ChatGPT

Stanford RNA 3D Folding Part 2: High-Ranking Solution Pipeline

Competition Overview

The **Stanford RNA 3D Folding Challenge (Part 2)** asks participants to predict RNA three-dimensional structures from sequence data alone. Submissions are scored by **TM-score**, comparing predicted 3D coordinates to known structures [1]. To ensure robust predictions, competitors must output five candidate structures per RNA, with the best-of-five contributing to the score [1]. This rigorous setup pushes models toward high accuracy and discourages trivial template matching. An official baseline model (**Boltz-1**) was provided as a starting point [2], but top teams greatly improved upon it with novel architectures and training tricks.

Top Solution Approaches

Leading teams leveraged advanced **deep learning models** and RNA-specific features to achieve top performance:

- **1st Place:** A transformer-based model with **dynamic positional encoding** and a CNN processing base-pair probability matrices (BPPMs) [3]. This approach addressed the challenge of **sequence length generalization** by using learnable biases that adapt to longer RNAs, avoiding the limits of absolute position encoding.
- **2nd Place:** A model built on a **Squeezeformer** (Conformer-like transformer) backbone, augmented with **BPP Conv2D attention** biases [4] [5]. The team used **ALiBi positional encoding** for better extrapolation to long sequences [6] and fed **predicted base-pairing probabilities (BPP)** into the attention layers as learnable bias terms, improving structure awareness [5].
- **3rd Place:** An **AlphaFold2-inspired "twin-tower" architecture** combined with a Squeezeformer model [7]. The twin-tower network (resembling AlphaFold's dual networks) captured 3D contacts, while the squeezeformer handled sequence features. Notably, all top-3 teams incorporated **BPP features** into their models' attention or bias mechanisms [8], underlining the importance of pairing information for RNA folding. Many also employed techniques like **weight sampling by signal-to-noise ratio**, **dynamic positional biases**, and **ensembling** to boost generalization [9] [10].

Pipeline Implementation (Train–Test–Use)

A high-ranking solution pipeline involves several stages, from data prep to inference. Below is an outline inspired by the top solutions:

1. **Data Preparation:** Download the competition data (e.g. training RNA sequences, known structures, multiple sequence alignments, and BPP files) from Kaggle. For instance, the *3rd place team* placed the provided CSV and MSA data under a designated folder and ran preprocessing scripts [11]. The RNA sequences are preprocessed (e.g. converted to a suitable format like a folded **parquet** file), and large supporting data like BPP matrices are processed via a script (e.g.

`preprocess_bpps.py`) to generate efficient indexed files [11]. This yields ready-to-use inputs such as numerical sequence features, evolutionary info from MSAs, and pairwise features (BPP or contact maps).

2. **Model Training:** Train one or more models on the prepared data. Top teams often trained multiple models in sequence. For example, one pipeline first trained a **“twin-tower” model** (a complex architecture for 3D contact prediction) using a distributed training script [12]. Then, a second model (e.g. a Squeezeformer-based network) was trained on the data. These training jobs were initiated with simple one-line commands using the teams’ code. For instance, the 3rd place team’s code uses:

   ```
   python train.py -C cfg_1
   ```

   to launch training with configuration `cfg_1` (for their Squeezeformer model) [13]. Likewise, to train the twin-tower model on 2 GPUs, they ran:

   ```
   python train_ddp.py -C cfg_2
   ```

   which executes the distributed data-parallel training with config `cfg_2` [12]. The training stage may involve monitoring validation TM-scores and possibly saving the best model checkpoints. (In practice, teams performed extensive hyperparameter tuning and cross-validation, given the importance of generalizing to novel RNA structures.)

3. **Synthetic Data Augmentation (Optional):** To further boost performance, some top teams generated additional training data. After the first model was trained, its predictions were used to create **synthetic structural data** for unlabeled or pseudo sequences. For example, the 3rd place solution trained their twin-tower model, then ran it on a large set of RNA sequences to produce pseudo-structures via:

   ```
   python generate_synthetic.py
   ```

   (generating predicted 3D coordinates) [14]. These synthetic samples (with model-predicted structures) were then combined with the real training set. The team next re-trained the Squeezeformer model on this **augmented dataset** (original + synthetic) [14]. This two-stage training (often called *knowledge distillation* or self-training) improved the model’s ability to handle difficult targets by exposing it to more structural examples.

4. **Model Inference & Ensemble:** With trained models ready, the next step is generating predictions on the test set. Participants wrote inference scripts (notebooks) that Kaggle would run in a restricted environment to produce output files. For instance, to get predictions from the Squeezeformer model, the 3rd place team provided an inference script invoked as:

   ```
   python inference_mdl_1_squeezeformer.py
   ```

   which loads the trained weights and outputs predicted 3D coordinates for each RNA in the test set [15]. Crucially, each submission had to output **5 structures per RNA**, so inference code often runs the model with different random seeds or dropout (to sample different folds) and then saves five predicted structures for every input sequence [1]. Top teams also **ensembled** multiple models' outputs to improve reliability. In the 3rd place pipeline, they blended the twin-tower and Squeezeformer predictions (averaging their coordinate outputs with 0.5 weight each) for the final results [16]. Such ensembling tends to cancel out individual model biases and gave a significant accuracy boost.

5. **Evaluation & Submission:** The generated structures were then evaluated using the provided **TM-score script** before final submission. Kaggle's evaluation ensured that only correctly aligned nucleotides contributed to the TM-score and averaged the scores of the best-predicted structure for each RNA [1]. Participants would validate their pipeline on a **public validation set** (if provided) or via local scoring using withheld structures. Finally, the inference pipeline (which includes model loading, prediction of five structures per RNA, and output formatting) was submitted to Kaggle for scoring on the hidden test set. A well-structured, reproducible pipeline that could be executed in Kaggle's environment was essential for a successful entry.

Each of these steps was automated as much as possible. The entire high-ranking solution workflow – from reading input sequences, generating features (like MSAs and BPPs), running deep learning models for structure prediction, to outputting multiple candidate 3D structures – was encapsulated in code. By combining advanced models (transformers with RNA-specific tweaks) and careful pipeline engineering, top teams produced solutions that **"played for the top of the leaderboard."** In summary, a one-line command (as shown above) could launch a training or inference job, but it is the integration of all the above pipeline stages that ultimately yielded a winning solution [14] [16].

Sources: The pipeline and methods described are based on the official competition data and top solution write-ups, including the 3rd place team's open-sourced code and README [13] [14], the 2nd place solution notes [6] [5], and the competition overview from Stanford/HHMI [1]. These references provide further details on model architectures and the successful strategies used by winners.

---

[1] Stanford-backed RNA folding challenge returns as AI tackles biology's next hard problem — EdTech Innovation Hub
https://www.edtechinnovationhub.com/news/stanford-backed-rna-folding-challenge-returns-as-ai-tackles-biologys-next-hard-problem

[2] Stanford RNA 3D Folding | Kaggle
https://www.kaggle.com/competitions/stanford-rna-3d-folding/discussion/576619

[3] [4] [5] [6] [7] [8] [9] [10] Stanford Ribonanza RNA Folding
https://kaggle.curtischong.me/competitions/Stanford-Ribonanza-RNA-Folding

[11] [12] [13] [14] [15] [16] GitHub - GosUxD/OpenChemFold: 3rd Place Solution for the Stanford Ribonanza RNA Folding Competition
https://github.com/GosUxD/OpenChemFold