# Comparing Computational Methods for RNA Secondary Structure Prediction

Harrison LaBollita[1] and Petr Šulc[2]

[1]Department of Physics, Arizona State University, Tempe, AZ 85281 USA
[2]Center for Biological Physics, Arizona State University, Tempe, AZ 85281 USA

**Abstract**

Text goes here.

## Contents

**1 Introduction** **1**

**2 Datasets** **1**

**3 CNN Predicts Secondary Structure** **2**
3.1 Matrix Representation of RNA . . . . . 2
3.2 Model . . . . . 2
3.3 Results and Discussion . . . . . 3

**4 Stem Level Gillespie Algorithm** **4**
4.1 Stem Level Implementation . . . . . 4
4.2 Results and Discussion . . . . . 6

**5 Conclusion** **7**

## 1 Introduction

## 2 Datasets

We used two independent datasets to benchmark the performance of our CNN and Gillespie algorithm, respectivley. The dataset we used to train, validate and test our CNN contained over 50,000 RNA sequences all of the length 30 nucleotides (ntds). This dataset was generated using NUPACK software [3]. While it is desirable for CNN to function with variable length sequences, to benchmark performance of our model it was sufficient to use homogenous length sequences.

For our CNN model, we must encode the sequence and dotbracket information in such a way for the computer to understand it. The sequences are transformed into a matrix, which is described in more detail in a later section. The dotbracket representation is also transformed into a $N \times 3$ matrix, where $N$ is the number of ntds in the sequence and the three columns represent '(', ')', or '.'. A 1 is placed in the

appropriate represenation and 0's are placed in the remaining columns. For example, if the dot bracket representation of the secondary structure was '((..))', then we would encode this information in the following matrix:

((..))

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

We created a second dataset from the Rfam database, which is an online colletion of thousands of RNA families that contain the sequence, secondary structure, and the free energy of the secondary structure [4, 7]. This dataset contains 27 sequences with variable lengths. The complete dataset statistics are contained in Figure 1. In a later section, we benchmark the performance of our stem level gillespie algorithm compared to a leading secondary structure predictor, Vienna RNA [2].

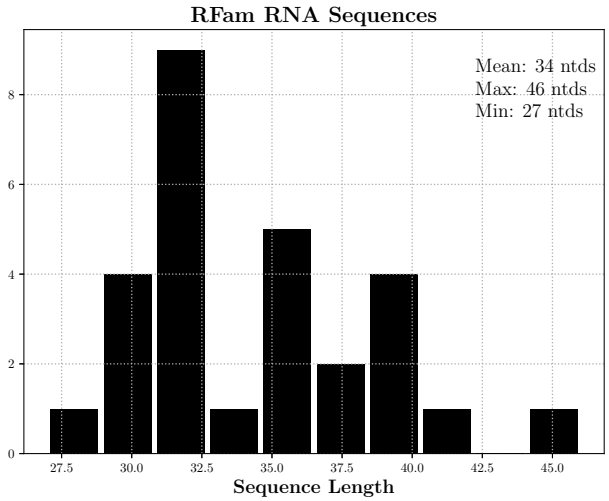


Figure 1: The Rfam dataset contains 27 total sequences with variable length. The average length is 34 ntds. The longest sequences is 46 ntds and the shortest sequence is 27 ntds.

# 3 CNN Predicts Secondary Structure

Convolutional neural networks (CNN)s are machine learning algorithms that are primarily used for image classification problems. The input layer is the images RGB values, which are then fed through multiple convolution layers, that are finally connected to a standard fully-connected feed forward neural network. Ultimately, to use a CNN one must have data that can be interrupted as an image. Following [9], we have replicated their CNN in PyTorch [6] in efforts to achieve similar results.

## 3.1 Matrix Representation of RNA

RNA sequences are comprised of any combination of four nucleotide bases: adenine (A), guanine (G), cytosine (C), and uracil (U). When using ML techniques for RNA secondary structure prediction, a formidable challenge is how to encode the sequence as input for the ML architecture. We have chosen to encode the sequence as a matrix in order to use a CNN architecture. The rules of folding a RNA secondary structure are very simple: A pairs with U, G pairs with C, and sometimes G pairs with C. The canonical pairs A-U and G-C are called Watson-Crick pairs (CITE), while the G-U pair is known as the wobble pair. Following these simple rules, we can build a weighted matrix based on which base is more likely to pair with whom. The complete matrix building algorithm is explain in [9], and our source code is available on this GitHub repository.

## 3.2 Model

The architecture of our CNN is simple. The model contains two convolutional layers, two pooling layers, and three fully-connected layers. We use a binary cross-entropy loss function, since we are predicting either a value of 0 or 1. We optimized the hyperparameters in our model via configuration space searching, where we randomly choose parameters from our con-

figuration space, train our network and log the performance. We found that a learning rate of 0.001 and momentum of 0.9 performed the best without overfitting.

We have illustrated the workflow of our CNN Model in Figure 2. We begin with an RNA sequence, convert the sequence into a matrix, then feed it into our CNN. The output of the CNN is a $N \times 3$ matrix, where $N$ is the number of nucleotides in the sequence and our model's predicted values of each nucleotide being paired or unpaired. From this information, we reconstuct the dotbracket representation by choosing the highest value in each row as the model's choice for paired ("(" or ")") or unpaired (".") and compare our predicted structure with the actual secondary structure for this sequence.

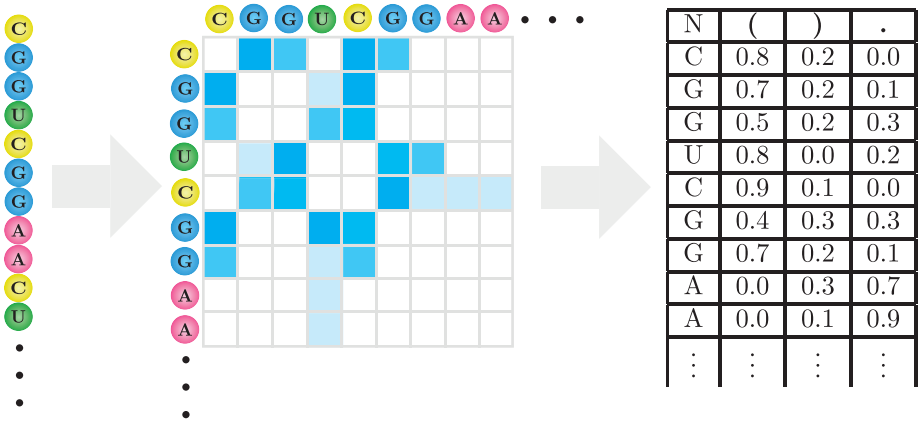


Figure 2: We convert our dataset of RNA sequences into a new dataset of RNA matrices. These matrices are used as inputs into our CNN. Notice that the darker blue squares correspond to heavier weighted base meaning more likely to be paired. The output of our network is the matrix on the far right which we use to reconstruct the dot bracket represenation of the secondary structure.

We trained and validated our CNN model on dataset containing 50,000 uniform length RNA sequences following the standard condition, where 80% of the data is used for training and 20% of the dataset was used for validation. We trained our network for 100 epochs with a batchsize of 100. We quantified the accuracy of our model by counting the number of mistakes our model made when predicting the secondary structure. A perfect prediction corresponds to 0 mistakes.

### 3.3 Results and Discussion

In this section, we present our CNN model's performance on RNA secondary structure prediction. In Figure 3, we have plotted the training and validation accuracy of our model throughout the training session. We can see that the model converges to about 84% accuracy for both training and validation accuracies.

Our CNN model achieves similar results to the [9], however, our model will sometimes predict non-physical structures. Therefore, it is necessary that we include a dynamic layer in our model that corrects for these non-physical outputs. This is the central weakness of all non-physical secondary structure predictors. Without including knowledge of physical considerations these models will always produce unphysical secondary structures.

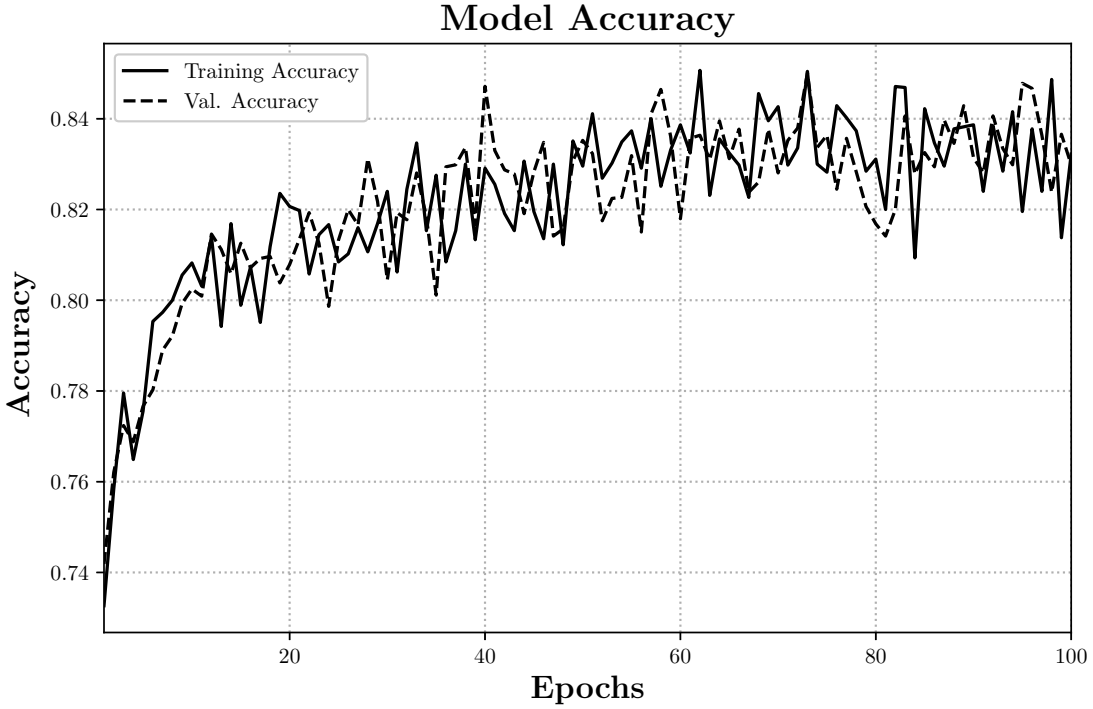


Figure 3: Training and validation accuracy over the 100 epochs. The solid black line is the training accuracy and the dotted black line is the validation accuracy.

# 4 Stem Level Gillespie Algorithm

## 4.1 Stem Level Implementation

There are two central approaches when writing a computer program to simulate RNA folding kinetics: base pairs or stems. The first approach considers the transitions from RNA states at the base pair level, which generates a microscopic picture of the RNA folding events [4]. The alternative approach is to consider large arrangements of base pairs or stems. The transition between RNA states is then completed via forming or breaking a complete stem rather than single base pairs. In this work, we propose a gillespie algorithm that works at the stem level. We intend for the algorithm to be used for long RNA stands ($>$ 100+ ntds), therefore, the stem level implementation is computationally more desirable than working at the base pair level.

As mentioned above we implement a gillespie algorithm, which is a stochastic simulation alogrithm to perform our RNA folding kinetics. The gillespie algorithm is very popular for simulating chemical reactions. For a more complete discussion of the gillespie algorithm please see Ref. [1]. We will briefly go over our algorithm and additionally, we have provided an illustration of our algorithm in Figure 4.

Our algorithm begins by first defining a set of moves and a set of transition rates that correspond to each move. There are two different rates: one for creating a stem, which we calculate from the entropy

$$\text{Rate of forming stem} = \exp\Big(\frac{\Delta S}{k_B}\Big), \quad (1)$$

where $k_B$ is the Boltzman constant in the correct units. The entropy term is the sum loop entropy, bond entropy, and duplex entropy for that state.

The rate of breaking is given by

$$\text{Rate of breaking stem} = \exp\Big(-\frac{\Delta G}{k_B T}\Big). \quad (2)$$

These rates form a probability distribution from which we can stochastically sample from to choose our next move. Upon transitioning from state $S_i \rightarrow S_j$ to our next state. We regenerate a new move set and re-calculate the transition rates between the current structure state and all other possible structure states. We have include a diagramtic illustration of our algorithm in Figure 4. Referencing the figure, one can see that we begin with a RNA sequence, then find the move set. From here we generate the probability distribution, where the sum of all of the transition rates sum to 1. We generate a random number and find in which been the random number falls. We then choose this move that corresponds to this rate whether it is to break a currently formed stem or form a new stem. We repeat this process until we reach a cutoff time that is set by the user. We experimented with different cutoff times to find a quasi-optimal time to let the algorithm run. This completes our description of our stem level gillespie algorithm in the next section we present our results and discussion.

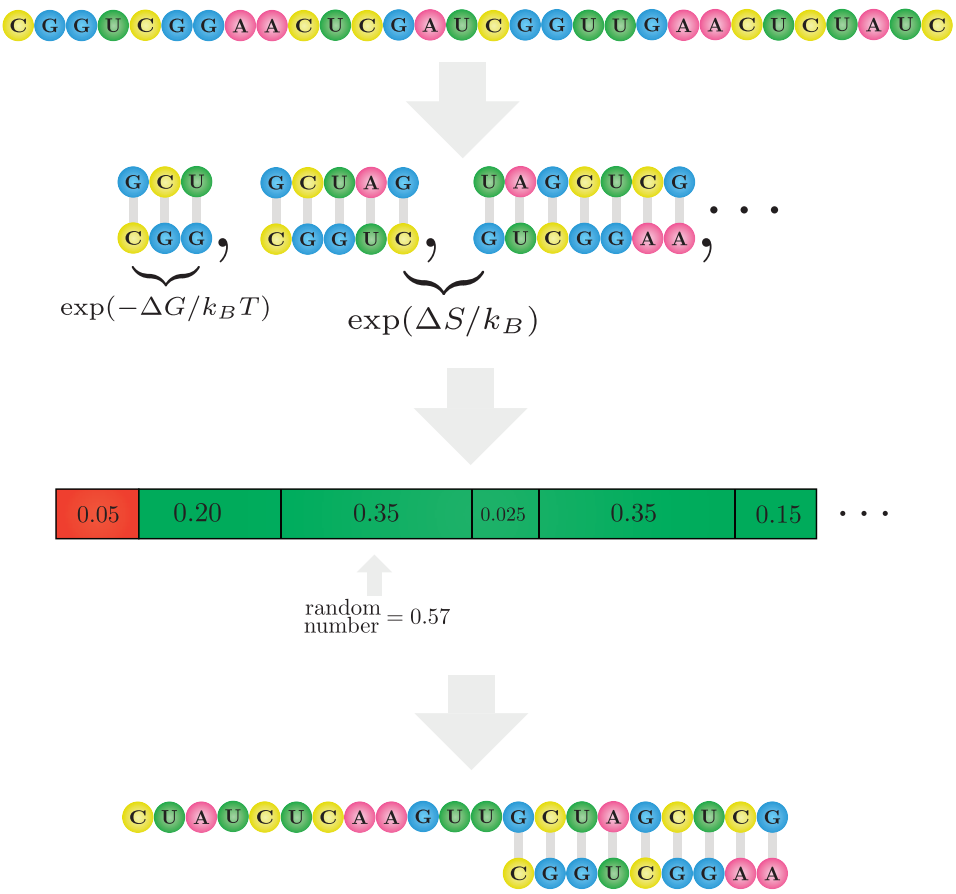


Figure 4: Above is an illustration of our algorithm. We begin with a open strand sequence. From here we generate a move set of all possible moves, in this case, all the possible stems that could form. We assign a transition rate from the current state $S_i$ to the next state $S_j$. We then stochastically choose a move to happen from our probability distribution. We propogate our structure to this next move, then regenerate a move set with new transtion rates. We continue to do this until we have reached our maximum running time. The result is our folded structure.

```
Sequence: CGGUCGGAACUCGAUCGGUUGAACUCUAUC
Time: 0.1945s | Added Stem: [[10, 17], [11, 16]] | Current Structure: ..........((....)).............
Time: 4.4331s | Added Stem: [[5, 29], [6, 28]] | Current Structure: .....((...((....)).........))
Time: 4.9914s | Added Stem: [[7, 19], [8, 18]] | Current Structure: .....((((.((....)))).........))
Time: 6.2564s | Added Stem: [[1, 24], [2, 23]] | Current Structure: .((..((((.((....)))).)).))..))
```

Figure 5: A sample output from our algorithm.

### 4.2 Results and Discussion

In this section, we present and discuss the benchmark results of our stem level gillespie algoritm and compare them with benchmark results of Vienna RNA program [2]. We tested our algorithm on the second dataset mentioned above, which was derived from the Rfam database. Recall this dataset consisted of approximately 30 variable length RNA sequences.

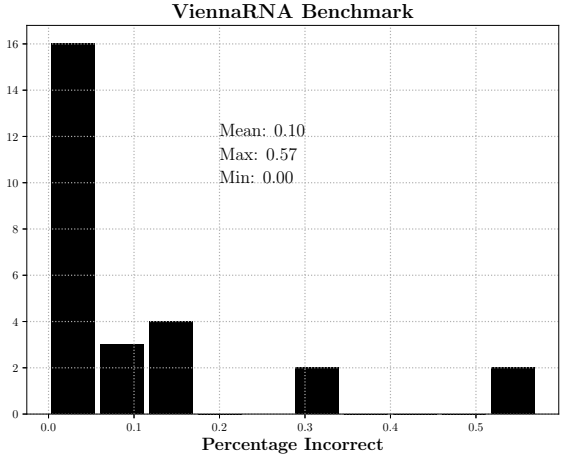


Figure 6: To benchmark our performance, we compared our algorithm to a popular RNA secondary structure predictor Vienna RNA [2]. The $x$-axis is the percentage of the structure that we incorrectly predicted. The mean was 10%, the worst prediction was 0.57%, and the best prediction was 57%.

In order to quantify the performance of our algorithm, we first obtain benchmark results on ViennaRNA's performance on our dataset. In Figure 6, we have plotted a histogram of ViennRNA's performance. The histogram shows the percentage of the secondary structure that ViennaRNA predicted incorrectly. We can see that the average error was about 10%. The worst prediction was 57% and the best prediction was 0% or 100% correct.

We now use the same data set with the Gillespie algorithm. Again, we have plotted a histogram of our program's performance. We can see from Figure 7 that on average we predicted 38% of the secondary structure incorrectly. Our best prediction was only 21% incorrect and our worst prediction was 54% incorrect.

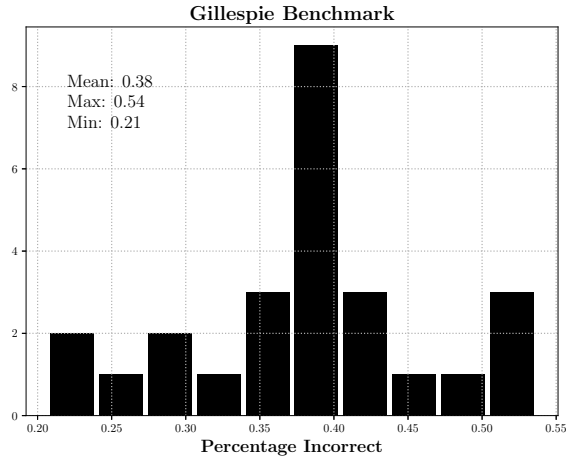


Figure 7: A histogram of the gillespie algorithm's performance. The $x$-axis represents the percentage of the structure that was incorrectly predicted by our algorithm. The mean is 38%, the worst predction was 54%, and the best prediction was 21%.

Compared to ViennaRNA our stem level Gillespie algorithm was inferior in performance. However, we have one advantage that ViennaRNA does not. Our algorithm considers pseudoknotted structures. This gives a tremendous advantage when compared to

# 5 Conclusion

The quick brown fox jumped over the lazy sheep dog.

## References


[1] Radek Erban, Jonathan Chapman, and Philip Maini. *A practical guide to stochastic simulations of reaction-diffusion processes*. 2007. arXiv: 0704.1908 [q-bio.SC].

[2] Ronny Lorenz et al. "ViennaRNA Package 2.0". In: *Algorithms for Molecular Biology* 6.1 (2011), p. 26. ISSN: 1748-7188. DOI: 10.1186/1748-7188-6-26. URL: https://doi.org/10.1186/1748-7188-6-26.

[3] Joseph N. Zadeh et al. "NUPACK: Analysis and design of nucleic acid systems". In: *Journal of Computational Chemistry* 32.1 (2011), pp. 170–173. DOI: 10.1002/jcc.21596. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.21596. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21596.

[4] Eric C. Dykeman. "An implementation of the Gillespie algorithm for RNA kinetics with logarithmic time update". In: *Nucleic Acids Research* 43.12 (May 2015), pp. 5708–5715. ISSN: 0305-1048. DOI: 10.1093/nar/gkv480. eprint: http://oup.prod.sis.lan/nar/article-pdf/43/12/5708/16659220/gkv480.pdf. URL: https://doi.org/10.1093/nar/gkv480.

[5] Ioanna Kalvari et al. "Rfam 13.0: shifting to a genome-centric resource for non-coding RNA families". In: *Nucleic Acids Research* 46.D1 (Nov. 2017), pp. D335–D342. ISSN: 0305-1048. DOI: 10.1093/nar/gkx1038. eprint: http://oup.prod.sis.lan/nar/article-pdf/46/D1/D335/23162120/gkx1038.pdf. URL: https://doi.org/10.1093/nar/gkx1038.

[6] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[7] Ioanna Kalvari et al. "Non-Coding RNA Analysis Using the Rfam Database". In: *Current Protocols in Bioinformatics* 62.1 (2018), e51. DOI: 10.1002/cpbi.51. eprint: https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/cpbi.51. URL: https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/cpbi.51.

[8] Ofer Kimchi et al. "RNA structure prediction including pseudoknots through direct enumeration of states". In: *bioRxiv* (2018). DOI: 10.1101/338921. eprint: https://www.biorxiv.org/content/early/2018/06/04/338921.full.pdf. URL: https://www.biorxiv.org/content/early/2018/06/04/338921.

[9] Hao Zhang et al. "A New Method of RNA Secondary Structure Prediction Based on Convolutional Neural Network and Dynamic Programming". In: *Frontiers in Genetics* 10 (2019), p. 467. ISSN: 1664-8021. DOI: 10.3389/fgene.2019.00467. URL: https://www.frontiersin.org/article/10.3389/fgene.2019.00467.