of position 75, we found that the original base pair was uracil (U) for all 2096 training sequences, likely leading to a greater perturbation effect than that in other bases. In general, it seems that our model suggests that only a few positions have a large effect (in magnitude) on their own reactivity values, while many of the positions in the sequence, especially near the middle, have minimal impact on their reactivity scores.

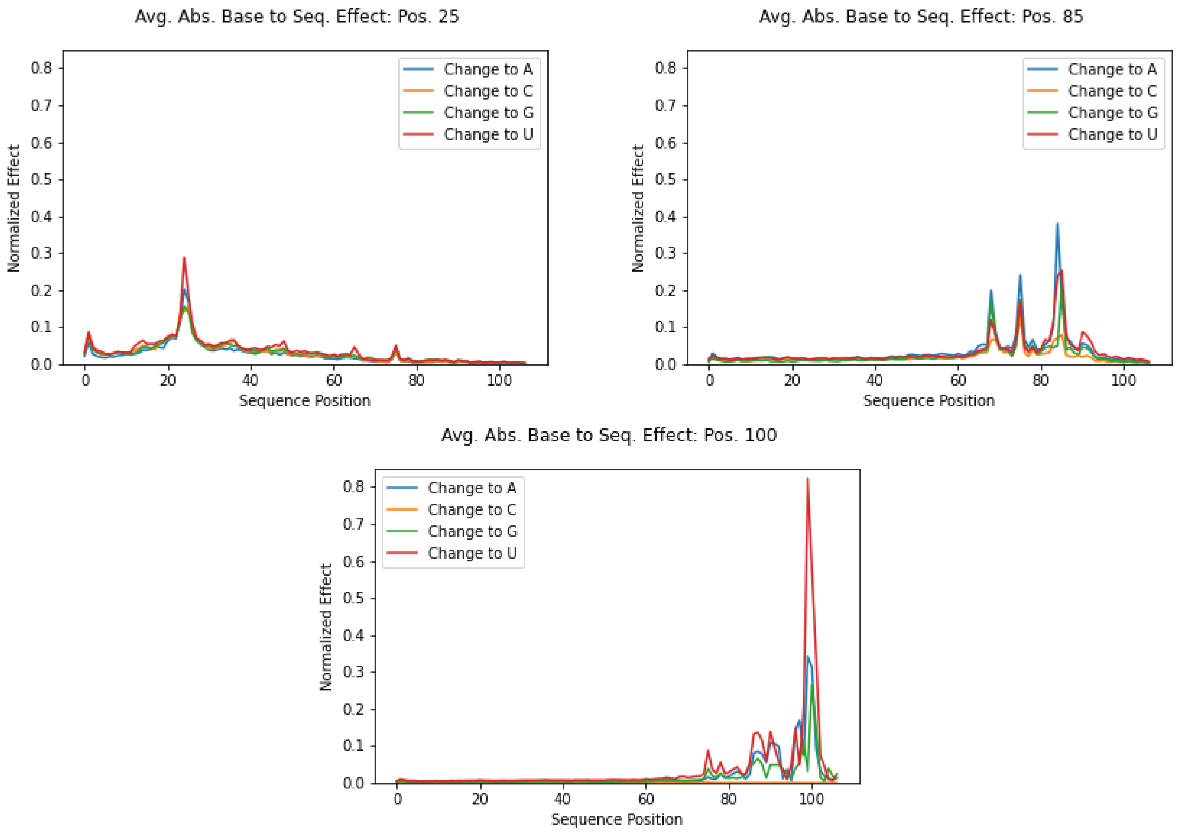


Figure 8: Absolute Average Base to Sequence Reactivity Plots

5.1.2 Point to Sequence Analysis

In **Point to Sequence Analysis**, we aim to understand the effect that changing the base value at position B has on the reactivity values of the rest of the sequence, and how these effects vary by the base value we perturb B to.

In order to understand how perturbing base values at position B impacts the whole sequence, we compute the average absolute base to sequence effect for every possible perturbation at position B. Specifically, for position B, we compute the 250 sequence reactivity profiles for all of the 4 possible perturbations. For each perturbation we average the absolute reactivity profile over the 250 sequences, so we are essentially looking at the effect that perturbing position B to base value N has, on average, upon the reactivity values of the entire sequence. In concrete terms, we compute the following for perturbation of position B to base value N:

$$\overline{\Lambda_{B,N}} = \frac{1}{250} \sum_i \Lambda_{iB_N} \text{ where}$$

$$\Lambda_{iB_N} = \{|\mu_{B,N}^{(j)}|\}_{j=1,..,107}^{i}$$

where i is the sequence number, B is the base position we are perturbing, N is the base value we perturb to, and j is the position in the sequence. Thus, $\overline{\Lambda_{B,N}}$ is the vector of average absolute reactivity effect for perturbing base position B to base value N. We show a sampling of the results below. Each plot overlays $\overline{\Lambda_{B,N}}$ for $N = A, C, G, U$ for a select few positions B. Note that the positions for which we have selected plots show patterns similar to their relative neighbors' average absolute base to sequence profiles.

We can see from the plots in Figure 8 that positions along the sequence display some similar features and some markedly different features. All position's have tapering effects; that is, a given position B's influence on reactivity values of positions far away from B is negligible. Changing position 25's base value, in particular, has steep effects on very close neighbors, with the effect tapering off very quickly as we move away from position 25 in either direction. Position 100 reflects a similar pattern. Position 85, however, seems to sustain its effect more sharply, as it results in spikes of reactivity for bases more than 15 positions away. Generally, these patterns show that the model does not seem to carry influence of changing one base very far.

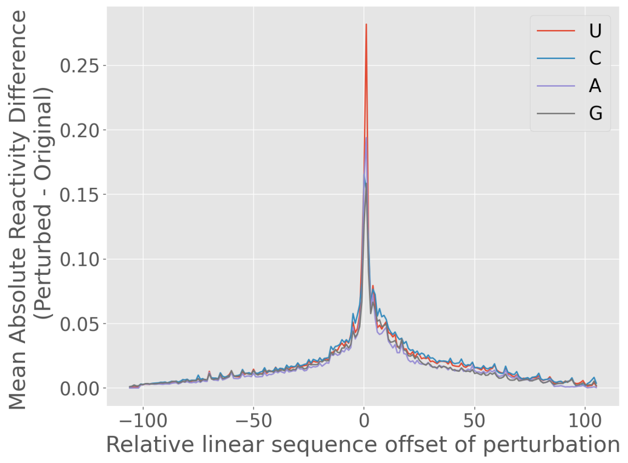
Interestingly, effect size also differs. While position 25 and 85 seem to have relatively similar effect sizes, around 0.3 to 0.4 absolute effect on reactivity, position 100 seems to have a much larger magnitude of effect size, reaching 0.8, when it is perturbed to U. For perturbations to other bases, however, its effect size is comparable to that of position 25 and 85.

This analysis suggests that the model has a "short-term" attention span; that is, the model does not tend to use information from positions far away from B when predicting B's reactivity value.

5.1.3 Sequence to Point Analysis

In **Sequence to Point Analysis**, we aim to understand the effect of perturbing the sequence some distance (`offset`) away from a given position in the sequence. Specifically, we measure the reactivity at a position B after perturbing bases +/- `offset` away from that position. We compute the effect of this perturbation by subtracting the non-perturbed (old) value from the perturbed (new) value.

We can compute these relative effects in a sliding-window fashion across all base positions in a sequence. We bin these effects by the `offset` as well as the base type (U, C, A, or G) that the base was perturbed to. We can then compute the mean effect of perturbations at `offset` for all base positions and sequences. Results are displayed in Figure 9.



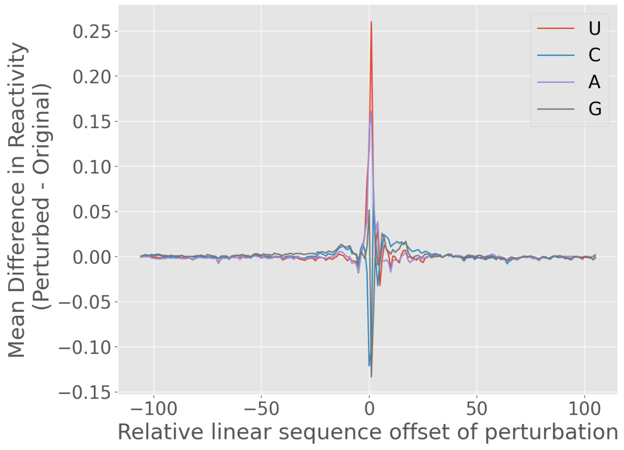


Figure 9: Sequence to base analysis results. Each figure describes the effect of perturbing a base, `offset` bases away from the reference. The left figure describes the mean absolute difference, whereas the right figure describes just the mean difference.

As depicted in Figure 9 (left), we can see that perturbations with small offsets have the greatest average absolute effect, with the maximum effect realized when perturbing the reference base itself (`offset`=0). The effect dissipates as the offset increases in magnitude, meaning that moving the perturbation further and further away corresponds to a decrease in the average effect. Intuitively, this makes sense, as the chemical properties of a base distant in linear sequence will, on average, have a smaller effect on the chemical properties of the reference base. This result primarily serves as a sanity check that the model behaves as we might expect.

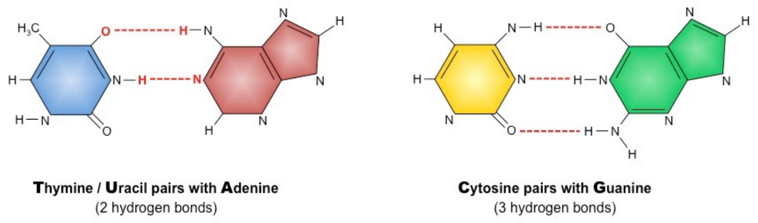

Thymine / Uracil pairs with Adenine
(2 hydrogen bonds)
Cytosine pairs with Guanine
(3 hydrogen bonds)

Figure 10: Complementary base pairing of nitrogenous bases. Notice that U-A pairing comprises two hydrogen bonds, whereas G-C pairing comprises three. Image Credit: [25].

Figure 9 (right), depicts the mean *raw* difference in predicted perturbation. We notice that the mean raw effect of perturbing bases beyond about +/- 25 is near zero, suggesting that the absolute effect observed in (left) at those offset values may simply be noise amplified by the absolute value. Within an offset of magnitude 10, however, we do see a noisy effect on the reactivity. Interestingly, we see a variable effect based on the perturbation type. On average, perturbations to U and A (red and purple traces) appear to have a local destabilizing effect (increase the reactivity of a base), while perturbations to C and G appear to have local stabilizing effects. This finding confers with a biological understanding of RNA base pairing. Bases G and C form a base-pair bond with three hydrogen bonds, whereas bases A and U form a base-pair bond with two hydrogen bonds (see Figure 10) [25]. Thus, a perturbation to A or U nearby has the likely effect of decreasing the number of base pair bonds by one, which has a destabilizing effect. Conversely, a perturbation to G or C nearby has the likely effect of increasing the number of base pair bonds by one, which has a stabilizing effect. Although these observations are preliminary, it is encouraging that our model appears to understand these differences in base pairing, despite never being "taught" the underlying chemistry of the base pair bonding.

6 Discussion

6.1 Model Performance

Our GCN model outperforms the CNN-based baseline by a large margin on both private and public tests. To better understand the model performance, we calculated the MSE (Mean Squared Error) and plotted the labels versus predictions for each task. The MSE for the reactivity, the degradation rate with Mg at pH =10, and the degradation rate with Mg at 50°C is 0.087, 0.255, and 0.125 respectively. From the MSE scores, we noticed that our model performs much better in predicting reactivity than predicting degradation rate with Mg at pH =10. Thus, performing task-specific optimizations, such as stopping some tasks early, or training a separate model to only predict degradation rates at pH =10, may be beneficial. As shown in Figure 11, our GCN model generally underestimates the degradation rate and rarely predicts any degradation rate of more than 5. In the training example, bases with degradation rate of more than 5 appear with a low frequency (less than 0.2 %), which explains why our model barely predicts any high degradation rate.

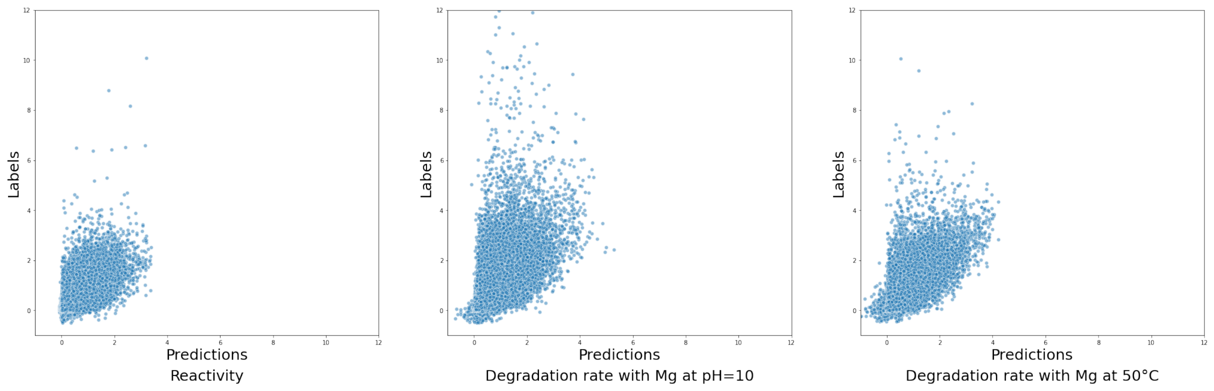


Figure 11: Labels vs. Predictions for each task

During our experiment, there is a consistent gap of performance between the public test set and private test set, which is also observed in models in the Kaggle competition. The MCRMSE on the private test set is 0.12 higher than that on the public test set. The measurement of degradation rates is up to 91 bases in the private test set and is only up to 68 bases in the public test and training set. Thus, we hypothesized that bases 69 $\sim$ 91 in the private test set will have higher loss, which contributes to the worse performance on the private test set. We calculated the MCRMSE scores and plotted labels versus predictions for the first 68 bases and bases 69 $\sim$ 91 in the private test set respectively. Surprisingly, as shown in Figure 12, our model has better performance on bases 69 $\sim$ 91 compared to the first 68 bases. Besides, the MCRMSE for the first 68 bases is 0.405 and the MCRMSE for the bases 69 $\sim$ 91 is only 0.317. Thus, the later bases do not contribute to the worse performance on the private test set.

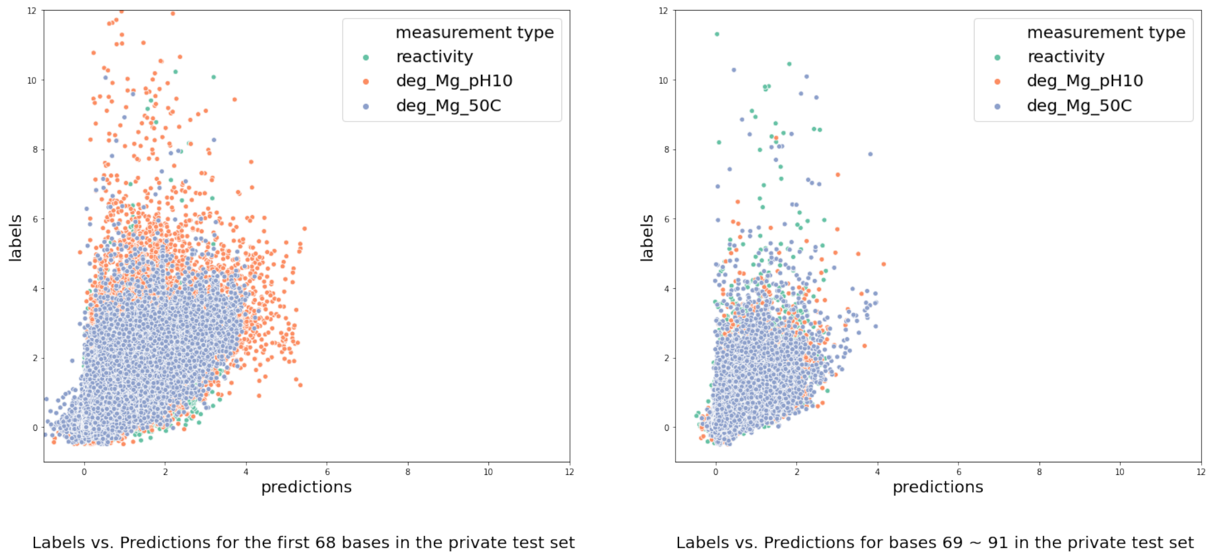


Figure 12: Labels vs. Predictions for the first 68 bases and bases 69 $\sim$ 91

Therefore, we hypothesized that the sequences in the two test sets are distinct, which contributes to the gap in performance. We analyzed the sequences in the training, public test set, and private test set using dimension reduction techniques as one of the Kaggle posts suggested. We encoded those sequences as an array of integers and performed t-SNE dimensionality reduction on them. The sequences in the private test sets are truncated to 107 bases to ensure they have the same dimension as that in the training and public test sets. The result of t-SNE reduction [26] on sequences is plotted in Figure 13. We found that the sequences in the private test set form a completely different distribution compared to the training and public test set. Although we truncated the sequences in the private test set, all sequences, including the training, public and private test sets, are not complete RNA molecules and were truncated before they were released. Thus, we believe the difference in the distribution contributes to the worse performance on the private test sets. To further improve the model, we think that data augmentation techniques are necessary to help the model to generalize better on the private test sets.

6.2 Relation to Winning Models from Kaggle

Our mRNA dataset is derived from a Kaggle competition, so we can conveniently compare our model's performance to others from the competition. Many of the top-performing competitors have released informal write-ups describing their approaches and general architectures. In Figure 14, we show a comparison of our best model's performance (the GCN with GRU architecture) against the top-three performing models.

We note that the winning models from the Kaggle competition outperform our best model, although only by 0.04 MCRMSE on the private test set. Notably, many of the leading models achieved very similar loss scores (around 0.34 overall), and the difference from 1st to 10th place is marginal (0.00375). This suggests that the top-performing models maxed out their performance at around the same level.

What, then, might account for the gap in performance between our best model and the best models of the Kaggle competition? As a disclaimer, we note that since we were not directly participating in the competition, we prioritized constructing and interpreting our models, rather than only improving our scores on the Kaggle leaderboard. Nevertheless, it is still useful to compare the different approaches.

Many of the top competitors reported basing their final models on an architecture released about midway through the competition: "AE pretrain + GCN + Attention." This architecture uses a graph convolutional network fit with attention modules. Its weights are pre-trained using an auto-encoder-like, unsupervised process whereby inputs are reconstructed

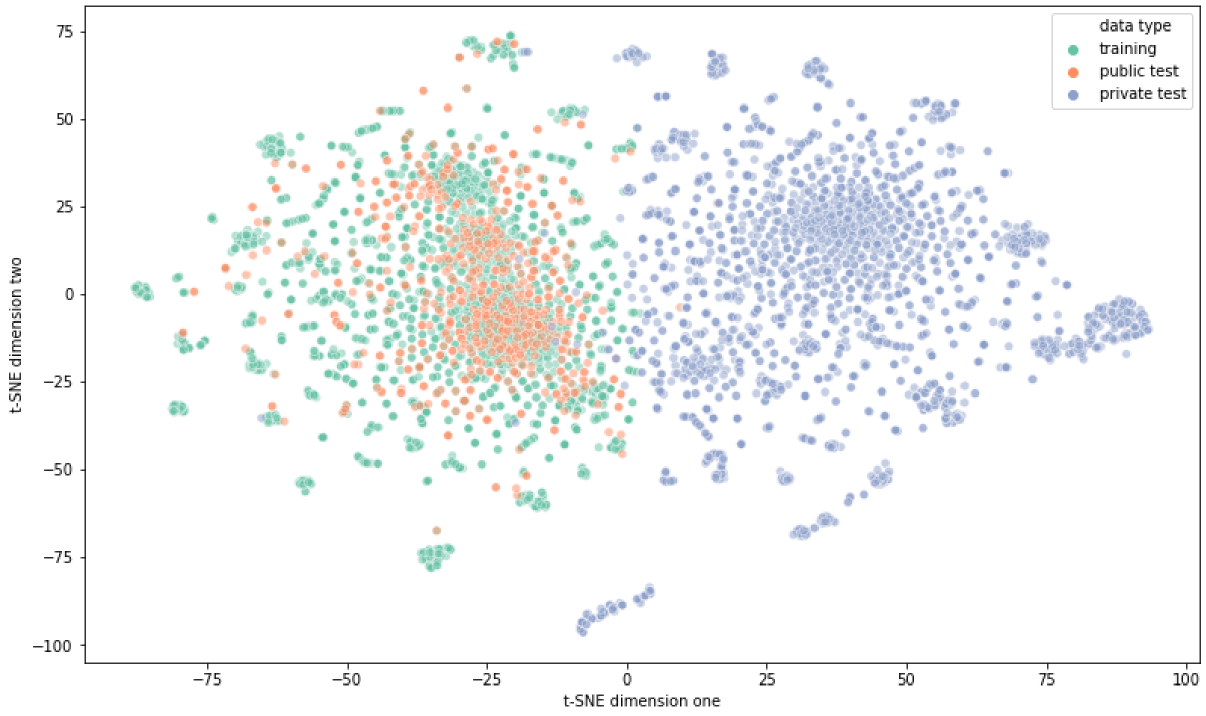


Figure 13: t-SNE reduction on the sequences in the training set, public test set, and private test set

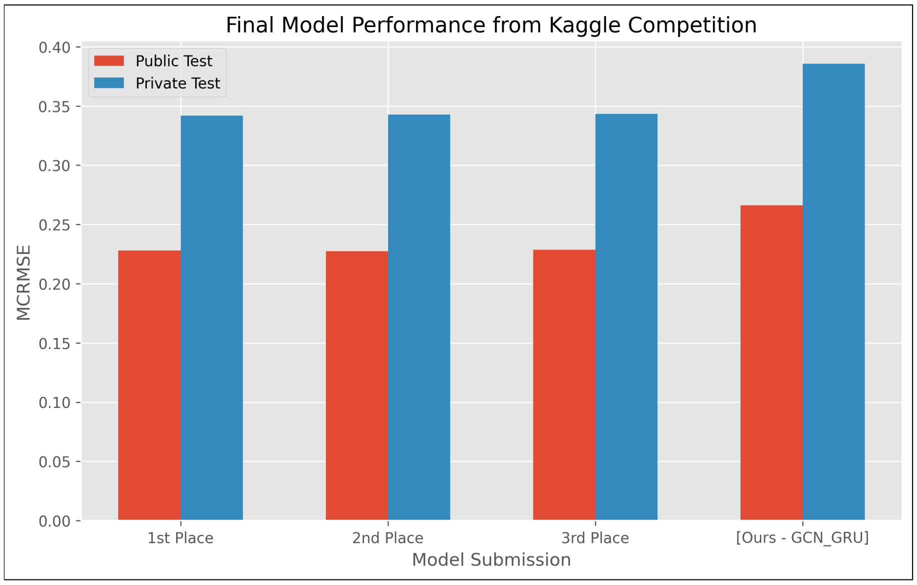


Figure 14: Comparison of our top-performing model (far right) against the three top-performing models from the Kaggle competition.

after random dropout in the sequence. We note that this approach is similar to what we performed with pretraining the node-embedding in the GCN. Indeed, pre-training the model to recover details of the RNA sequence seems to offer a benefit to training. We also note the addition of attention in this architecture. Future work could investigate the benefits of using attention with our model architecture.

Another general theme of the top-performing models is extensive and creative data augmentation. As aforementioned, the overall dataset is relatively small, so it makes sense that data augmentation strategies would be important to avoid pitfalls like overfitting. In addition, as depicted in Figure 13, the private test set is distributionally distinct from the training and public test sets, so any methods that successfully augment training data to account for this difference would be beneficial in model generalization to the private test set. Competitors report a variety of data augmentation strategies such as: pseudolabelling, inverting the sequence, and perturbing the RNA sequence at various base pairs. Future work could investigate robust and effective methods for data augmentation in RNA structure data.

7 Conclusion

Here, we investigate the utility and interpretability of deep neural network architectures for the relevant problem of predicting mRNA sequence degradation. We find that graph convolutional neural networks (GCNs) consistently outperform standard convolutional neural network (CNN) architectures for the task of base-wise reactivity prediction. However, for both our models and the top models submitted to the Kaggle competition, much is left to be desired in terms of accurate predictions (recall Figure 12) at a base-wise level. Increased availability of ground truth data, rational data augmentation strategies, and a better understanding of distributional shifts across different datasets (recall Figure 13) would be most beneficial in developing and training better models.

Despite the existing data and modelling challenges, inference on and preliminary interpretation of our trained model was still able to provide valuable insights. We find that the model, when predicting on a given position B, is paying attention to the identity of relevant bases near B's vicinity. Additionally, the model is able to recover, without prior knowledge, the effects of biochemical differences in base pair bonding characteristics (i.e. the H-bond count in G-C versus A-U). These preliminary investigations suggest that a well-trained model with access to a representative dataset could provide valuable clues to researchers working to understand important factors of mRNA degradation. Additionally, a well-trained model may guide engineering insight into the process of developing synthetic mRNA molecules.

8 Code and Data Availability

All of our code can be found at: `https://github.com/schlagercollin/covid-mrna-degradation`. All of the data used can be found through the Kaggle competition website: [8]: `https://www.kaggle.com/c/stanford-covid-vaccine/data`.

References

[1] Esteban Ortiz-Ospina Max Roser, Hannah Ritchie and Joe Hasell. Coronavirus pandemic (covid-19). *Our World in Data*, 2020. https://ourworldindata.org/coronavirus.

[2] Mangalakumari Jeyanathan, Sam Afkhami, Fiona Smaill, Matthew S. Miller, Brian D. Lichty, and Zhou Xing. Immunological considerations for COVID-19 vaccine strategies. *Nature Reviews Immunology*, 20(10):615–632, September 2020.

[3] Peter Loftus, Jared Hopkins, and Bojan Pancevski. Moderna and pfizer are reinventing vaccines, starting with covid. *Wall Street Journal*, Nov 2020.

[4] Nicholas A. C. Jackson, Kent E. Kester, Danilo Casimiro, Sanjay Gurunathan, and Frank DeRosa. The promise of mRNA vaccines: a biotech and industrial perspective. *npj Vaccines*, 5(1), February 2020.

[5] Ambro van Hoof and Roy Parker. Messenger rna degradation: beginning at the end. *Current Biology*, 12(8):R285–R287, 2002.

[6] Anne-Lise Fabre, Marthe Colotte, Aurélie Luis, Sophie Tuffet, and Jacques Bonnet. An efficient method for long-term room temperature storage of rna. *European Journal of Human Genetics*, 22(3):379–385, 2014.

[7] Jamie Ducharme. Why you may not be able to get pfizer's frontrunner covid-19 vaccine. *Time magazine*, Nov 2020.

[8] OpenVaccine. Openvaccine: Covid-19 mrna vaccine degradation prediction. *Stanford University, Eterna*, Sept 2020.

[9] Ronny Lorenz, Stephan H Bernhart, Christian Höner Zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for molecular biology*, 6(1):26, 2011.

[10] Rune B Lyngsø and Christian NS Pedersen. Rna pseudoknot prediction in energy-based models. *Journal of computational biology*, 7(3-4):409–427, 2000.

[11] Rune Lyngsø and Christian Pedersen. Rna pseudoknot prediction in energy-based models. *Journal of computational biology : a journal of computational molecular cell biology*, 7:409–27, 02 2000.

[12] Kyle E Watters and Julius B Lucks. Mapping rna structure in vitro with shape chemistry and next-generation sequencing (shape-seq). In *RNA Structure Determination*, pages 135–162. Springer, 2016.

[13] Matthew G Seetin, Wipapat Kladwang, John P Bida, and Rhiju Das. Massively parallel rna chemical mapping with a reduced bias map-seq protocol. In *RNA Folding*, pages 95–117. Springer, 2014.

[14] Zichao Yan, William L Hamilton, and Mathieu Blanchette. Graph neural representational learning of RNA secondary structures for predicting RNA-protein interactions. *Bioinformatics*, 36(Supplement_1):i276–i284, 07 2020.

[15] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[21] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[22] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, 2013.

[23] Robert M. Dirks, Milo Lin, Erik Winfree, and Niles A. Pierce. Paradigms for computational nucleic acid design. *PMC*, 34(4):1392–1403, 2004.

[24] Padideh Danaee, Mason Rouches, Michelle Wiley, Dezhong Deng, Liang Huang, and David Hendrix. bprna: large-scale automated annotation and analysis of rna secondary structure. *Nucleic Acids Research*, 46(11):5381–5394, 2018.

[25] Brent Cornell. Nitrogenous bases, 2016.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.