RNA Secondary Structure Prediction using a Transformer Encoder

Marcel Arian Hadi
Hasso-Plattner-Institut
marcelarian.hadi@student.hpi.uni-potsdam.de

Peregrin Wahle
Hasso-Plattner-Institut
peregrin.wahle@student.hpi.uni-potsdam.de

Abstract

Predicting RNA secondary structure from nucleotide sequence is an important problem in computational biology. In this work, we study the task of translating RNA sequences into dot-bracket secondary structure representations using a Transformer encoder model. We systematically evaluate multiple model configurations and training settings, and assess their performance using both token-level and structure-aware metrics, including sequence-level exact match, paired F1 score, and structural validity. In addition, we analyze the impact of simple postprocessing strategies for correcting invalid structures and perform a downstream structure-level analysis based on the repaired dot-bracket outputs. Our findings show that Transformer encoders are capable of capturing relevant structural patterns in RNA sequences, but that standard token-level objectives alone are insufficient for producing valid structures. Incorporating structure-aware evaluation and lightweight postprocessing significantly improves the quality and validity of the predicted outputs.

1 Introduction

Ribonucleic acid (RNA) plays a central role in many biological processes, including gene regulation, catalysis, and protein synthesis. While the primary structure of RNA is given by its nucleotide sequence, many of its functional properties depend on how this sequence folds into a secondary structure. Predicting RNA secondary structure from sequence alone is therefore a fundamental task in computational biology.

A common representation of RNA secondary structure is the dot-bracket notation, in which unpaired nucleotides are represented by dots and paired nucleotides by matching bracket symbols. This representation turns the prediction problem into a structured sequence labeling task: for each position in the RNA sequence, the model must predict a structural symbol while at the same time respecting global consistency constraints such as valid base-pair matching. As a result, the task is not only about assigning locally plausible labels, but also about producing outputs that form valid overall structures.

This makes RNA secondary structure prediction challenging for standard neural sequence models. Long-range interactions between distant nucleotides are common, and local token-level accuracy alone does not guarantee structurally valid predictions. A model may predict many positions correctly while still producing an invalid dot-bracket sequence. For this reason, evaluating such systems requires not only token-wise metrics, but also structure-aware measures that reflect global correctness and validity.

Preprint. Under review.

In this work, we investigate whether a Transformer encoder model can learn to translate RNA sequences into dot-bracket secondary structure representations. Transformer-based architectures are a natural candidate for this task because self-attention allows the model to capture dependencies between distant positions in the sequence. At the same time, the structured nature of RNA outputs makes this a meaningful test case for the strengths and limitations of such models.

Our study focuses on building and analyzing a working prediction system under realistic computational constraints. We compare multiple model and dataset variants, investigate postprocessing methods for repairing invalid predictions, and perform an additional structure-level analysis of repaired outputs.

2 Data and Problem Formulation

Our task is to predict RNA secondary structure from nucleotide sequence input. We formulate this as a token-level sequence labeling task in which each nucleotide position is mapped to one structural symbol in dot-bracket notation. On the sequence side, the basic RNA alphabet consists of the four standard nucleotides guanine (G), uracil (U), adenine (A), and cytosine (C), while some dataset variants additionally contain ambiguous symbols.

The dataset used in this project is derived from bpRNA data Oregon State University [2018] and was constructed from two sources of structure files, referred to as DBN and STA, which required several preprocessing steps before they could be used for model training.

First, raw sequence-structure pairs were extracted from both archives. Files with malformed content were discarded, for example when sequence and structure length did not match or when the structure contained invalid or inconsistent bracket symbols. In addition, the two sources used different bracket conventions for structural annotation. To make the data comparable, all structures were normalized into a shared dot-bracket representation with a consistent bracket inventory.

Second, duplicate observations were consolidated. Repeated sequence-structure pairs from DBN and STA were merged by counting occurrences per sequence and structure, thereby removing duplicate entries while preserving frequency information.

Some sequences appeared with more than one distinct structure. Since our training setup requires a clear sequence-to-structure mapping, we retained only single-structure examples.

Additional sequence filtering was applied to define biologically consistent input alphabets. In particular, we considered three preprocessing modes that correspond to our internal `base_mode` settings. `base_mode=0` denotes a strict `GUAC` setting containing only the four standard RNA nucleotides. `base_mode=1` denotes a `GUACN` setting in which ambiguous IUPAC symbols are mapped to `N`. `base_mode=2` denotes an extended `GUAC+` setting that retains ambiguous IUPAC symbols explicitly. These variants allowed us to study how dataset tokenization influences the learning problem.

In addition, we used a maximum-length filter rather than truncation. Sequences longer than the chosen threshold were removed instead of being cut, ensuring that all retained examples corresponded to complete biological sequences and structures without artificial shortening or partial targets. For the experiments reported in this work, we focused on datasets with a maximum sequence length of 1024. This choice provided a practical compromise between retaining long RNA molecules and respecting the memory and runtime limits of the available hardware.

Finally, the cleaned data was converted into a tagging format suitable for model training. The resulting dataset consists of unique sequence-structure pairs together with metadata such as the sequence alphabet, structure alphabet, normalized bracket pairs, and maximum sequence length. This final representation provides a consistent and well-defined basis for learning RNA sequence-to-structure mappings.

After preprocessing, the resulting dataset sizes differ slightly across modes: the strict `GUAC` setting (`base_mode=0`) contains 54,379 sequences, while `GUACN` (`base_mode=1`) and `GUAC+` (`base_mode=2`) contain 55,786 and 55,793 sequences, respectively.

3 Method

3.1 Problem Setup

Let $x = (x_1, \ldots, x_T)$ denote an RNA sequence of length T, where each x_t belongs to the input alphabet Σ_{seq}. Let $y = (y_1, \ldots, y_T)$ denote the corresponding RNA secondary structure in dot-bracket notation, where each y_t belongs to the structure alphabet Σ_{str}. We formulate RNA secondary structure prediction as a sequence tagging problem, that is, we seek to learn a function

$$f_\theta : \Sigma_{\text{seq}}^T \to \Sigma_{\text{str}}^T$$

parameterized by θ, which maps each nucleotide position to a structural token.

3.2 Input and Output Encoding

Each input sequence is converted into integer IDs via a lookup map

$$\phi_x : \Sigma_{\text{seq}} \to \{0, \ldots, |\Sigma_{\text{seq}}| - 1\},$$

and each target structure is encoded analogously by

$$\phi_y : \Sigma_{\text{str}} \to \{0, \ldots, |\Sigma_{\text{str}}| - 1\}.$$

Within each mini-batch, sequences are padded to the maximum batch length. Let $m_t \in \{0, 1\}$ denote a binary mask indicating whether position t is valid ($m_t = 1$) or padded ($m_t = 0$). Padded target positions are excluded from the training objective.

3.3 Transformer Encoder Architecture

Our prediction model is a Transformer encoder tagger. Given an encoded input sequence $(x_1, \ldots, x_T)$, each token is first mapped to a learned embedding vector,

$$e_t = E[x_t] \in \mathbb{R}^{d_{\text{model}}},$$

where $E \in \mathbb{R}^{|\Sigma_{\text{seq}}| \times d_{\text{model}}}$ is the embedding matrix. Sinusoidal positional encoding $p_t \in \mathbb{R}^{d_{\text{model}}}$ is then added to obtain

$$h_t^{(0)} = e_t + p_t.$$

The sequence of hidden states is processed by L Transformer encoder blocks. For layer $\ell \in \{1, \ldots, L\}$, the update takes the form

$$\tilde{h}^{(\ell)} = h^{(\ell-1)} + \text{MHSA}\big(\text{LN}(h^{(\ell-1)})\big),$$

$$h^{(\ell)} = \tilde{h}^{(\ell)} + \text{FFN}\big(\text{LN}(\tilde{h}^{(\ell)})\big),$$

where LN denotes layer normalization, MHSA multi-head self-attention, and FFN a position-wise feed-forward network. This corresponds to a pre-normalization architecture with residual connections and dropout.

3.4 Self-Attention

For each attention head, queries, keys, and values are computed as

$$Q = HW_Q, \qquad K = HW_K, \qquad V = HW_V,$$

where $H \in \mathbb{R}^{T \times d_{\text{model}}}$ is the input to the attention layer. Attention scores are given by

$$A = \frac{QK^\top}{\sqrt{d_h}},$$

where $d_h = d_{\text{model}}/n_{\text{heads}}$ is the per-head dimension. Since RNA secondary structure depends on potentially long-range interactions, we use non-causal self-attention, allowing every valid position to attend to every other valid position in the sequence. Padding positions are masked out in the attention computation.

3.5 Output Layer

After the final encoder block, each contextualized representation $h_t^{(L)}$ is mapped to logits over the structure vocabulary:

$$z_t = W_{\text{out}} h_t^{(L)} + b_{\text{out}},$$

with

$$z_t \in \mathbb{R}^{|\Sigma_{\text{str}}|}.$$

The conditional distribution over structure tokens at position t is then

$$p_\theta(y_t \mid x) = \text{softmax}(z_t).$$

The raw prediction is obtained independently at each position by

$$\hat{y}_t = \arg\max_{c \in \Sigma_{\text{str}}} p_\theta(c \mid x).$$

3.6 Training Objective

Training is performed with token-level cross-entropy loss over all non-padded positions. For a single sequence, the loss is

$$\mathcal{L}(x, y) = -\sum_{t=1}^{T} m_t \log p_\theta(y_t \mid x),$$

and the overall training objective is the empirical mean over the training set. Thus, the model is optimized only with local per-position supervision. No explicit structural validity constraint is included in the loss.

3.7 Postprocessing

Because independent token predictions do not guarantee a valid dot-bracket structure, we additionally evaluate two repair procedures after decoding.

Kill-to-Dot (KTD). This method scans the predicted structure from left to right and maintains a stack for each bracket type. Unmatched closing brackets are replaced by dots immediately, and any unmatched opening brackets remaining at the end are also replaced by dots. Thus, KTD transforms an invalid prediction into a valid one by deleting inconsistent bracket assignments.

Dynamic-Programming-Guided Salvage (DPGS). This method uses the model logits to repair predictions based on model confidence, rather than only deleting inconsistent brackets.

Since a joint optimization over all bracket types would be computationally too expensive, DPGS uses a type-wise heuristic. For each bracket type (o, c) separately, where o and c denote the opening and closing symbols of that type, it considers only positions currently assigned to one of the three symbols $\{., o, c\}$ and computes, by dynamic programming, the highest-scoring valid balanced sequence under the corresponding per-position log-probabilities. Thus, the repair is globally optimal for the current bracket type, but not globally optimal over the full structure.

The bracket types are processed in the order given by the dataset metadata, namely `bracket_type_order`. This order is determined during preprocessing from bracket frequencies in the training data, with more frequent bracket types handled first. Positions that cannot be incorporated consistently into a valid sequence are mapped to dots. In this way, DPGS preserves plausible bracket assignments whenever possible while remaining computationally tractable.

3.8 Downstream Structure Labeling

We additionally perform a separate structure-level analysis on postprocessed outputs. This step is not part of training or decoding, but serves as an additional structural interpretation layer.

Given a repaired dot-bracket sequence, we apply a deterministic labeling algorithm that assigns each position a higher-level structural role. In particular, paired positions are labeled as stem positions (`S`), while unpaired positions are classified into categories such as hairpin loop (`H`), bulge (`B`), internal

loop (`I`), multiloop (`M`), exterior region (`E`), or exterior mixed region (`X`), depending on their location relative to enclosing base pairs and nested substructures.

The algorithm extracts base-pair relations, derives a hierarchical parent-child structure of primary base pairs, and assigns labels according to structural context.

4 Experimental Setup

4.1 Training Protocol

All experiments reported in this work were trained from scratch using `mode=new`, that is, by starting a fresh training run together with a newly generated data split. We did not use `mode=current`, which refers to resuming from an existing training state. As a result, all reported model comparisons are based on independent runs rather than resumed optimization trajectories.

For each dataset, we used a fixed random train/validation/test split of 80% / 10% / 10%, generated with split seed 1337 and stored for reproducibility. This ensures that experiments conducted on the same dataset are directly comparable.

Training used the AdamW optimizer together with a cosine learning-rate schedule. The learning rate was updated across epochs, starting from its initial value and decaying smoothly over the course of a run towards a predefined minimum. Model selection was based on validation loss.

During each run, we stored both the latest model state (`last`) and the best validation-loss model state (`best`). In addition, we saved training snapshots at regular intervals and at the end of training or upon early stopping. These snapshots allow us to compare model quality across training stages and to evaluate postprocessing over the course of learning.

4.2 Compared Model Variants

Rather than performing an exhaustive hyperparameter search, we evaluated a focused set of model variants chosen to study a small number of practically relevant design axes.

Our base model uses embedding dimension $d_{\text{model}} = 256$, 8 attention heads, 4 encoder layers, feed-forward dimension $d_{\text{ff}} = 1024$, dropout 0.1, learning rate $5 \cdot 10^{-4}$, minimum learning rate $\eta_{\min} = 10^{-5}$, and weight decay 10^{-2}. Starting from this configuration, our comparisons include deeper variants obtained by increasing the number of encoder layers, larger variants obtained by jointly increasing model width and feed-forward size, and a small set of alternative weight decay values.

Across these comparisons, the core architecture remained the same.

4.3 Dataset Variants

In addition to architectural comparisons, we evaluated the same modeling approach on different dataset variants induced by sequence preprocessing. Following the variants introduced in Section 2, we compared the three input settings `base_mode=0,1,2` (`GUAC`, `GUACN`, and `GUAC+`). This allows us to study how different treatments of ambiguous nucleotide symbols affect prediction quality.

All reported experiments in the main model comparison use the same maximum sequence length of 1024. Thus, differences between runs are not caused by sequence-length changes, but by architecture, regularization, or tokenization.

4.4 Evaluation Metrics

To assess model quality, we report validation loss, token accuracy, sequence exact match, paired-position F1, and invalid rate.

Let B denote the number of sequences in the evaluation set, let T_b denote the valid length of sequence b, let $y_t^{(b)}$ be the gold label at position t, and let $\hat{y}_t^{(b)}$ be the predicted label. Further, let $m_t^{(b)} \in \{0, 1\}$ indicate whether position t is valid or padded.

Token accuracy is computed over all non-padded positions:

$$\text{TokenAcc} = \frac{\sum_{b=1}^{B} \sum_{t=1}^{T_b} m_t^{(b)} \mathbf{1}\left[\hat{y}_t^{(b)} = y_t^{(b)}\right]}{\sum_{b=1}^{B} \sum_{t=1}^{T_b} m_t^{(b)}}.$$

Sequence exact match requires the complete predicted structure of a sequence to be correct:

$$\text{SeqExact} = \frac{1}{B} \sum_{b=1}^{B} \mathbf{1}\left[\forall t:\ m_t^{(b)} = 0 \text{ or } \hat{y}_t^{(b)} = y_t^{(b)}\right].$$

For paired-position F1, all bracket symbols are treated as paired labels and dots as unpaired labels. Let $P_t^{(b)}$ denote whether the gold label at position t is paired, and let $\hat{P}_t^{(b)}$ denote the corresponding prediction. Then TP counts valid positions that are paired in both gold and prediction, FP counts valid positions that are predicted as paired although the gold label is unpaired, and FN counts valid positions that are paired in the gold structure but predicted as unpaired. We compute

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN},$$

and

$$\text{PairedF1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Finally, invalid rate measures the fraction of predicted sequences that violate dot-bracket consistency constraints:

$$\text{InvalidRate} = \frac{1}{B} \sum_{b=1}^{B} \mathbf{1}\left[\hat{y}^{(b)} \text{ is not a valid dot-bracket sequence}\right].$$

Together, these metrics capture complementary aspects of performance: local labeling quality, full-structure correctness, paired-position prediction quality, and structural well-formedness.

4.5 Postprocessing Comparison

To evaluate the effect of structural repair, we compared three decoding conditions on validation data: raw argmax predictions, Kill-to-Dot (KTD), and Dynamic-Programming-Guided Salvage (DPGS). These comparisons were carried out across saved training snapshots, allowing us to analyze not only final model quality but also how postprocessing interacts with model quality over time.

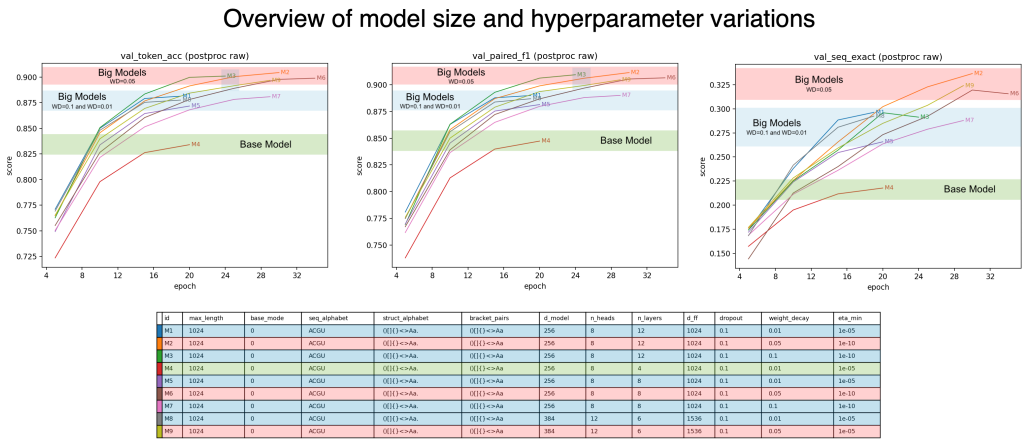
4.6 Structure-Level Analysis

Besides the main prediction metrics, we performed an additional structure-level analysis on post-processed predictions. For this purpose, repaired dot-bracket outputs were converted into structural element labels using the downstream labeling procedure described in Section 3.8. This analysis was used to examine whether the predicted structures recover higher-level RNA components such as stems, hairpins, bulges, internal loops, and multiloops.

Since this labeling step is deterministic and applied after decoding, it does not affect training or model selection, but provides an additional perspective on structural quality.

5 Results

We first assess whether the Transformer encoder can learn meaningful sequence-to-structure mappings for RNA secondary structure prediction. Across the evaluated configurations, the models achieve non-trivial performance on the validation set, indicating that the task is learnable within this framework. We then examine the effects of model size, depth, weight decay, postprocessing, and dataset composition, followed by a separate structure-level analysis based on downstream structural labels.



id	max_length	base_mode	seq_alphabet	struct_alphabet	bracket_pairs	d_model	n_heads	n_layers	d_ff	dropout	weight_decay	eta_min
M1	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	12	1024	0.1	0.01	1e-05
M2	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	12	1024	0.1	0.05	1e-10
M3	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	12	1024	0.1	0.1	1e-10
M4	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	4	1024	0.1	0.01	1e-05
M5	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	8	1024	0.1	0.01	1e-05
M6	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	8	1024	0.1	0.05	1e-10
M7	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	256	8	8	1024	0.1	0.1	1e-10
M8	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	384	12	6	1536	0.1	0.01	1e-05
M9	1024	0	ACGU	()[]{}<>Aa.	()[]{}<>Aa	384	12	6	1536	0.1	0.05	1e-05

Figure 1: Overview of model size and hyperparameter variations. The small model underperforms, while scaled and deeper models form a performance cluster. Within this cluster, lower and higher weight decay values (0.01 and 0.1) yield similar performance, while 0.05 performs better.

5.1 Overview of Model Size and Hyperparameters

We evaluated a range of model sizes and hyperparameter configurations to understand their impact on RNA secondary structure prediction. Figure 1 presents an overview of all tested variants, including the small/base model, scaled models (all parameters $\times 1.5$), deeper models (layer $\times 2$ and $\times 3$), and models trained with different weight decay values. We also explored different minimum learning rates (η_{min}) in the cosine scheduler, but did not observe a clear effect on performance; these variations are therefore not included in the overview figure.

Among all evaluated configurations, the best model achieved, without postprocessing, a validation token accuracy of 90.4%, a paired F1 of 91.1%, and a sequence exact match of 33.6%.

As shown in Figure 1, the small model clearly underperforms compared to larger variants. In general, models with scaled parameters or increased depth form a cluster with similar performance, especially in sequence-level metrics. Within this cluster, weight decay values of 0.01 and 0.1 result in comparable performance, while 0.05 performs better than both. This analysis motivates a more detailed examination of individual factors such as depth and weight decay in the following sections.

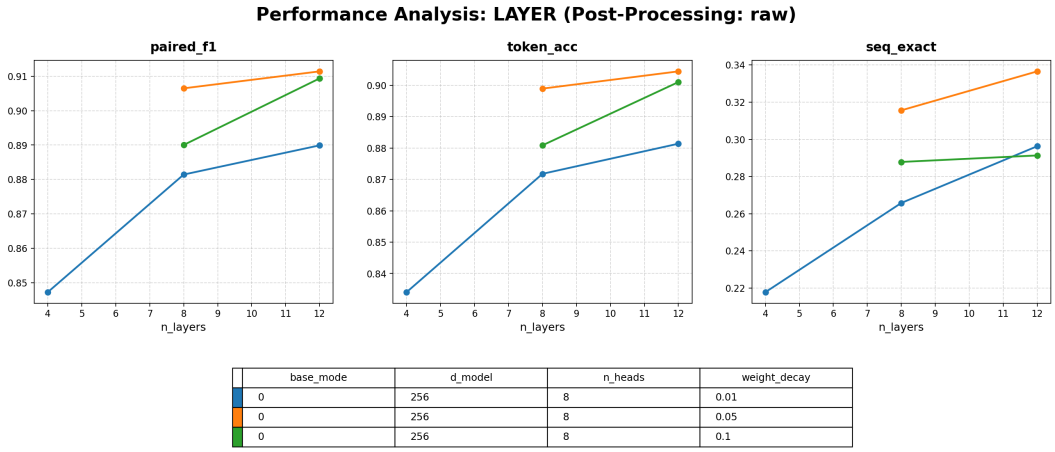
5.2 Effect of Model Depth

To isolate the effect of network depth, we plotted models that differ only in the number of layers (Figure 2). The results indicate a consistent improvement in token-level accuracy and paired F1 with increasing layer count. Sequence-level exact match also increases for the deeper variants, suggesting that additional depth helps the model capture the dependencies required for structured RNA prediction.

5.3 Effect of Weight Decay

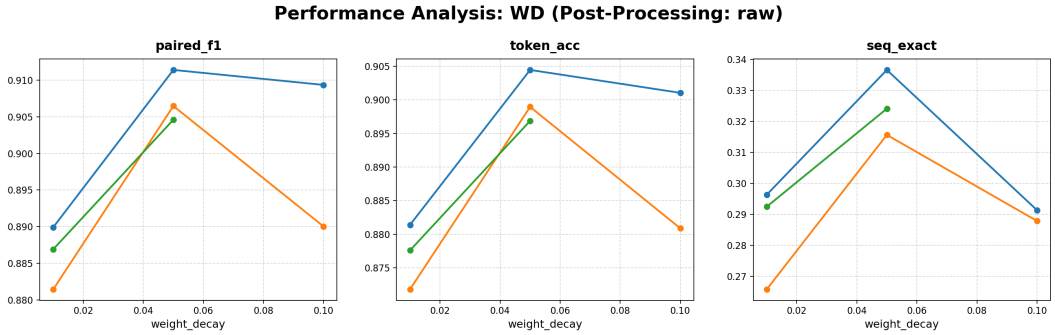
We further investigated the influence of regularization by training models with different weight decay values (Figure 3). For both models, a moderate weight decay (0.05) yields the strongest overall performance among the tested values. Increasing weight decay to 0.1 leads to a substantial drop across all metrics for the medium-depth model.

In contrast, the deep model is more robust to stronger regularization: F1 and accuracy decrease only slightly, while sequence-level exact match drops more noticeably. This suggests that deeper models can better tolerate increased regularization at the token level, but sequence-level structural correctness remains sensitive.



	base_mode	d_model	n_heads	weight_decay
	0	256	8	0.01
	0	256	8	0.05
	0	256	8	0.1

Figure 2: Impact of model depth on performance. Increasing the number of layers consistently improves token-level metrics (F1, accuracy) and sequence-level exact match.



	base_mode	d_model	n_heads	n_layers
	0	256	8	12
	0	256	8	8
	0	384	12	6

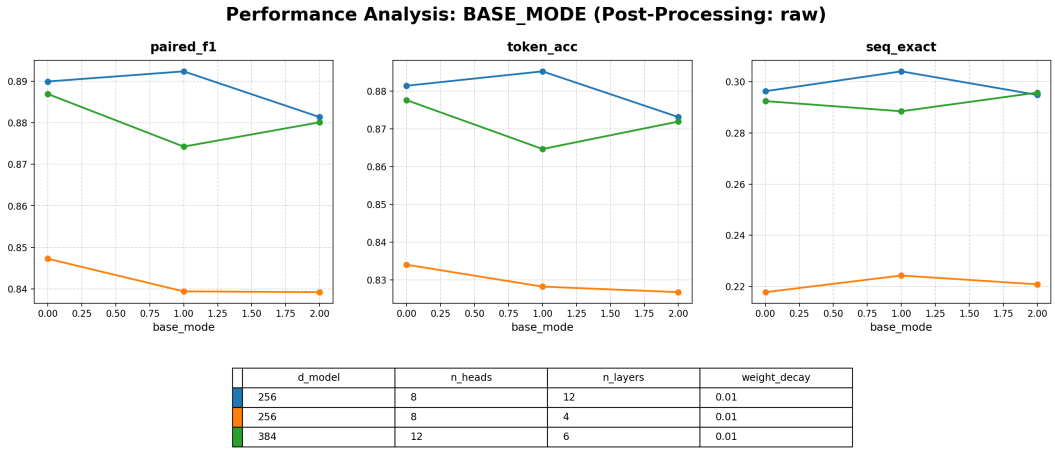
Figure 3: Impact of weight decay on model performance. A moderate weight decay (0.05) yields the best overall results for both models. Increasing weight decay to 0.1 leads to a clear performance drop for the medium-depth model, while the deep model remains more robust in F1 and accuracy but shows a noticeable decrease in sequence-level exact match.

5.4 Effect of Dataset Tokenization

To assess how input representation affects learning, we trained the same models on datasets with different token sets. Following the preprocessing modes described in Section 4.3, `base_mode=0` uses only the four standard nucleotides (G, U, A, C), `base_mode=1` maps ambiguous symbols to N, and `base_mode=2` retains ambiguous IUPAC symbols. These settings allow us to examine the sensitivity of the models to tokenization choices.

The results, shown in Figure 4, reveal that the impact of tokenization depends on model architecture. For the small model, `base_mode=0` achieves the highest performance, followed by `base_mode=1` and `base_mode=2`. For the larger model, `base_mode=0` also performs best, while `base_mode=2` slightly outperforms `base_mode=1`. For the deep model, `base_mode=1` provides the best token-level and sequence-level performance, with `base_mode=2` performing worst.

These trends suggest that ambiguous bases (represented as `N` in `base_mode=1`) introduce uncertainty in the input, which generally reduces performance for smaller models. The deep model, however, can better handle these uncertainties due to its increased capacity, explaining why it performs



	d_model	n_heads	n_layers	weight_decay
	256	8	12	0.01
	256	8	4	0.01
	384	12	6	0.01

Figure 4: Performance of the same model on datasets with different token sets (`base_mode=0, 1, 2`). Including ambiguous bases (`N`) introduces uncertainty in the input, which generally reduces performance for the smaller model. The deeper model can partially compensate for these uncertainties, resulting in higher performance with `base_mode=1`. These results illustrate the model-dependent sensitivity to dataset tokenization and the handling of ambiguous bases.

slightly better with `base_mode=1` compared to 0 or 2. Overall, the results highlight that the optimal tokenization strategy is model-dependent, emphasizing the importance of biologically consistent input representations.

These results are presented separately from the hyperparameter overview (Figure 1) to emphasize that the observed differences arise from dataset composition rather than cross-model hyperparameter variations.

5.5 Postprocessing Effects on Structural Validity

Postprocessing has a significant impact on structural validity. While raw token predictions achieve reasonable token-level scores (paired F1), they can still violate bracket consistency. As shown in Figure 5, KTD performs worse than raw predictions in paired F1, yet it achieves higher sequence-level exact match. This is consistent with the fact that KTD enforces structural validity by construction. DPGS shows only minimal gains in paired F1 but substantially improves sequence-level exact match and achieves even better results than KTD. These results highlight that different postprocessing approaches have markedly different impacts on token-level versus sequence-level performance, emphasizing the importance of structure-aware evaluation in RNA secondary structure prediction.

5.6 Structure-Level Analysis

To further analyze the predicted RNA secondary structures, we developed a post-hoc annotation script that assigns structural labels (e.g., stems, loops, exterior strands) to each position based on the dot-bracket notation. This allows us to move beyond token-level evaluation and assess the data at a higher structural level. To ensure the technical correctness of this tool, we performed a comprehensive validation against the $102,318$ sequences in our dataset.

During this validation, we implemented two mutually exclusive annotation modes: *Strict* and *Pragmatic*. These modes represent different biological interpretation standards, particularly regarding the classification of boundary cases. The results of this cross-check, summarized in Table 1, notably highlighted a fundamental characteristic of the dataset's ground truth.

While the two modes together account for all sequences in the validation set, neither mode alone matches the dataset labels completely. Specifically, 30.64% of the sequences are matched exclusively by the strict mode, whereas 5.47% are matched exclusively by the pragmatic mode. This indicates that the dataset does not follow a single annotation convention uniformly.

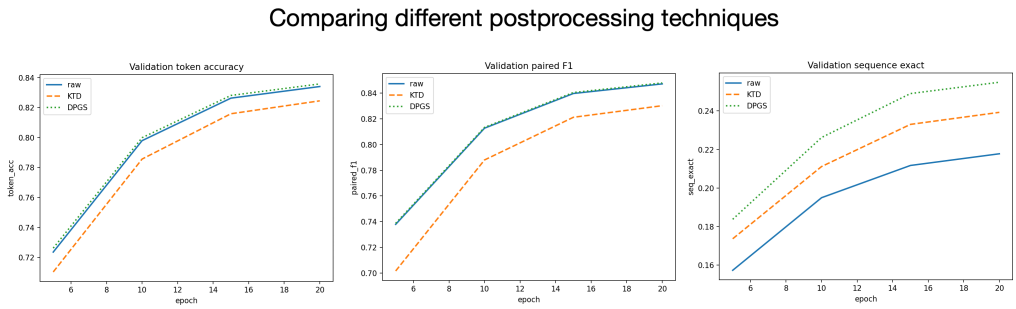


Figure 5: Comparison of different postprocessing strategies for one of our models. KTD performs significantly worse than raw predictions in paired F1 but improves sequence-level exact match by ensuring all outputs are valid. DPGS further boosts sequence-level exact match, with only minimal changes in paired F1.

Table 1: Validation of the Annotation Script against Dataset Labels

Metric	**Count**	**Percentage**
Total Sequences Validated	102,318	100.00%
Matches Strict Logic	96,719	94.53%
Matches Pragmatic Logic	70,969	69.36%
Matches *Exclusively* Strict	31,349	30.64%
Matches *Exclusively* Pragmatic	5,599	5.47%
Total Explained (Either Mode)	**102,318**	**100.00%**

These discrepancies therefore appear to result primarily from differences in annotation conventions within the dataset rather than from implementation errors in the labeling procedure. As a consequence, structural labels should be interpreted relative to the annotation convention used by the underlying source.

6 Discussion

The results suggest that Transformer encoders are a suitable choice for RNA secondary structure prediction because they can capture long-range interactions between distant sequence positions. This is a key requirement of the task, since paired bases are often separated by large distances. At the same time, our experiments show that raw token prediction alone is not sufficient to guarantee structurally valid outputs.

A key aspect of this work is that the main objective was not to find the best possible hyperparameter setting through exhaustive search. Instead, the focus was on developing a complete and working prediction pipeline and evaluating whether the chosen modeling approach is effective for the task. Within the scope of this project, the approach was successful in the sense that the model learned meaningful sequence-to-structure mappings and the evaluation setup captured both predictive accuracy and structural correctness.

There are, however, several limitations. First, the number of tested configurations was limited by computational resources. Second, all experiments were trained from scratch, which made each run costly. Third, although we compared several reasonable settings, the explored parameter space remains small. Future work could evaluate larger models, longer training schedules, and more systematic configuration searches. It would also be interesting to integrate structural constraints more directly into the decoding process instead of relying on postprocessing after prediction.

7 Conclusion

We presented a Transformer-based approach to RNA secondary structure prediction in dot-bracket notation and evaluated it within a complete end-to-end pipeline covering preprocessing, training, decoding, postprocessing, and downstream structural analysis. Our experiments show that Transformer encoders can learn meaningful sequence-to-structure mappings, but also that token-level optimization alone does not ensure structurally valid outputs. Structure-aware postprocessing substantially improves the validity and sequence-level quality of the predictions. Overall, the results support Transformer encoders as a practical baseline for RNA secondary structure prediction under realistic computational constraints.

References

Oregon State University. bprna-1m: Rna secondary structure database, 2018. URL `https://bprna.cgrb.oregonstate.edu/`. Accessed: 2026-03-30.