UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

MASTER THESIS No. 2699

# EVALUATION OF RNA ATOM DISTANCE PREDICTION MODELS

Jelena Šarić

Zagreb, February 2022

UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

MASTER THESIS No. 2699

# EVALUATION OF RNA ATOM DISTANCE PREDICTION MODELS

Jelena Šarić

Zagreb, February 2022

**UNIVERSITY OF ZAGREB**
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

Zagreb, 15 October 2021

# MASTER THESIS ASSIGNMENT No. 2699

| | |
|---|---|
| Student: | **Jelena Šarić (0036498474)** |
| Study: | Computing |
| Profile: | Computer Science |
| Mentor: | prof. Mile Šikić |

Title: **Evaluation of RNA Atom Distance Prediction Models**

Description:

This thesis aim is an evaluation of neural network architectures currently used for predicting the 3-dimensional structure of large biological molecules. The field has attracted tremendous interest since a deep neural network model AlphaFold made a breakthrough on protein folding task. A large part of AlphaFold model is dedicated to making accurate predictions on the distance between every two atoms in the protein. Several neural network layers, such as convolution, gated self-attention or triangular update layers, have been used for this purpose. It is important to evaluate and compare the performance of all of them on the closely related problem of RNA folding - prediction of distances between RNA atoms. As indicated by previous works, it is expected that self-attention outperforms convolutions but at the cost of significantly higher compute time and memory requirements. The solution should be implemented in Python with the PyTorch or similar computational library. The source code should be documented using comments and should follow the Google Python Style Guide when possible. The complete application should be hosted on GitHub under an OSI approved license.

Submission date: 04 February 2022

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 15. listopada 2021.

# DIPLOMSKI ZADATAK br. 2699

| | |
|---|---|
| Pristupnica: | **Jelena Šarić (0036498474)** |
| Studij: | Računarstvo |
| Profil: | Računarska znanost |
| Mentor: | prof. dr. sc. Mile Šikić |

Zadatak: **Vrednovanje modela za predikciju udaljenosti između RNA atoma**

Opis zadatka:

Cilj ove teze je vrednovanje slojeva neuronskih mreža trenutno korištenih za predviđanje trodimenzionalne strukture velikih bioloških molekula. Ovo područje je privuklo velik interes nakon što je AlphaFold model duboke neuronske mreže napravio iskorak u predviđanju smatanja proteina. Velik dio AlphaFold modela je posvećen točnom predviđanju udaljenosti između bilo koja dva atoma u proteinu. U tu svrhu korišteno je nekoliko vrsta slojeva dubokih mreža, kao što su konvolucijski, upravljani slojevi samo-pažnje i triangulacijsko osvježavanje. Važno je vrednovati i usporediti performanse svakog od njih na bliskom problemu vezanom uz smatanja RNA - predviđanju udaljenosti između RNA atoma. Kako je naznačeno u proteklim radovima, očekivano je da slojevi samo-pažnje postižu bolje rezultate od konvolucijskih, ali uz cijenu značajno duljeg vremena izvođenja i memorijskog zauzeća. Rješenje je potrebno implementirati u programskom jeziku Python koristeći PyTorch ili sličnu biblioteku za matrični izračun. Izvorni kod je potrebno dokumentirati koristeći komentare i razvijati prema Google Python Style Guide kada je to moguće. Cijeli programski proizvod potrebno je postaviti na GitHub pod jednu od OSI odabranih licence.

Rok za predaju rada: 4. veljače 2022.

# CONTENTS

# 1. Introduction

Owing to its architecture, an RNA molecule has the ability to fold and form three-dimensional structures. Although structures understanding is imperative since it can lead to skyrocketing discoveries in medicine and pharmacy, the number of known RNA structures is still at a disappointing level due to the difficulty of determining it.

Inspired by the recent progress in solving similar problems, the *RNAFold* model predicts a three-dimensional RNA structure using deep learning methods.

In favour of getting an overview and a direction in the first stages of RNAFold's implementation, this thesis aims to analyse different model configurations. By observing multiple performance metrics, the objective is to find patterns between model architecture and performance, and determine which factors may improve current model implementation.

Firstly, the problem will be thoroughly explained in the following chapter to understand the motive behind this model and its aim. Additionally, similar projects will be referenced to explain the inspiration behind the model implementation.

The introduction will be followed by the chapter that focuses on the model interpretation, model's expected input and output, architecture, losses and modifications to the original version that will be included to define different analysis versions.

The *Implementation* chapter will give a quick overview of analysis implementation, its organisation and used hyperparameters.

Furthermore, testing results, tables and visual representations of model performances shall be included in the *Results* chapter to give a detailed comparison between the model versions based on the different observed metrics.

Observed results will be analysed and commented on in the *Discussion*, followed by the *Conclusion* chapter.

# 2. Problem

Both proteins and RNA molecules have a tendency to fold and consequently form a three-dimensional structure. Their functions and effect on the organism vary depending on their structure. Mechanisms of RNA function, designing synthetic RNAs and discovering RNA-targeted drugs are just one of many examples of why knowledge of such structures is of great importance. [3]

The problem mentioned above is well known as the RNA structure prediction problem. It focuses on determining the three-dimensional shape of RNA from its nucleotide sequence.

However, the problem is that predicting such structures has been shown to be very challenging due to a limited number of determined structures. Therefore, computational approaches have become increasingly popular.

According to the biannual *Critical Assessment of Structure Prediction* (*CASP*) experiments, deep learning methods have demonstrated significant improvement compared to other more traditional methods. Among plenty of models that apply the deep learning approach using data from the rich database of known protein structures (*PDB*), *AlphaFold* and *TrRosetta* are instances of those with remarkable results.

The latest experiment *CASP14* has raised the bar due to AlphaFold2's[2] impressive performance on proteins for which the experimental structures were not public at the time. [1] AlphaFold2 outcompeted other participants of around 100 groups by achieving accuracy competitive with experimental structures in most cases. [2]

Although there has been significant progress in recent years, the RNA structure prediction problem still faces various challenges. In addition, RNA structure prediction has proven to be even more arduous than predicting the structure of a protein.

Some methods that have positively impacted protein structure prediction have not triumphed as much in the corresponding RNA problem since structure data of the related RNAs is not as available. Furthermore, even if that was not the case, sequence coevolution information did not show too promising results when determining tertiary contacts in RNA.[3]

A neural network named *Atomic Rotationally Equivariant Scorer* (*ARES*) tackles previously mentioned obstacles by using many processing layers and recycling outputs to learn effectively with a modest dataset. Given a model specified by each atom's 3D coordinates and chemical element type, ARES predicts the structural model's root mean square deviation (RMSD) from the unknown true structure. Based on the corresponding article - *Geometric deep learning of RNA structure* - ARES continuously outperformed the competition by identifying accurate structural models compared with other scoring functions (*Rosetta*, *Ribonucleic Acids Statistical Potential* (*RASP*) and *3dRNAscore*).[3]

The RNAFold model, on which this thesis focuses, aims to utilize the aforementioned protein prediction techniques on the corresponding RNA folding problem. The idea behind the implementation is to learn from the best approaches in predicting protein structures and apply the most promising practices to predict RNA 3D structures. Since AlphaFold2 showcased outstanding CASP performance, the model implementation is mainly based on the AlphaFold2.

Given RNA as nucleotide sequence and aligned sequences of homologues as inputs, the RNAFold predicts the three-dimensional coordinates of all heavy atoms. The network consists of two distinct components - *Evoformer* and *Structure module*. While Evoformer illustrates the first structure sketch, Structure module provides explicit 3D structure.

Since the model's implementation is still in progress and consumes a considerable amount of time and memory, the idea is to explore whether other model configurations achieve similar, if not better, results with less time and memory.

The main focus will be on the Evoformer network block that exchanges information between the *Multiple Sequence Alignment* (*MSA*) and pair representations to provide relevant spatial and evolutionary relationship information. It consists of both attention-based and non-attention-based components.

Instead of using very complex attention layers, various combinations of convolutional layers will be tested since they have a history of performing well in related problems and require less trainable parameters.

# 3. Model

The RNAFold model tackles the problem of predicting the three-dimensional structure of RNA based on its nucleotide sequence.

The model's architecture is complex and consists of several components. On the input, the network expects RNA data in the form of MSA features and pair representations. Said features are preprocessed before the model forwards input data to the aforementioned Evoformer block. Once prepared, the Evoformer block applies multiple operations to mix information between the pair and MSA representations for structure generation within the Structure module. The Structure module's output and thus the RNAFold model's output produces a three-dimensional structure in the form of a rotation and translation for each atom of the nucleotide.

In pursuance of higher accuracy, iterative refinement is applied all over the network by repeatedly reusing the outputs as inputs into the same models and including the final loss in outputs.

Each network's components will be explained in detail in the following sections.

## 3.1. Feature preprocessing

A single model input is implemented as a tensor dictionary, where every dictionary has components listed in Table 3.1. Corresponding constants are defined in Table 3.2.

During the training and validation phases, samples are chosen randomly within training and validation sets. On the other hand, during testing, they are referenced by index.

Since not all input sequences have the same length, a random crop is applied to every sample. In other words, instead of using the entire sequence length, only a random sequence window is observed. Starting position is uniformly sampled from $[0, n - N_c + 1]$, where $n$ is a sequence length and $N_c$ is a fixed crop size.

*Feature preprocessing* module also expects recycled outputs on the input according to the refinement that the network constantly applies. Therefore, recycled MSA and

pair features are added to the input value.

Figure 3.1 illustrates feature preprocessing flow. While `unsqueeze` refers to adding one additional dimension, `R` stands for applying recycled outputs.

`relpos` operation is defined through Algorithm 1, and its purpose is to provide information about the positions of nucleobases in the chain. The algorithm calculates relative distance within a chain, encodes it as a one-hot vector and performs linear projection.

Input features enter the preprocessing stage in order to provide final input representation by embedding input values. Consequently, both MSA features and pair features have embeddings instead of one-hot encoding representation.

**Table 3.1:** Input Details

| Feature | Dimension | Definition |
|---|---|---|
| | Input Features | |
| sequence | $N_c \times N_b$ | a nucleobase sequence tensor where every nucleobase is encoded as a one-hot encoding vector |
| index sequence | $N_c$ | a tensor containing range $\{0, 1, ..., N_c - 1\}$ |
| MSA features | $N_{al} \times N_c \times N_t$ | a tensor containing MSA information: nucleobases of aligned sequences where every nucleobase is encoded as one-hot encoding vector and deletions vector |
| pairing probabilities | $N_c \times N_c$ | pairing probabilities between every two positions in a sequence |
| | Target Features | |
| atom coordinates | $N_c \times N_{at} \times 3$ | a tensor containing three-dimensional coordinates for every atom of nucleotide |
| atom coordinates mask | $N_c$ | a tensor mask indicating whether three-dimensional coordinates are defined for a corresponding position in a sequence |
| prediction features | $N_c \times N_p$ | a tensor containing values for every prediction feature for every corresponding position in a sequence |

**Table 3.2:** Constants

| constant | description |
|---|---|
| $N_c$ | crop size, i.e. number of sequence positions included in the crop window |
| $N_b$ | number of differentiating nucleobases; size of a vector encoding single position in a nucleobase sequence |
| $N_{al}$ | number of alignments observed in a MSA feature |
| $N_t$ | size of a vector encoding single position in a MSA feature |
| $N_p$ | number of prediction features |
| $N_e$ | embedding size |

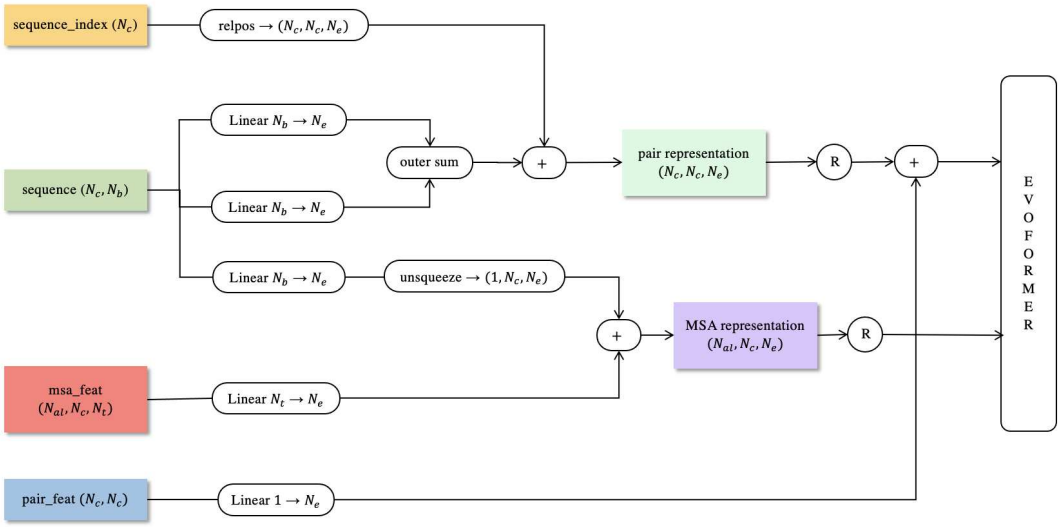



**Figure 3.1:** Feature preprocessing

**Algorithm 1** Relative position encoding

**function** RELPOS($\mathbf{f}, \mathbf{v}_{bins} = [-32, -31, ..., 32]$) ▷ $\mathbf{f} \in \mathbb{N}_0^{N_c}$
  $\mathbf{d} \leftarrow$ outer sub($\mathbf{f}, \mathbf{f}$) ▷ $\mathbf{d} \in \mathbb{Z}^{N_c \times N_c}$
  $\mathbf{b} \leftarrow$ arg min($|\mathbf{d} - \mathbf{v}_{bins}|$) ▷ $\mathbf{b} \in \mathbb{N}_0^{N_c \times N_c}$
  **out** $\leftarrow$ Linear(one_hot($\mathbf{b}$)) ▷ **out** $\in \mathbb{R}^{N_c \times N_c \times N_e}$
  **return out**
**end function**

## 3.2. Evoformer

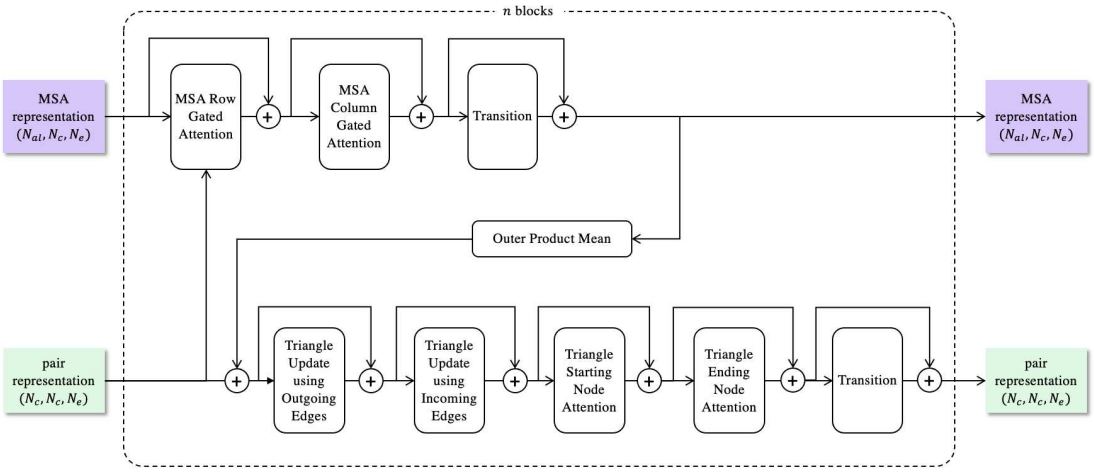



**Figure 3.2:** Evoformer v1

The Evoformer module consists of multiple blocks with shared architecture but no shared weights.

Besides the fact that different combinations of attention and dropout layers are applied on the Evoformer input, there is continuous communication between the MSA and pair representation through an outer product.

Since this prediction problem can be interpreted as a graph prediction problem with sequence positions presenting nodes and pair representation illustrating the information about the edges, the triangle inequality on the edges must be satisfied. Consequently, the Evoformer stack includes both attention and non-attention based triangle updates.

### 3.2.1. MSA row-wise gated self-attention with pair bias

The operation illustrated in Figure 3.3 updates a row in MSA feature by exchanging information between elements that belong to the same target alignment. Both MSA and pair representations are provided as inputs.

For the sake of obtaining attention weights, processed queries and keys are multiplied, and projected pair representation is included as bias.

Queries, keys, values and gate values are calculated by grouping projected MSA input into heads.

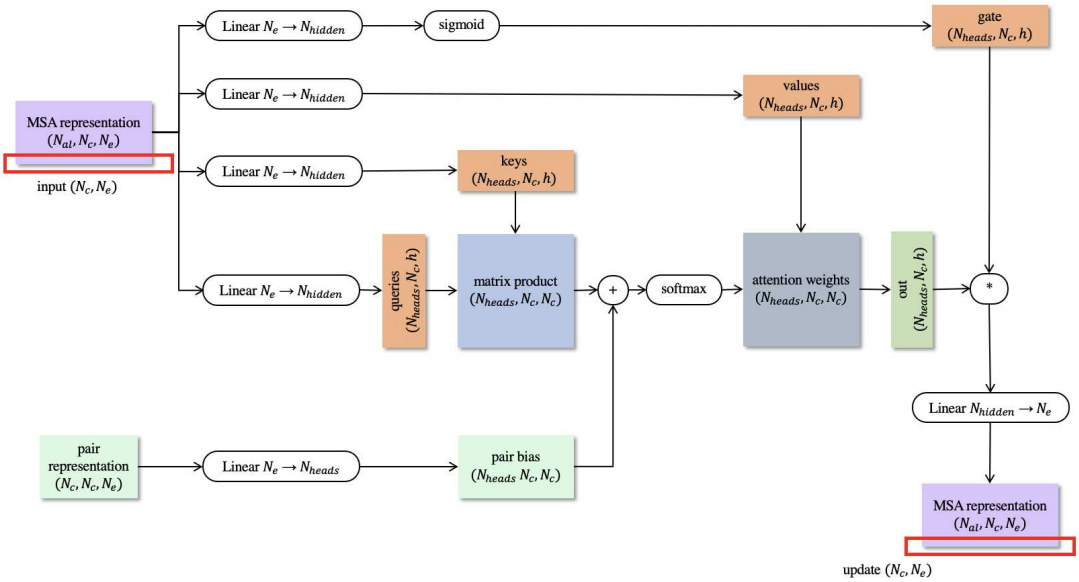



**Figure 3.3:** MSA row-wise gated self-attention

### 3.2.2. MSA column-wise gated self-attention

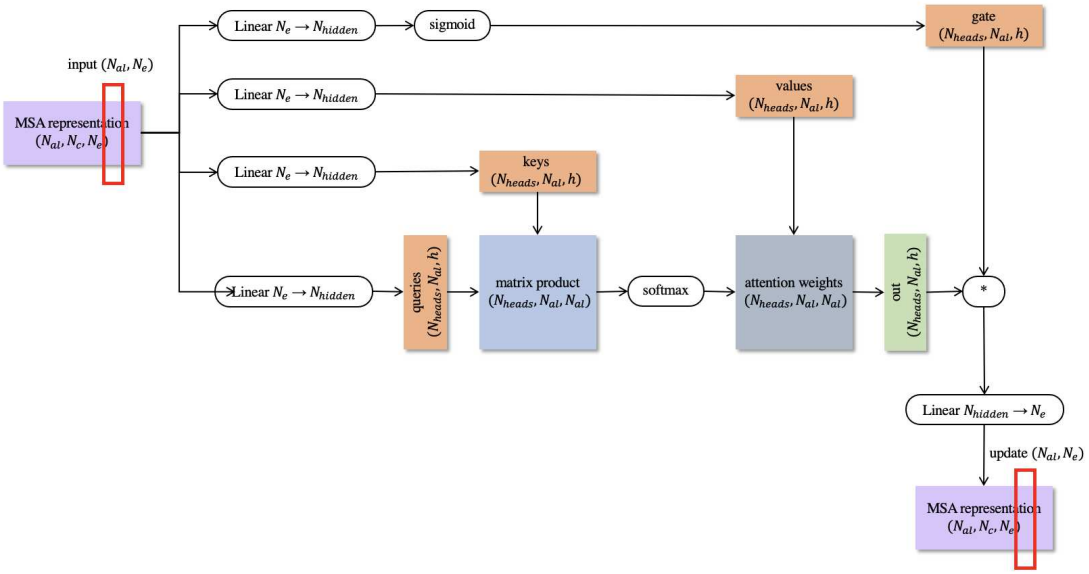



**Figure 3.4:** MSA column-wise gated self-attention

Communication between elements that belong to the same target nucleotide updates a column in the MSA feature, as shown in Figure 3.4.

Since only MSA is expected on input, there is no additional bias when calculating attention weights.

### 3.2.3. Transition

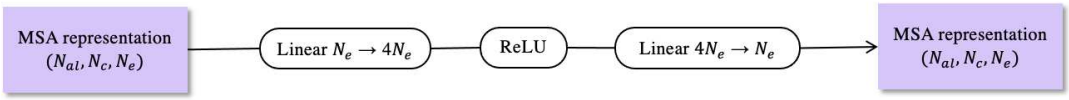



**Figure 3.5:** MSA transition

Transition layer is present both in MSA and pair stack, as demonstrated in Figure 3.5. It includes two linear projections and one activation using Rectified Linear Unit (ReLU).

### 3.2.4. Outer product mean

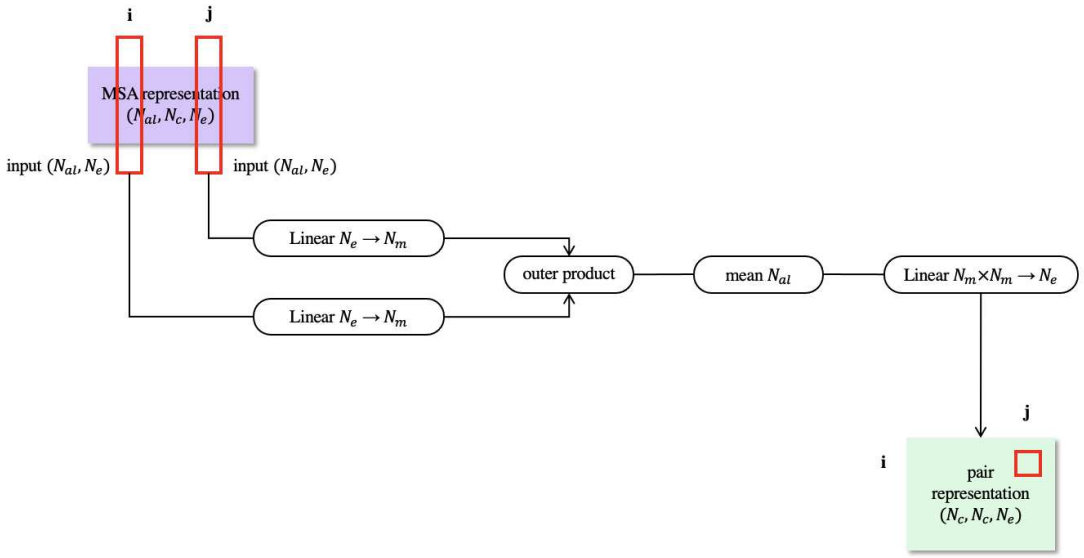



**Figure 3.6:** Outer product mean

Figure 3.6 demonstrates Outer product mean where MSA feature performs outer product by itself. It is followed by averaging values by alignments and finishes with linear projection to update pair representation.

### 3.2.5. Triangular update

By interpreting every position in a sequence as a graph node and a pairing probability as graph edge, this operation updates the edges of a graph based on the information from the other two edges from every possible triangle.

The model differentiates two symmetric versions of this update, with the only difference being whether edges are interpreted as incoming or outgoing edges. For instance, when updating edge $ij$ using the outgoing edges version, edges $ik$ and $jk$ will be observed from $ijk$ triangle. On the other hand, edges $ki$ and $kj$ will be considered when using the incoming edges version.

The process flow is shown in Figure 3.7.

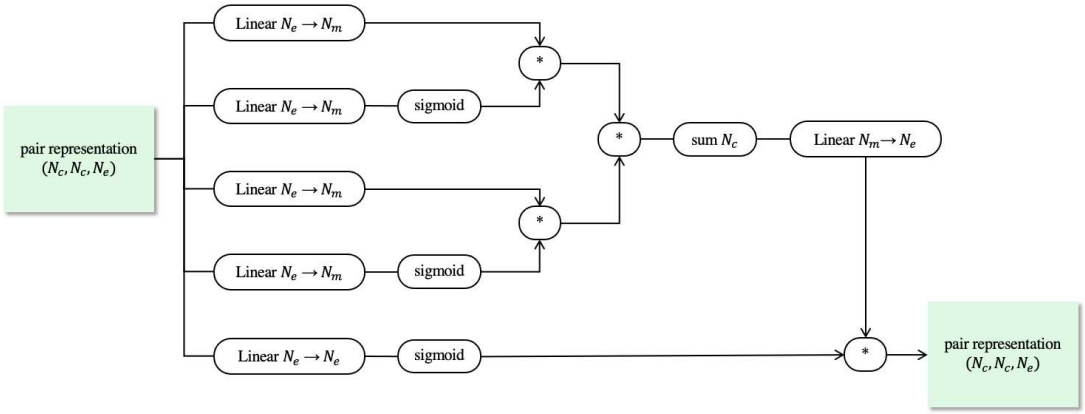



**Figure 3.7:** Triangular update using outgoing/incoming edges

## 3.2.6. Triangular self-attention

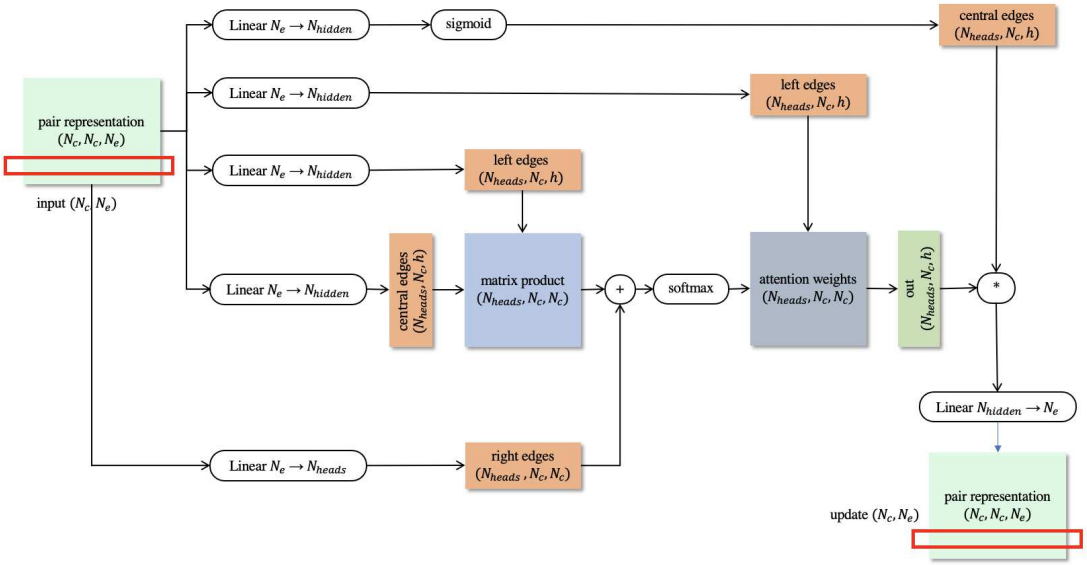



**Figure 3.8:** Triangular self-attention around starting node

Transformation illustrated in Figure 3.8 also updates pair representation by interpreting the prediction problem as a graph prediction problem.

Depending on whether starting or ending node is observed, different graph edges are taken into account.

Edge $ij$ is updated based on the information obtained from all other edges with the same starting node $i$. Analogously, in case of triangular attention with an ending node, edge $ij$ is modified using edges ending with the same node $j$.

## 3.3. Structure module

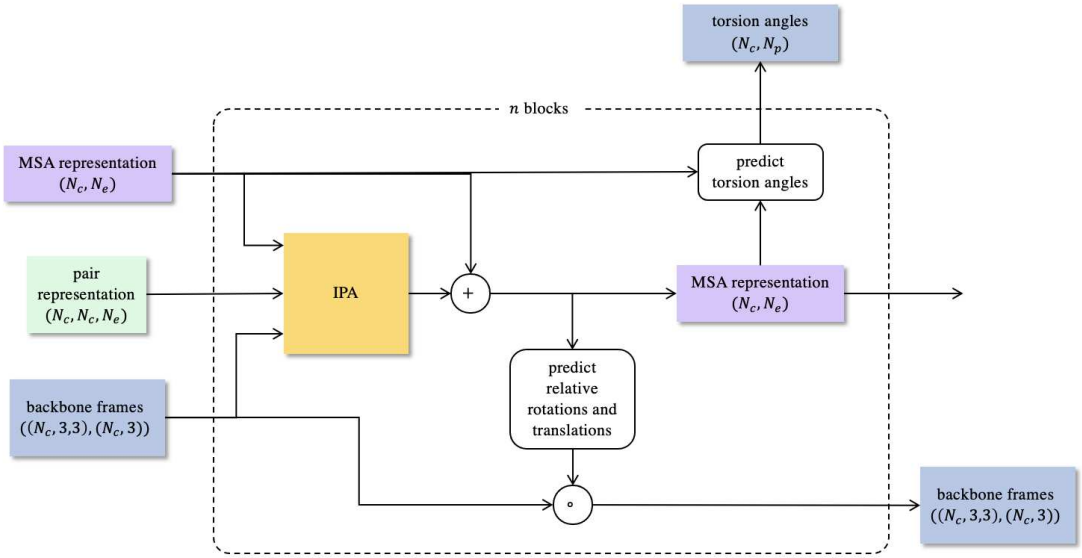



**Figure 3.9:** Structure module

The Structure module receives pair representation and a single MSA representation corresponding to the original sequence.

In contrast to Evoformer, the Structure module uses multiple blocks both with shared architecture and shared weights.

Firstly, *frames* are initialised and delegated to *Invariant Point Attention Module* (*IPA*) input along with single MSA and pair representation to update MSA representation.

Term frame is used within the model to illustrate affine transformation for every position in a sequence, i.e. translation and rotation. A single translation is defined as a three-dimensional vector representing translations for every dimension - $x$, $y$ and $z$, while rotation is determined through a $3 \times 3$ rotation matrix.

Once IPA output is calculated, multiple operations are applied to update MSA. Such MSA is used for predicting torsion angles and relative rotations and translations, i.e. relative frames.

Finally, backbone frames are updated by applying composition between current and relative backbone frames.

## 3.4. Losses

A total loss that the network aims to minimize constitutes of the following components:

1. **distance loss**: calculates the cross-entropy loss of predicting a distance bin
2. **mask loss**: calculates the cross-entropy loss of predicting a mask
3. **total frame-aligned point error (FAPE) loss**: compares predicted atom positions to actual positions
4. auxiliary losses: losses by every structure module iteration
   (a) **FAPE loss**
   (b) **torsions loss**: measures the difference between predicted torsion angles to true torsion angles
   (c) **sugar puckering loss**: calculates the mean squared error of predicting sugar puckering

## 3.5. Analysis modifications

Training such a powerful but complex network requires exceptional computer power, and even with such resources, single training takes a tremendous amount of time.

Due to the fact that this analysis' focus is the possibility of Evoformer's improvement, the initial version has been slightly modified. Namely, there is no need for the Structure module because other network components do not affect the metrics of interest. Consequently, the total loss that the optimisation process aims to minimise consists of distance loss and mask loss.

In addition, the goal is to compare different Evoformer configurations using similar, if not the same, hyperparameters. Therefore, there is no need for hyperparameter values that add to the model's complexity and training time.

In the continuation of this chapter, all tested versions will be described and illustrated. Novelties shall be marked red to demonstrate which modifications have been introduced compared to the original configuration.

## v1

Firstly, the original Evoformer configuration will be tested for a starting point. No modifications are introduced in this version. Used Evoformer architecture is illustrated in Figure 3.2.

## v2

Evoformer's second version, shown in Figure 3.10, introduces a convolutional layer instead of the MSA column gated attention layer.

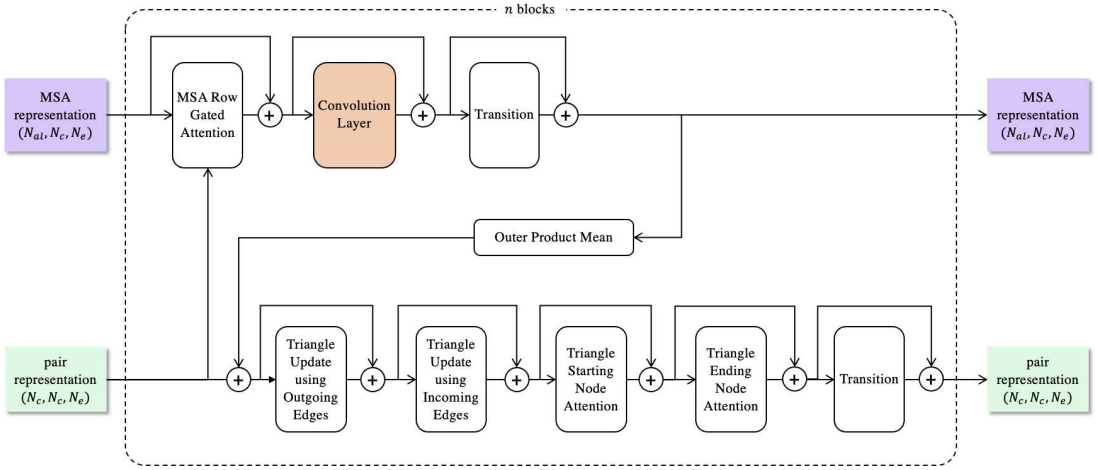



**Figure 3.10:** Evoformer v2

## v3

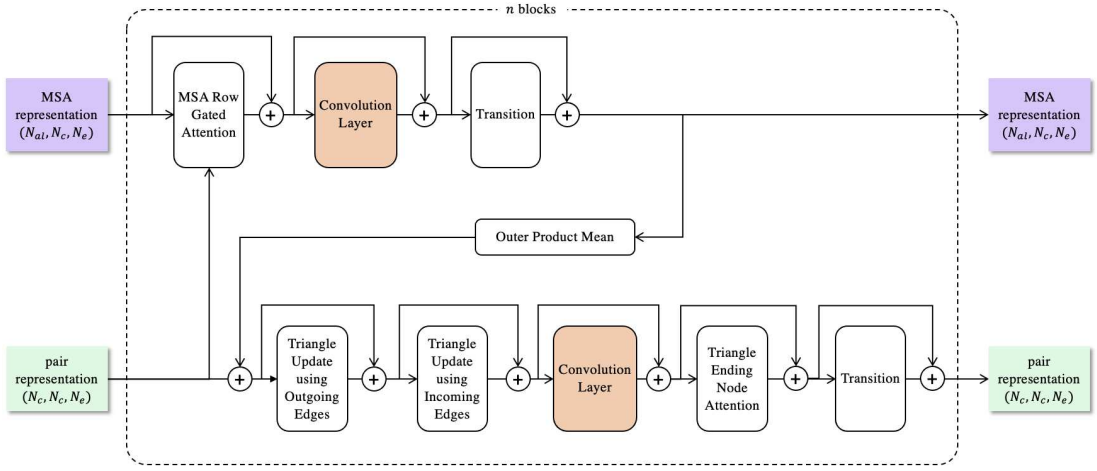



**Figure 3.11:** Evoformer v3

Figure 3.11 demonstrates the Evoformer version that uses two convolutional layers instead of the MSA column gated attention layer and the triangular starting node attention layer.

## v4

The convolutional layer replacing the triangular starting node attention layer is present in the Evoformer's fourth version, as shown in Figure 3.12.

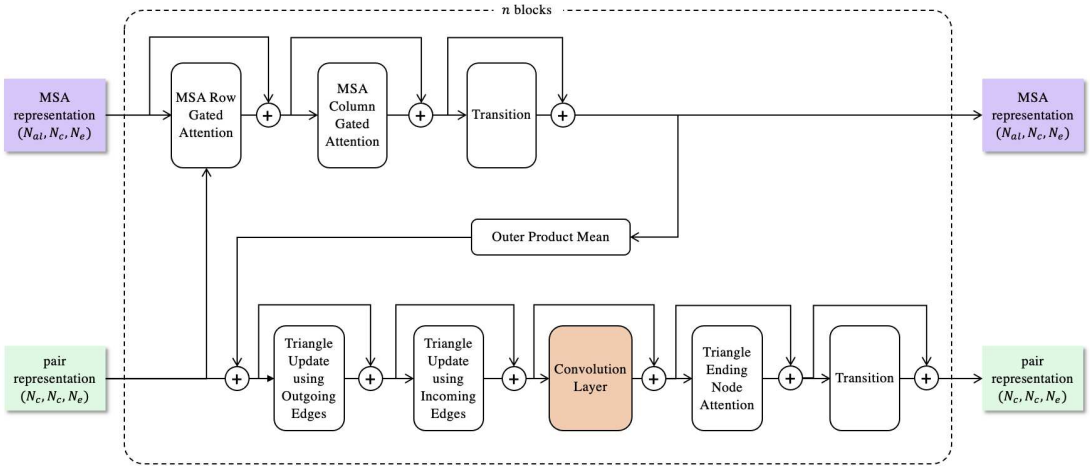



**Figure 3.12:** Evoformer v4

## v5

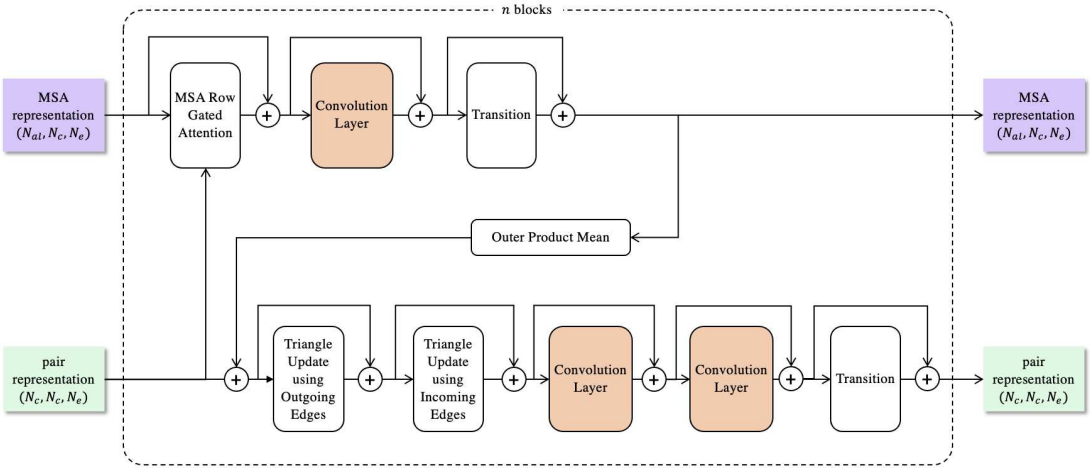



**Figure 3.13:** Evoformer v5

Usage of convolutional layer instead of the MSA column gated attention layer,

triangular starting node attention layer and triangular ending node attention layer is illustrated in Figure 3.13.

## v6

Figure 3.14 corresponds to Evoformer's sixth version, where both the triangular starting node attention layer and the triangular ending node attention layer are substituted with the convolutional layer.

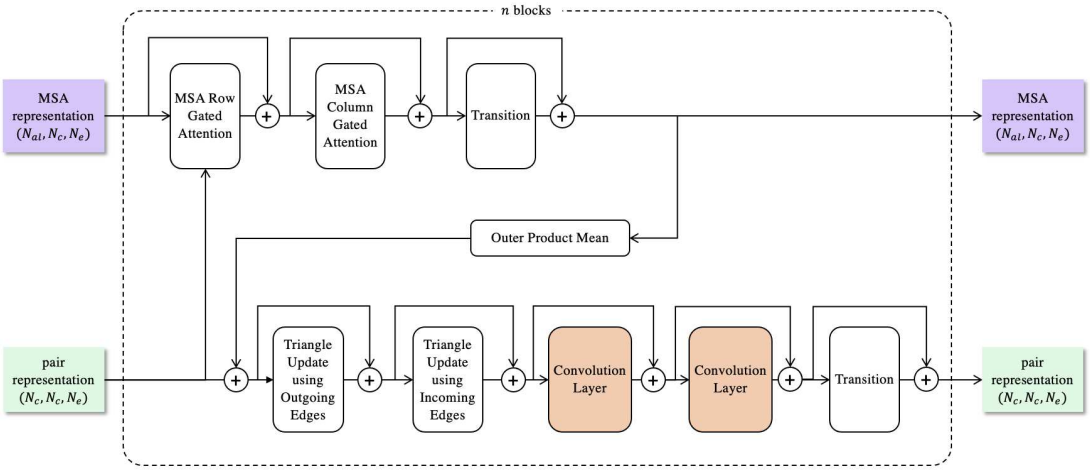



**Figure 3.14:** Evoformer v6

## v7

The MSA column gated attention layer and the triangular ending node attention layer are replaced with a convolutional layer in the Evoformer's seventh version, as demonstrated in Figure 3.15.

## v8

The eighth version, shown in Figure 3.16, is characterised by substituting the triangular ending node attention layer with a convolutional layer.

## v9

RoseTTAFold's[1] Pair2MSA layer is included in the ninth Evoformer version to replace the MSA row gated attention layer, as presented in Figure 3.17.

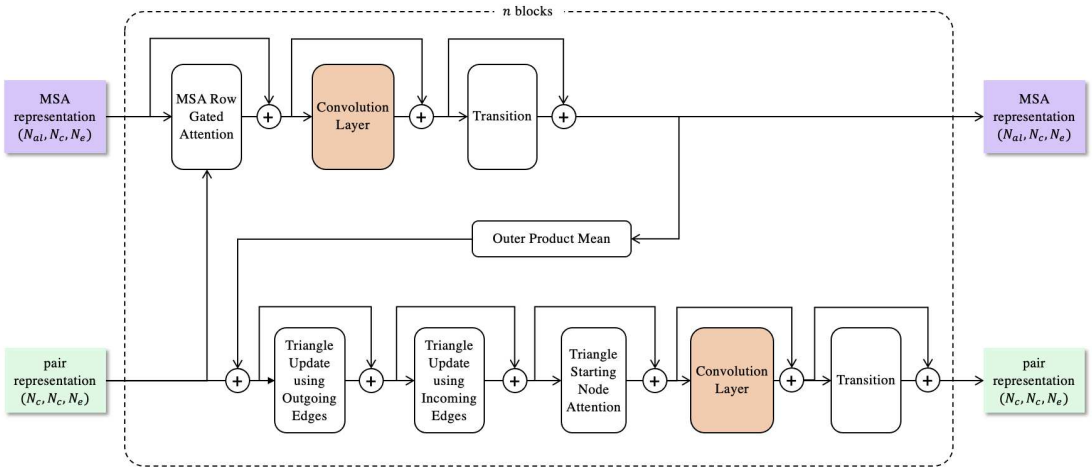



**Figure 3.15:** Evoformer v7

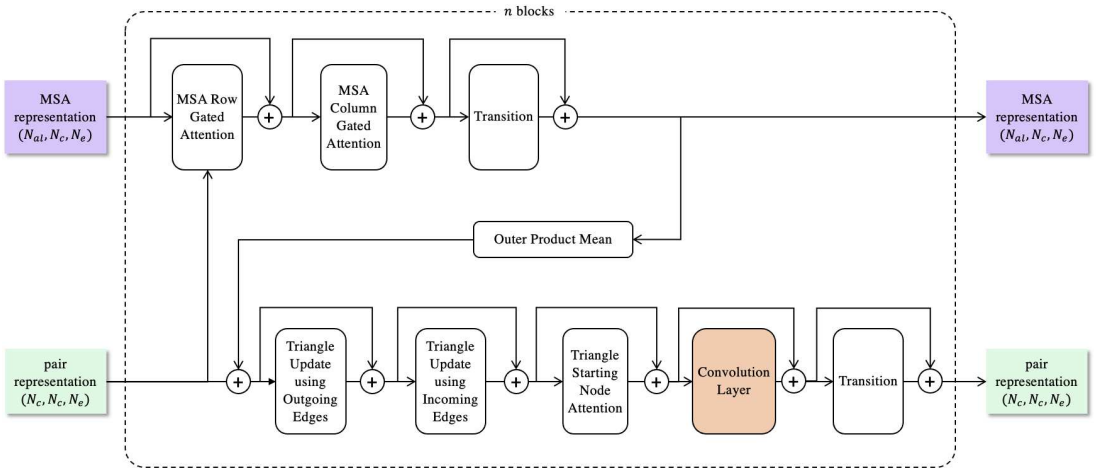



**Figure 3.16:** Evoformer v8

## v10

A combination of the Pair2MSA layer substituting the MSA row gated attention layer and a convolutional layer replacing the MSA column gated attention layer specifies the tenth version shown in Figure 3.18.

## v11

Figure 3.19 illustrates Evoformer's eleventh version. Again, the Pair2MSA layer substitutes the MSA row gated attention layer, while convolutional layers replace the MSA column gated attention layer and the triangular starting node attention layer.

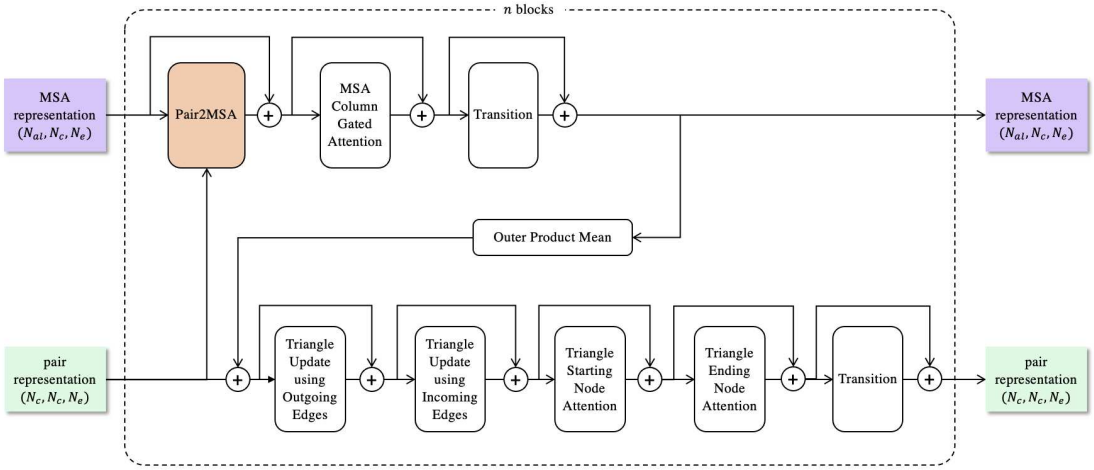



**Figure 3.17:** Evoformer v9

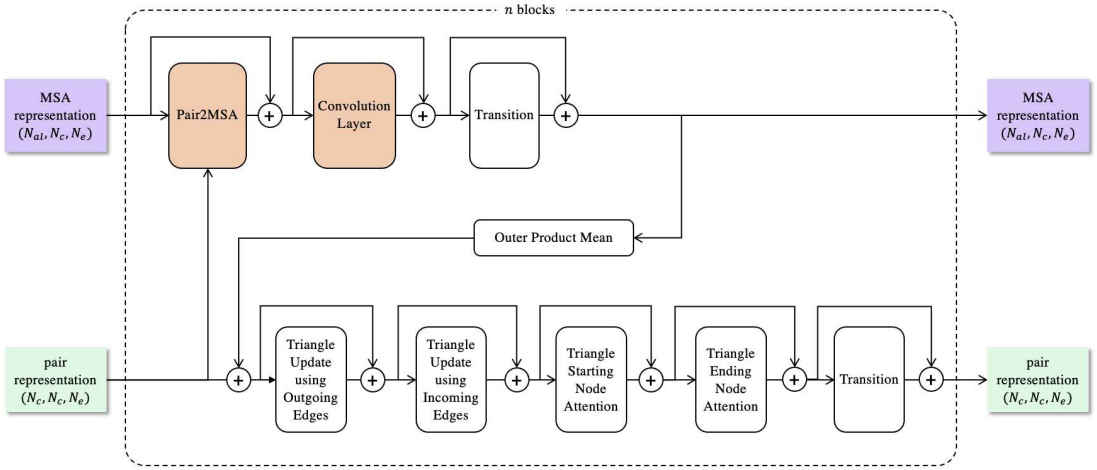



**Figure 3.18:** Evoformer v10

## v12

The MSA row gated attention layer is replaced with the Pair2MSA layer and the triangular starting node attention layer with a convolutional layer in this Evoformer's version, as illustrated in Figure 3.20.

## v13

Evoformer's 13th version, as shown in Figure 3.21, introduces a convolutional layer instead of the MSA column gated attention layer, triangular starting node attention layer, triangular ending node attention layer, and the Pair2MSA instead of the MSA row gated attention layer.

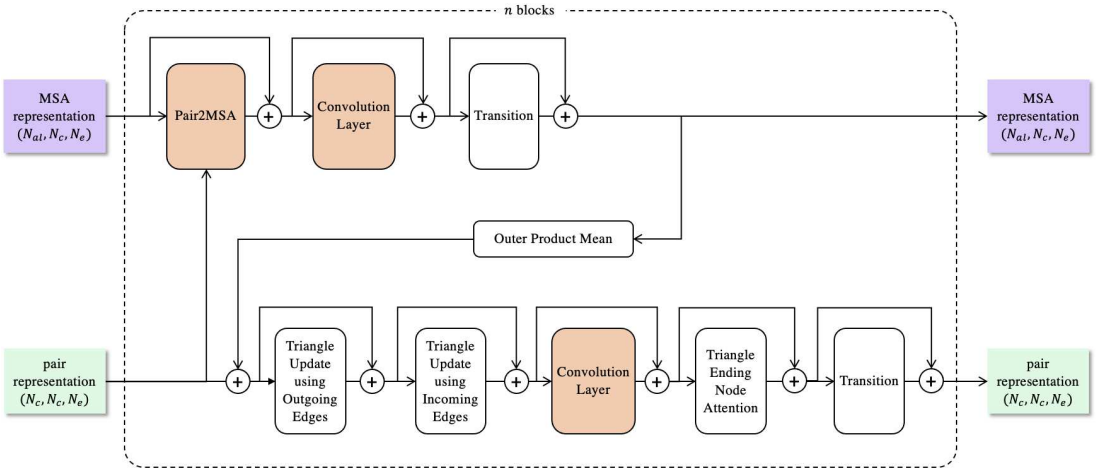



**Figure 3.19:** Evoformer v11

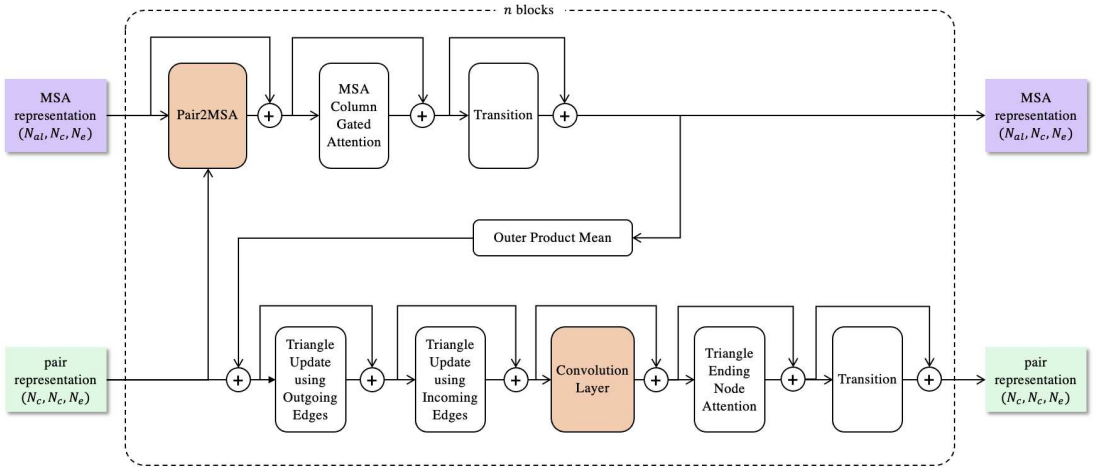



**Figure 3.20:** Evoformer v12

## v14

Usage of convolutional layers instead of the triangular starting node attention layer and the triangular ending node attention layer, and the Pair2MSA layer instead of the MSA row gated attention layer is present in Evoformer's 14th version, as demonstrated in Figure 3.22.

## v15

Evoformer's 15th version, as in Figure 3.23, is specific for replacing the MSA column gated attention layer and the triangular ending node attention layer with a convolutional layer, and the MSA row gated attention layer with the Pair2MSA layer.

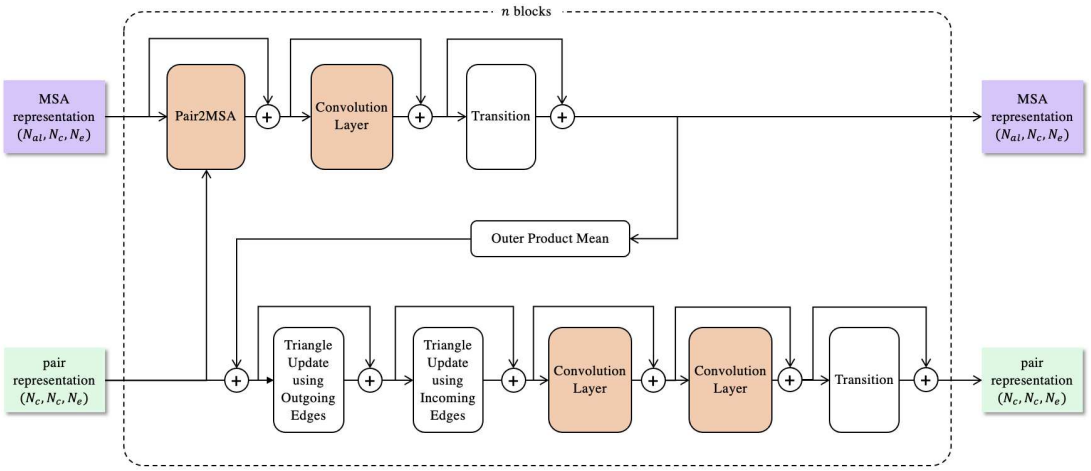



**Figure 3.21:** Evoformer v13

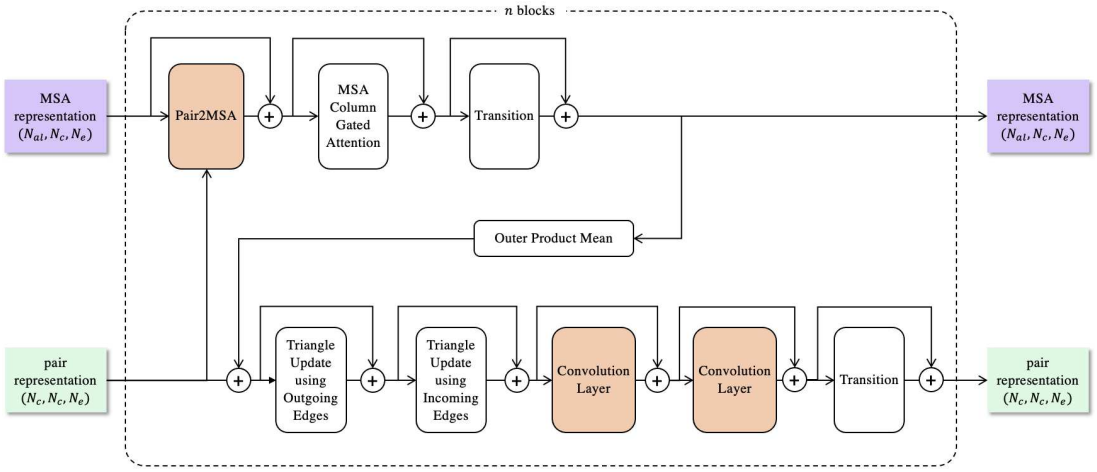



**Figure 3.22:** Evoformer v14

## v16

Finally, Figure 3.24 demonstrates the final analysis version in which the Pair2MSA layer substitutes the MSA row gated attention layer and a convolutional layer replaces the triangular ending node attention layer.

### 3.5.1. Pair2MSA

Bearing in mind that the goal is to replace all attention-based layers with convolutional layers, the problem arises with the MSA row gated attention layer due to the fact that it expects both pair and MSA representations on input.

Instead of introducing even more complexity by answering how to combine these two inputs into one, RoseTTAFold's layer Pair2MSA will be included in the analysis

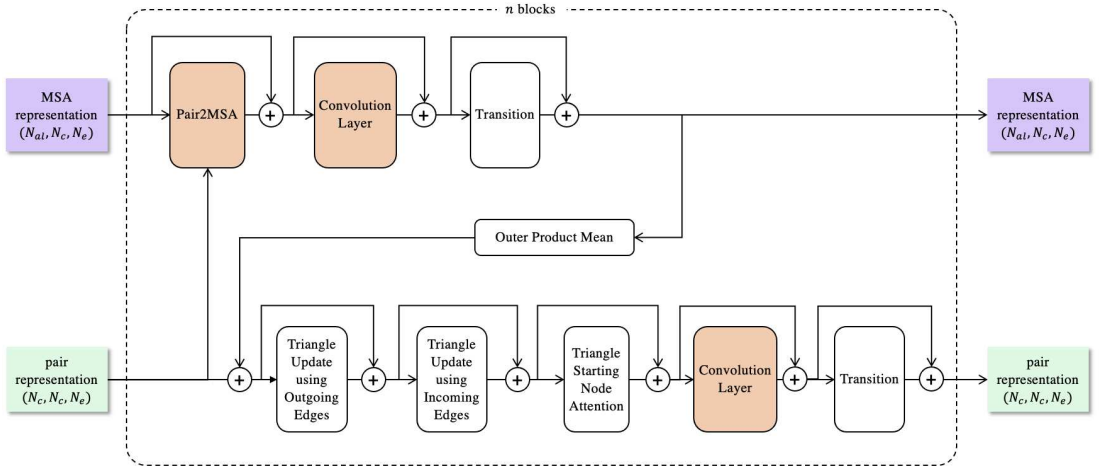



**Figure 3.23:** Evoformer v15

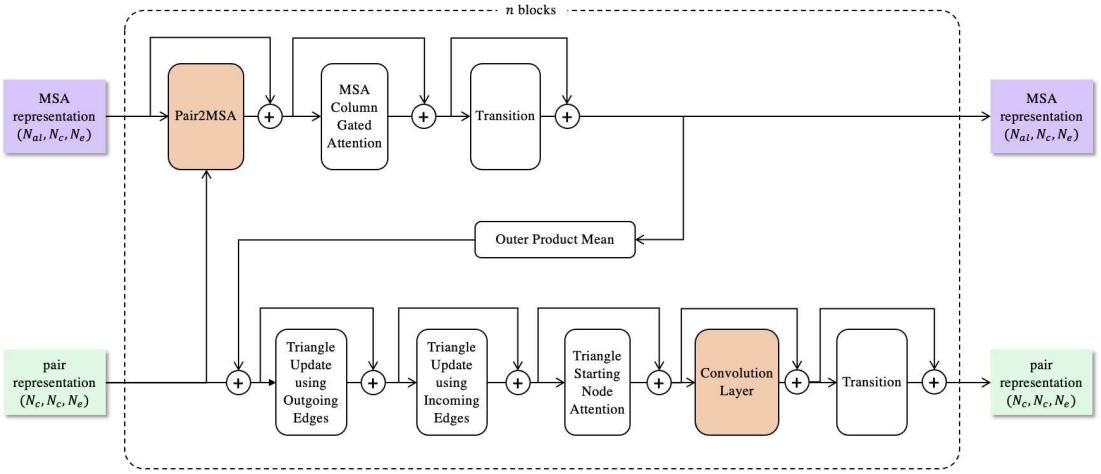



**Figure 3.24:** Evoformer v16

to replace the MSA row gated attention.

Pair2MSA's flow is demonstrated in Figure 3.25. As with the MSA row gated attention layer, it updates MSA representation using pair and MSA representations.

Operator `symmetrize` calculates the outer sum of pair representation and then divides it by two to average it. The process is followed by Direct attention layer and Feed forward layer.

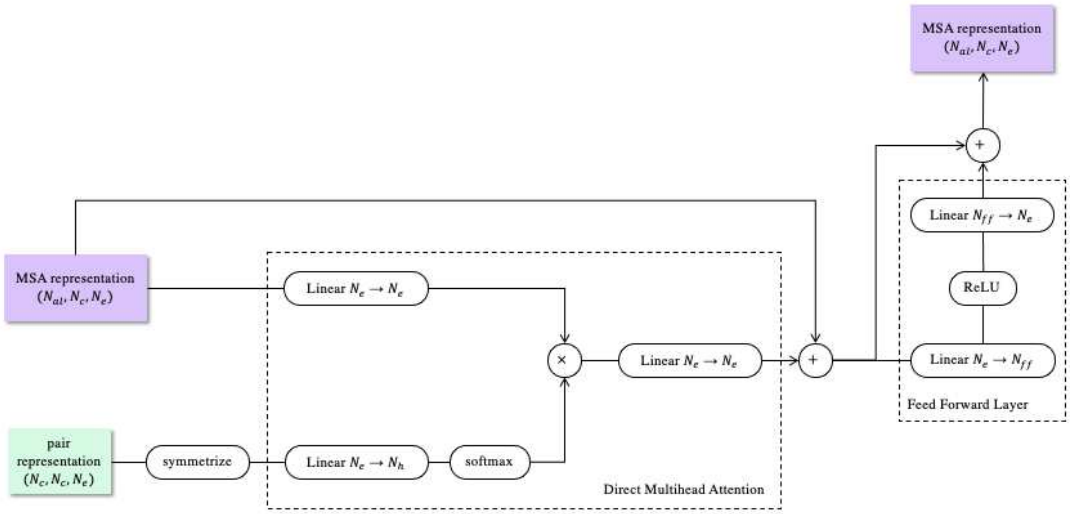



**Figure 3.25:** RoseTTAFold's Pair2MSA layer

# 4. Implementation

The RNAFold is entirely implemented using Python programming language in addition to numerous Python libraries, with the most important being:

- `PyTorch` - used for building a deep learning network and working with tensors
- `PyTorch-Lightning` - used for a more scientific approach when working with deep learning models
- `Matplotlib` - used for plotting distograms

Because the model's implementation is still in progress, the code implementation required for this analysis is in the forked GitHub repository with the following address `https://github.com/jelena54321/RNAFold`. That way, the forked model can easily be updated with the newest version from the upstream branch and modified without fear of interfering. On the other hand, the original model's address is `https://github.com/RJPenic/RNAFold`.

All used Evoformer versions are placed in the `RNAFold/rnafold/layers/evoformer-versions` folder. In addition, the code implementation for the Pair2MSA layer can be found in the `RNAFold/rnafold/layers/attentions.py` file.

A Python script named `RNAFold/rnafold/visualizations/distograms_plotting.py` was used and modified to plot distograms and show training results.

The most important used hyperparameters are listed in Table 4.1. Others were not modified and can be found on the previously mentioned GitHub repository.

**Table 4.1:** Hyperparameters

| name | description | value |
|---|---|---|
| `msa_embedding_dim` | MSA features embedding size | 128 |
| `pair_embedding_dim` | pair features embedding size | 128 |
| `num_btypes` | size of a vector representing a single position in input sequence | 7 |

| | | |
|---|---|---|
| num_msa_tokens | size of a vector representing a single position in alignment | 8 |
| heads | heads size | 8 |
| evoformer_blocks | number of Evoformer blocks | 4 |
| num_ensemble | number of iterary refinements | 1 |
| bins_high_val_start | upper boundary of the first bin | 7 |
| num_bins | number of bins | 16 |
| bin_size | bin size | 3 |
| mask_prob | masking probability | 0.04 |
| rand_swap_prob | probability of a masked token being masked with special token | 0.15 |
| spcl_tkn_prob | probability of a masked token being replaced with a random base | 0.7 |
| batch_size | batch size | 16 |
| max_epochs | maximal number of epochs during training | 20 |
| gradient_clip_val | the value at which to clip gradients | 0.1 |
| accelerator | trainer accelerator type | ddp |
| lr | learning rate | 0.001 |
| TRAIN_CROP_SIZE | random crop window size | 32 |
| MODEL_NR_MSA_ALIGNMENTS | number of alignments in MSA feature | 16 |
| mid_projection_dim | hidden layer size | 32 |
| conv1_channels | number of input and output channels in the first convolutional layer | 17 |

| | | |
|---|---|---|
| conv2_channels | number of input and output channels in the second convolutional layer | 32 |
| conv_kernel | size of a convolution kernel | (5,13) |
| conv_padding | convolutional layer padding | (2, 6) |

# 5. Results

For a fair model version comparison, all versions have been trained using the same hyperparameter values. The following metrics were observed for the analysis to provide information concerning the model's performance and the expense of required resources for training:

- training time
- memory necessary for storing trainable parameters
- distance loss
- distance accuracy

Among data that Evoformer outputs, the point of interest is the pair distance distribution. Instead of predicting the exact distance between every pair $(i, j)$, the model predicts the distance bin that the pair belongs to. The number and size of distance bins are predefined and are equal to `num_bins` and `bin_size`, respectively.

As previously mentioned, the distance loss is calculated as a cross-entropy loss between the pair distance distribution and the target bins. On the other hand, the distance accuracy informs how many pairs are mapped to an accurate distance bin, given that the predicted bin is the one with the highest probability.

Table 5.1 refers to the time it required every model version to finish all epochs of training. According to the results, the original version has the shortest, and the third version has the longest training time. Bolded values indicate the edge cases. In other words, the shortest and the longest training times.

To demonstrate memory usage by model version, Table 5.2 indicates the exact number of parameters stored in memory during model execution and the total estimated parameters size in MB. Interestingly, the model's fifth version keeps the smallest number of parameters, and the ninth version has the biggest number and equals 14.4 MB.

**Table 5.1:** Training times

| version | training time [s] |
|---|---|
| v1 | **4 575** |
| v2 | 5 147 |
| v3 | **6 693** |
| v4 | 5 232 |
| v5 | 6 209 |
| v6 | 5 747 |
| v7 | 5 690 |
| v8 | 5 215 |
| v9 | 4 631 |
| v10 | 5 210 |
| v11 | 5 775 |
| v12 | 5 300 |
| v13 | 6 283 |
| v14 | 5 815 |
| v15 | 5 762 |
| v16 | 5 284 |

**Table 5.2:** Parameters

| version | total number of parameters [M] | total estimated parameters size [MB] |
|---|---|---|
| v1 | 3.3 | 13.053 |
| v2 | 3.0 | 12.035 |
| v3 | 2.9 | 11.765 |
| v4 | 3.2 | 12.783 |
| v5 | **2.9** | **11.495** |
| v6 | 3.1 | 12.513 |
| v7 | 2.9 | 11.765 |
| v8 | 3.2 | 12.783 |
| v9 | **3.6** | **14.378** |
| v10 | 3.3 | 13.360 |
| v11 | 3.3 | 13.090 |
| v12 | 3.5 | 14.109 |
| v13 | 3.2 | 12.820 |
| v14 | 3.5 | 13.839 |
| v15 | 3.3 | 13.090 |
| v16 | 3.5 | 14.109 |

## 5.1. Test dataset results

For the sake of getting a general overview of the model's performance, distance loss and distance accuracy were calculated on the test dataset of 27 RNA examples.

Table 5.3 lists distance loss and distance accuracy for every model version. The most significant loss and the poorest accuracy score correspond to model v6. That is, to the model in which the convolutional layers replace triangular self-attention layers.

Interestingly enough, model v2 has demonstrated remarkable performance due to the lowest distance loss and one of the highest accuracy scores. Moreover, the model with the highest distance accuracy is also the one that, like model v2, substitutes the MSA column gated attention with the convolutional layer.

**Table 5.3:** Test results for test dataset

| version | distance loss | distance accuracy |
|---|---|---|
| v1 | 2.02918 | 0.31735 |
| v2 | **1.93663** | 0.36039 |
| v3 | 2.23006 | 0.33977 |
| v4 | 2.36739 | 0.31706 |
| v5 | 2.36208 | 0.31771 |
| v6 | **2.44934** | **0.29210** |
| v7 | 2.20778 | 0.34021 |
| v8 | 2.13482 | 0.34339 |
| v9 | 2.04905 | 0.32436 |
| v10 | 2.03223 | **0.36661** |
| v11 | 2.11000 | 0.35576 |
| v12 | 2.30562 | 0.30664 |
| v13 | 2.38193 | 0.31916 |
| v14 | 2.36760 | 0.30042 |
| v15 | 2.14414 | 0.35315 |
| v16 | 2.28534 | 0.32791 |

The two following sections will cover models performances on two RNA examples: 6TB7_1#16-48 and 6POM_2#0-32. The idea is to show one example (6TB7_1#16-48) that should be easier for the model to predict and the other (6POM_2#0-32) expected to result in weaker predictions.

## 5.2. Results for RNA example 6TB7_1#16-48

Table 5.4 shows a list of distance losses and accuracies for RNA 6TB7_1#16-48 example by model versions. Even before taking a more detailed look at Table 5.4, it is

immediately visible that obtained results are far better than those in Table 5.3.

Consistent with Table 5.3, model v6 again performed the poorest with both the most significant loss and the lowest accuracy. However, the model that performed the best due to the lowest distance loss and highest accuracy is model v16.

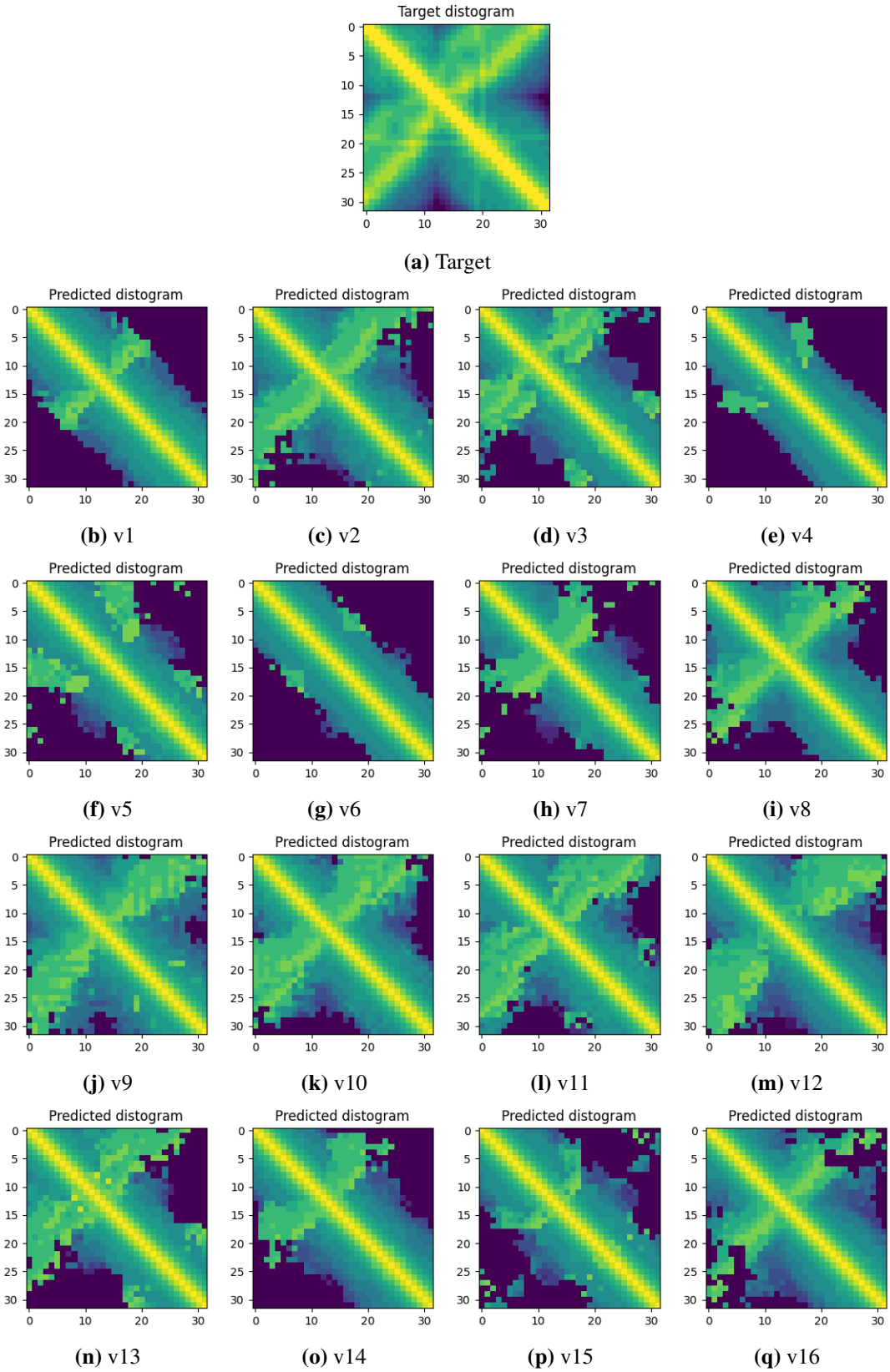
**Table 5.4:** Test results for RNA example 6TB7_1#16-48

| version | distance loss | distance accuracy |
|---|---|---|
| v1 | 1.77925 | 0.37109 |
| v2 | 1.45128 | 0.48828 |
| v3 | 1.78845 | 0.40234 |
| v4 | 1.80632 | 0.36523 |
| v5 | 1.73819 | 0.38281 |
| v6 | **1.93929** | **0.35156** |
| v7 | 1.71086 | 0.38867 |
| v8 | 1.60905 | 0.44922 |
| v9 | 1.56121 | 0.48242 |
| v10 | 1.47216 | 0.47656 |
| v11 | 1.53960 | 0.46875 |
| v12 | 1.47882 | 0.48242 |
| v13 | 1.68781 | 0.42578 |
| v14 | 1.87696 | 0.38672 |
| v15 | 1.54877 | 0.44531 |
| v16 | **1.40275** | **0.51367** |

To provide a visual representation of distance accuracies by models, so-called *distograms* will be used. Distogram is a plot illustrating a $N_c \times N_c$ matrix where every pixel $(x, y)$ shows the model's bin prediction for a pair of positions $x$ and $y$. A single pixel indicates to which distance bin belongs the distance of $y$ from $x$ according to the model's distance distribution. The closest pairs are shown in lighter colours, while darker tones present the farthest pairs.

Distograms for RNA example 6TB7_1#16-48 are illustrated in Figure 5.1. A target distogram is presented with Figure 5.1a, while models outputs are shown in other 16 distograms.

In accordance with distance accuracy values from Table 5.4, models v1, v4 and v6 show the weakest performance and are able to predict well only near the main diagonal. Similarly, models v2, v12 and v16 show indications of good performance.




**(a)** Target **(b)** v1 **(c)** v2 **(d)** v3 **(e)** v4 **(f)** v5 **(g)** v6 **(h)** v7 **(i)** v8 **(j)** v9 **(k)** v10 **(l)** v11 **(m)** v12 **(n)** v13 **(o)** v14 **(p)** v15 **(q)** v16

**Figure 5.1:** Target and prediction distograms by model versions for RNA example with ID 6TB7_1#16-48

## 5.3. Results for RNA example 6POM_2#0-32

Test results for the RNA example are recorded in Table 5.5.

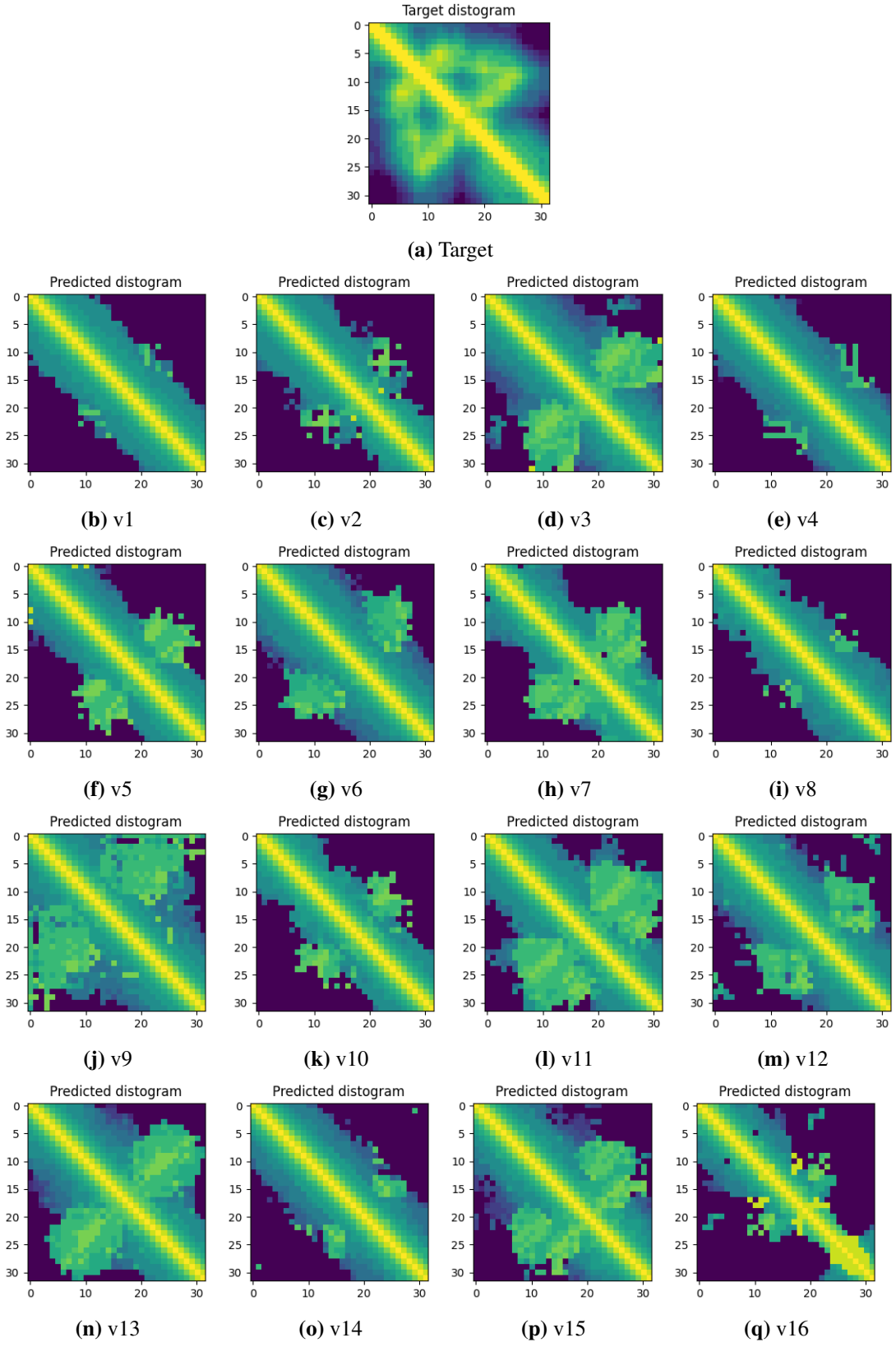
Compared to the previous RNA example, models performances are not as consistent, and consequently, the worst loss and accuracy values do not belong to the same models. In addition, results are weaker due to the more complex nature of the target distogram.

Although all models performed relatively poorly, model v13 has the highest accuracy. What is more, model v6 surprisingly ended with the smallest loss value.

Based on the prediction distogram of model v16, one cannot be surprised that the same model has the highest loss and one of the lowest accuracy scores.

**Table 5.5:** Test results for RNA example 6POM_2#0-32

| version | distance loss | distance accuracy |
|---|---|---|
| v1 | 1.95733 | 0.32617 |
| v2 | 2.10344 | 0.32031 |
| v3 | 1.98009 | 0.36133 |
| v4 | 2.10383 | 0.32813 |
| v5 | 2.10186 | 0.32227 |
| v6 | **1.89568** | 0.34961 |
| v7 | 1.98326 | 0.33789 |
| v8 | 2.06508 | 0.34180 |
| v9 | 1.90857 | 0.33398 |
| v10 | 1.97941 | 0.35742 |
| v11 | 1.93878 | 0.36914 |
| v12 | 1.99913 | 0.32227 |
| v13 | 1.95204 | **0.37891** |
| v14 | 2.00158 | **0.30664** |
| v15 | 2.00041 | 0.37109 |
| v16 | **2.10572** | 0.30859 |




**(a)** Target

**(b)** v1 **(c)** v2 **(d)** v3 **(e)** v4

**(f)** v5 **(g)** v6 **(h)** v7 **(i)** v8

**(j)** v9 **(k)** v10 **(l)** v11 **(m)** v12

**(n)** v13 **(o)** v14 **(p)** v15 **(q)** v16

**Figure 5.2:** Target and prediction distograms by model versions for RNA example with ID 6POM_2#0-32

# 6. Discussion

Bearing the results of the test dataset in mind, several observations concerning Evoformer configurations can be made.

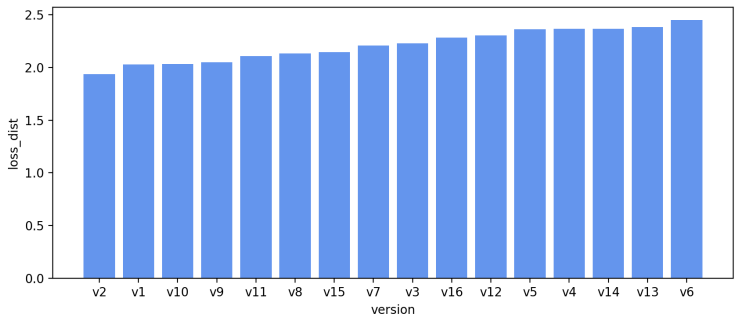



**Figure 6.1:** Versions sorted by distance loss

Firstly, it is interesting to notice that the smallest loss does not correspond to the original version but rather the one where the MSA column gated attention layer is replaced with a convolutional layer.

In addition, if one looks at the architecture of the best five models according to the distance loss, a particular pattern can be noticed. All of those versions - v2, v1, v10, v9 and v11 - have the triangular ending node attention layer. That is, it is not replaced with a convolutional layer. What is more, four out of five have the triangular starting node attention layer.

Now focusing on the distance accuracy, it is worth mentioning that all five models with the worst distance accuracy - v6, v14, v12, v4 and v1 - have the MSA column gated attention, i.e. the substitution of the MSA column gated attention layer with convolutional layer is not present. On the other hand, four out of five best models have a convolutional layer in that position.

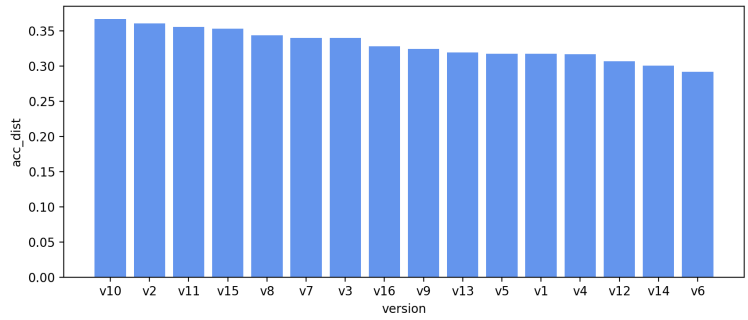



**Figure 6.2:** Versions sorted by distance accuracy

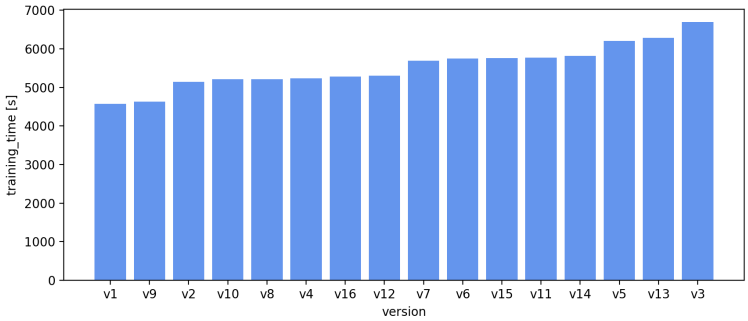



**Figure 6.3:** Versions sorted by training time

When comparing models based on training time, one can remark that the shortest training times belong to the models with a smaller number of substitutions compared to the original model. Analogously, versions with the longest training time - v11, v14, v5, v13 and v3 - have two or more replacements.

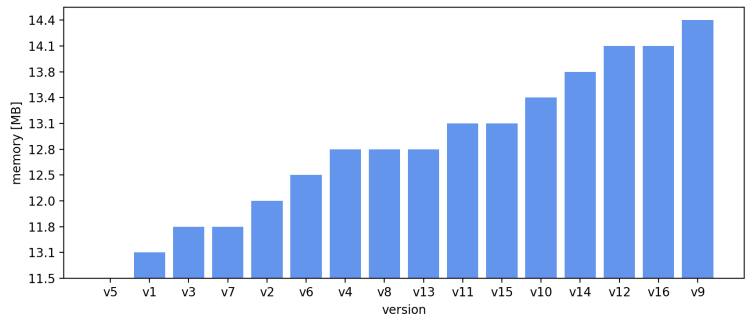



**Figure 6.4:** Versions sorted by memory usage

Model versions v5, v1, v3, v7 and v2, which have the smallest memory usage, all have one thing in common - there is no RoseTTAFold's Pair2MSA layer. Furthermore, out of five models, four have a convolutional layer instead of the MSA column gated

layer. Likewise, the models with the most stored parameters have a Pair2MSA replacing the MSA row gated attention layer, while the majority has the MSA column gated attention layer.

As expected, similar patterns can be noticed when comparing results on the test dataset to the results for RNA example 6TB7_1#16-48. Models v2 and v10 show promising results in both cases with one of the lowest distance losses. Additionally, models v6, v14 and v13 score in the bottom three.

Models v2 and v10 share the same Evoformer architecture, with the only exception being the presence of the Pair2MSA layer. On the other hand, all three models, v6, v14 and v13, have convolutional layers instead of triangular self-attention layers.

When comparing the first eight and last eight versions in Figure 5.1, one can notice that the first half performs weaklier farther from the main diagonal and has relatively smaller accuracy.

Although all models performed poorly with RNA example 6POM_2#0-32, the models with the best accuracy, v13, v15 and v11, have Pair2MSA layer and a convolutional layer replacing the MSA column gated attention layer.

Since this RNA example is far more complex than the previous one, there are fewer shared patterns between results for this example and the whole test dataset.

# 7. Conclusion

To begin with, it is evident that all configurations that include the replacement of the triangular ending node attention layer and the triangular starting node attention layer with convolutional layers have one of the worst performances based on the distance loss metric.

Furthermore, examples of promising candidates to replace the original version v1 would be models v2 and v10 due to their high distance accuracy and low distance loss. In addition, both of them are one of the models with the shortest training time.

Another conclusion worth mentioning would be that number of substitutions is somewhat proportional to training time. That is, three or four modifications would immediately implicate a longer training process.

Even though using the Pair2MSA layer instead of the MSA row gated attention layer has resulted in the distograms most similar to the target distograms, its application uses a noticeably larger amount of memory.

Some models have achieved more success than others when comparing their distograms to the target distograms, but all of them still have too sharp transitions and weaker performances near the edges.

RNA example 6TB7_1#16-48 demonstrated how models perform generally, but RNA example 6POM_2#0-32 was considerably more complex, and consequently, even the most promising versions fell short.

Distograms have shown to be an excellent visualisation for distance accuracy. Therefore, their usage in this kind of problem is a great model performance indicator and a key to solving performance issues.

Although this analysis has given some significant deductions in the model behaviour, there are still ways this exploration and model performance could be improved in the future.

For instance, new Evoformer versions could be easily introduced by including another replacement that isn't a convolutional layer, such as another RoseTTAFold's layer. One could also observe performances in batches where every batch is defined

with a new hyperparameter combination. Furthermore, using more powerful computational resources would implicate a more complex and faster environment and make this analysis even better.

All in all, this analysis has shown to be very beneficial to understand the model behaviour better and discover that there actually may be less complex versions with better performance overall.

# BIBLIOGRAPHY

[1] Baek et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021. doi: 10.1126/science.abj8754. URL https://www.science.org/doi/abs/10.1126/science.abj8754.

[2] Jumper et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2. URL https://doi.org/10.1038/s41586-021-03819-2.

[3] Townshend et al. Geometric deep learning of rna structure. *Science*, 373 (6558):1047–1051, 2021. doi: 10.1126/science.abe5650. URL https://www.science.org/doi/abs/10.1126/science.abe5650.

**Evaluation of RNA Atom Distance Prediction Models**


**Abstract**

RNA molecules can fold and form connections with themselves. Since such transformations lead to various possible three-dimensional structures and a diversity of corresponding functions, the knowledge of such structures is of great importance.

The RNAFold model predicts a three-dimensional RNA structure using deep learning methods and practices that have shown remarkable progress in related problems.

This thesis aims to analyse different model configurations to get an overview and a direction in the first stages of RNAFold's implementation. By observing multiple performance metrics, the objective is to find a correlation between model architecture and performance and determine which factors may improve current model implementation.



**Keywords:** RNA,folding,model,deep-learning,analysis


**Vrednovanje modela za predikciju udaljenosti između RNA atoma**


**Sažetak**

RNA molekule mogu se savijati i uspostavljati veze same sa sobom. S obzirom na to da takve transformacije kao posljedicu daju bogati spektar trodimenzionalnih struktura i funkcija, znanje o takvim strukturama od velike je važnosti.

RNAFold model daje predikciju o trodimenzionalnoj strukturi koristeći metode dubokih neuronskih mreža i prakse koje su pokazale izvanredan napredak u srodnim problemima.

Ovaj diplomski rad nastoji analizirati različite konfiguracije modela kako bi se dobio pregled i smjer u prvim fazama implementacije modela. Promatrajući više metrika performanse, cilj je pronaći povezanosti između arhitekture i performanse modela kako bi se ustanovili faktori koji bi mogli unaprijediti trenutnu implementaciju.



**Ključne riječi:** RNA,savijanje,model,duboko-učenje,analiza