AN ABSTRACT OF THE DISSERTATION OF

Dezhong Deng for the degree of Doctor of Philosophy in Computer Science presented on December 10, 2019.

Title: Computational Approaches for RNA Structure Prediction with Dynamic Programming and Deep Neural Networks

Abstract approved: ______________________________

David A. Hendrix

Our goal is to build a system to model the RNA sequences that reveals their structural information by using efficient dynamic programming algorithms and deep learning approaches. We aim to 1) achieve linear-time for RNA secondary structure prediction based on existing minimum free energy models; 2) utilize deep neural networks to learn high-level features directly from RNA sequences without looking at any indirect information from MFE models, in order to predict RNA secondary structures directly; 3) we also investigate RNA structure visualization approaches.

Here we organize our line of research all the way from a novel annotated dataset to systematic RNA secondary structure prediction using deep learning, including bpRNA (a RNA structure annotation tool with its generated, large-scale RNA

meta-database bpRNA-1m), LinearFold (a linear-time dynamic programming algorithm for RNA secondary structure prediction), DeepSloop (a deep learning approach that learns complex rules to detect stem-loop-forming RNA sequences), and DeepStructure (an end-to-end RNA secondary structure prediction approach via deep neural networks). We also presented bpRNA-Visual for RNA structure visualization purposes.

Computational Approaches for RNA Structure Prediction with Dynamic Programming and Deep Neural Networks

by

Dezhong Deng

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented December 10, 2019
Commencement June 2020

ACKNOWLEDGEMENTS

First I would like to thank my parents, my grandparents, and my girlfriend. I cannot finish this PhD journey without the strong support from them.

Thanks to my advisor, professor David Hendrix, for taking me as a PhD student, guiding me with the research and supporting me for graduation.

Thanks to my committee members, including professor Alan Fern, professor Glencora Borradaile, professor Prasad Tadepalli, and professor Leonard Coop, for guiding me with the dissertation and a better-quality research.

Thanks to my collaborators, including Padideh Danaee, Mason Rouches, Michelle Wiley, Liang Huang, Kai Zhao, He Zhang, David Mathews, Douglas Holman, Juneki Hong, and others, for collaborating with me and helping me.

Thanks to people in the Hendrix Lab, including Padideh Danaee, Lillian Padgitt-Cobb, Rosalyn Fey, Juneki Hong, Nathan Waugh, Patrick Morar, Joseph Valencia, and others, for providing a nice and warm group for me, and helping me with the work.

I would like to also thank the Oregon State University and the Department of EECS for providing good environment and a position for me, and supporting me in these years.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDIX FIGURES

Chapter 1: Introduction

Ribonucleic Acid (RNA) is a polymeric molecule which plays essential roles in various biological systems, in coding, decoding, regulation and expression of genes. We are interested in building a system that can reveal structural information of an RNA sequence. For example, given an RNA sequence `"CUCGUCUAGUCAUUUCUUGCCCC ACUGGAGGUCGAG"`, we are curious if it forms a stem-loop, or more basic, what does its secondary structure looks like. Such a system should be trained on a labeled RNA dataset with minimal information.

The secondary structures and base-pairing interactions of RNAs reveal information about their functions [7, 8, 21, 39]. Starting from RNA secondary structure prediction [57], computational approaches for RNA biology have emerged and become an important part of computational biology. While traditional methods, especially the widely-used minimum free energy models, have made decent progress on modeling RNA sequences with structures[38, 36], deep learning approaches have shown strong potential to break the limitations of those models with the advantage of learning high-level features with a huge (up to billions) number of parameters from large-scale datasets [22].

Therefore, we are looking for such a system that can utilize these advantages of deep neural networks, and take on the structure prediction tasks well enough. Motivated by this, we present our line of research all the way from a novel annotated dataset to the systematic deep learning RNA secondary structure prediction approach.

In this dissertation, we will first go through the background knowledge, getting the basic concepts of the proposed techniques; then we cover part of the research of bpRNA which is our presented RNA data annotation toolkit including a large-scale dataset bpRNA-1m; we next introduce part of the research of LinearFold, in which we presented the first incremental linear-time dynamic programming algorithm to predict RNA secondary structures in linear-time.

While the algorithm of LinearFold has to be based on the MFE energy parameters (explained in the background section) which is heavily manual designed, we next explored the prediction of RNA structural information without it. The work DeepSloop is followed, describing our proposed Recurrent Neural Network approaches that can learn complex rules to detect stem-loop-forming RNA sequences; as a next step of DeepSloop, we have the research DeepStructure which predicts the whole RNA secondary structure directly from RNA sequences. Both DeepSloop and DeepStructure conclude achieved our goal, which are able to avoid looking at any energy knowledge or other manual designed knowledge, but directly learn to predict RNA structural information directly from large-scale datasets.

We also investigated RNA structure visualization tools, and presented bpRNA-Visual for RNA structure visualization purposes. This work is put into the Appendix section.

Chapter 2: Background

2.1 RNA sequence and its structure

Fig. 2.1 shows a segment of RNA sequence and its secondary structure. Consider an RNA sequence r which is a sequence of nucleotides $r[i] \in \{A, C, G, U\}$. We define its structure S_r to be a set of base-pairs $(r[i], r[j])$, with the restriction of each nucleotide cannot be paired twice: $S_r = \{(r[i], r[j])\}$. In Fig. 2.1, eight base-pairs are formed with blue lines.

There are exponential number of possible structures S'_r given one RNA sequence r, but under a certain environment, only one of them S_r is considered as the (most stable) secondary structure of it. Our goal is to design a system that can take r as an input and S_r as an output. Moreover, S_r as a base-pairing set, cannot fully reveal the structural information of RNA sequences [10], thus a systematic method is needed here. We will cover this part in Chapter 3 , which introduces bpRNA.

2.2 Minimum free energy models

Minimum free energy models [57], also known as MFE models, is the conventional way to solve the secondary structure prediction problem. The model has a energy functionf_E which takes the RNA sequence r and a structure S'_r and returns its free energy: free energy of $S'_r = f_E(r, S'_r)$. In thermodynamics, a RNA structure with

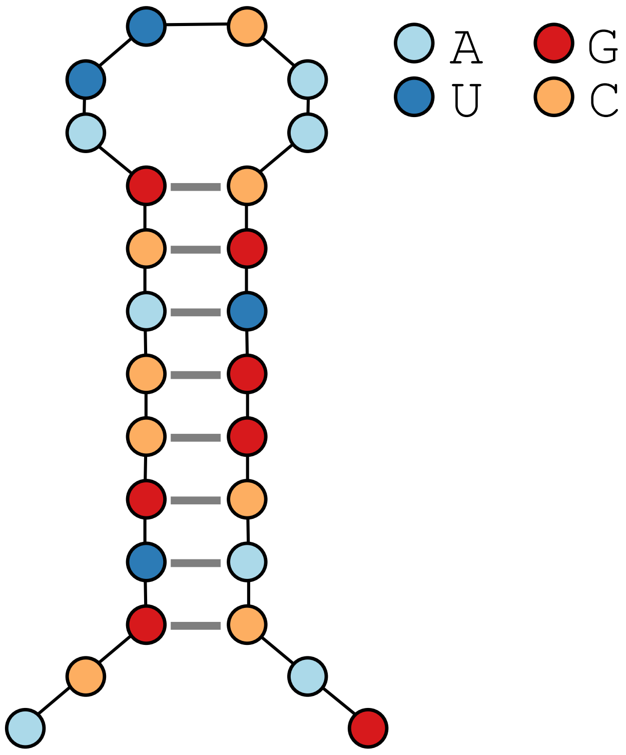


Figure 2.1: The secondary structure of an RNA sequence.

a higher energy is considered unstable and tend to fold (form a base-pair) and/or unfold to transform to another structure which has a lower free energy. Therefore, a structure with the minimum free energy could be a reliable way to model the secondary structure: $S_r = \text{argmin}_{S'_r} f_E(r, S'_r)$.

However, there is no exact method to calculate f_E: physical experiments can be performed to measure on a RNA sequence with its secondary structure, but this is resource consuming and can't be generalized to an algorithm. As a computational solution, the energy function is decomposed with local features for different structure types, and its value is being widely researched and manual designed, to simulate the original energy function f_E. The decomposition is designed to fit into a bottom-up dynamic programming algorithm (CKY algorithm), leading to a cubic-time, quadratic-memory solution for calculating the simulated minimum free energy, f^*_E, and its secondary structure, $S^*_r = \text{argmin}_{S'_r} f^*_E(r, S'_r)$. Fig. 2.2 is an example of energy decomposition, using the sum of local energy to estimate the free energy of the whole sequence; Fig 2.3 explains these local types for decomposed energy calculation.

However, despite the convenience from automatically predicting secondary structures from sequences, experimental results show that the simulated energy function f^*_E is far from perfect: models built on this decomposition can hardly reach 70% of the pairwise accuracy (accuracy of the base pairs) [33, 14] and thus limits the application of secondary structure prediction.

2.3 Deep neural networks and sequence models

Consider an input layer is a matrix representation (with numerical values) of what we want to model on, and an output layer is another matrix (can be a single node) of what we want to predict. A deep neural network (DNN) is multiple layers of network mapping the input layer all the way to output layer. Such a layer can be regarded as a linear and/or nonlinear computation from a matrix to another matrix, usually with a large parameter set. There is a large world of deep neural networks [32], and we just focus on part of it for our system.

The core part of a deep neural network is its model architecture, and usu-

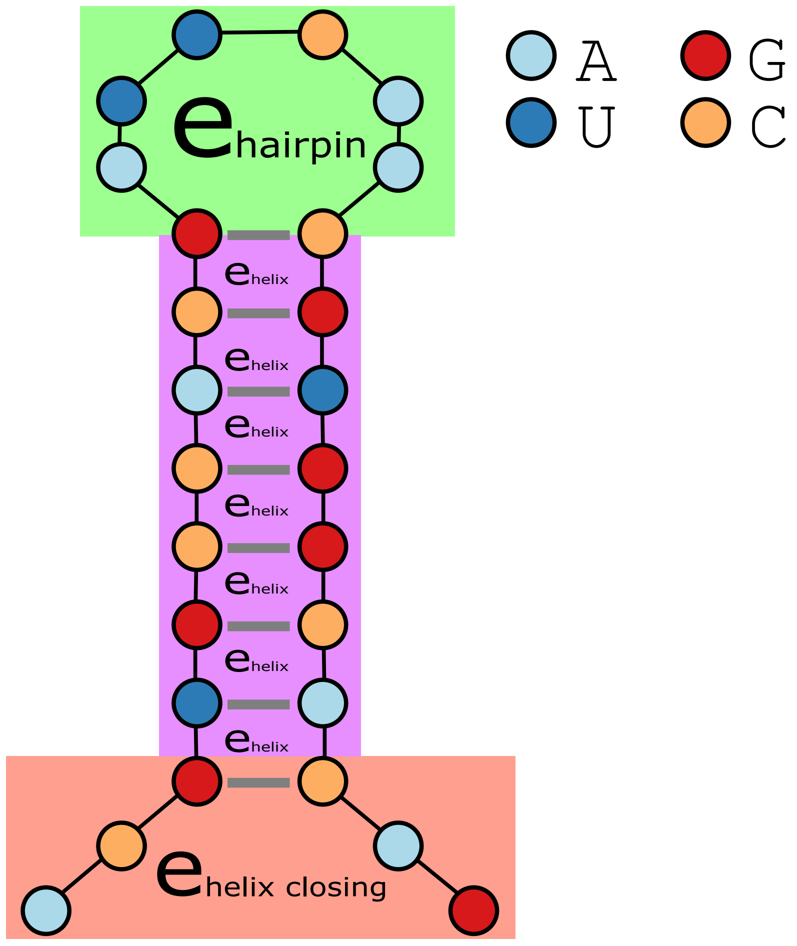


Figure 2.2: An example of the energy decomposition of RNA secondary structure.

ally with a reliable training strategy. For sequential data input (e.g., sentences in human language, RNA sequence), models including convolutional neural networks (CNN) [29], recurrent neural networks (RNN, sometimes combined with CNN layers) [35], and others (e.g., Transformer networks..) [53]. The work in this dissertation is mostly related to the bi-directional LSTM technique and the LSTM plus Attention architecture.

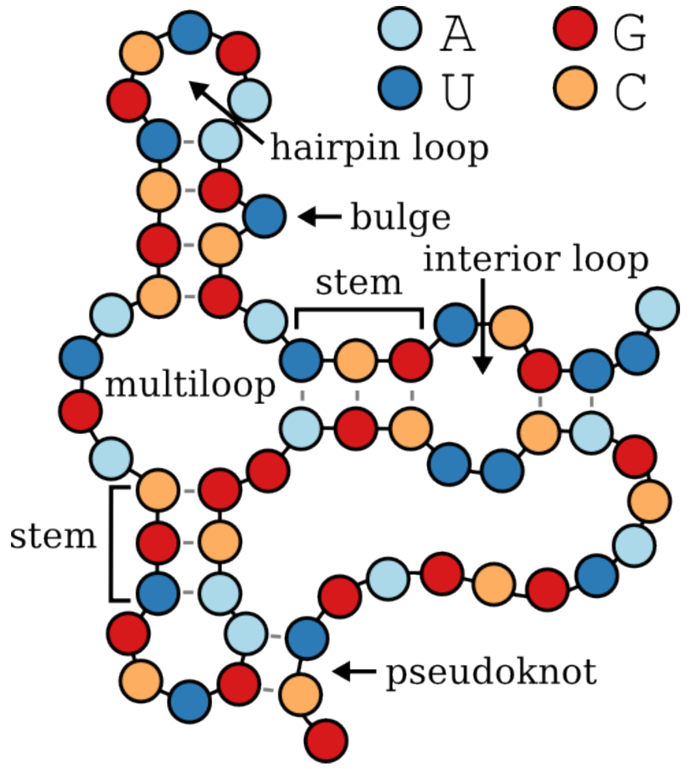


Figure 2.3: A brief example of RNA structure types. MFE models divide free energy into these types, and use a set of simulated, hand-engineered features for calculating the energy within each type. This figure was published in [10].

2.4 Bidirectional LSTM

Generally, the bi-directional LSTM technique is able to model sequential data, and map a (fixed or any) length of sequential input to either a scalar/vector (which could be a classification label, a regression number, or a rich class of features represented by the vector), or with a (fixed or any) length of sequence of vectors. See Fig. 2.4 for the overview.

Taking a vector of inputs, the bi-directional LSTM layers use a forward LSTM

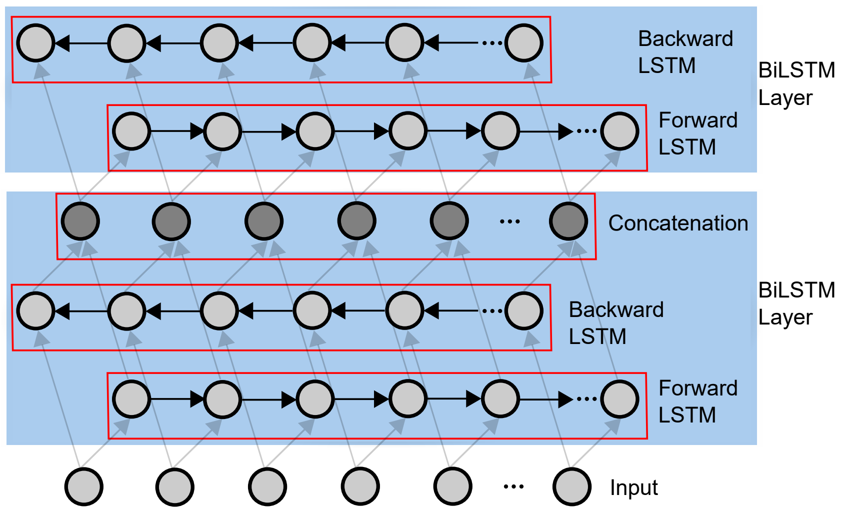


Figure 2.4: An illustration of the bi-directional LSTM technique as a sequence model. This architecture is used in part of the DeepSloop model.

and a backward LSTM (with the LSTM cell unit explained in Fig. 2.5), and each independently produces an output sequence of vectors (and a hidden sequence of vectors that would be used later or ignored). The sequential output of a BiLSTM layer is a concatenation of these two sequences of vectors to one sequence of vectors; the vector output of a BiLSTM layer takes a concatenation of the end of these two sequences of vector and produces one vector.

We used the BiLSTM technique in DeepSloop and DeepStructure which would be detailedly described in the later chapters.

2.5 Seq2Seq Models and Encoder-Decoder Networks

The Seq2Seq (sequence-to-sequence) models generally stand for a set of neural network models including an encoder layer and a decoder layer, which both the

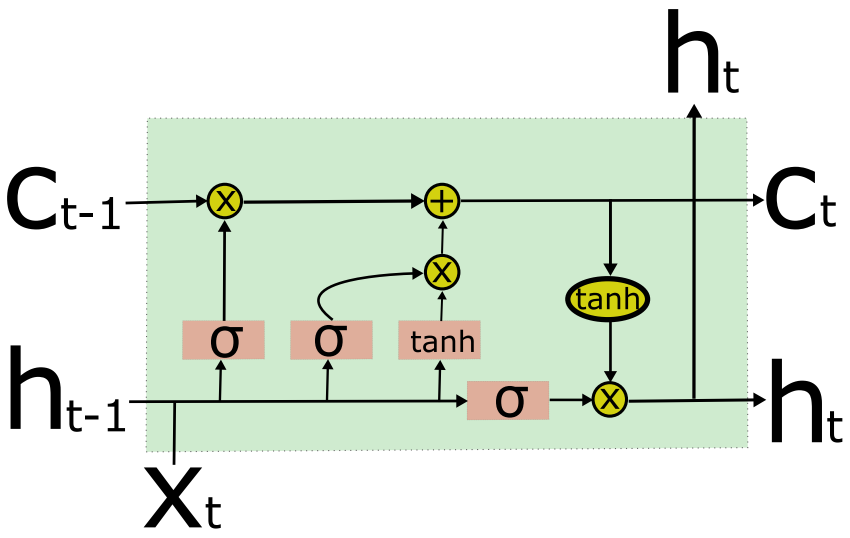


Figure 2.5: An LSTM unit, which takes a vector of input and a pair of vector (c, h), using math computations to produce the next pair of vector (c, h).

encoder and the decoder uses RNN (e.g., BiLSTM for encoder, LSTM for decoder) layers to produce a mapping from a sequence of vectors to another sequence of vectors. The Encoder-Decoder Network idea is from machine translation[48] and widely used in sequence-to-sequence tasks including speech recognition, image processing and natural language processing [30].

2.6 LSTM with Attention Layer

The LSTM with Attention architecture is a typical variant of the Seq2Seq framework. The attention technique here is to take the encoder output (the output can be regarded as two parts: an output vector and a sequence of output hidden vectors; attention works on this sequence) and provide extra information for the decoder layer; this extra information is usually a vector, either directly fed into the decoder input (first cell), or dynamically map the decoder hidden vector from any decoder cell and concatenate to the input of the next cell. The overview showed in Fig. 2.6 is a typical LSTM plus attention architecture and used in DeepStructure framework.

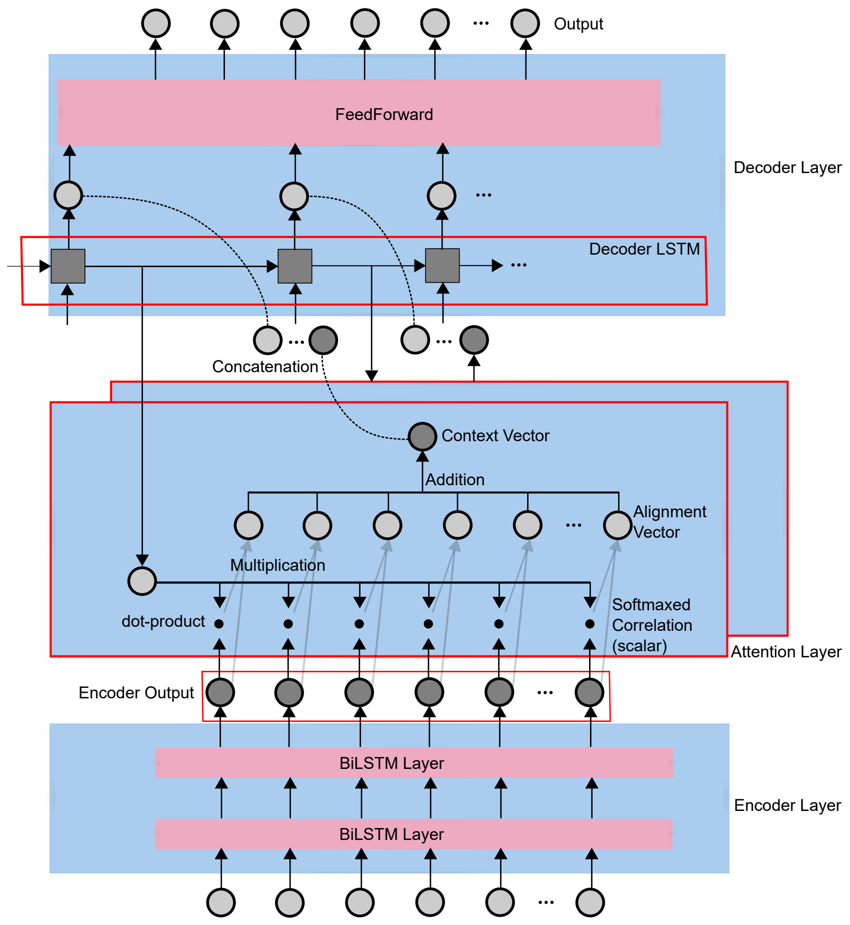


Figure 2.6: A brief overview of the LSTM plus Attention architecture. In this framework, both encoder layer and decoder layers are required to be RNNs which works with the input/output and hidden vector for each cell. This architecture is used in part of the DeepStructure Model.

2.7 Transformer Encoder and Transformer Decoder

Transformer Networks [53, 9] generally stack attention-based neural network layers in the Seq2Seq framework with multiple transformer encoder and decoder layers; each layers takes a sequence of vectors as input and output. A transformer encoder layer usually consists a self-attention layer and a feed-forward layer, while a transformer decoder layer usually consists a self-attention layer, a encoder-decoder attention layer (which takes the encoder output sequential vectors and makes attention with the current input sequential vectors), and a feed-forward layer. See Fig. 2.7, 2.8 for the layer overview.

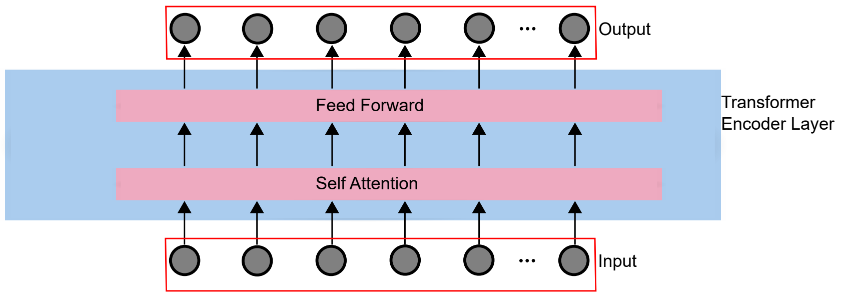


Figure 2.7: An overview of the Transformer encoder layer.

In the DeepStructure project, we have two architectures utilizing Transformer Networks: one uses both the Transformer encoder and the Transformer decoder; another one uses LSTM encoder with the Transformer decoder.

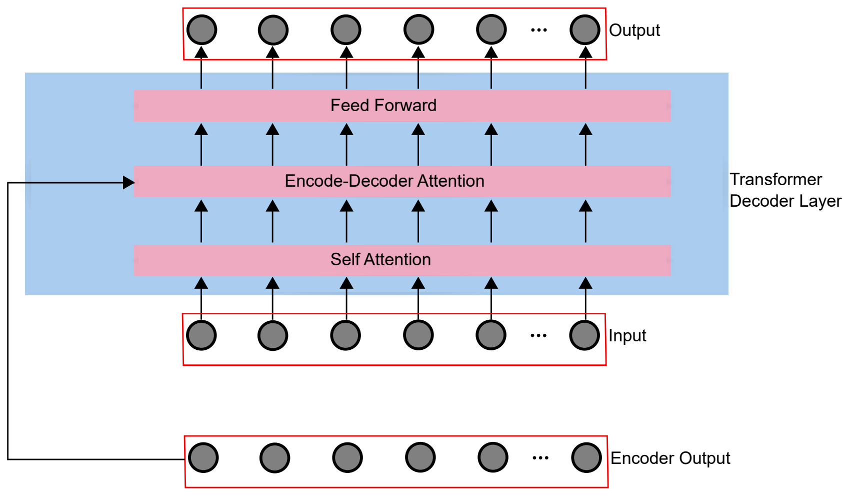


Figure 2.8: An overview of the Transformer decoder layer.

Chapter 3: bpRNA: Large-scale Automated Annotation and analysis of RNA Secondary Structure

3.1 Abstract

bpRNA is a novel annotation tool capable of parsing RNA structures, including complex pseudoknot - containing RNAs, to yield an objective, precise, compact, unambiguous, easily-interpretable description of all loops, stems, and pseudoknots, along with the positions, sequence, and flanking base pairs of each such structural feature. bpRNA also introduces several new informative representations of RNA structure types to improve structure visualization and interpretation. bpRNA-1m is further generated as a web-accessible meta-database, 'bpRNA-1m', of over 100,000 single-molecule, known secondary structures; this is both more fully and accurately annotated and over 20-times larger than existing databases. Both the bpRNA method and the bpRNA-1m database will be valuable resources both for the specific analysis of individual RNA molecules and large-scale analyses such as are useful for updating RNA energy parameters for computational thermodynamic predictions, improving machine learning models for structure prediction, and for benchmarking structure-prediction algorithms.

3.2 Overview of bpRNA

Ribo-nucleic acid (RNA) plays an essential role in all lives, with various functions including molecular scaffolding, gene regulation, and encoding proteins. Research studies focus on the secondary structures and base-pairing interactions of RNAs which reveal information about their functions [7, 8, 21, 39]. With the tremendous improvements on RNA structure prediction, the limited and outdated data resources start to constrain the research in this field. For example, the most detailed meta-database, RNA STRAND v2.0 [2], contains only ¡5,000 entries, with information not been updated in a decade. Another common fact for the resources is that, even with the base pairing data, it can be hard to infer their structural features since there's no systematic way to resolve the structural topology and identify all structural features given the base pairing relationships. Therefore, it is needed for a system analyzing RNA base pairing data and identify and annotate structural features.

We present 'bpRNA', a program for RNA structural topology that parses base pair data into detailed structure 'maps' providing relevant contextual data for stems, internal loops, bulges, multi-branched loops (multiloops), external loops, hairpin loops, and pseudoknots. This work is aiming to fill a long-standing vacancy that previous work to parse RNA structural topology does not handle pseudoknots or only analyze tertiary structures. A detailed meta-database consisting of more than 100,000 single-molecule RNA secondary structures called 'bpRNA-1m', is also released. We use 'bpRNA-1m' as the standard dataset for training in the projects of this proposal.

bpRNA is focused on these aspects of improvements against previous work: 1) it handles a detailed analysis of complex structures with pseudoknots, including Page-wise pseudoknot analysis; 2) it outputs both high-level and detailed-level of RNA structural information to help users understand the structure; 3) a complete bpRNA-1m dataset is released with accurately generated dot-bracket sequences for all structures including pseudoknots. Here we focus on the first one which the author of this proposal mainly contributes to.

3.3 RNA structure types, pseudoknots, and page number

RNA structure types include hairpin loops, bulges, stems, internal loops, multi-branch loops, external loops, and pseudoknots, showing in Fig. 2.3.A. An unpaired sequence with both ends meeting at the two strands of a stem region forms a hairpin loop (Fig. 2.3.B). An internal loop is defined as two unpaired strands flanked by closing base pairs on both sides; if only one of the strands has length zero, this forms a bulge (Fig 3.1.C/D). Multi-branch loops (multiloops) consist of more than two unpaired strands in one cycle, connected by stems (Fig 2.3.E); if this is not connected in a cycle, we call it external loops.

Pseudoknots (PKs) are defined by the crossing of the base-paired positions. Generally, consider two set of base pairs (usually stems), S_{bp0}, S_{bp1}, if $\forall (r[i], r[j]) \in S_{bp0}, (r[i'], r[j']) \in S_{bp1}, i < i' < j < j'$ or $i' < i < j' < j$, then S_{bp0}, S_{bp1} forms a pseudoknot. Pseudoknots are related to the planar graph since a pseudoknot-free

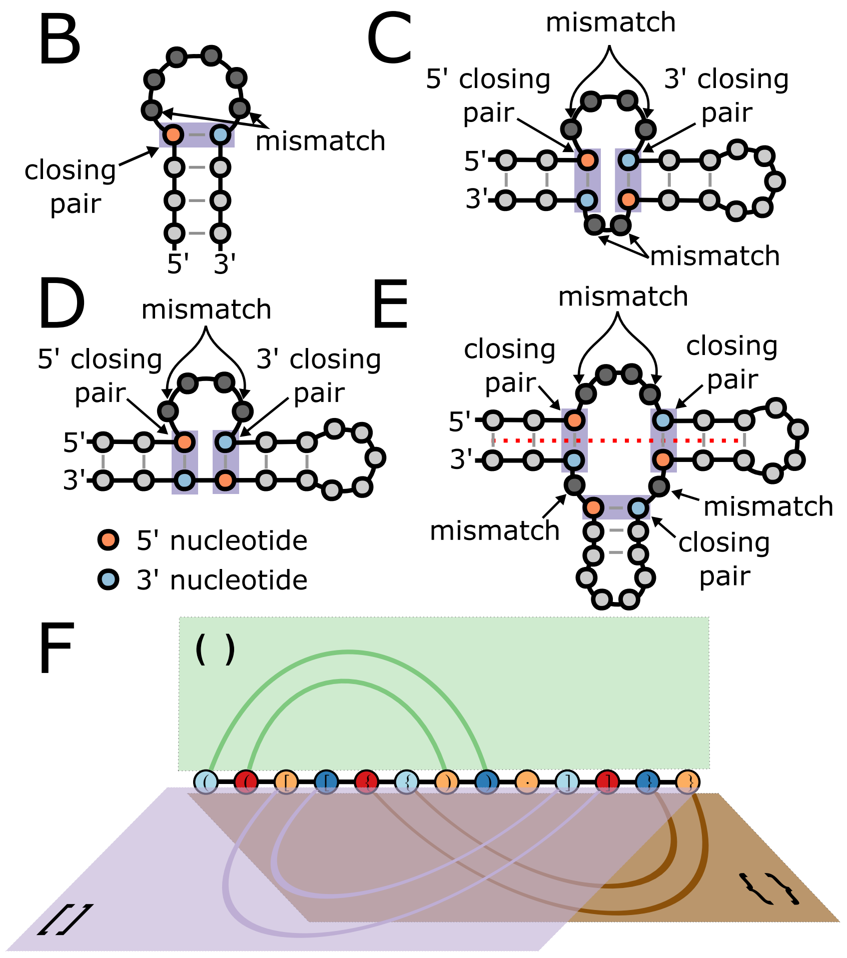


Figure 3.1: RNA structure types. B shows a hairpin loop, C shows a internal loop, D shows a bulge loop, and E shows a multi-stem loop. F is the page hierarchy of the pseudoknots. This figure was published in [10].

structure can be represented by non-crossing pairing arcs in a half-plane with the boundary of RNA strand; but a pseudoknotted structure needs several distinct half-planes to present the secondary structure (Fig. 2.3.F). A structure with k planar can be represented as the dot-bracket format of k types of brackets; this k is called page number.

Finding the minimum page number of pseudoknots is challenging, as no existing work solved this in polynomial time. Therefore, pseudoknots cause the gaps between the base-pairings and the structural features of the RNA sequence: the base-pairings can be parsed to a unique structure if it is pseudoknot-free, but introducing pseudoknots would make this task non-deterministic and complicated.

3.4 Maximizing first-page pair numbers

We are interested in a sustainable algorithm to parse the structural information from a set of base-pairs with pseudoknots. Specifically, since most structure prediction algorithms are focused on pseudoknot-free structures, it is important for bpRNA providing a pseudoknot-free structure, i.e., a set of first-page base-pairs, as the first step.

We design and develop a dynamic programming algorithm that can guarantee to find the PK-free set of the maximum number of base-pairs. The algorithm MaxFirstPagePairs (Algorithm 1) uses a bottom-up CKY-approach [27] to identify the maximum set of base pairs to produce a pseudoknot-free structure.

We also use the backtracking algorithm to get the dot-bracket format for the

Algorithm 1 Identify the maximum set of base pairs to produce a pseudoknot-free structure

```
function MaxFirstPagePairs($S, L$)
    ▷ input is a set of base pairs $S$ for an RNA of length $L$
    ▷ matrix score stores the number of pairs on each span $[i, j]$
    $score \leftarrow$ 2D array
    ▷ backptr for the corresponding $[i, j]$ span
    $backptr \leftarrow$ 2D array
    ▷ initialize the 0-length/1-length spans
    for $i \in 0 \cdots L$ do
        $score[i][i] \leftarrow 0$
        $score[i][i+1] \leftarrow 0$
    end for
    ▷ bottom-up CKY
    for $span \in 2 \cdots L$ do
        for $i \in 0 \cdots L - span$ do
            $j \leftarrow i + span$
            ▷ pair (i, j-1)
            if $(i, j-1) \in S$ then
                $score[i][j] \leftarrow 1 + score[i+1][j-1]$
                $backptr[i][j] \leftarrow (PAIR, -1)$
            else
                $score[i][j] \leftarrow 0$
            end if
        end for
    end for
    ▷ loop over all split points
    for $k \int i + 1 \cdots j - 1$ do
        if $score[i][k] + score[k][j] > score[i][j]$ then
            $score[i][j] < -score[i][k] + score[k][j]$
            $backptr[i][j] < -(SPLIT, k)$
        end if
    end for
    return backtrack$(0, L)$
end function
```

first-page structure (see Algorithm 2).

Algorithm 2 Recursively track back MaxFirstPagePairs to get structures

```
function BACKTRACK($(i, j)$)
    if $score[i][j] = 0$ then
        return "." $\times (j - i)$
    else if $backptr[i][j][0] = PAIR$ then
        return "(" $+ backtrack(i + 1, j - 1) +$ ")"
    else if $backptr[i][j][0] = SPLIT$ then
        $k = backptr[i][j][1]$
        return $backtrack(i, k) + backtrack(k, j)$
    end if
end function
```

3.4.1 Proof of the Algorithm

Here we prove that the algorithm is always able to find the maximum number of the first-page pairs. We instead prove: **for any segment $r_i ... r_j$ with a structure that contains m brackets, the state (defined as the maximum number of first-page pairs in a segment (i, j)) $s[i, j] \geq m$.**

We formulate the transition system as follows. The core part of a transition system is the states and the transitions, representing the max first-page pair number of a segment and the transition of it to other states. The dynamic programming algorithm processes the transition system and gives values for all states.

$$\begin{aligned}
\texttt{state}: & s[i,j], 0 \le i < j < n \\
\texttt{axiom}: & s[i,i+1], 0 \le i < n-1 \\
\texttt{goal}: & s[0,n-1] \\
\texttt{initialization}: & s[i,i+1] = 0, 0 \le i < n-1 \\
\texttt{transition}: &
\end{aligned}$$

$$s[i,j] = max \begin{cases} s[i,j-1](A) \\ s[i+1,j](B) \\ 1 + s[i+1,j-1], \ \texttt{if} \ (i,j) \in S(C) \\ s[i,k] + s[k+1,j], k \in [i+1,j-1](D) \end{cases}$$

We further define the transition graph. A transition graph is a collection of states and state transitions and the states are connected by transitions, just like nodes and edges in the graph.

Now we consider a sequence segment $r_i...r_j$ with a valid first-page structure, there is a unique mapping from this sequence-structure (segment-substructure) pair to a transition graph as a tree structure. The uniqueness requires removing (A) rules in the transition system thus the transition could be determined by the pairing status of x_i each time; removing it would not cause the proof to be much different, but adds simplicity.

The transition graph mapping is defined here:

- if i unpaired, then $s[i,j] \longleftarrow s[i+1,j](B)$;

- if (i, j) paired, then $s[i, j] \longleftarrow s[i + 1, j - 1] + 1(C)$;
- otherwise, if (i, k) paired where $k \neq j$, then $s[i, j] \longleftarrow s[i, k] + s[k + 1, j](D)$.

From the unique mapping, we can see that any valid first-page structure of a segment could map to a state transition graph. Assuming a structure contains m brackets, and only rule (C) transits the bracket and increments the value of the state, thus there is m number of (C) rules being used in the transition graph. This would lead to $s[i, j] \geq m$ since the transition would guarantee the $s[i, j]$ value has been incremented m times.

Therefore, this algorithm would guarantee to achieve the maximum number of first-page pairs.

3.5 Minimizing page numbers by approximation

We are looking for minimizing the page numbers of the RNA structures as much as possible, despite this problem is challenging and not yet solved by any polynomial algorithms. For example, RNA STRAND v2.0 had RNA sequences with high page number as high as 30, presenting a very complex analyzed structure; novel methods are needed to further optimize these structures, and then convert these structures to multi-bracket dot-bracket representation for bpRNA.

Intuitively, if we have an algorithm A parsing the base-pair set S and get a set of first-page basepairs S_{first}, then we can perform a recursive algorithm:

- Step 1: run algorithm A to get $S_{first} = A(S)$, mark base-pairs in S_{first} as page number 1;

- Step 2: take the rest of the base-pairs $S_{rest} = S - S_{first}$;
- Step 3: run algorithm A on S_{first} to get $S_{second} = A(S_{first})$, mark the BPs as page number 2, then go back to step 2;
- run Step 2-3 until no base-pairs are left; return the current parsed structure.

We modify Algorithm 1 to generate multiple first-page parsing algorithms. The left-to-right and right-to-left order are used instead of bottom-up order for dynamic programming, which would lead to different results since the search order affects the tie-breaker. We also do the reverse direction of pointers in bottom-up order: we compute from the right side instead of the left side for generating a different set of first-page base-pairs from tie-breakers.

With different versions of Algorithm 1, we take the steps described above on each of the sequences. Since each version might generate a different page number, we take the minimum page number found.

In this way, we reduce the page number of many RNA structures from the available resources and represent them in the bpRNA-1m dataset. We are able to represent all structures with a page number less than or equal to 7 (reduced from 30 of RNA STRAND v2.0), and 99.46% of the structures with a page number of 2 or less. For all 1,497 structures where bpRNA differs from RNA STRAND v2.0, we produce a lower page number, and thus a simpler dot-bracket sequence, see Fig 3.2.

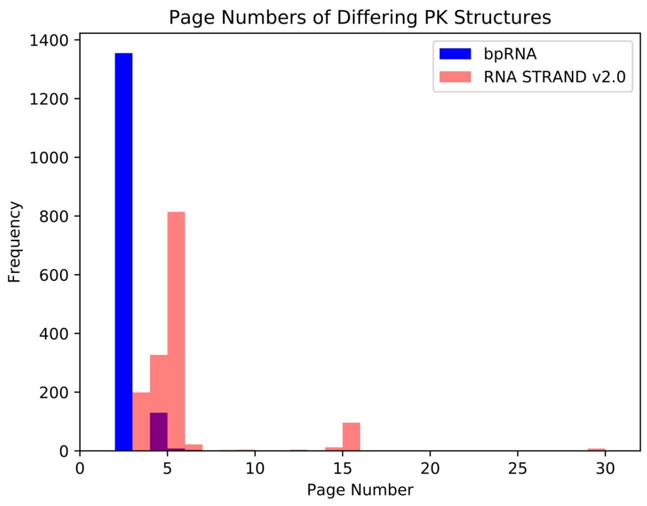


Figure 3.2: Our proposed algorithm reduces the page number of 1,497 sequences and simplifies their structure representation. This figure was published in [10].

Chapter 4: LinearFold: Linear-Time Prediction of Secondary Structures

4.1 Abstract

Predicting the secondary structure of an RNA sequence with speed and accuracy is useful in many applications such as drug design. The state-of-the-art predictors have a fundamental limitation: they have a run time that scales cubically with the length of the input sequence, which is slow for longer RNAs and limits the use of secondary structure prediction in genome-wide applications. To address this bottleneck, the first linear-time algorithm is designed for this problem. which can be used with both thermodynamic and machine-learned scoring functions. Our algorithm, like previous work, is based on dynamic programming (DP), but with two crucial differences: (a) LinearFold incrementally processes the sequence in a left-to-right rather than in a bottom-up fashion, and (b) because of this incremental processing, the beam search pruning is employed to ensure linear run time in practice (with the cost of exact search). Even though our search is approximate, surprisingly, it results in even higher overall accuracy on a diverse database of sequences with known structures. More interestingly, it leads to significantly more accurate predictions on the longest sequence families in that database (16S and 23S Ribosomal RNAs), as well as improved accuracies for long-range base pairs

(500+ nucleotides apart).

4.2 Overview of LinearFold

As described in the previous chapters, RNA is involved in numerous cellular processes, and its structure reveals the functions they have, especially for noncoding RNAs (ncRNAs) which have intrinsic functions without being translated to proteins [15]. However, on the one hand, predicting the (pseudoknot-free) secondary structure of RNA sequences remains challenging and costs at least cubic time given a minimum free energy model [57]; on the other hand, RNA sequences with long length ($>$ 1,000nt) widely exist in non-coding RNAs, showing a need for faster prediction algorithms.

Therefore, we present LinearFold, a linear-time global RNA secondary structure prediction algorithm. while the classical cubic-time dynamic programming if bottom-up, our algorithm firstly design a transition system to solve the optimal structure in a left-to-right fashion; we then use the beam search to keep top b highest-scoring states at each position, making the prediction algorithm linear with the approximation. Practically, with $b = 100$, our approach leads to a slightly better prediction accuracy than the baselines, by using either thermodynamic model or machine-learned models.

4.3 Dynamic Programming with Minimum Free Energy Models

As mentioned in chapter 2.2, applied minimum free energy approaches are based on energy decomposition. The core part of the decomposition is that the decomposed energy function $f_E^*(r, S'_r)$ is monotonic in the transition of every rule in the folding grammar, i.e., the best of the left-hand structure depends on the best of the right-hand substructures only. Thus, dynamic programming can be applied to reach the structure with the minimum free energy: $S_r^* = \text{argmin}_{S'_r} f_E^*(r, S'_r)$.

From the previous prediction systems, two types of model weights, i.e., free energy parameters, on top of this dynamic programming algorithm remain popular:

1. free energy parameters from thermodynamics, which is hand-engineered to simulate the thermodynamic free energy. We use Vienna RNAfold system with Turner 2004 Free Energy Parameters [36] as the baseline.

2. free energy parameters from machine learning, which is based on the MFE models and the same parameter set, but learning weights from data directly. We use CONTRAfold system [14] with its pretrained model as the baseline.

4.4 Incremental Linear-Time Dynamic Programming

The basic idea of linear-time prediction is to predict incrementally from left to right, labeling each nucleotide as unpaired “.”, opening “(”, or closing “)”. We require this dot-bracket string to be well-balanced as we only consider pseudoknot-free structures.

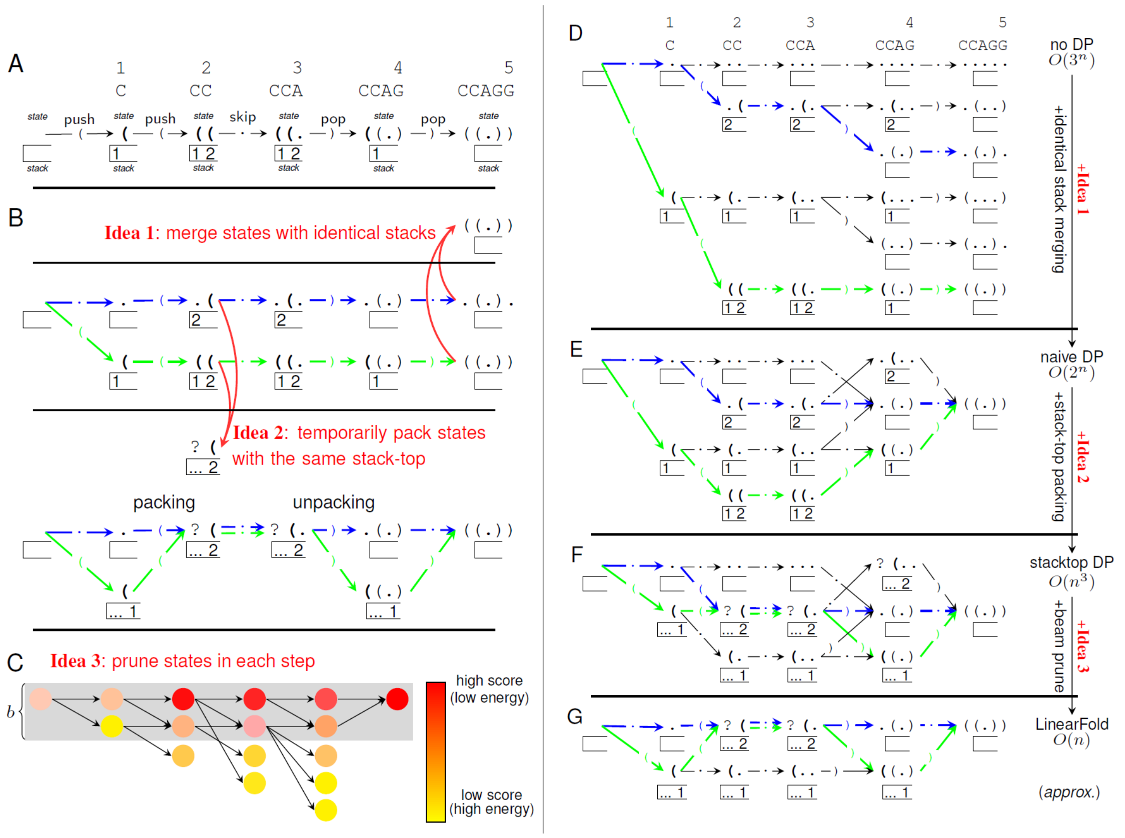


Figure 4.1: Illustration of the LinearFold algorithm, using a short sequence CCAGG and the simple Nussinov model (maximizing the number of base pairs). **A**: the optimal path, showing states (predicted prefix structures), actions (push "(", skip ".", and pop ")"), and stacks (unpaired open brackets which are shown in bold in states). **B**: two example paths (the optimal one in green and a suboptimal one in blue) and two essential ideas of left-to-right dynamic programming: merging equivalent states with identical stacks (Idea 1) and packing temporarily equivalent states sharing the same stack top, and corresponding unpacking upon pop (Idea 2). **C**: illustration of beam search, which keeps top b states (those in the shaded region) per step (Idea 3). **D**: the whole search space of the naive algorithm ($O(3^n)$ time). **E**: improving to $O(2^n)$ time with Idea 1. **F**: further improving to $O(n^3)$ time with Idea 2. **G**: further improving to $O(n)$ time (but with approximate search) with Idea 3. In **B, F**, and **G**, each blue/green arrow pair is actually a single arrow, denoting two paths temporarily packed as one; we draw paired arrows to highlight that two states .(and ((are performing skip action together. This figure was published in [12].

Following the CONTRAfold annotation, given an input RNA sequence $\mathbf{x} = x_0 x_1 \ldots x_{n-1}$ where $x_i \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$, our algorithm aims to find the best structure $\mathbf{y} = y_0 y_1 \ldots y_{n-1}$ where $y_i \in \{$“.”, “(”, “)”$\}$ with minimum free energy (or minimum model cost):

$$f(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} c(\mathbf{x}, \mathbf{y}; \mathbf{w}). \tag{4.1}$$

Here $\mathcal{Y}(\mathbf{x})$ is the set of all possible structures, i.e., $\{\mathbf{y} \mid \mathbf{y}$ has balanced parentheses$\}$, c is the cost function (i.e., free energy function), and $\mathbf{w}$ is the model (and parameters).

4.4.1 Naive exhaustive incremental prediction: $O(3^n)$ time

By exhaustively predicting y from left-to-right, we traverse all the possible structures in $\mathcal{Y}(\mathbf{x})$, and pick the one with the minimum free energy or model cost. We formalize each state at step j ($j \in \{0, \ldots, n\}$) to be a triple, $\mathrm{s} = \langle \sigma | i,\ j \rangle : \mathbf{y}$, where $\sigma | i$ is a stack consisting of unmatched openings so far where i is the top of the stack, meaning x_i is the last unmatched opening nucleotide. $\mathbf{y}$ is the corresponding dot-bracket (sub)sequence up to x_{j-1}. For each state, it can transition to a subsequent state, taking one of the three actions: push , which labels the current nucleotide x_j as a left bracket “**(**”, putting it on top of the stack, skip , which labels x_j as a dot “**.**”, leaving the stack unchanged, and pop , which labels x_j as a right bracket “**)**”, if it matches x_i and popping i from the stack. See Fig. 4.2 (a) for the deductive system. This algorithm takes $O(3^n)$ time to exhaustively traverse all possible structures (see Figure 4.1 D).

4.4.2 Dynamic Programming via Full Stack Merging: $O(2^n)$ time

Now we apply dynamic programming on top of this exhaustive method to exploit shared computations. Consider a simple case that two states can be merged: if there are two states in the same step j, $\langle \sigma,\ j\rangle : \mathbf{y}$ and $\langle \sigma,\ j\rangle : \mathbf{y}'$, sharing the exact same stack σ but with different dot-bracket strings $\mathbf{y}$ and $\mathbf{y}'$, we say that these two states are "equivalent" and we can merge them (and only keep the better scoring between $\mathbf{y}$ and $\mathbf{y}'$). Fig. 4.1 E illustrates this merging. Although we merge to reduce the number of states, it is still exponential time, since there could be exponentially many different stacks in each step. This algorithm takes $O(2^n)$ time.

4.4.3 Dynamic Programming via Graph Structured Stacks: $O(n^3)$ time

To avoid considering exponentially many states, we further merge states with different stacks. Consider two states in the same step j, $\langle \sigma_0|i, j\rangle$ and $\langle \sigma_1|i, j\rangle$, which share the last unpaired opening i (i.e., stack top). We call these states "temporarily equivalent", since they can be treated as exactly the same until the unpaired opening x_i is closed (and thus popped from the stack). In other words we can represent both stacks $\sigma_0|i$ and $\sigma_1|i$ as $...i$ where ... denotes part of the history that we do not care at this moment. This factorization of stacks is called "Graph-Structured Stacks" (GSS) by Tomita [51]. After merging, we define the new state to be $\langle i, j\rangle$ and therefore we maintain $O(n^2)$ states. For each state $\langle i, j\rangle$, the pop action can take worst-case $O(n)$ time because $\langle i, j \rangle$ can combine with every $\langle k, i \rangle$ from step i. Thus the overall time complexity is $O(n^3)$. See Fig. 4.1 F for an example of the merging process and Fig. 4.2 for the deductive system.

4.4.4 Dynamic Programming via Beam Search: $O(n)$ time

In practice, the exact search algorithm still runs in $O(n^3)$ time. But this left-to-right $O(n^3)$ search is easily "linearizable" unlike the traditional bottom-up $O(n^3)$ search used by all existing systems for RNA structure prediction. We further employ beam search pruning [24] to reduce the complexity to linear time. Generally, we only keep the b top-scoring states $\langle i, j \rangle$ for each step. This way all the lower-scoring states are pruned, and if a structure survives to the end, it must have been one of the top b states in every step. This pruning also means that in a pop action, a state (i, j) can combine with at most b states (k, i) from step i. Thus the overall time complexity is $O(nb^2)$. However, instead of generating b^2 new states from a pop action, we use **cube pruning** [23] to generate the best b states, which would take $O(b \log b)$ time. Thus the overall running time over a length-n sequence is $O(nb \log b)$, see See Figure 4.1 G for beam search.

4.4.5 Dataset, Evaluation Metrics and Significance Testing

We choose the ArchiveII dataset [45], a diverse set of over 3,000 RNA sequences with known secondary structures. But since the current CONTRAfold machine-learned model (v2.02) is trained on the S-Processed dataset [3] we removed those sequences appeared in the S-Processed dataset. The resulting dataset we used contains 2,889 sequences over 9 families, with an average length of 222.2 nt. Due to the uncertainty of base-pair matches existing in comparative analysis, we consider a base pair to be correctly predicted if it is also slipped by one nucleotide on a strand, accordingly([45]). Generally, if a pair (i, j) is in the predicted structure, we claim it's correct if one of (i, j), $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$ is in the ground truth structure. We report both Sensitivity and PPV where

$$\text{Sensitivity } = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}, \text{PPV } = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

We use the paired two-tailed t-test to calculate the statistical significance, with the type I error rate, consistent with the previous methods [55].

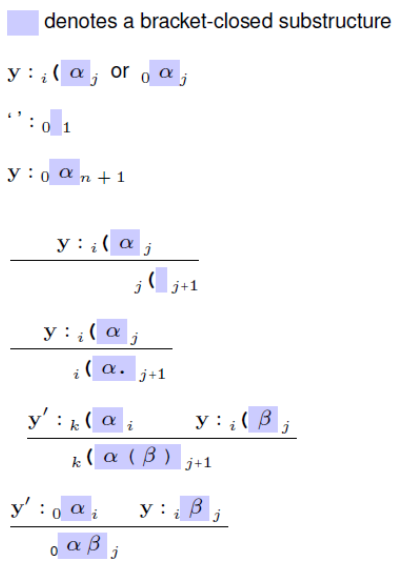
input	$x_1 \ldots x_n$
state	$\langle \mathbf{y},\ \sigma,\ j\rangle : s$
axiom	$\langle \text{`'},\ \text{`'},\ 1\rangle : 0$
goal	$\langle \mathbf{y},\ \text{`'},\ n+1\rangle : \_$
push	$\dfrac{\langle \mathbf{y},\ \sigma,\ j\rangle : s}{\langle \mathbf{y} \circ \text{`('},\ \sigma \mid j,\ j{+}1\rangle : 0}$
skip	$\dfrac{\langle \mathbf{y},\ \sigma,\ j\rangle : s}{\langle \mathbf{y} \circ \text{`.'},\ \sigma,\ j{+}1\rangle : s + \delta}$
pop	$\dfrac{\langle \mathbf{y},\ \sigma \mid i,\ j\rangle : s}{\langle \mathbf{y} \circ \text{`)'},\ \sigma,\ j{+}1\rangle : s + \xi_{ij}}$ $\quad (x_i, x_j)$ match

(a) exhaustive, $O(3^n)$

input	$x_1 \ldots x_n$
state	$\langle \sigma,\ j\rangle : \langle \mathbf{y}, s\rangle$
axiom	$\langle \text{`'},\ 1\rangle : \langle \text{`'}, 0\rangle$
goal	$\langle \text{`'},\ n+1\rangle : \langle \mathbf{y}, \_\rangle$
push	$\dfrac{\langle \sigma,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle \sigma \mid j,\ j{+}1\rangle : \langle \mathbf{y} \circ \text{`('}, 0\rangle}$
skip	$\dfrac{\langle \sigma,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle \sigma,\ j{+}1\rangle : \langle \mathbf{y} \circ \text{`.'}, s + \delta\rangle}$
pop	$\dfrac{\langle \sigma \mid i,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle \sigma,\ j{+}1\rangle : \langle \mathbf{y} \circ \text{`)'}, s + \xi_{ij}\rangle}$ $\quad (x_i, x_j)$ match

(b) Idea 1: identical-stack merge , $O(2^n)$

input	$x_1 \ldots x_n$
state	$\langle i,\ j\rangle : \langle \mathbf{y}, s\rangle$
axiom	$\langle 0,\ 1\rangle : \langle \text{`'}, 0\rangle$
goal	$\langle 0,\ n+1\rangle : \langle \mathbf{y}, \_\rangle$
push	$\dfrac{\langle i,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle j,\ j{+}1\rangle : \langle \text{`('}, 0\rangle}$
skip	$\dfrac{\langle i,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle i,\ j{+}1\rangle : \langle \mathbf{y} \circ \text{`.'}, s + \delta\rangle}$
pop	$\dfrac{\langle k,\ i\rangle : \langle \mathbf{y}', s'\rangle \quad \langle i,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle k,\ j{+}1\rangle : \langle \mathbf{y}' \circ \mathbf{y} \circ \text{`)'}, s' + s + \xi_{ij}\rangle}$ $\quad (x_i, x_j)$ match,
combine	$\dfrac{\langle 0,\ i\rangle : \langle \mathbf{y}', s'\rangle \quad \langle i,\ j\rangle : \langle \mathbf{y}, s\rangle}{\langle 0,\ j\rangle : \langle \mathbf{y}' \circ \mathbf{y}, s' + s\rangle}$



(c) Idea 2: stack-top packing, $O(n^3)$

Figure 4.2: The actual deductive system implemented in LinearFold program. Here δ^E, δ^M, ξ^M_{ij}, ξ^E_{ij}, ξ^H_{ij}, ξ^S_{ij}, and ω^S_{ijlk} are the various energy or scoring parameters (E stands for external loop, M for multiloop, S for single loop, and H for hairpin loop). Our LinearFold algorithm can linearize any dynamic programming-based pseudoknot-free RNA secondary structure prediction algorithm. The next (i, j) returns the next position after x_j that can pair with x_i; this is the "jumping" trick used in CONTRAfold and ViennaRNA to speedup from the theoretical runtime of $O(n^3)$ to the empirical $O(n^2)$ to $O(n^3)$. Our final two rules also use this jumping trick in the righthand side loop. The only cubic-time rule is reduce (intermediate step in multiloop), again inspired by CONTRAfold source code. This figure was published in [12].

Chapter 5: DeepSloop: A Recurrent Neural Network Learns Complex Rules to Detect Stem-Loop-Forming RNA Sequences

5.1 Abstract

The analysis of RNA structure is an important problem because RNA plays essential roles throughout all living things. Increasingly, RNA plays pivotal roles in biotechnology including RNA silencing, guide RNAs for CRISPR, ligand-binding aptamers, and drug design. While several studies have used deep learning models for analyzing RNA structure, these approaches rely on the outputs of traditional dynamic programming algorithms based on known thermodynamic energy parameters during training. The challenge remains to train neural networks to independently learn thermodynamic principles, so that potentially new biology can be uncovered. DeepSloop is a deep recurrent network capable of distinguishing hairpin-forming sequences from non-hairpin-forming, and that is trained purely on class-labeled nucleotide sequences. DeepSloop can accurately classify hairpin stem-loops, and thus learned a distinct internal representation of structure that includes destabilizing energies of bulges and internal loops.

5.2 Overview of DeepSloop

The secondary structures of RNA molecules reveal information about their functions [6, 52]; therefore, the prediction of how RNA sequences fold into structures is an important area of computational biology.

Modern RNA secondary structure prediction approaches have found great success in algorithms based on thermodynamic principles, especially the minimum free energy (MFE) models [37, 4, 38], which decompose free energies into individual loops to evaluate their stability in terms of thermodynamic energy parameters. Based on this MFE model, several approaches have been proposed for predicting RNA structures, mostly with the conventional dynamic programming algorithm [33, 25, 56].

More recently, deep recurrent neural network models have made remarkable progress on sequence modeling, including the analysis of sequential speech signals [19], video frames [47], and natural language texts [13]. These deep learning approaches have also been applied to study RNA secondary structure [54, 31, 40]. While these approaches have demonstrated that deep neural networks are capable of learning complex secondary structural patterns from sequence information, several challenges remain in the application of these models to RNA structure. An important remaining challenge is to train a neural network without directly using the MFE calculations, or secondary structure predictions from dynamic programming algorithms. The goal of this work is to train neural networks to independently learn their own concept of energy, without specific input from calculations using existing thermodynamic models.

In our recent work, we created a large, richly annotated meta-database of RNA secondary structures, bpRNA-1m, derived from both comparative sequence analysis and biophysical experiments [10]. For example, this database contains 708,144 hairpin loops. As a fundamental RNA secondary structure type, hairpins, also known as stem-loops, consist of a paired helical region that can contain bulges and internal loops, with a hairpin loop on one end. Stem-loops can direct RNA folding, protect structural stability for messenger RNA, provide recognition sites for RNA binding proteins, and serve as a substrate for enzymatic reactions [49]. Traditionally, hairpins are detected via thermodynamics secondary structure prediction, followed by pattern matching on the dot-bracket sequence. As a proof-of-principle that neural networks can independently learn thermodynamic rules, we present "DeepSloop", a deep neural network that is trained to recognize RNA sequences that form stem-loops (S-loops). We extracted hairpin sequences from bpRNA-1m [10], trained our deep neural network model to distinguish hairpins from randomized non-hairpin-forming sequences, and then analyzed what features our network models learned in order to properly recognize hairpin patterns. Additionally, we perform various destabilization tests and mutation experiments to show that our model could learn distinct thermodynamic rules without being given energy parameters.

5.3 Results

5.3.1 Data Preprocessing

In order to train our model, we needed to first build a collection of hairpin-forming and non-hairpin-forming sequences. We demonstrate our data preprocessing pipeline in Fig. 5.1A. As the first step, we source stem-loops from bpRNA-1m [10], resulting in 532,012 stem-loops, applied a length-filtering procedure, and removed highly similar sequences. This results in a filtered set consisting of 36,698 positive sequences.

Because there is no defined set of non-hairpin-forming sequences, we sought to create one with the same length distribution, nucleotide and dinucleotide composition as our positive set. We achieved this by performing dinucleotide shuffling on the positive sequences, resulting in a set of shuffled random sequences [1]. Because of the possibility of a shuffled sequence also forming a hairpin, we developed a statistical approach to further refine our data sets. First, we defined two metrics for scoring a putative hairpin. The balance score quantifies the degree of symmetry in base pairing in the dot-bracket representation, and is defined as the proportion of left brackets on the left side, plus the proportion of the right brackets on the right side. The base pair density (BPD) describes how densely concentrated base pairs are, and is defined as the number of base pairs per length of RNA. Because we only have structures for the positive set, we used RNAfold [33] to predict the secondary structure of each sequence for computing the balance score and BPD for all sequences. The statistical distribution of these two metrics on the stem-loop set

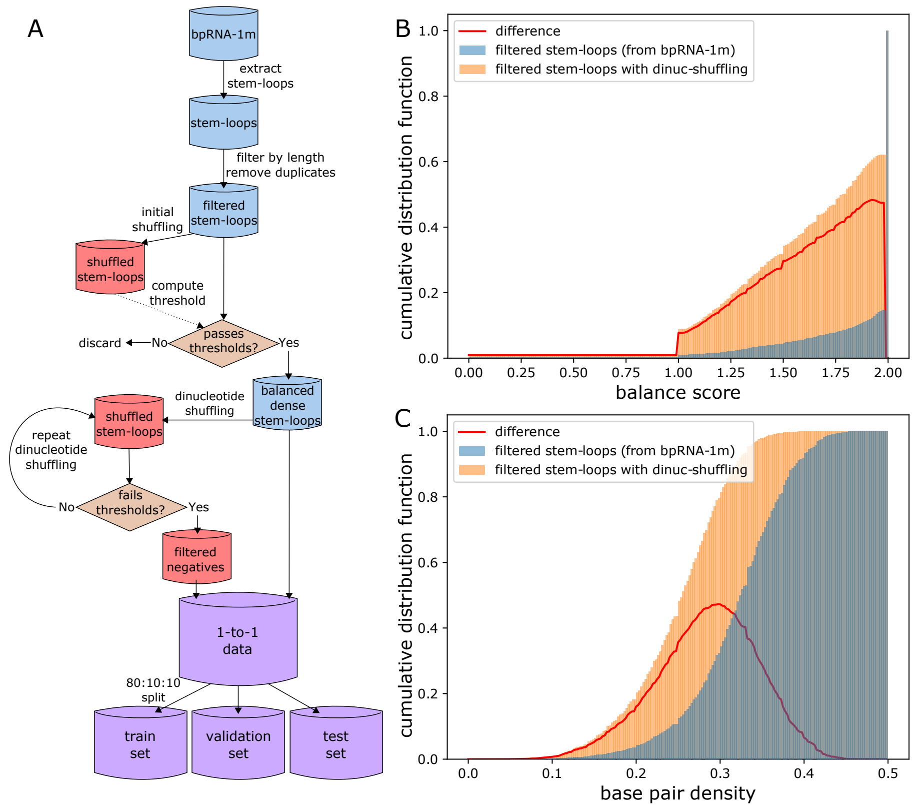


Figure 5.1: A. The data preprocessing pipeline, step by step from bpRNA-1m data set to the split training/validation/test data set for DeepSloop. B. The balance score CDF comparing our filtered stem-loops and dinucleotide shuffled counterparts. C. The base pair density CDF comparing our filtered stem-loops and dinucleotide shuffled counterparts.

and the shuffled set are shown in Fig. 5.1B-C. We defined our threshold for balance score as the maximum separation of the cumulative distribution function (CDF) for the stem-loops and shuffled sequences respectively. We defined the threshold for BPD to be halfway between its theoretical limits.

After the boundary was decided, we refined our stem-loops by selecting a subset of symmetric, dense stem-loops that pass both thresholds. We performed a 1-to-1 negative data generation process, resulting in a data set balanced in positive and negative examples. We required that each positive example should generate exactly one negative example that fails both thresholds, so an iterative process was used to repeat the dinucleotide shuffling until a sequence that fails both of our thresholds was found. We then split this data set into training, validation, and test sets, with the proportion of 80%:10%:10%.

5.3.2 Stem-Loop Prediction Model

Our design includes a bidirectional LSTM (BiLSTM) [43] centralized neural network model architecture, shown in Fig. 5.2A. We use a one-hot encoding for the nucleotide input, followed by two BiLSTM layers, stacked with a three-layer fully connected neural network. The hidden vector is mapped to a scalar, called the DeepSloop score, followed by a sigmoid function that outputs the probability of the sequence forming a stem-loop.

To train our model with RNA hairpins, we sourced stem-loop segments from the bpRNA-1m database [10]. Next, we applied the length filter, and removed

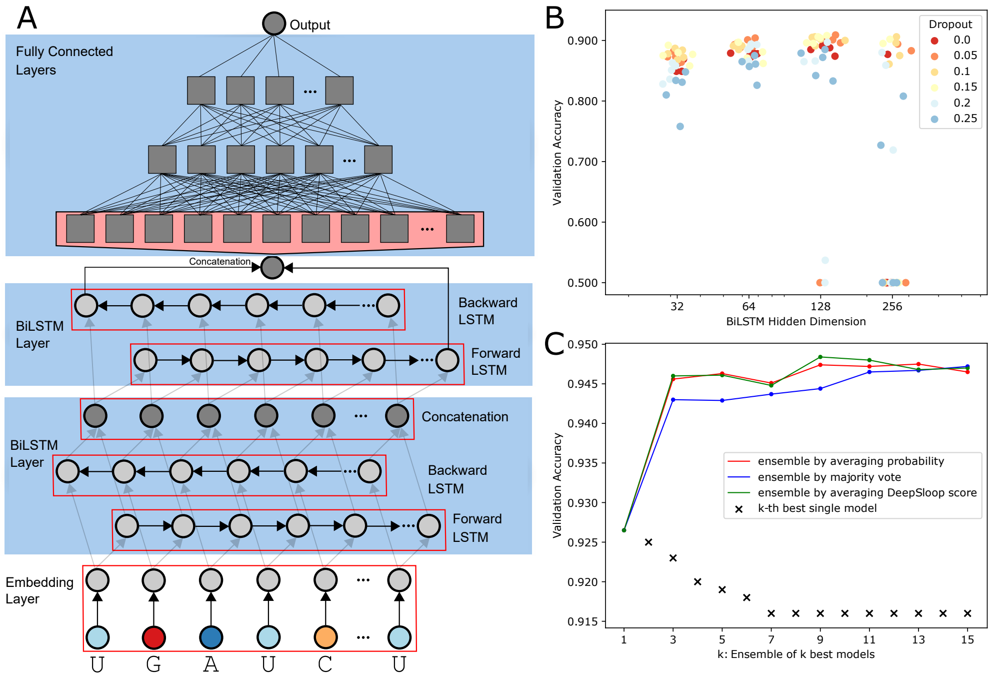


Figure 5.2: A. The model architecture of DeepSloop, which consists of 1) an embedding layer transforming the RNA sequence to a vector sequence by one-hot embedding on each nucleotide; 2) stacked bidirectional LSTM layers, mapping the sequential input to a sequence vector through stacked layers of LSTM cells; 3) a fully connected layer, mapping the sequence vector to a single output (DeepSloop score). The sigmoid transformation of DeepSloop score indicates the probability of the input sequence being a hairpin. B. Demonstration of (part of) our hyperparameter tuning, showing a relationship between validation accuracy and two factors: the hidden dimension of BiLSTM, and the input dropout rate during training. C. The validation accuracy through ensemble methods, regarding of 15 best models from 40 runs. Accuracy from three ways of ensembling top-k models are plotted in different colors, and the accuracy of k-th best single model are scattered.

highly similar sequences, resulting in 36,698 stem-loops. We further generated a negative set of sequences by dinucleotide shuffling, and applied structural filters to remove shuffled sequences that form spurious hairpins. We also randomly added nucleotides at both sides of copies of each sequence, preventing the first and last nucleotides from always being complementary. The whole data preprocessing procedure is illustrated in Fig. 5.1A. Therefore, our final training and testing data are strictly balanced, with the same distribution of nucleotides and dimers on positive and negative data set, ensuring our model would not learn simple statistics features to process the classification. We provided no RNA structural or thermodynamic information to the network, only the RNA sequence and stem-loop binary class labels during the training time, and we designed multiple ways of testing whether our trained models learned structural patterns of RNA sequences.

We performed hyperparameter tuning on our training and validation sets; we then performed augmented training by adding point mutations to the training data set. We extended the training set up to 11 times the original size for “pre-training”. We then used the pre-trained model to train on our original data set resulting in our final model. This pre-training approach improved our training time validation accuracy from 91.6% to 92.4%. We also performed ensemble testing to improve the performance of our system. From our 40 different trained models, we ensembled the top k-best models, resulting in improved testing time validation accuracy from 92.7% (single model) to 94.7% (15 models).

The hidden vector dimension of LSTM and input dropout rate are determined by our hyperparameter tuning on the validation data, shown in Fig. 5.2B. During hyperparameter tuning, we used grid search on each parameter set with 5 replicates for each. We selected the best parameter set of the averaged accuracy. As shown, DeepSloop prefers a BiLSTM hidden dimension of 128 and input dropout rate of 0.1.

5.3.3 Overall Performance

We show our ensemble method for testing has a higher prediction accuracy over single models, in Fig. 5.2C. We trained 40 replicates on an augmented data set using our parameter set, and ranked them by their validation accuracy. We selected the top-k models and ensembled them together in different ways, and plotted their validation accuracy. The ensemble of multiple models leads to a consistent improvement of validation accuracy (Fig. 5.2C).

We evaluated three different approaches with our test set: the best ensemble model (best, i.e., with the highest accuracy on validation set), the best single model, and the best single model without augmented training. The results show that our data augmentation contributed 0.47% improvement of the testing accuracy, while the ensemble methods contribute the other 2.77%. Our overall performance on the test set reaches 94.79% classification accuracy, benefiting from both data augmentation methods and the ensemble method.

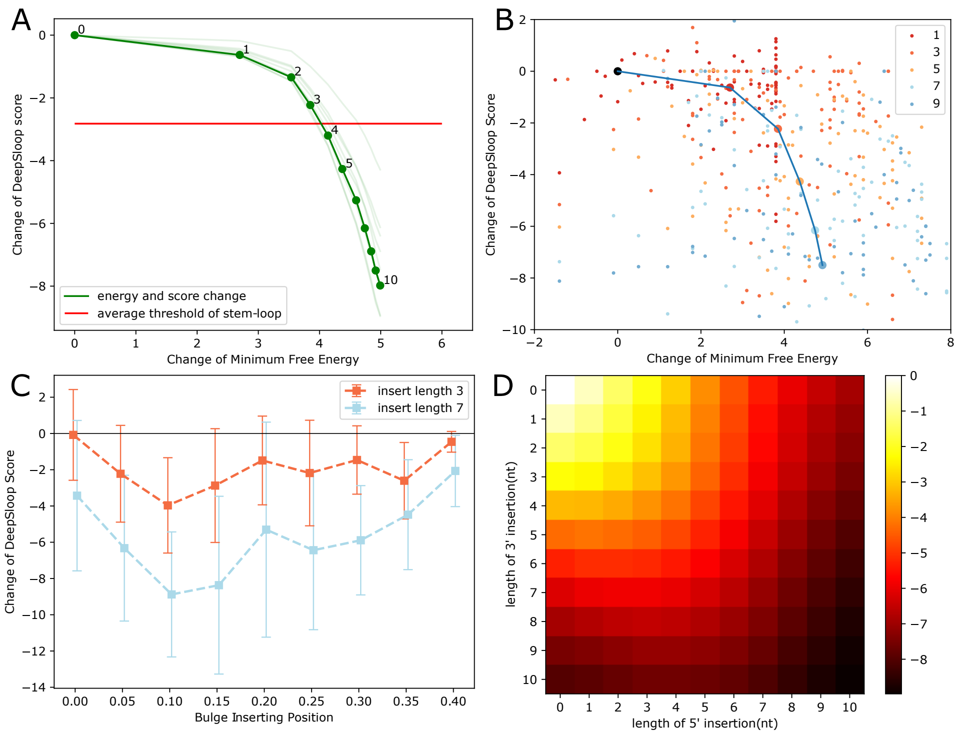


Figure 5.3: A. The DeepSloop score change against MFE change in the bulge destabilization test, when 0,1,...,10 'A' nucleotides are inserted. The red line indicates the average threshold of predicting as a hairpin or not by DeepSloop. Light curves show this change using single models instead of the ensembled one. B. Scatterplots of the DeepSloop score / MFE change of sequences, when 1,3,5,7,9 'A' nucleotides are inserted. C. The DeepSloop score change against the insertion position, when 3,7 'A' nucleotides are inserted. The squared nodes are the averages and the segments show the standard deviation. D. The DeepSloop score change in the internal loop destabilization test, when 0,1,...,10 'A' nucleotides are inserted in each direction.

Model	Test Accuracy
single model	91.55%
+data augmentation	92.02%
+ensemble method	94.79%

Table 5.1: Accuracy of the test set.

5.3.4 Energy Destabilization

In addition to the stem-loop prediction accuracy, we designed tests to assess whether our model learned patterns associated with hairpin formation. The results of the bulge and internal-loop destabilization test are shown in Fig. 5.3.

For the bulge destabilization test, we analyzed each stem-loop sequence from the validation set, then inserted k 'A' nucleotides into the middle of the longest helix on the left side of the sequence and observe the score change, with k from 1 to 10. Fig 5.3A shows the DeepSloop score change due to the bulge destabilization, compared with the MFE change (from Vienna RNAfold [33]), averaging from all positive sequences in the validation set. In addition, we visualized the specific values for 100 randomly sampled stem-loops in Fig 5.3B. Though our model only learns from the RNA sequences with minimal structural information, the DeepSloop score is able to represent the destabilization change due to bulges similar to the MFE score. The DeepSloop score decreases consecutively when more destabilizing nucleotides are inserted, and decreases less after there are already 10 nucleotides inserted. From the scatter plot, we observe that the DeepSloop score decrease is varied, but increases with the length of the inserted bulge.

We also investigated the impact of bulge destabilization against the bulge insertion position in the sequence, shown in Fig 5.3C. The curves show that the decrease of the DeepSloop score is lowest when the insertion position is close enough to the middle of the sequences, where our hairpin is located. An insertion of a length 3 would not change the output classification at most positions on average, while a longer insertion of length 7 would generally change the prediction.

We performed a similar experiment for internal-loop destabilization. We randomly selected a position within the longest helix, and insert two loops of length $l1$ and $l2$ consisting of 'A' nucleotides, and observe the DeepSloop score change. The heatmap of the DeepSloop score change shown in Fig 5.3D demonstrates a consistent decrease, and is also symmetric against two lengths, $l1$ and $l2$.

5.3.5 Point Mutation Analysis

We further performed several mutation tests on the DeepSloop model. First, we performed point mutation tests, where we sampled stem-loops from the validation set, applied a point mutation on each of the positions. For each position, we identified the most destabilizing point mutation based on the most negative DeepSloop score change. Fig. 5.4A visualizes the point mutation test on stem-loop sequences. Nucleotides in the long helices are unlikely to have a large DeepSloop score decrease due to the mutation, but for nucleotides near a bulge, internal loop, or open structure, the mutation is more likely to destabilize. The DeepSloop captures patterns of destabilization that are consistent with known energy functions.

We also designed an incremental selection process (ISP) using DeepSloop to

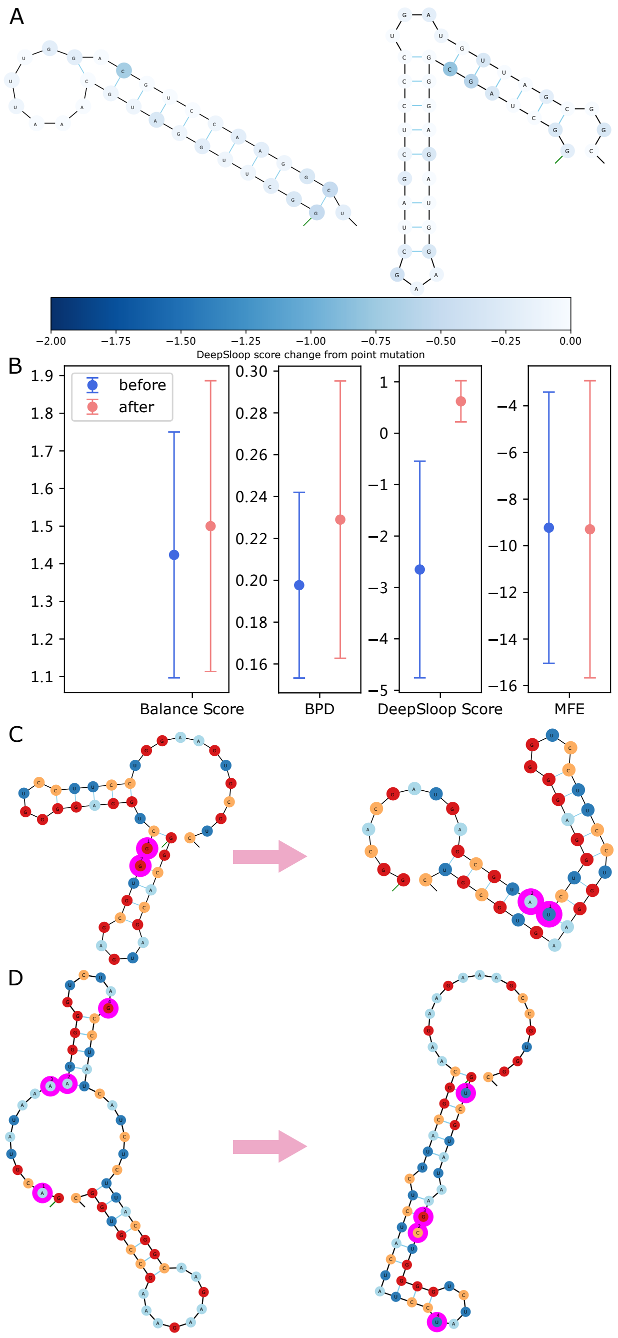


Figure 5.4: A. Point mutation test for two different RNA hairpin sequences. The color of each nucleotide indicates the DeepSloop score change from the worst point mutation of this position among three different mutations. The darker color corresponds to mutations with greater disruptive potential. B. Statistics comparing the balance score, base pair density, DeepSloop score, and MFE change due to an incremental selection process (ISP), compared the initial RNA structures. C,D. Examples of our DeepSloop-designed hairpin structures that result from ISP. The left shows the sequence/structure before ISP, and the right is after. All the mutated positions are highlighted with order numbers labeled.

mutate a random RNA sequence towards a hairpin. In the ISP, we selected an RNA sequence from the negative set; then we performed all the possible point mutations and selected the mutation with the highest increase in DeepSloop score. This process is like evolving the sequence; by repeating this process until convergence, we observe whether we could transform the non-stem-loop-forming sequences to stem-loop-forming. Fig. 5.4B shows the statistics of sequences before and after. On average, ISP increases the balance score, base pair density, DeepSloop score, and decreases of the MFE. The structure and mutation process of two examples are shown in Fig. 5.4C-D. The mutated nucleotides are highlighted to show a step-by-step procedure.

5.3.6 Detection of Hairpins within Long Noncoding RNAs

We assessed whether DeepSloop could detect hairpin structures within long noncoding RNAs (lncRNAs), despite the extra sequence information flanking the hairpin locations. We selected lncRNA sequences including HOTAIR (Domain1) [46], the polycistronic microRNA precursor MIR17HG [20] and bpRNA_CRW_12228 (16S rRNA Bacilli/DQ169500) [10]. For HOTAIR and bpRNA_CRW_12228, the structures are known from experiments and comparative sequence analysis, respectively, and the MIR17HG is predicted by RNAfold. We used a collection of fixed-size sliding windows over each sequence, computing the hairpin-forming potential (DeepSloop probability) of each position. We then plotted a smoothed curve of the probabilities with Savitzky-Golay filter [42] for each sequence, and

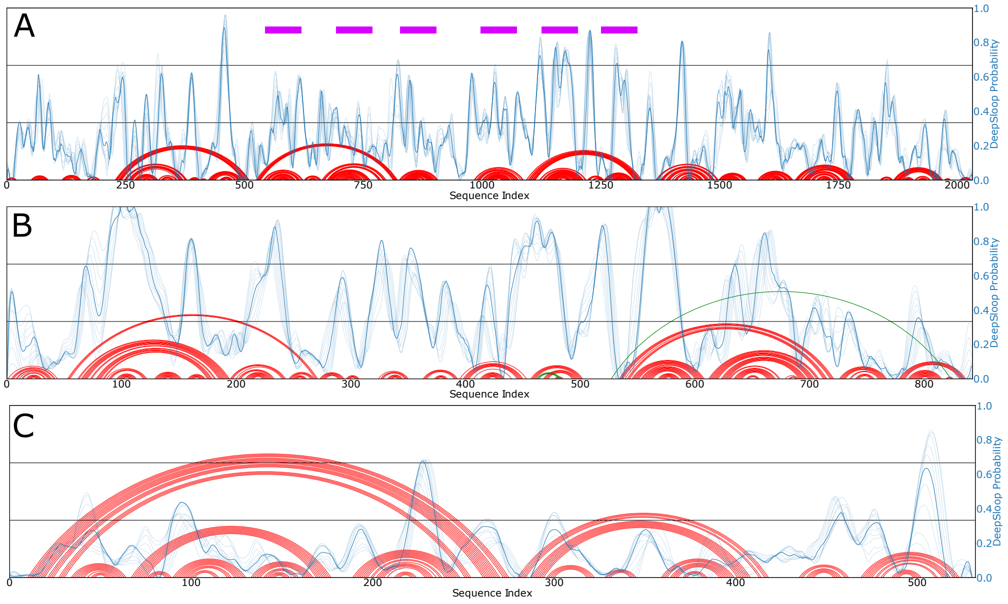


Figure 5.5: Examples of DeepSloop hairpin prediction of lncRNAs; A: MIR17HG, B: bpRNA_CRW_12228(Bacilli/DQ169500), C: HOTAIR_Domain1. The X-axis is the sequence index, and the Y-axis is the DeepSloop hairpin probability. We calculated the DeepSloop hairpin probability using a sliding window with size 51 over the whole sequence, smoothed by the Savitzky-Golay filter with width 35 and degree 5, and plotted in the opaque blue color. All other light blue curves are from window-size 41 to 61. The original secondary structure of the sequence is shown using arcplot, where each base-pair is plotted using a red arc in the figures; green-arcs show pseudoknots; purple boxes in the MIR17HG indicate the positions of microRNAs.

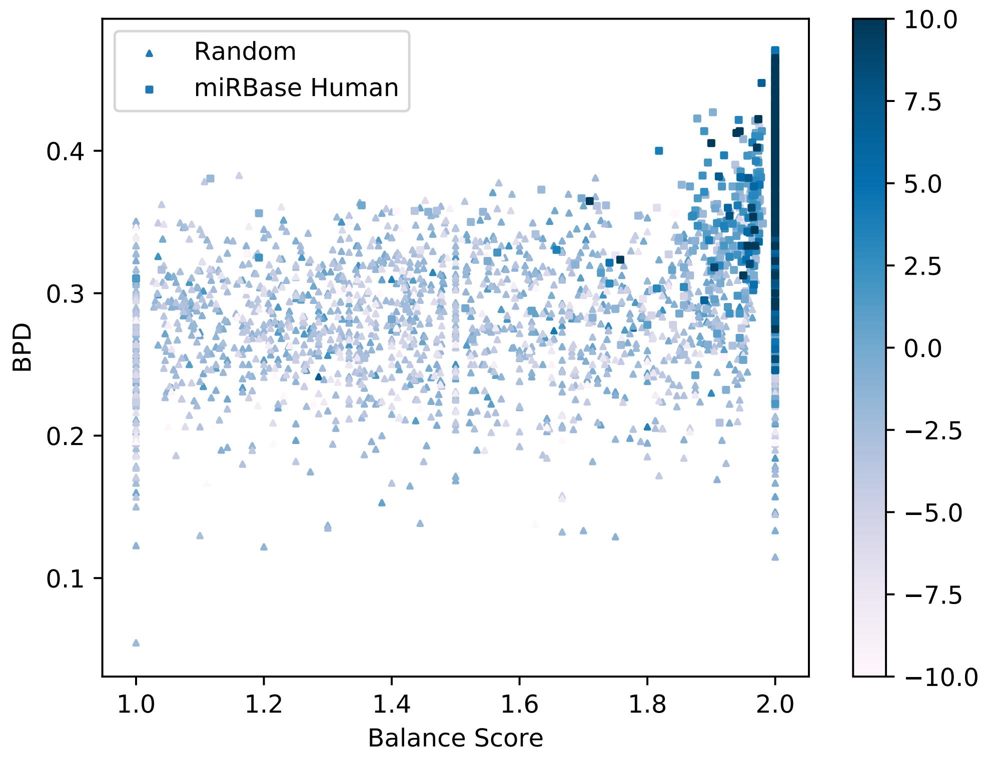


Figure 5.6: The scatter of the miRBase human test, comparing base pair density (Y axis) against balance score (X axis). The color of nodes represents the DeepSloop score. Triangles represent the sequences in the negative set (random), and squares represent the positives.

compared with its original secondary structure, shown in Fig. 5.5. We observed that most of the strongest predicted peaks correspond to hairpins in the reference structure.

5.3.7 miRBase human Test

We also evaluated the performance of DeepSloop on an external data set, using the miRBase human data [20] consisting of 1,917 hairpin precursor sequences. We generated an equal number of negative examples by mono-nucleotide shuffling. DeepSloop achieved 77.54% accuracy for classifying miR precursors as hairpins, with 68.96% sensitivity and 86.12% specificity. We also visualized the miRBase human set in Fig. 5.6, which shows a dense concentration of the positives on the top right corner with mostly high DeepSloop scores, and a sparse distribution of the negatives on the rest of the figure with mostly negative DeepSloop scores.

5.4 Discussion

In this study, we have shown that a BiLSTM deep neural network model can successfully predict RNA hairpin-forming potential. This result demonstrates that neural networks have the capacity to learn and encapsulate distinct thermodynamic rules of RNA structure formation, breaking away from the dependence on energy parameters.

DeepSloop learned rules that are very similar to known thermodynamic principles in some instances, and different rules in others. We conclude from the bulge and internal-loop destabilization tests that our model learned that the length of the inserted region is proportional to the degree of destabilization. In the case of bulge insertions, we observed that our model is not very sensitive to the bulges of length 1, but incurs a larger score decrease when the inserted length increases from 2 to

5. This is in contrast to traditional bulge destabilization costs computed by most RNA structure prediction algorithms [56, 38], which show the greatest increase in destabilization at the first nucleotide. In the case of internal loops, DeepSloop learned that symmetric internal loops, with both loops having the same length, have the lowest cost. Similarly to bulges, the destabilization cost of internal loops is less sensitive to the first inserted nucleotide than established thermodynamic parameters. Thus, while DeepSloop disfavors these unpaired regions, it is more tolerant of single-nucleotide bulges and internal loops, consistent with the fact that short bulges and internal loops are the most frequent [10].

DeepSloop also has strong potential to reveal and quantify the stability of different regions of RNA sequences without knowing their structures, which suggests that the network has an internal representation of the structure. From the mutation test, we demonstrated that the most disruptive positions in hairpin sequences correspond to base-paired nucleotides. In many examples, these positions correspond to base pairs that are adjacent to bulges or other loops, which can be pivotal for stability. Our model is also able to evolve a random non-hairpin-forming RNA sequence to a hairpin-forming sequence by performing a series of point mutations in a few steps. These experiments show that DeepSloop favors mutations that introduce more base pairs and a more balanced structure.

Future work could explore the application of deep neural networks on the analysis of RNA structure more generally. Because hairpins play a pivotal role in guiding some RNA structures, their prediction could be part of a multi-step process of structure prediction by intelligent systems. The current DeepSloop model is sensitive to the window size when applied to long sequences using a sliding-window approach. Future work is needed to detect hairpin forming sequences within the context of surrounding sequence. This problem might be overcome by improved design of the training data, such as with additional random flanking sequence padding added to training sequences. Alternatively, a model trained on a different task, such as site-labeling positions or windows as either hairpin- or non-hairpin-forming across the full-length RNA, may result in better hairpin detection performance for long sequences.

5.5 Methods

5.5.1 Data Preprocessing

The bpRNA-1m database contains detailed structural information including stem-loops [10]. We extracted 532,012 stem-loops, and applied a length filter, where we keep RNA segments with stem lengths from 20 nt to 150 nt, with hairpin loop lengths from 3 nt to 22 nt. This resulted in 72,586 stem-loop candidates. To remove similar sequences, we ran CD-HIT-EST [18], a tool for clustering and comparing nucleotide sequences, with 90% as the sequence identity threshold, and 70% as a coverage threshold.

5.5.2 Dinucleotide Shuffling and Boundary Detection

We generated a negative data set to have a nucleotide distribution similar to our positive set, coaxing our classifier to learn high-level structural information instead of simple features to separate them. We then performed the default folding settings from RNAfold[33] to determine the secondary structure for each sequence. Next, we defined two metrics based on the secondary structure of a sequence for filtering positive and negative stem-loops, the BPD, and the balance score. The BPD is defined as the number of base-pairs over the sequence length, ranging from $[0, 0.5]$.

$$\texttt{BPD}(\texttt{S}_\texttt{r}) = \frac{|\texttt{S}_\texttt{r}|}{|\texttt{r}|}$$

The theoretical range of the BPD score is from 0.0 for a completely unpaired RNA to 0.5 for an RNA that lacks unpaired nucleotides. Therefore, we set the BPD threshold halfway at 0.25, corresponding to half of the nucleotides being paired, as the maximal CDF difference proved too restrictive.

Balance score is computed from the dot-bracket sequence, and is defined as the proportion of left brackets on the left side, plus the proportion of the right brackets on the right side. The range of the balance score is $0 \cup [1, 2]$. The balance score is large when a structure is close to a stem-loop, which should have all the left brackets on the left side, and similarly for right brackets.

$$\texttt{BS}(\texttt{S}_\texttt{r}) = \frac{|\texttt{left brackets on left}|}{|\texttt{left brackets}|} +$$

$$\frac{|\texttt{right brackets on right}|}{|\texttt{right brackets}|}$$

We selected the balance score threshold boundary based on the balance score distribution of our positive stem-loop data set and negative data set. Our first step is to shuffle our initial positive data set, denoted as P_0. We performed dinucleotide shuffling, resulting in a data set noted as N_{ds}. We then computed the difference between the CDFs of balance score on these two data sets, P_0 and N_{ds}, using a bin size of 0.01, shown in Fig. 5.1. We selected the balance score that has the largest gap between these two CDF curves, which corresponds to the boundary, $T_{\texttt{BS}} = 1.86$.

5.5.3 Data set Generation, Augmentation and Split

After setting up two boundaries, we start generating our positive and negative data set based on P_0.

1. $P = \phi, N = \phi$
2. $\forall r \in P_0$, if $\texttt{BPD}(\texttt{S}_\texttt{r}) \geq \texttt{T}_\texttt{BPD} \wedge \texttt{BS}(\texttt{S}_\texttt{r}) \geq \texttt{T}_\texttt{BS}$, then process steps below; otherwise drop r.
   (a) $r' = \texttt{DinucShuffle}(\texttt{r})$
   (b) if $\texttt{BPD}(\texttt{S}'_\texttt{r}) < \texttt{T}_\texttt{BPD} \wedge \texttt{BS}(\texttt{S}'_\texttt{r}) < \texttt{T}_\texttt{BS}$, then $P = P \cup \{r\}, N = N \cup \{r'\}$, else go back to `DinucShuffle`; drop r if no r' passed thresholds after 50 shuffles.

3. return P and N of equal size.

After generating equal-size P and N, we split these into $D_{train}, D_{val}, D_{test}$, with 80%:10%:10% proportion.

To train our model, we used a binary label 1/0 for each of the RNA sequences, indicating whether it forms a stem-loop.

5.5.4 Model Architecture

Our model architecture is presented in Fig. 5.2A. We explored alternative model architectures as well. Instead of one-hot encoding, we tried learning an embedding layer during training, and observed minimal differences between them, thus one-hot is used. We also added a convolutional layer between the embedding layer and the LSTM layers; however, this resulted in a slower training time and no improved performance.

5.5.5 Hyperparameter Tuning

During the tuning and training step, we evaluated the validation accuracy on D_{val} after each epoch. Our training stops when the validation accuracy is not improved in the most recent 5 iterations, i.e., our early stopping patience is 5. We tuned our model architecture on D_{train}, using grid search on all possible sets of hyperparameters, with five replicates. We explored a set of hyperparameters including the number of BiLSTM layers, input dropout rate, batch size, BiLSTM hidden vector size, and whether using a starting convolution layer.

5.5.6 Training and Augmented Training

Besides training our model on the original training set D_{train}, we also applied augmented training to improve the performance of our models. We augmented our data by using point mutations. For each of the RNA sequences in D_{train}, we randomly generated 10 copies, where each copy has a point mutation; all the sequences generated are added to the data set. We use D_{train}^{aug} to represent our augmented training set, $|D_{train}^{aug}| = 11|D_{train}|$. To perform augmented training, we first trained from scratch on D_{train}^{aug}; after the model converged, or was halted by early stopping, we saved our model for the next step. We further trained this saved model on the original data set D_{train}, resulting in the output model. Experimental results show that by performing augmented training, our single model accuracy on the validation set improved from 91.6% to 92.4%. Additionally, to prevent Deep-Sloop from learning that the first and last nucleotides are always complementary, we performed an additional data augmentation step by adding a random prefix and suffix to each copy of length $0, 1, 2, 3$ nt.

5.5.7 Ensemble Testing

Our training resulted in 40 model replicates, each different due to the random initialization and the stochastic nature of learning. We combined the top-k models together, with k in $1, 3, ..., 15$, and ensemble them using different approaches. We computed the validation accuracy of the ensembled models, and we measured the performance on the testing set, using our best single model and the ensembled model compiled from 15 individual models.

5.5.8 Destabilization Tests

The destabilization tests and mutation tests are based on mapping the change of the RNA sequence to a DeepSloop score change. For an input RNA sequence r, we calculated its DeepSloop score $S_{DS}(r)$; then made a small change on r to r', and calculated $S_{DS}(r')$. We compute $\Delta S_{DS}(r, r') = S_{DS}(r') - S_{DS}(r)$ to assess whether the DeepSloop score change reflects this small perturbation.

In our bulge destabilization test, we first identified helical regions to insert a disruptive bulge. For every positive sequence r in the validation set, we first used RNAfold to label its secondary structure in dot-bracket format, for example "`...((((((...))))))...`"; then we find the longest helix segment on the left (longest consecutive left parentheses) and denote its middle position as i. Our next step is to insert multiple 'A' nucleotides at i to destabilize the structure. We insert k nucleotides with $k = 1, 2, 3, ..., 10$, making 10 destabilized sequences for each sequence r, and compute $\Delta S_{DS}(r, r')$.

For the internal loops, we performed a similar test to find the insertion position i. The only difference is that two loops of length k_1, k_2 from $1, 2, .., 10$ are generated, resulting in 100 destabilized sequences for each sequence r.

5.5.9 Point Mutation Analysis

We designed two different mutation experiments. The first examines whether some nucleotides in an RNA sequence are more unstable than others. For each position i in the sequence r, we mutate the nucleotide, resulting in three different sequences $r' \in \texttt{mutate}(\texttt{r}, \texttt{i})$. We define the worst $\Delta S_{DS}(r, r')$ from these mutations, for each position i. We then visualized the most negative score change of all the positions.

Our ISP experiment evolves a random RNA sequence to a stem-loop incrementally. We performed $3|r|$ point mutations and selected the highest $\Delta S_{DS}(r, r')$ to replace the original r with r' at each step, until the best DeepSloop score becomes positive, resulting in a hairpin-forming sequence.

5.5.10 Point Mutation Analysis

To further evaluate the performance of DeepSloop, we also evaluated performance on miRBase human data as an external data set. We first source 1,917 miRNA precursor sequences as positives. For each of the positive sequences, we generated a negative counterpart by mono-nucleotide random shuffling. Thus a data set with balanced positives and negatives is generated for testing.

5.5.11 Detection of Hairpins within Long Noncoding RNAs

We applied DeepSloop to lncRNAs by predicting hairpin-forming potential within a sliding window that is moved over the full length of transcripts greater than 200 nt. Using the window-size ranging from 41 to 61, for each window-size $2k + 1$, we loop over all indices i, taking segment from the original sequence with index $[i - k, i + k]$, with a padding of 'A' nucleotides at the beginning or end of the sequence. We tested the hairpin probability of each and plotted the score from the ensembled DeepSloop model. For smoothing, we used Savitzky-Golay filter [42], averaging nearby 35 data points with a polynomial degree of 5. We selected window-size of 51 as the center window-size, and also plotted curves with window-size $[41, 61]$ using light color. The original secondary structure of the sequences are presented as well for comparison.

Chapter 6: DeepStructure: A Neural Network Approach for RNA Secondary Structure Prediction from Scratch

6.1 Abstract

It is valuable to further investigate RNA secondary structure prediction using deep neural networks. While deep neural network models have made remarkable progress in many classification tasks on DNA and RNA sequences, predicting RNA secondary structure is still challenging due to the structural complexity of the output. DeepStructure is an end-to-end approach learning to predict secondary structures from RNA sequences from scratch, and no longer relies on the energy parameters. DeepStructure utilizes Bidirectional LSTM and Attention Networks as the core part of its neural network architecture and learns the RNA folding principles independently from the large-scale dataset bpRNA-1m. Experimental results show that DeepStructure reaches a competitive performance against traditional RNA secondary structure prediction approaches.

6.2 Overview of DeepStructure

Our goal is to explore the RNA secondary structure prediction technique using deep neural networks, without the involvement of free energy models. On the one hand, we proposed DeepSloop for prediction of stem-loops and model structural information, which shows the feasibility for neural network models to learn structural information purely from the RNA sequences, in the last chapter; on the other hand, recent researches show the success of deep neural networks modeling sequence-to-sequence or sequence-to-structure problems, including machine translation, syntax parsing, and speech recognition [32].

However, modeling RNA sequences is different from modeling text or speech signals, as the core challenge is that the nucleotide (A/C/G/U) sequences provide too little information at each position. From the previous studies, we found that 1) researchers have applied neural networks to model RNA / DNA / protein sequences for variety of tasks, but hard to solve problems beyond binary classification without extra information for training or testing [22]; 2) the current neural network models are still relied on the free energies [54] computed by MFE models, which is basically learning features from the decomposed energy model and simulate its calculation, instead of learning real structural information from RNA sequences.

We focus on an end-to-end neural network system that models RNA sequences and predicts RNA secondary structures directly, without the help of any energy parameters. In the previous DeepSloop work, we showed the potential that the neural network is able to learn complex structural patterns and predict stem-loops reliably. On the other hand, attention techniques are widely used in Seq2Seq models, such as Attention Layers between encoder and decoder[34], and Transformer Networks [53, 9].

Thus, we present DeepStructure, an end-to-end neural network approach for

RNA secondary structure prediction. Starting from the LSTM-Attention model, we trained our basic model from the (preprocessed version of) bpRNA-1m dataset; we next designed and performed base-pair-aware data augmentation, allowing the complex model to be trained on a larger-scale dataset with a significantly improved performance; at last, we tuned the Transformer architecture and used the LSTM-Transformer framework instead, in which we further boosted the prediction accuracy. Experimental results show that our model can reach competitive performance against traditional RNA secondary structure prediction techniques without hand-design features nor structure prediction algorithms.

6.3 Results

6.3.1 Model Architecture

We conclude our LSTM-attention model architecture in Fig. 6.1. Generally, for an input sequence, we first map each nucleotide into a vector; then we feed a sequence of vectors into the encoder layer, i.e., two stacked BiLSTM layers; next, we start decoding in a mono-directional LSTM by taking the output of the encoder layer; in each step of decoding, we utilize an attention layer to calculate the context vector between the current decoder hidden vector and the sequential encoder hidden vectors; the context vector is used by concatenated to the end of the input vector for each LSTM cell; after a sequential output, if processed from the decoder LSTM, we map the vectors to the dot-bracket labels and produce the output of the network.

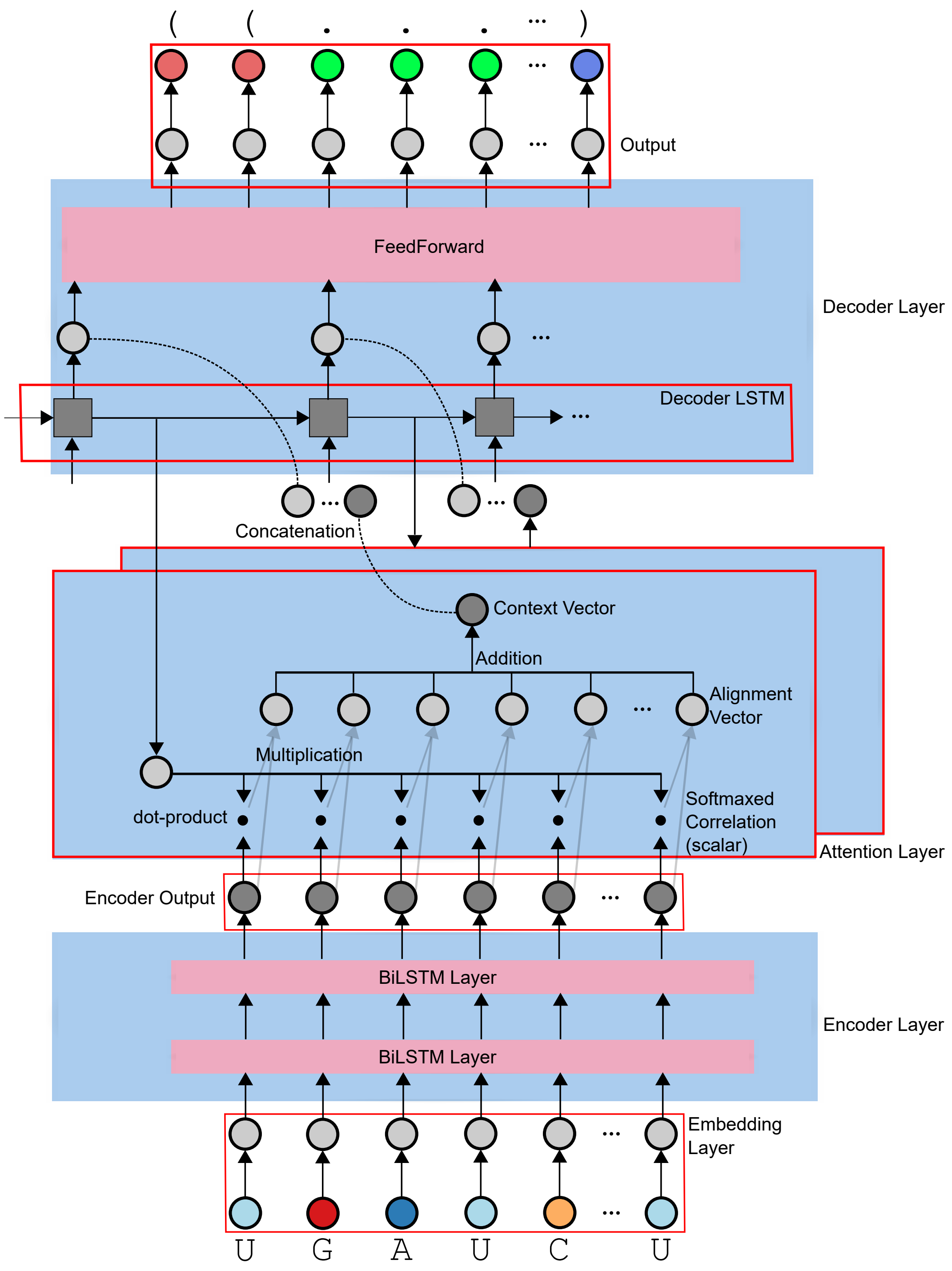


Figure 6.1: An overview of the LSTM-Attention Architecture of DeepStructure.

Moreover, we combined the LSTM encoder with the Transformer decoder and designed the LSTM-Transformer framework, shown in Fig. 6.2. In the LSTM-Transformer framework, The encoder LSTM outputs a sequence of vectors which is used for the input of the first decoder layer and the encoder-decoder attention for every decoder layers; for the stacked Transformer decoder layer, each layer contains a self-attention component, an encoder-decoder attention component, and a feed-forward component. The attention component works similarly to our previous shown attention technique, except a sequence of the attention actions is performed thus the output is kept as a sequence of vectors. 6 stacked decoder layer is used.

The LSTM-Transformer framework can be regarded as an advanced version of the LSTM-Attention framework since it utilizes a deeper structure of attention technique, but it also requires a larger dataset to train since the parameter size is a lot larger.

6.3.2 Base-Pair Prediction Quality

We evaluate the performance of our models on the test set, and show the base pair precision, recall, and F-score in Tab. 6.1. As shown in the table, we reach 42.9 F-score using the LSTM-Transformer framework with base-pair-aware data augmentation.

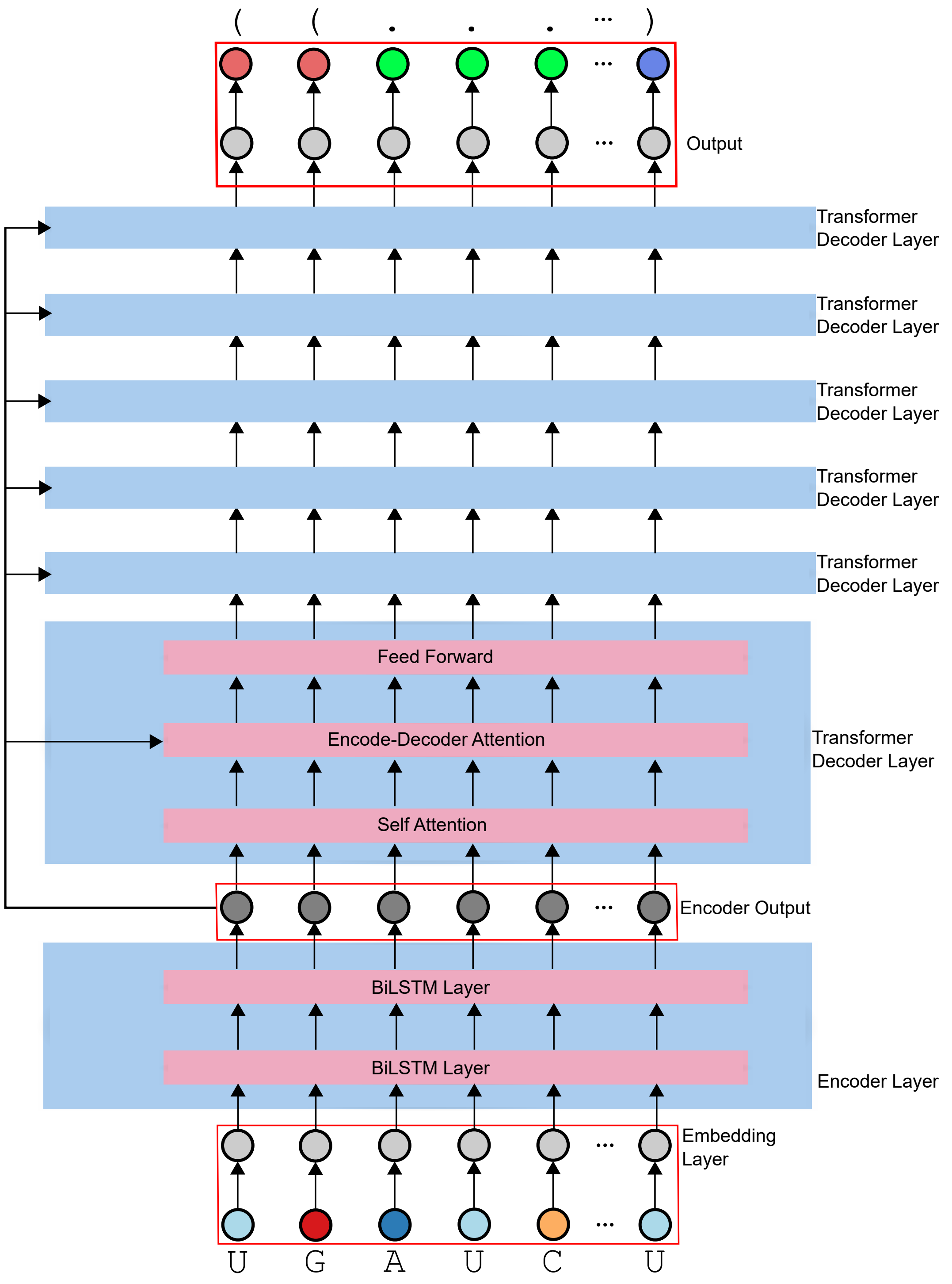

((. . . ...)
Output
Transformer Decoder Layer
Transformer Decoder Layer
Transformer Decoder Layer
Transformer Decoder Layer
Transformer Decoder Layer
Feed Forward
Encode-Decoder Attention
Self Attention
Transformer Decoder Layer
Encoder Output
BiLSTM Layer
BiLSTM Layer
Encoder Layer
Embedding Layer
U G A U C U

Figure 6.2: An overview of the LSTM-Transformer Architecture of DeepStructure.

Model	BP Precision	BP Recall	BP F-score
LSTM+Attention	27.7%	26.4%	27.0
+ BP-Aware Augment	34.1%	33.5%	33.8
LSTM+Transformer	35.4%	35.2%	35.3
+ BP-Aware Augment	43.6%	42.2%	42.9
Transformer-only	18.3%	5.2%	8.1
+ BP-Aware Augment	17.0%	18.4%	17.7

Table 6.1: The base pair prediction performance of the RNA secondary structure using our different settings.

6.3.3 Position Accuracy and Structure Accuracy

The position-wise accuracy and the structure exact match rate are shown in Tab. 6.2, in which our LSTM-Transformer approach reaches 81.9% position accuracy and 27.4% structure accuracy.

Model	Position Accuracy	Structure Accuracy
LSTM+Attention	71.2%	19.6%
+ BP-Aware Augment	74.5%	25.8%
LSTM+Transformer	74.5%	25.2%
+ BP-Aware Augment	81.9%	27.4%
Transformer-only	57.5%	4.1%
+ BP-Aware Augment	58.1%	12.6%

Table 6.2: The position-wise accuracy and the structure (exact match) accuracy on the test set.

6.3.4 Balances of the Predicted Structures

Since our framework does not involve any traditional structural prediction algorithms, there is no guarantee that our predicted structure is balanced in terms of the bracketing. We further investigate the rate of balanced structures out of all the structures we predicted; we also show the predicted left bracket rate and the right bracket rate and compare to the ground truth, shown in Tab. 6.3.

As we can observe, the LSTM decoder (in the LSTM-Attention framework) actually outperforms our Transformer decoder (96.6% vs. 78.6%), although the latter reaches the highest accuracy among all these models.

Model	Structure Balanced	LB / RB Rate	LB / RB Rate Diff
LSTM+Attention	69.4%	22.5% / 20.9%	+0.6% / −1.0%
+ BP-Aware Augment	96.6%	21.4% / 21.3%	−0.5% / −0.6%
LSTM+Transformer	89.0%	22.0% / 21.9%	+0.1% / +0.0%
+ BP-Aware Augment	81.5%	21.4% / 21.3%	−0.5% / −0.6%
Transformer-only	92.8%	6.6% / 6.3%	−15.3% / −15.6%
+ BP-Aware Augment	71.1%	24.5% / 23.7%	+2.6% / +1.8%
Gold	-	21.9% / 21.9%	-

Table 6.3: The balance rate and the predict left/right bracket rate of the predicted structures.

6.4 Methods

6.4.1 Dot-Bracket Representation

We regard the RNA secondary structure prediction as a sequence-to-sequence task, by using the dot-bracket representation as to the structure representation. For example, "`GGGAAACCC (((...)))`" represents a structure with three GC pairs covering three A unpaired regions.

6.4.2 Data Preprocessing

We first used a length filter $[20, 200]$ on the full bpRNA-1m dataset, resulting in 76,009 RNA sequences with labeled structures; we then ran cd-hit [18] to remove replicate sequences with 90% or more similarity, and split the 23,445 sequences into train, val, test set with a $80 : 10 : 10$ proportion.

We next recover sequences from the cd-hit preprocess step, allowing the removed sequences highly similar to some training sequences, but not validation or testing, to be added back to our training set. In this way, 40,360 sequences were included in the training data, and val, test set size were kept at 2,344 and 2,345.

6.4.3 BP-Aware Data Augmentation

As for labeling the structure of an RNA sequence requires other structure prediction tools, we aim to produce more RNA sequences with given structures in the training set. Assume we have a RNA sequence with known structure, the core idea of the base-pair-aware data augmentation is to transform the sequence based on the base-pair relations such that its structure would be kept as stable as possible, e.g., switching two nucleotides in a helix stem would be very unlikely to break the stability of the stem itself.

Practically, for each sequence in the training data, we found all the base pairs that are inside the stems, and then generated up to 10 new RNA sequences each by switching one of the base pairs and kept the original structure. Our final augmented data contains $407k$ RNA sequences as training.

6.4.4 LSTM-Attention Framework

Our encoder architecture is similar to the DeepSloop model, where two stacked layers of BiLSTM are used on top of the one-hot embedding for the inputs. The Attention part is illustrated in Fig. 2.6, where an attention layer is used to connect the encoder (stacked BiLSTM) and the decoder (mono-directional LSTM).

6.4.5 LSTM-Transformer Framework

We introduced LSTM encoder back to the Transformer architecture to combine LSTM encoder and Transformer decoder together. (We also performed experiments on the conventional Transformer-only architecture [53] as the comparison.)

For the conventional Transformer framework, both the encoder and the decoder uses the stacked-attention structure, wherein the encoder, 6 encoder layer was used, and each consists a self-attention layer and a feed-forward layer; in the decoder, 6 decoder layer was used and each consists a self-attention layer, an encoder-decoder attention layer (described in the LSTM-attention section) and a feed-forward layer.

6.4.6 Learning

Generally, we kept the most hyperparameters and/or settings the same as the OpenNMT work [30]. We performed $200,000$ batches as the one training run; we kept Adam as the default optimizer, as the stochastic gradient descent method didn't have a significant difference with or without decayed learning rate comparing to Adam.

In the embedding layer, we started with a random embedding for each nucleotide and kept it updated through the backpropagation of the neural network.

For the LSTM-Attention framework, we used a batch size of 64 during training, with a 0.3 dropout rate in LSTM layers and 0.1 dropout rate in Attention layer. The LSTM hidden dimension was kept to 512.

For the LSTM-Transformer framework, we used a batch size of 1024, having 6 layers of 8-head attentions; 0.1 dropout rate was used during training.

6.4.7 Testing and Performance Metrics

During testing, we used the following metrics to measure the quality of our produced RNA secondary structures are dot-bracket format, giving the length of the output dot-bracket sequence is guaranteed to be equivalent to the input RNA sequence.

- position-wise accuracy: the number of correctly predicted labels (dots and brackets) divided by the total number of labels by comparing each position of each predicted and gold dot-bracket structure.

- structure exact-match accuracy: the number of the correctly predicted examples (RNA structure for a sequence) divided by the total number of sequences.

- base-pair performance, including precision, recall, and F-score: comparing the predicted base-pair set against the gold base-pair set for all examples. If a predicted dot-bracket structure is not bracket-balanced, we simply remove the redundant brackets.

- structure balance rate: the number of the balanced predicted examples divided by the total number of sequences.
- label predicted rate: predicted rate of each label (left bracket / dot / right bracket) comparing to the gold.

Chapter 7: Conclusion

We presented a line of research, focusing on revealing structural information (stem-loops, secondary structures) from RNA sequences. We have explored different techniques on it, starting from an elegant RNA Structure annotation tool with a organized large-scale dataset; then a BiLSTM-based deep learning framework is proposed to predict stem-loops from RNA sequences; at last, we presented a BiLSTM-Attention framework to directly predict the whole sequential RNA secondary structure from scratch, which achieves the sequence-to-sequence and end-to-end characteristic.

The core idea of this line of research is to avoid using the conventional way to reveal RNA structural information, which is based on the hand-designed energy parameters and limited by the imperfect modeling of it. Our presented line of research instead reveals structural information from training a deep neural network model directly from the large-scale dataset, and is able to achieve the goal without using any traditional features.

Alternatively, we explored an efficient dynamic programming algorithm for RNA secondary structure prediction using MFE models with a slightly higher accuracy; we also investigated how to visualize RNA secondary structures and conclude our novel approach into bpRNA-visual to produce better visualizations than previous work. These research pieces are also proved to be useful in the RNA structure prediction area of research and could contribute on different topics of studies.

Future work on this line of research could focus on exploring different deep learning architectures for the RNA secondary structure prediction, including Transformer networks, structure prediction algorithms on top of the neural network, which might be more accurate and efficient on the performance. The topic of a generalized data augmentation approach on training the neural network models could also be addressed, which we explored but not fully in the DeepSloop and DeepStructure research; if the data augmentation approach would be generalized on any learning-to-predict research on RNA sequences/structures, this would be a solid research as well.

Bibliography

[1] Stephen F Altschul and Bruce W Erickson. Significance of nucleotide sequence alignments: a method for random sequence permutation that preserves dinucleotide and codon usage. *Molecular biology and evolution*, 2(6):526–538, 1985.

[2] Mirela Andronescu, Vera Bereg, Holger H Hoos, and Anne Condon. Rna strand: the rna secondary structure and statistical analysis database. *BMC bioinformatics*, 9(1):340, 2008.

[3] Mirela Andronescu, Anne Condon, Holger H Hoos, David H Mathews, and Kevin P Murphy. Efficient parameter estimation for rna secondary structure prediction. *Bioinformatics*, 23(13):i19–i28, 2007.

[4] Mirela Andronescu, Anne Condon, Holger H Hoos, David H Mathews, and Kevin P Murphy. Computational approaches for RNA energy parameter estimation. *RNA*, 16(12):2304–2318, 2010.

[5] David Auber, Maylis Delest, Jean-Philippe Domenger, and Serge Dulucq. Efficient drawing of rna secondary structure. *J. Graph Algorithms Appl.*, 10(2):329–351, 2006.

[6] Philip C Bevilacqua, Laura E Ritchey, Zhao Su, and Sarah M Assmann. Genome-wide analysis of RNA secondary structure. *Annual review of genetics*, 50:235–266, 2016.

[7] Jamie H Cate, Anne R Gooding, Elaine Podell, Kaihong Zhou, Barbara L Golden, Craig E Kundrot, Thomas R Cech, and Jennifer A Doudna. Crystal structure of a group i ribozyme domain: principles of rna packing. *Science*, 273(5282):1678–1685, 1996.

[8] Carl C Correll, Betty Freeborn, Peter B Moore, and Thomas A Steitz. Metals, motifs, and recognition in the crystal structure of a 5s rrna domain. *Cell*, 91(5):705–712, 1997.

[9] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[10] 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.

[11] Kévin Darty, Alain Denise, and Yann Ponty. Varna: Interactive drawing and editing of the rna secondary structure. *Bioinformatics*, 25(15):1974, 2009.

[12] Dezhong Deng, Kai Zhao, David Hendrix, David Mathews, and Liang Huang. Linearfold: Linear-time prediction of rna secondary structures. *bioRxiv*, page 263509, 2018.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] Chuong B Do, Daniel A Woods, and Serafim Batzoglou. Contrafold: Rna secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 2006.

[15] Sean R Eddy. Non–coding rna genes and the modern rna world. *Nature Reviews Genetics*, 2(12):919, 2001.

[16] Richard Eliáš and David Hoksza. Rna secondary structure visualization using tree edit distance. 2016.

[17] Richard Elias and David Hoksza. Traveler: a tool for template-based rna secondary structure visualization. *BMC bioinformatics*, 18(1):487, 2017.

[18] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, 2012.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[20] Sam Griffiths-Jones, Russell J Grocock, Stijn Van Dongen, Alex Bateman, and Anton J Enright. miRBase: microRNA sequences, targets and gene nomenclature. *Nucleic acids research*, 34(suppl_1):D140–D144, 2006.

[21] ME Harris, AV Kazantsev, JL Chen, and NR Pace. Analysis of the tertiary structure of the ribonuclease p ribozyme-substrate complex by site-specific photoaffinity crosslinking. *Rna*, 3(6):561–576, 1997.

[22] Maya Hirohara, Yutaka Saito, Yuki Koda, Kengo Sato, and Yasubumi Sakakibara. Convolutional neural network based on smiles representation of compounds for detecting chemical motif. *BMC bioinformatics*, 19(19):526, 2018.

[23] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

[24] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics, 2012.

[25] Liang Huang, He Zhang, Dezhong Deng, Kai Zhao, Kaibo Liu, David A Hendrix, and David H Mathews. Linearfold: linear-time approximate RNA folding by 5'-to-3'dynamic programming and beam search. *Bioinformatics*, 35(14):i295–i304, 2019.

[26] Brian Johnson and Ben Shneiderman. *Tree-maps: A space-filling approach to the visualization of hierarchical information structures*. IEEE, 1991.

[27] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.

[28] Peter Kerpedjiev, Stefan Hammer, and Ivo L Hofacker. Forna (force-directed rna): simple and effective online rna secondary structure diagrams. *Bioinformatics*, 31(20):3377–3379, 2015.

[29] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[30] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[31] Rohan V Koodli, Benjamin Keep, Katherine R Coppess, Fernando Portela, Eterna participants, and Rhiju Das. Eternabrain: Automated RNA design through move sets and strategies from an internet-scale RNA videogame. *PLoS computational biology*, 15(6):e1007059, 2019.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[33] Ronny Lorenz, Stephan H Bernhart, Christian Hoener 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.

[34] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[35] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[36] David H Mathews. Using an rna secondary structure partition function to determine confidence in base pairs predicted by free energy minimization. *Rna*, 10(8):1178–1190, 2004.

[37] David H Mathews, Matthew D Disney, Jessica L Childs, Susan J Schroeder, Michael Zuker, and Douglas H Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–7292, 2004.

[38] David H Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of rna secondary structure. *Journal of molecular biology*, 288(5):911–940, 1999.

[39] Francois Michel and Eric Westhof. Modelling of the three-dimensional architecture of group i catalytic introns based on comparative sequence analysis. *Journal of molecular biology*, 216(3):585–610, 1990.

[40] Seunghyun Park, Seonwoo Min, Hyunsoo Choi, and Sungroh Yoon. deepmirgene: Deep neural network based precursor microrna prediction. *arXiv preprint arXiv:1605.00017*, 2016.

[41] Andrew Pavlo. Interactive, tree-based graph visualization. 2006.

[42] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[43] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[44] Ben Shneiderman. Tree visualization with tree-maps: A 2-d space-filling approach. Technical report, 1998.

[45] Michael F Sloma and David H Mathews. Exact calculation of loop formation probability identifies folding motifs in rna secondary structures. *RNA*, 22(12):1808–1818, 2016.

[46] Srinivas Somarowthu, Michal Legiewicz, Isabel Chillón, Marco Marcia, Fei Liu, and Anna Marie Pyle. Hotair forms an intricate and modular secondary structure. *Molecular cell*, 58(2):353–361, 2015.

[47] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[48] I Sutskever, O Vinyals, and QV Le. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

[49] P Svoboda and A Di Cara. Hairpin RNA: a secondary structure of primary importance. *Cellular and Molecular Life Sciences CMLS*, 63(7-8):901–908, 2006.

[50] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

[51] Masaru Tomita. Graph-structured stack and natural language parsing. In *26th Annual Meeting of the Association for Computational Linguistics*, 1988.

[52] Lee E Vandivier, Stephen J Anderson, Shawn W Foley, and Brian D Gregory. The conservation and function of RNA secondary structure in plants. *Annual review of plant biology*, 67:463–488, 2016.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[54] Michelle J Wu, Johan OL Andreasson, Wipapat Kladwang, William J Greenleaf, Rhiju Das, et al. Prospects for recurrent neural network models to learn rna biophysics from high-throughput data. *bioRxiv*, page 227611, 2017.

[55] Zhenjiang Xu and David H Mathews. Multilign: an algorithm to predict secondary structures conserved in multiple rna sequences. *Bioinformatics*, 27(5):626–632, 2010.

[56] Michael Zuker, David H Mathews, and Douglas H Turner. Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. In *RNA biochemistry and biotechnology*, pages 11–43. Springer, 1999.

[57] Michael Zuker and Patrick Stiegler. Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.

APPENDICES

Appendix A: bpRNA-Visual: A Compact RNA Secondary Structure Visualizer with Layout Optimization via Genetic Algorithm

A.1 Abstract

Visualizing RNA secondary structure can be extremely important and useful for RNA structure research. As visualizing RNA secondary structure is a complex and challenging task, major drawbacks on the existing implementations include 1) lack of simplicity on the data structure design and program implementation, and 2) the layout of visualization is ease-to-fail due to the massive overlapping issues. To address these problems, a compact visualizer, bpRNA-Visual, is introduced for RNA secondary structures with clean algorithm design and layout optimization via genetic algorithms. Visualization and optimization results show the improved performance of our presented approach.

A.2 Overview of bpRNA-Visual

Visualizing RNA secondary structure can be extremely important and useful for RNA structure research. While visualizing RNA secondary structure is a complex and challenging task, there exist some visualization systems from previous work, including VARNA[11], forna[28], and TRAVeLer[17].

However, there are two major drawbacks to the existing implementations. First of all, the currently available visualizers are based on the complex data structure design and program implementation, making a long road to explain and code, thus not suitable for beginners to learn and implement the visualizer of RNA secondary structures. This is contrary to the natural properties of RNA secondary structure which is defined in a simple way and the core part of a visualizer shouldn't require any highly complicated data structure and algorithms. Moreover, the layout of visualizing RNA secondary structure is ease-to-fail due to the massive overlapping issues from stems and loops, especially for long-ranged RNA sequences. Existing works are either not able to solve the structure overlaps in the visualization, or tend to lead to non-intuitive shapes for long-ranged RNA sequences, or heavily based on hand-designed templates.

To address these problems, we first designed a recursive layout algorithm, assigning positions to all the nucleotides through a simple and elegant strategy based on the local structure; then we designed and implemented a novel genetic algorithm to optimize the layout, minimizing the overlaps of structures revealed in a systematic way; the presented visualizer is named as **bpRNA-Visual**. The basic layout algorithm visualizes RNA secondary structures in 200 lines of Python code and uses the basic data structure and a well-defined recursive algorithm to achieve all the basic functions, and the advanced genetic algorithm solves the massive layout problem and results in a rational layout without any hand-designed templates.

Our designed data structure could lead to a unique mapping to the tree structures defined in the previous work [16]; we also reviewed the currently available

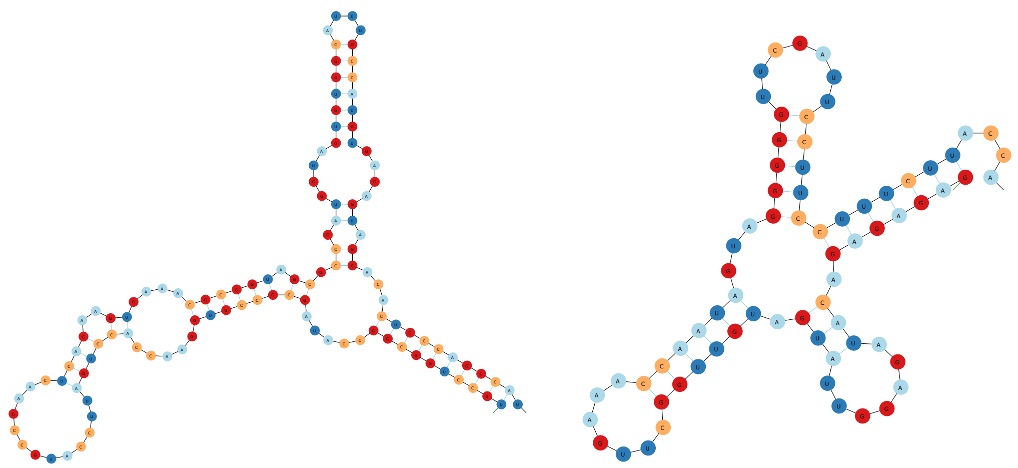


Figure A.1: Example visualizations via the recursive layout algorithm only. Left: 5s Aeromonas salmonicida rRNA; Right: tRNA tdbR00000403.

RNA secondary structure visualizers [5, 11, 28, 17]; the presented genetic algorithm gathered insights from the tree structure visualization literature [50, 26, 44, 41].

A.3 Results

A.3.1 Visualizing RNA Secondary Structures via Recursive Layouts

Fig. A.1 shows some example visualizations through our recursive layout algorithm only (no optimization). Generally, the algorithms starts from the 5' and 3' end, drawing two points with a unit distance and makes a recursion based on the local structure: if the local structure on top of the current two nucleotides is a loop, then a circle is drew (top right part of tRNA in Fig. A.1); if the local structure forms a stem (helix), then the double-helix structure is drew on top of a long rectangle. After all the subsequences are solved using recursion, a fixed visualization

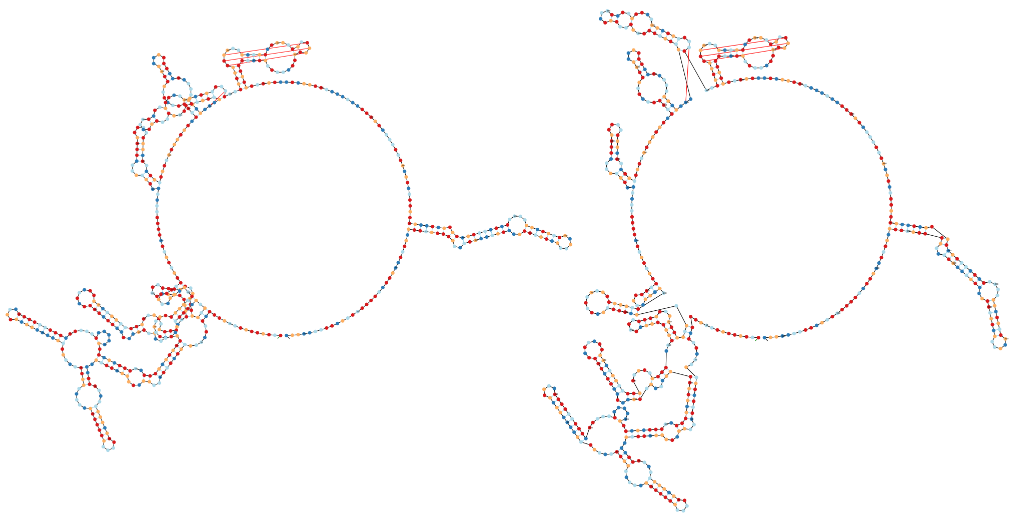


Figure A.2: Example of the layout optimization by the proposed genetic algorithm. Left: bpRNA-CRW-95 before optimization; Right: after optimization. Pseudoknots exist in this structure, marked as red segments between nucleotides.

is generated for the output.

A.3.2 Layout Optimization via Genetic Algorithm

The layout generated by the recursive algorithm could solve most of the RNA structures with smaller sequence length ($\leq 200nts$) without overlaps, while there are some exceptions due to the direction assignment of substructures in a circle is without any heuristics. For long RNA sequences, the structures could be a lot more complicated (Fig. A.2 Left, Fig. A.3 Left) and adjusting the directions of substructures might not be enough to solve all the overlap structures. Thus both rotation actions and stretch actions are needed.

We designed and implemented the genetic algorithm, aiming to solve the overlapping issues by performing rotation and stretch actions on substructures in a heuristic way. Fig. A.2 and Fig. A.3 shows two examples before and after layout optimization through genetic algorithm. On Fig. A.2(bpRNA_CRW_95, length:563nts), our algorithm ran through the overlapping substructures and solved all the overlapping parts in the end; on Fig. A.3(bpRNA_CRW_1, length:$1,434nts$), the algorithm ran out of all 30 epochs, resulting in the final structure layouts with no overlapping nucleotides and a single-digit number of crosses of segments.

Generally, the genetic algorithm runs several epochs; at the beginning of each epoch, we shuffle all the substructures in a different order and perform rotation and stretch actions on substructures according to this order; after each action is performed, we keep track of the number of overlaps, and keep the updated layout in a high probability (Monte-Carlo strategy) if the number of overlaps has reduced from this action; the algorithm stops after a non-overlapping layout is found or we run out of all the epochs. The algorithm is generally doing an optimization task on the number of overlaps of the whole structure; when the layout is updated, the number of overlaps is guaranteed to be reducing. The stretch parameter is also controlled by the epoch index since long stretches might not lead to a dense layout. The detailed algorithm is described in the method section below (see Algorithm 3).

A.4 Discussion

We present a novel visualization tool for RNA structures, bpRNA-Visual, which optimizes both data structure / program design and the layout strategy. bpRNA-

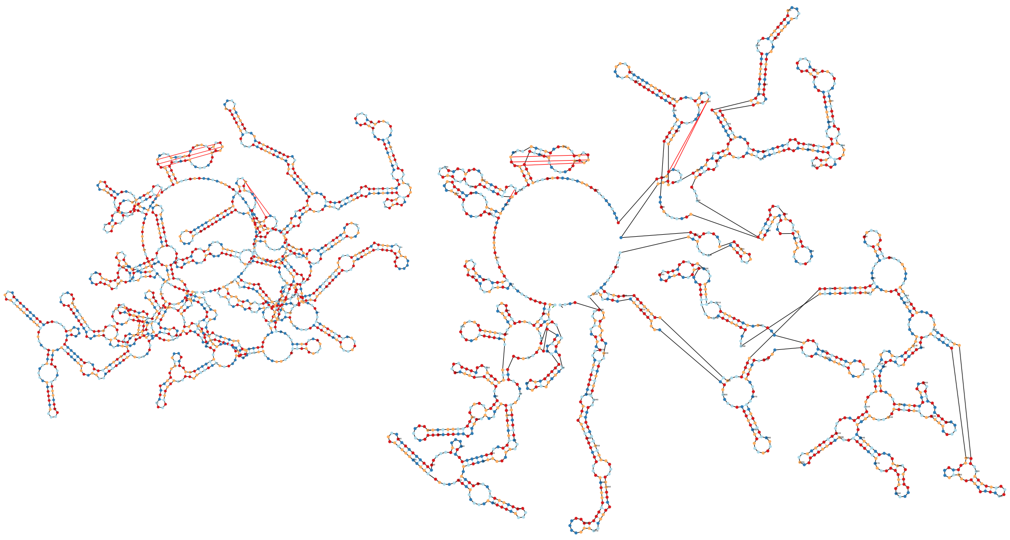


Figure A.3: Layout optimization in complex scenarios. Left: bpRNA-CRW-1 (1,434nts) before optimization; Right: after optimization.

Visual has 200 lines of code for the basic RNA visualization program, which is a lot simpler than any existing approaches and could be used as tutorials or teaching materials for people getting started with computational RNA research; which is also good to reveal the veil of RNA structure visualization, letting people understand how visualization works on drawing these secondary structures in an easier way. The advanced version could be used for general-purpose, producing a rational layout for RNA sequences, and even long-ranged RNA sequences. Comparing to previous studies, our approach could efficiently solve the overlapping issues in a genetic way and does not rely on any hand-designed templates, so that even the RNA shape is far away from any templates we have in the database, the program could still produce the optimized layout of the structures.

Future work may focus on two directions: 1) reducing the complexity of computing the number of overlaps. The current complexity here is $O(k * n)$ where k is the number of nucleotides in the moved substructure and n is the number of all nucleotides. Since we would need to check this optimizing goal after every set of actions, this occupies the majority of the run-time. An approximation strategy could be used here to significantly reduce the run-time; 2) pruning redundant actions. There is only a few numbers of actions that could minimize our goal especially in the later epochs, which makes the program performing too many redundant actions. We could possibly calculate some heuristics by the local layouts to try actions only if it would not cause new overlaps against the current layout. This might boost the efficiency of bpRNA-Visual as well.

A.5 Methods

A.5.1 The Recursive Layout Algorithm

To visualize the secondary structure of an RNA sequence, we need the sequence of nucleotides and its base pairs as the inputs of the program. We denote the input RNA sequence as $s = s_1 s_2 ... s_n$, and the base pair set as $B_s = \{(i, j)\}$. We also use $s[i, j]$ to represent the sub-sequence $s_i ... s_j$.

The goal of the recursive layout algorithm is to go through the input structure and assign each nucleotide with a position in the plane, i.e., an (x, y) coordinate. Once the position assignment is finished, we could either start plotting the structure with these positions, or use high-level algorithms to optimize the layout positions.

We define the state and transition as follows.

A.5.1.1 State

To start with the layout of $s = s[1, n]$, we select two points in the plane for the 5' end and the 3' end, and choose one of the two directions (vertical to the segment of the two points) to expand the layout.

We define a state as a tuple of a sub-sequence $s[i, j]$, a pair of points p_1, p_2, and a unit vector as the direction D:

$$state = (s[i, j], p_1, p_2, D).$$

The intuition of the state definition can be conclude as, if $(i, j) \in B_s$ and $(i + 1, j-1) \in B_s$, we would simply transit the state to $(s[i+1, j-1], p_1+D, p_2+D, D)$; and otherwise there would be a loop containing either s_i, s_j which we need to assign the layout and plot the loop circle.

For the starting state, we could easily choose a fixed set of values of p_1, p_2, and D, i.e., $p_1 = (-0.5, 0), p_2 = (0.5, 0), D = (0, 1)$.

$$state_0 = (s[1, n], (-0.5, 0), (0.5, 0), (0, 1))$$

A.5.1.2 Transitions

We consider three different cases for transiting a state $(s[i,j], p_1, p_2, D)$ to another: 1) $s[i,j]$ only contains one or two nucleotides; 2) helix, $(i,j), (i+1,j) \in B_s$; 3) otherwise, i.e., a loop would contain s_i and s_j.

- if $s[i,j]$ contains only one nucleotide s_i, we assign its position as the middle point of p_1 and p_2; if it contains only two nucleotides, we directly assign p_1 to s_i and p_2 to s_j. In this case, we close the state without transiting further.
- if $(i,j), (i+1,j) \in B_s$ (helix), we simply assign p_1 to s_i, p_2 to s_j, and then transit the state to $(s[i+1, j-1], p_1 + D, p_2 + D, D)$.
- otherwise, we use a circle shape to represent the current loop containing s_i, s_j. We go through $s_i s_{i+1} ... s_j$ and collect all k nucleotides in the current loop, note as $s_{l_1} s_{l_2} ... s_{l_k}$, where $i = l_1, j = l_k$.

  We then calculate the position of the circle center C which is at the D direction of the middle point of p_1, p_2, with an L distance, such that all k nucleotides could be uniformly distributed on the circle with the unit distance against their neighbors.

  $$L = \frac{1}{2\tan(\frac{\pi}{k})}$$

  By rotating the coordinates of p_1 with the center of C by $\frac{2\pi}{k}$ each time, all the positions of $s_{l_1} s_{l_2} ... s_{l_k}$ are calculated and assigned to $p_{l_1}, p_{l_2}, ..., p_{l_k}$.

We then consider nucleotides in $s_{l_1}s_{l_2}...s_{l_k}$ into two cases:

- s_{l_u} is unpaired and unrelated to any sub-structure, i.e., $l_{u-1} = l_u - 1, l_{u+1} = l_u + 1$; in this case we don't further transit to another state;
- $(l_u, l_{u+1}) \in B_s$, in this case $s[l_u, l_{u+1}]$ expands a sub-structure out of the current loop, we transit to a state $(s[l_u, l_{u+1}], p_{l_u}, p_{l_{u+1}}, D')$, where D' is the direction from the current circle center C to the midpoint of p_{l_u} and p_{l_u+1}.

The recursive layout algorithm starts from $state_0$ and executes the transitions until the positions of all nucleotides are assigned. We denote positions of each nucleotide as P_s, and the set of all the states as ST_s. Since each state corresponds to a base pair in the input structure except $state_0$ may or may not, we have $|B_s| \leq |ST_s| \leq |B_s| + 1$, thus this algorithm runs in linear time against the length of the input sequence n.

A.5.2 The Genetic Algorithm

The recursive layout algorithm would lead to a fixed structure layout for each of the input RNA, with a set of states ST_s and positions of each nucleotide P_s. However, as well as previous studies, the fixed structure layout might have overlapping issues. We propose a genetic algorithm to optimize towards minimum overlapping by exploring a combination of transforming substructure layouts, i.e., performing state actions.

A.5.2.1 State Actions

The core idea of transforming a substructure layout of $s[i, j]$ is to perform rotation and stretch transformation of all the nucleotides within $s[i, j]$, based on our defined state $(s[i, j], p_1, p_2, D)$.

Consider a state $(s[i, j], p_1, p_2, D)$, we define three different actions, left rotation, right rotation, and stretch. The term "left" and "right" are based on the direction D, as we could always claim p_1 is on the left side and p_2 is on the right side given D as the direction.

- Left rotation of an angle θ ($0 < \theta \leq \frac{\pi}{2}$). Regarding p_1 as the center, consider every nucleotide within $s[i, j]$, we rotate its position by a degree of θ in the counterclockwise direction.

- Right rotation of an angle θ ($0 < \theta \leq \frac{\pi}{2}$). The rotation is clockwise and the center is p_2 instead.

- Stretch by a distance of L. Perform a stretch transformation of every nucleotide within $s[i, j]$ towards the D direction by an L distance.

A.5.2.2 Greedy and Monte-Carlo Strategy

The basic intuition of the genetic search is to 1) perform a state action on one state; 2) check if the overall number of overlaps is reduced; 3) if yes, decide if we would keep this action to update our layout. Whenever the overlap count is reduced, by using the greedy strategy, we always keep the action; and by the Monte-Carlo strategy, we use a fixed probability of α to control the process, and ignore this action with an α probability.

A.5.2.3 Genetic Search

Algorithm 3 describes our genetic search algorithm. Generally, we run the genetic search for a number of epochs E, with default 30; for each epoch, we generate a different order of ST_s, and loop over the states in order; for each state, we perform rotate actions and stretch actions, and update the layout with the probability $1 - \alpha$ if the new layout has less number of overlaps. The algorithm stops if a non-overlapping structure visualization is found, otherwise, we return the layout with the minimum number of overlaps in the current search. Fig. A.2, A.3 show the layout optimization via genetic search.

Algorithm 3 Genetic Search Algorithm for Layout Optimization

```
1: $actions_{rotate} \leftarrow \{rotate(\text{clockwise, counter-clockwise})\} \times \{degree(15°, 30°, ..., 150°)\}$
2: function GeneticSearch($ST_s, P_s, B_s, E, \alpha$)
3:     ▷ input is the state set $ST_s$, positions of nucleotides $P_s$ base pairs $B_s$, epochs $E$, and Monte-Carlo probability $\alpha$
4:     $m \leftarrow CalculateNumberOfOverlaps(P_s, B_s)$
5:     if $m = 0$ then
6:         return $P_s$
7:     end if
8:     ▷ $m$ is the number of overlaps of current layout
9:     for $e \in E$ do
10:        $states \leftarrow$ a shuffled array of $ST_s$
11:        ▷ shuffle states at each epoch
12:        for $state \in states$ do
13:            for $action_{rotate} \in actions_{rotate}$ do
14:                for $action_{stretch} \in (e+1) \times stretch(random(0, 2e))$ do
15:                    ▷ $e+1$ different lengths are randomized ranging from 0 to $2e$
16:                    $P'_s \leftarrow PerformAction(P_s, state, action_{rotate})$
17:                    $P'_s \leftarrow PerformAction(P'_s, state, action_{stretch})$
18:                    $m' \leftarrow CalculateNumberOfOverlaps(P'_s, B_s)$
19:                    if $m' < m$ and $random(0, 1) > \alpha$ then
20:                        $P_s \leftarrow P'_s$
21:                        $m \leftarrow m'$
22:                        if $m = 0$ then
23:                            return $P_s$
24:                        end if
25:                    end if
26:                end for
27:            end for
28:        end for
29:    end for
30:    return $P_s$
31: end function
```