Supplementary S1-S5

S1 The validity of sequence-structure alignment encoding representation

We compared an eight-bit one-hot encoding representation (SSR) of a sequence containing a secondary structure with a four-bit one-hot encoding representation of a nucleotide-only sequence under the same CNN architecture (Note: they are nine-bit and five-bit, to highlight difference we do not mention about one bit of gap). The validity of the secondary structure information contained in the SSR can be seen in Fig.1, which suggests the significance of our innovative alignment approach of the secondary structure.

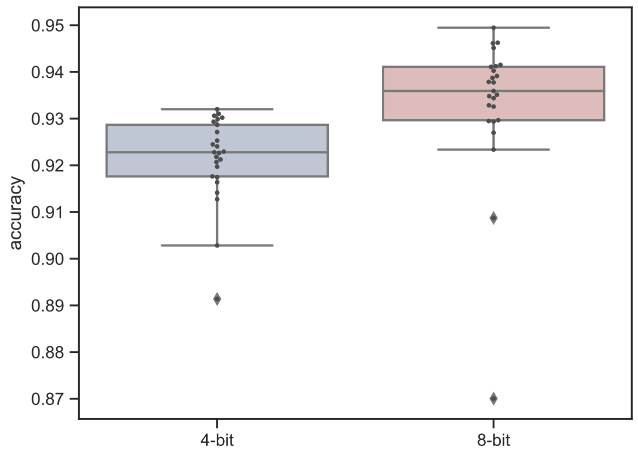

Fig. 1. This box plots show the result of 25 epoch training of the CNN architect, in this compare experiment the everything is the same but the sequence-structure alignment encoding representation is different, on the left is the 4-bit representation (without fused secondary structure), right is the 8-bit representation.

S2 The classification results of RNA structure image representation by pre-training model

We used three pre-training models to extract the features of the RNA structure images. Each model calculates image features by forward-propagating and restricts the output feature shape by adding a fully connected layer. In this way, the image can be transferred into a vector with a fixed length, which can classify different ncRNA preliminary and reduce our model training complexity. Regarding the selection of pre-training models and the influence of their extracted features on the overall multi-view features, please refer to the following three experiments:

Multi-classification experiment.Based on the generated RNA grayscale images (Fig.2), we tested the ability of some common pre-training models to discriminate between images of different ncRNAs (The first two columns of Table 1). Results showed that three models, VGG16, ResNet101, and AlexNet, can discriminate the images.

Binary-classification experiment. In addition to exploring their ability to distinguish between images of different families, we also explored their ability to discriminate in pairwise ncRNAs whether they belong to the same family (the two columns in the Table 1 with the '*' symbol). The results showed the pre-trained model can effectively discriminate whether their corresponding ncRNAs belong to the same family with the help of the alignment-based comparison. Finally, three pre-trained models (AlexNet, ResNet, and Vgg16) with the top F-values in the binary-classification experiment were selected to extract the features of the grayscale images (The pre-trained models with the top three F-values in the binary-classification experiment was the same as the pre-trained models with the top three F-values in the multi-classification experiment). In the binary-classification experiment, we explored the feature vector of the grayscale image extracted by a pre-trained model to discriminate in pairwise ncRNAs whether they belong to the same family. The process of binary-classification and the difference from multi-classification are as follows:

1. Calculate grayscale images for all ncRNAs (dataset is the same as in the main text).
2. Extract the vector features of the grayscale images using a pre-trained model.
3. Splice the vector features of two different ncRNAs horizontally. The label is positive if they belong to the same family and negative if they do not (the construction method is also the same as in the main text). There is no such construction procedure for multi-classification and the label is the family.
4. As with multi-classification, multiple machine learning models (AdaBoost, GradientBoost, LogisticRegression, RandomForest, and DecisionTree) are trained and tested to obtain evaluation metrics.

Removing the grayscale image features experiment. Removing the grayscale image features experiment result showed that the Accuracy was 0.9989 and the F-value was 0.9942 after removing the grayscale image features within the overall multi-view feature representation (MVS) under the 10-fold cross-validation. Compared to the classification results based on the complete MVS (Accuracy was 0.9991 and the F-value was 0.9973), both Accuracy and F-value are decreased. After removing the feature part of grayscale image features, Accuracy reduce from 0.9952 to 0.9914, and F-value reduce from 0.9935 to 0.9887 under the 5-fold cross-validation. Under the 3-fold cross-validation, the reduction is 0.05 (Accuracy reduce from 0.9905 to 0.9855) and 0.045 (F-value reduce from 0.9852 to 0.9807), respectively. These results show that the grayscale image features can lead to performance improvements and should be integrated into MVS. After removing gray image features, Accuracy and F-value were slightly reduced. This is mainly due to the following reasons: i) The grayscale image itself contains a variety of possible base-pairing possibilities, features are extracted by the pre-training model, and the final extracted features account for a small proportion of multi-view feature representation. ii) SSR accounted for the largest proportion in multi-view features, and only relying on SSR can achieve Accuracy 0.9945, F-value 0.9834. The remaining features are mainly used to enhance robustness. More details of these two experiments can be found following. In the experiment by removing the grayscale image features, we removed the grayscale image features extracted by the pre-trained model from the MVS to study the effect of this feature within the overall. Its overall experimental procedure is the same as in the main text, except for the removal of parts of the grayscale image feature. The model is also the integration of the CNN module and the GcForest module under the optimal parameters in the main text.

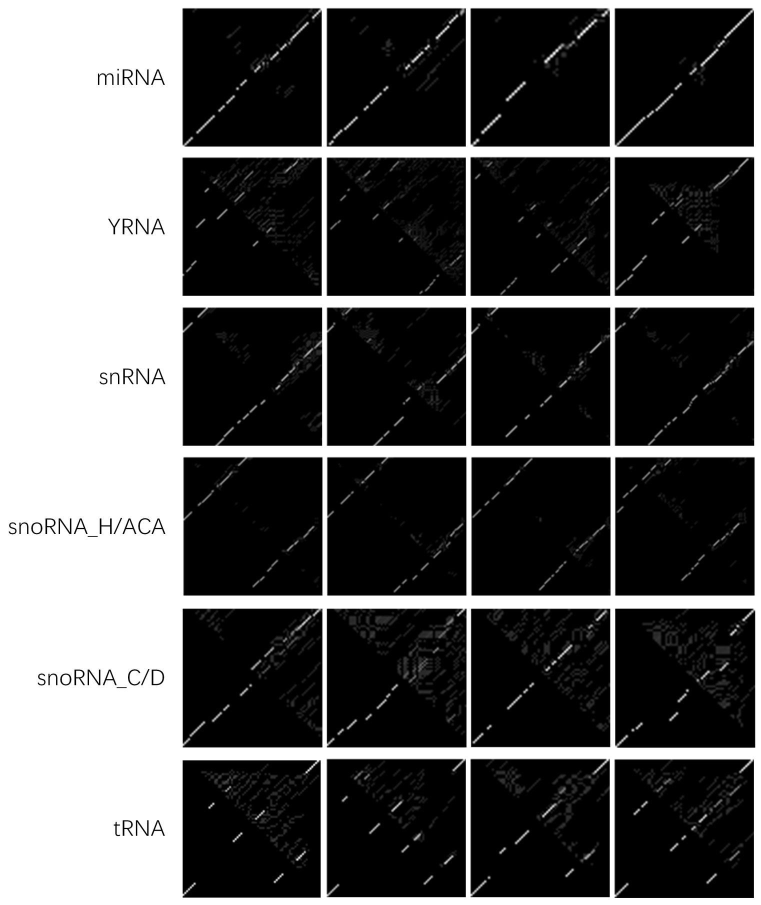

Fig. 2. The first column is the name of ncRNA families, and there are four examples of grayscale images in each row.

Table 1. The classification results of pre-trained model

Model	Accuracy	F-value	Accuracy*	F-value*
alexnet	**0.4317**	**0.4200**	**0.7492**	**0.7384**
resnet18	0.1900	0.1855	0.6719	0.6733
resnet101	**0.3350**	**0.3321**	**0.7246**	**0.7214**
resnet152	0.1800	0.1705	0.6679	0.6666
resnext50_32x4d	0.2000	0.1798	0.6799	0.6773
resnext101_32x8d	0.1850	0.1819	0.6666	0.6669
vgg16	**0.3300**	**0.3282**	0.7200	**0.6834**
vgg19	0.2817	0.2740	**0.7415**	0.6688

note: For resnext where the 32 is the data input first basic unit's groups and the 4d is depth means the channel of each group. Besides, the Accuracy and F-value were calculated max by AdaBoost, GradientBoost, LogisticRegression, RandomForest, and DecisionTree based on 10-fold cross-validation. Especially, these results (without the '*' symbol) were multi-classification, the number of classes is as same as the number of families. These results (with the '*' symbol) were binary-classification (determine whether the feature vector of two different ncRNAs extracted by a pre-training model can distinguish whether the ncRNAs belong to the same family or not).

S3 The pseudo-code of structure subunit

The RNA shape sequence can be calculated by a dynamic programming algorithm based on the RNA structural subunit (Algorithm 1).

Algorithm 1 Calculate the shape structure of the nucleic acid at its location

```
Input: secondary structure $Ss$
Output: sequence of shape structure $SAR$
 1: according to the situation that the $i$ and $j$ nucleic acid pairs are obtained layer by layer from the outside to the inside of the point bracket, if their are
    paired then interchanged the sequence number $i$ and $j$, then, we get $pairs$. $len$ is the length of sequence.
 2: for j = i; j < len; j++ do
 3:   if $Ss_j$=="." then
 4:     search for the nearest brackets location k, m on the left and right
 5:     if There's no match to the right then
 6:       $SAR_j$ = "E"
 7:     else if $Ss_k$=="(" & $Ss_m$==")" then
 8:       $SAR_j$ = "H"
 9:     else if $Ss_k$==")" & $Ss_m$==")" then
10:       if $pairs_{m+1}$==$pairs_k$ then
11:         $SAR_j$ = "B"
12:       else if  then
13:         $SAR_j$ = "T"
14:       end if
15:     else if $Ss_k$==")" & $Ss_m$=="(" then
16:       if there's any i-j pair where i<k and j>m then
17:         $SAR_j$ = "M"
18:       else if  then
19:         $SAR_j$ = "E"
20:       end if
21:     else if $Ss_k$=="(" & $Ss_m$=="(" then
22:       if $pairs_{m+1}$==$pairs_k$ then
23:         $SAR_j$ = "B"
24:       else if  then
25:         $SAR_j$ = "N"
26:       end if
27:     end if
28:   else if  then
29:     if $Ss_k$=="(" then
30:       $SAR_j$ = "L"
31:     else if  then
32:       $SAR_j$ = "R"
33:     end if
34:   end if
35:   for j=0; j<len; j++ do
36:     if $SAR_j$=="N" then
37:       if there are more than two loops between k, m then
38:         $SAR_j$ = "M"
39:       else if  then
40:         $SAR_j$ = "T"
41:       end if
42:     end if
43:   end for
44: end for
45: return $SAR$
```

S4 The list of hyperparameters in GCFM to be tuned

By adjusting the hyperparameters within ranges shown in Table 2, we finally selected the parameter settings for the convolution module and GcForest module with better test results.

Table 2. The list of hyperparameters in GCFM to be tuned

Hyperparameter	Range
Convolution module	
Kernel size for convolution	3, 5, 7, 15, 20, 28, 32, 38
Number of kernels	16, 32, 64, 128, 256
Pooling method	max pooling, average pooling
Window size for pooling	2, 4, 6, 8, 10, 12, 14, 16, 18
Stride of pooling	4, 6, 8, 10, 12, 14
Learning algorithm	Adam, AdaGrad, MomentumSGD
Number of units in hidden layer (ratio to input layer)	1/3, 1/2, 2/3, 3/4, 1
Gcforest module	
Maximum tree depth	3, 5, 7, 9, 11, 13, 15
Number of tree	3, 5, 7, 10, 11, 12, 15, 17
Maximum number of rounds without accuracy increasing that cascade level will stop growing	2, 3, 4, 5, 6, 7
Maximum number of cascade layers	5, 7, 9, 13, 15, 17

We have further analyzed the effect of different hyperparameters of the CNN module on the downstream GcForest model (Table 3). More details are listed as follows. (i) The setting of the convolution kernel and the number of hidden layer units have a significant influence on the performances of the CNN module and the GcForest module. Too small or large convolutional kernel size settings will degrade performance. We selected the proper size for the best front-end features for optimal GcForest performance. As the number of units in the hidden layer decreases, the performance of the downstream GcForest also decreases. Too many units of hidden layers don't provide significant improvement in the performance while using more commutating resources. (ii) Changes of kernel size in the pooling layer have less impact on the CNN module and GcForest module.

Another important point about the impact of the architecture of the CNN module is that the simplification of CNN module architectures leads to the poor performance of the downstream GcForest module. In the last line in Table 3, can be seen that although the Accuracy of the GcForest module improves, the F-value decreases. This means that after combining with the simplified architecture CNN module and the GcForest module, the ability of the integration model to predict true negative cases is improved, but some of the ability to predict true positive cases is lost. Therefore, to ensure availability and robustness, the upstream CNN module needs to contain at least two convolutional layers and two pooling layers.

Table 3. The robustness of downstream GcForest under different CNN modules

CNN modules	Accuracy	F-value	Accuracy*	F-value*
pool(k1_size:14, k2_size:8)	0.9787	0.9876	0.9872	0.9850
pool(k1_size:16, k2_size:12)	0.9816	0.9893	0.9870	0.9951
pool(k1_size:18, k2_size:16)	0.9804	0.9887	0.9862	0.9923
pool(k1_size:18, k2_size:16, stride:4)	0.9291	0.9590	0.9566	0.9832
pool(k1_size:18, k2_size:16, stride:16)	0.9829	0.9900	0.9879	0.9946
conv(k1_size:5)	0.8523	0.9085	0.9199	0.9349
conv(k1_size:7)	0.9298	0.9473	0.9649	0.9784
conv(k1_size:38)	0.9308	0.9594	0.9336	0.9641
conv(k1_n:32, k2_n:64)	0.9457	0.9688	0.9591	0.9784
conv(k1_n:128, k2_n:256)	0.9257	0.9573	0.9507	0.9792
2x the number of hidden layer units	0.9910	0.9933	0.9985	0.9962
1/2 the number of hidden layer units	0.9463	0.9330	0.9849	0.9538
1/4 the number of hidden layer units	0.9298	0.8949	0.9649	0.9239
only one layer of convolution and pooling	0.9080	0.8873	0.9382	0.8705

note: Columns marked with '*' are GcForest module performances, unmarked are CNN module performances. The unadjusted parameters are the same as the original CNN module. k1 and k2 represent kernels (convolution or pooling) in the first and second layers, respectively. n represents the number of kernels.

S5 Phylogenetic tree trees constructed by other methods

The phylogenetic tree in main text is from the Neighbor-Joining (NJ) algorithm based on the classification matrix from GCFM (distance measured as the probability that two ncRNAs belong to the same family). The NJ algorithm starts with an N x N (N is assumed to be the number of ncRNAs) distance matrix D (here, it is a classification matrix generated from GCFM with probability). The NJ algorithm transforms the distance matrix D to a matrix Q, find a minimum element, and join these two nodes. This will lead to a new edge in the tree, two new edge lengths, and reduced D to an (N-1) x (N-1) matrix. Such a process will be iterated until the tree is completely resolved to determine all splits and branch lengths.

Additionally, we showed four mainstream phylogenetic tree construction methods, including Maximum Parsimony, UPGMA, Maximum Likelihood, and Minimum Evolution, to demonstrate that GCFM can indeed contribute to constructing phylogenetic trees. Based on phylogenetic trees from the four methods (Fig.3), we found the three methods (UPGMA, Maximum Likelihood, and Minimum Evolution) have only three errors except for Maximum Parsimony. This suggested that phylogenetic tree construction methods, like UPGMA and Minimum Evolution, which are called distance methods, can all be applied to distance matrix generated based on GCFM and get good results. However, the discrete character methods cannot apply to the distance matrix, like Maximum Likelihood, Maximum Parsimony, and Maximum Parsimony, and they have more errors than the distance methods.

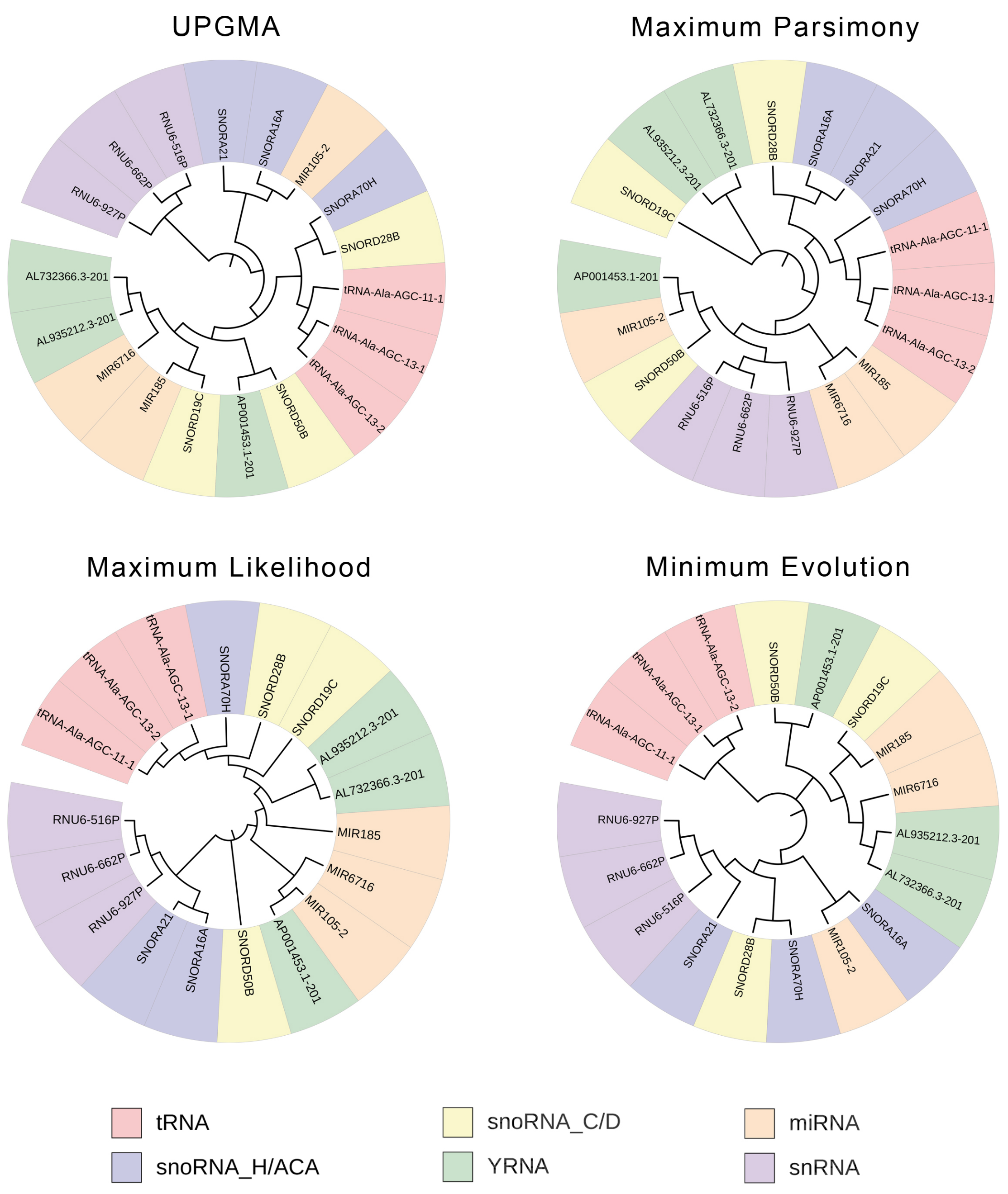

Fig. 3. Results of the mainstream phylogenetic tree construction method.