POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Bachelor's thesis

QUALITY ASSESSMENT OF 3D RNA STRUCTURES USING GRAPH NEURAL NETWORKS

Bartosz Adamczyk, 148163
Maciej Biliński, 148221
Mikołaj Bartkowiak, 148164
Szymon Stanisławski, 150192

Supervisor
dr hab. inż. Maciej Antczak, prof. PP

POZNAŃ 2024

Contents

Chapter 1

Introduction

RNA, or ribonucleic acid, is a vital biomolecule essential for various cellular processes, e.g., gene regulation or protein synthesis [1]. Composed using nitrogenous bases (i.e., adenine, guanine, cytosine, and uracil), RNA's primary structure is a linear sequence of these nucleotides. However, the true complexity and functionality of RNA emerge from its spatial conformation. The three-dimensional structure of RNA molecules is pivotal in dictating their specific functions, as it determines how they interact with other biomolecules [2, 3].

The RNA-Puzzles competition is a collective blind experiment that aims to address the challenging task of accurately predicting three-dimensional RNA structures. RNA-Puzzles provides a structured framework for participants to evaluate the strengths and weaknesses of existing computational approaches for predicting three-dimensional RNA structures and examining advancements and identifying any obstacles impending progress in the field [4, 5].

On the other hand, the CASP-RNA competition, an integral component of the Critical Assessment of Structure Prediction initiative aims to evaluate the state-of-the-art in protein and RNA structure prediction. CASP-RNA focuses on assessing the accuracy of computational methods in predicting the three-dimensional structures of RNA molecules by providing a standardized platform for the evaluation of diverse prediction methods [6].

The hierarchical notation of RNA structures captures the multi-level organization of RNA molecules, encompassing various levels of structural complexity from the primary sequence to higher-order structures. The primary structure represents the linear sequence of nucleotides, denoted by the specific arrangement of adenine (A), guanine (G), cytosine (C), and uracil (U) nitrogenous bases. Moving to the secondary structure, interactions such as base pairing lead to the formation of structures: helices, bulges, loops, and junctions of various kind. Secondary structure is often represented using diagrams, where base pairs are depicted using connected lines. Tertiary structure involves the atom locations in three-dimensional space. Quaternary structure refers to the interactions between multiple RNA molecules or other biological compounds.

The 3D structures of RNA are experimentally determined using a number of techniques, e.g., X-ray crystallography, Nuclear Magnetic Resonance (NMR) spectroscopy, or cryogenic electron microscopy (cryoEM) [2]. Unfortunately, experimental methods are often expensive, difficult to apply, and not always effective due to the structural specificity, flexibility or dynamics of 3D RNA structures. Therefore, the results of computational methods often support experimenters to refine experimental assumptions or reduce the time needed to obtain satisfactory results. There are many methods for 3D RNA structure prediction currently available, but how to select the one that usually returns reliable results?

The assessment of the quality of three-dimensional RNA structures represents a complex chal-

lenge within the realm of structural bioinformatics, and despite numerous existing solutions, a universally reliable method for accurately recognizing native three-dimensional RNA structures remains elusive. The inherently dynamic nature of RNA molecules, coupled with the diverse range of conformations they can adopt, poses a significant obstacle for accurate quality assessment.

One of the fundamental metrics employed in the quality assessment of 3D RNA structures is the root-mean-square deviation (RMSD) of atomic positions [7]. RMSD provides a measure of structural similarity, quantifying the spatial displacement between corresponding atoms in predicted and reference structure. In the context of 3D RNA structures, RMSD serves as a crucial tool for evaluating the accuracy of computational predictions or experimental methods. Lower RMSD score indicates a higher degree of agreement between the predicted and reference structure.

Machine learning methods have already proven their effectiveness in many applications. *AlphaFold* [8] is a deep learning system used for 3D structure prediction of proteins. Its accuracy and performance is equivalent or even outperforms other methods. In the CASP14 protein structure prediction competition [9], *AlphaFold* managed to beat the competition in the category involving new molecules (so called new-folds).

The most popular and best-performing deep learning-based methods encompass a diverse range of architectures. Convolutional neural networks (CNN) excelling in tasks such as image classification, object detection and facial recognition [10] have already been used for quality assessment of 3D RNA structures [11]. Transformer architectures, exemplified by models like GPT [12], have achieved remarkable success in a wide array of natural language processing tasks, leveraging self-attention mechanisms for contextual understanding. Generative adversarial networks (GANs) have emerged as powerful tools for image synthesis [13], while reinforcement learning techniques excel in autonomous decision-making scenarios, such as game playing [14].

Graph convolutional networks (GCNs) are a class of neural networks designed for processing and analyzing graph-structured data [15]. Graphs consist of nodes and edges, where nodes represent entities and edges capture relationships between these entities. GCNs leverage this inherent structure to perform node-level or graph-level tasks.

In conventional neural networks, individual layers process the entire input in isolation. In contrast, graph convolutional networks (GCNs) take into account features of both a node and its adjacent neighbors, enabling the model to capture the relational information inherent in the graph structure. This is accomplished through graph convolution operations that aggregate information from adjacent neighboring nodes, leading to the updating of features of each node in a manner that considers its connectivity within the graph.

Graph attention networks (GATs) [16] are a type of graph neural network architectures which assign varying attention coefficients to different nodes during the aggregation phase, where information is collected from adjacent neighboring nodes. The attention coefficients are determined using a function called the attention function.

This allows GATs to assign importance to neighboring nodes throughout aggregation phase. The input to a single graph attentional layer is a set of node features, $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$, where the number of nodes is given by N, and $\vec{h}_i \in \mathbb{R}^F$, where F represents the number of features in each node. The layer produces an output $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_N\}$, $\vec{h}'_i \in \mathbb{R}^{F'}$. A shared linear transformation, defined by weight matrix $\mathbf{W}$ is applied uniformly to all nodes to elevate input features to a higher-level. Following this, a self-attention mechanism is used on all nodes, with the following attention coefficients

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

indicating the importance of node j's features to node i. The attention mechanism, denoted by a is a single-layer feed-forward neural network characterized by a weight vector $\tilde{\mathbf{a}}$ which employs *LeakyReLU* non-linearity. When expanded, the coefficients calculated by the attention mechanism can be expressed as follows:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp\left(\text{LeakyReLU}\left(\tilde{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\tilde{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_k]\right)\right)}$$

where T represents transposition, $\parallel$ represents concatenation, and $\mathcal{N}_i$ represents the neighborhood of node i in the graph. After calculating the normalized attention coefficients, they are used to calculate a linear combination of the corresponding features, serving as output features for each node, with the possibility of applying some non-linearity, σ:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

Three-dimensional RNA structures can be represented as a graph, prompting the exploration of graph-based machine learning methods, particularly GCNs and GATs, to address the challenges associated with analyzing such complex structures. Given the inherent graph-like nature of three-dimensional RNA structures, the application of graph neural networks holds promise for delivering satisfactory results to the problem at hand.

One such application is the *Atomic Rotationally Equivariant Scorer* (ARES) [17]. It is a deep learning system using graph neural networks for quality assessment of 3D RNA structures. Trained on 18 small RNA molecules, ARES consistently surpasses both human performance and other techniques, yielding the best results. Thus, we decided to compare our developed models against ARES during evaluation process.

In the context of describing 3D RNA structures, various features and properties are essential for comprehensive analysis. Parameters such as distances in both sequence and three-dimensional space, torsion angles and secondary structure information are among the crucial features that precisely characterize three-dimensional RNA structures.

One crucial aspect of developing machine learning models involves the need for appropriate data. The data must be diverse and reliable to ensure optimal model performance. In the case of RNAQuANet, the model specifically requires data on 3D RNA structures. However, acquiring such RNA structures is a non-trivial task.

The Protein Data Bank [19] is the primary source of experimentally determined structure data, but it offers a limited number of standalone RNA structures. Additionally, certain segments of RNA may be intertwined with DNA structures or contain extraneous elements such as ligands or other biological compounds. Consequently, the evolutionary progress of RNA-based solutions is slower compared to DNA.

Before incorporating these structures into the learning process, it is imperative to clean them. While cleaning can be performed in-house, it is a time-consuming process. Fortunately, there are dedicated repositories that provide pre-cleaned RNA structures. The RNAQuANet dataset, for instance, is derived from the RNAsolo database [20], which stores RNA cleaned 3D structures obtained through various experimental methods. The structures in that repository are organized into equivalence classes to supply non-redundant sets for learning purposes [21].

In the context of assessment of three-dimensional RNA structures, one approach involves performing local analysis. This paradigm is rooted in the assumption that the immediate neighborhood of the nucleotide and its structural properties are correlated. Considering this approach is

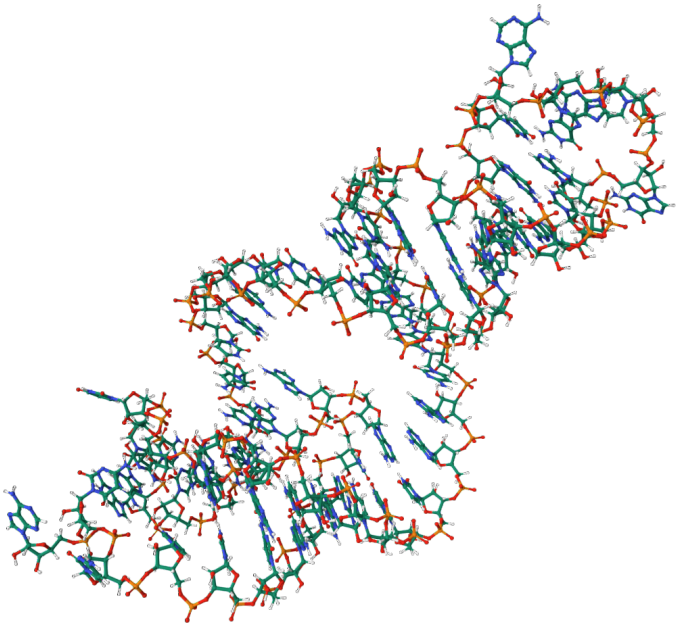


FIGURE 1.1: The example 3D RNA structure (*E. coli* cysteinyl-tRNA and *T. aquaticus* elongation factor EF-TU:GTP ternary complex) visualised in Mol* [18].

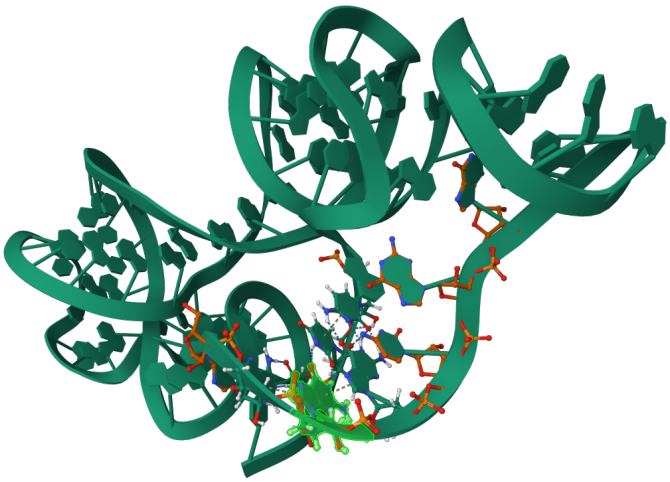


FIGURE 1.2: 3D structure of 16S RIBOSOMAL RNA visualized in Mol* (PDB ID: 1G1X, with atoms located in the immediate neighborhood of the third nucleotide).

useful in the case when conventional dataset containing 3D RNA structures might be too insignificant for deep learning purposes.

Local analysis in the context of RNA structures involves studying specific regions or segments rather than considering entire structure as a whole. These specific substructures are called 3D RNA local descriptors [22]. In our framework, structural descriptors describe nucleotide neighborhood in pre-determined distance of 16 Å within our dataset. Thanks to this approach deep learning techniques can focus on essential features of RNA tertiary structure.

Another approach to consider if conventional learning process might yield unsatisfactory results is the use of transfer learning techniques. Transfer learning [23] is a machine learning paradigm that involves utilizing knowledge acquired from solving one task and applying it to a different but related task. Instead of training a model from scratch for a specific task, transfer learning begins with a pre-trained model on a source task and fine-tunes it for the target task. This approach is particularly effective when the source and target tasks share common patterns or features, allowing the model to capitalize on the learned representations.

Transfer learning is especially useful when the target task lacks sufficient labeled data, as the pre-trained model can bring valuable knowledge from a data-rich source task. The method is commonly applied in diverse domains, including natural language processing, computer vision and speech recognition, where large pre-trained models have shown significant improvements in performance and efficiency when adapted to specific applications.

1.1 Scope of the work

The scope of the thesis encompasses design, training, evaluation, and implementation of graph neural network models with the primary goal of efficient and reliable quality assessment of spatial 3D RNA structures, as well as construction of a representative, diverse, and non-redundant training set including three-dimensional RNA structures.

The crucial problem relies on an adequate representation of three-dimensional RNA structures effective for training deep neural networks and applying different graph neural network-based models to reliably assess the quality of structures considered. Furthermore, we aim to develop a fully-automated pipeline that can process any 3D RNA structure and evaluate its quality using proposed models. We also analyze the architecture and the significance of its components and perform hyperparameter optimization for promising models. Finally, an user-friendly user interface will be developed to allow users to assess quality of their 3D RNA structures with models we aim to provide.

1.2 Structure of the work

The diploma thesis consists of 10 chapters.

Chapter 1 focuses on stating the problem and describing the importance of assessing the quality of three-dimensional RNA structures. We describe some advancements made using deep learning techniques and how graph neural networks can be effectively applied for analysis of 3D RNA structures.

Chapter 2 presents the methods used to prepare a representative, diverse, and non-redundant training dataset. It contains information about how we performed prediction and extraction of RNA structure properties. It also describes the tools we used and presents measurable statistics describing the training data.

Chapter 3 focuses on extracting meaningful features from the input dataset and how they are represented onto a graph-based structure to be applied in the learning process. It lists all the node- and edge-based properties used by our graph representation of three-dimensional RNA structures and how each property is transformed and represented in graph neural networks.

Chapter 4 describes the graph neural network architectures we experimented with. It explains the representation of the graph in *PyTorch Geometric* [24] and how node features in a graph neural network are updated during the training process. Additionally, it provides an insight to architectures we used.

Chapter 5 presents differing approaches we used concerning the learning process. It describes the contents of each file representing a distinct dataset (train, validation, and test subset), as well as different training processes on small and large datasets independently and when a transfer learning approach was finally applied.

Chapter 6 describes the methods and metrics used during evaluation of our models. It goes in depth into what is the accuracy of the models trained on different variations of the input datasets, additionally considering descriptors' dataset of variable segment length.

Chapter 7 contains in-detail analysis of the results obtained. Effective assessment of the model's performance is an essential step. Using comparative analysis, we compared different approaches we have used, especially evaluating differences in accuracy and performance with ARES, another model focused on quality assessment of three-dimensional RNA structures.

Chapter 8 describes our work in the context of technological decisions and tools we used. It focuses both on how we conducted work on our model, using *Python* command line interface scripts and *PyTorch Geometric*, during implementation of graph neural networks. Moreover, it also describes the technology we used during the development of the web application we provided as a way to evaluate 3D RNA structures using our model.

Chapter 9 presents the web application and its user interface. The demo interface allows one to upload their own files with 3D RNA structures. The chapter focuses on presenting the application and contents of the results page. It also provides an introduction to the command line interface.

Chapter 10 contains the summary of our work performed within the thesis, as well as an URL pointing to a GitHub repository where all the learning scripts and the models developed were published.

1.3 Work distribution

The successful completion of the project was a collaborative effort involving contributions of four team members, each specializing in distinct aspects. Additionally, the team collaborated closely on significant tasks that required joint efforts. The work distribution is summarized as follows:

- **Bartosz Adamczyk**:
  - Development and supervision of the workflow for creating training datasets and its parallelization.
  - Creation of ARES Docker Image crucial for the evaluation.
  - In-detailed analysis of training datasets.
  - Preparation and parallelization of feature extraction pipeline.
  - Evaluation and benchmarking the ARES model.

- **Maciej Biliński**:
  - Project management: repository configuration, tasks assignment, code review.
  - Preparation of the development environment: creating a Docker image, developing the project architecture, introducing a configuration file, and "How to Use" documentation.
  - Implementation of parameterizable graph neural network architecture as well as efficient loading mechanisms, including stream feature.
  - Experimenting with neural network parameters and training several models on different datasets.
  - Evaluation and comparison among the models developed in terms of their accuracy.
- **Mikołaj Bartkowiak**:
  - Managing command line interface scripts, implementation of argument parsing framework in early stages.
  - Contribution to the preprocessing and creation of the pipeline, mostly by optimizing developed scripts to ensure efficiency.
  - Keeping the codebase clean by formatting and restructuring the code, adding type annotations and comments.
  - Working on the early stages of containerizing the application.
  - Involvement in analysis of the datasets developed within the thesis.
- **Szymon Stanisławski**:
  - Development of the web user interface for the project.
  - Development of the API (Application Programming Interface) connecting the web interface with the processing layer.
  - Implementation of a database to store and manage user tasks' data within the API.
  - Integration of a job queue system to enhance the efficiency of asynchronous tasks within the API.
  - Efficiency optimization of the whole application's processing.

Chapter 2

Preparation of a diverse training set

Establishing a representative, diverse, and non-redundant dataset is a crucial component for the future success of a model. The diversity can be achieved in at least two possible ways:

- by using significant amount of existing experimentally determined 3D RNA structures and their in silico predictions allowing us to compute the RMSD between them,
- by molecular dynamics simulations performed on existing experimentally determined 3D RNA structures.

The dataset used in the *Atomic Rotationally Equivariant Scorer* (ARES) [17] is the sole dataset that fulfills the criteria, but it was imperative to create an artificial set. Consequently, the RNAQuANet dataset is established from a non-redundant collection of 3D RNA structures [21]. This approach helps mitigate the risk of model overfitting.

From the set of non-redundant RNA structure representatives downloaded from the RNAsolo [20], structures were processed in a specific way to achieve diversity:

- 3D RNA structure resolution has to be less than or equal to 3 Å to ensure high-quality,
- at least two nucleotides have to form base pairing,
- at least one nucleotide has to be unpaired,
- amount of paired nucleotides in the structure has to be greater or equal than the amount of unpaired nucleotides,
- filter out structures of size less than 10 or greater than 200.

737 out of 1840 structures passed through the filtering process. Each structure was subject to minor modifications to create the desired target 3D structure.

2.1 Generating representative dataset of 3D RNA structures

Preparation of a training dataset by modifying data entities requires specific tools capable of making subtle adjustments. When it comes to predicting alternative conformation of RNA structures, the best approach involves utilizing 3D RNA structure prediction tools. These tools predict 3D RNA structures based solely on the secondary structure, making them versatile and efficient. Additionally, their results rather closely correspond to the align with reference. While this behaviour might not be ideal for their original purpose, it is well-suited for our problem, where modifications are plausible and likely to feature user expectations.

For *RNAComposer* [3], which requires secondary structure at the input, the structure can be extracted either from a reference structure or can be computationally predicted based on the reference sequence.

Secondary structure extractor (*RNApdbee*) and predictors [25] used during the dataset preparation include: *RNAfold*, *Contrafold*, *ContextFold*, *CentroidFold*, *IPknot*, *RNAstructure*, *RNAshapes*, *HotKnots*.

Employing a diverse range of predictors enables *RNAComposer* to produce a broader range of results, as these predictors exhibit variations in accuracy during base-pair predictions. This diversity is a crucial step in generating a highly varied structure dataset.

FIGURE 2.1: Mol* illustration of Crystal Structure of the *E. coli* Aspartyl-tRNA Synthetase:Yeast tRNAasp:aspartyl-Adenylate Complex (PDBID: 1IL2, chain: C) with generated (orange) and reference (green) structures depicting RMSD: 16.222 Å.

The method generates up to 30 3D RNA structures for each reference and evaluates them using the *RNAQUA* tool, developed for the RNA-Puzzles contest [26]. *RNAQUA* calculates the RMSD for each predicted 3D model and stores the results in XML format for each reference structure. The tool considers only normalized PDB file atoms considered in the RNA-Puzzles contest. Specifically, it retains atoms for the bases (C2, C4, C6, C8, N1, N2, N3, N4, N6, N7, N9, O2, O4, and O6) and for the sugar-phosphate backbone (C19, C29, C39, C49, C59, O29, O39, O49, O59, OP1, OP2, and P) [27].

This process is depicted in Figure 2.2.

2.2 Detailed analysis of the dataset

The RMSD value cuttoff used for predicted 3D models is equal to 60 Å in order to eliminate outliers from the dataset.

Final range of the RMSD values and sequence length are provided in Table 2.1.

Figures 2.3 and 2.4 depict histograms illustrating that the generated dataset includes a range of RMSD values, encompassing both high-quality and poorly predicted 3D RNA structures. This

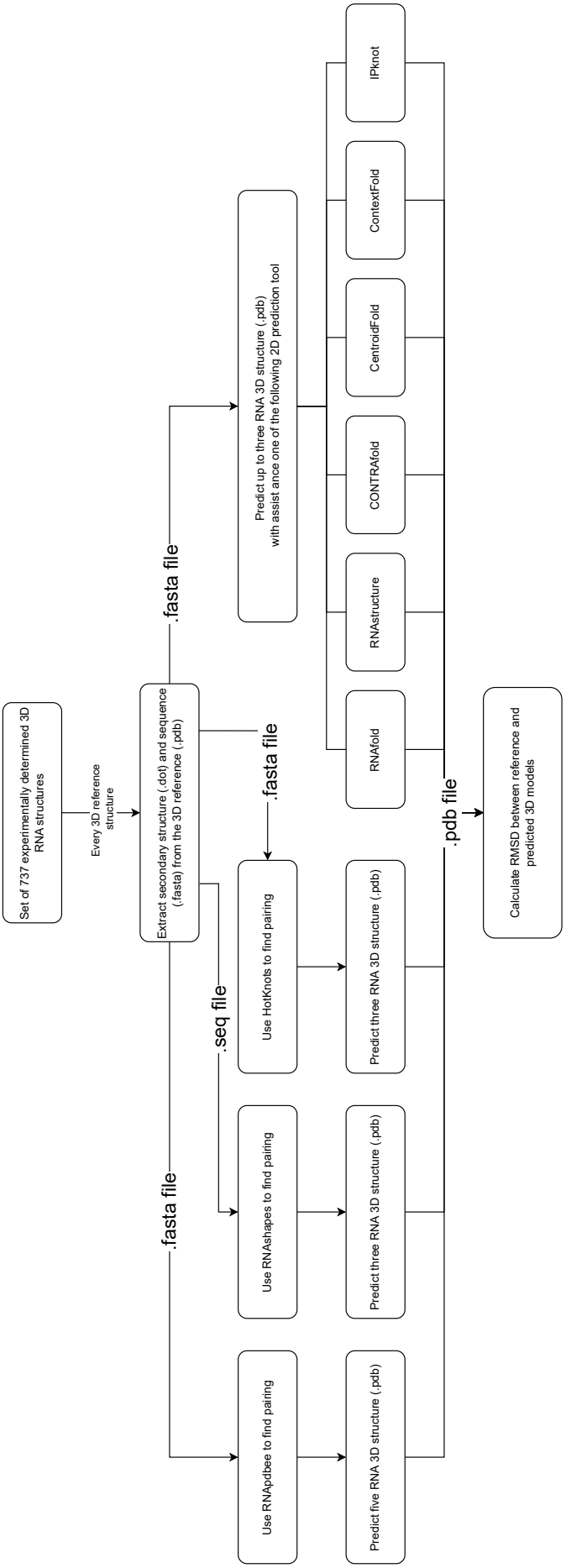


FIGURE 2.2: A representative, non-redundant dataset generation workflow.

Parameter	Min	Max
RMSD	0.268	58.753
Sequence length	10	190

TABLE 2.1: Range of values for RMSD and sequence length.

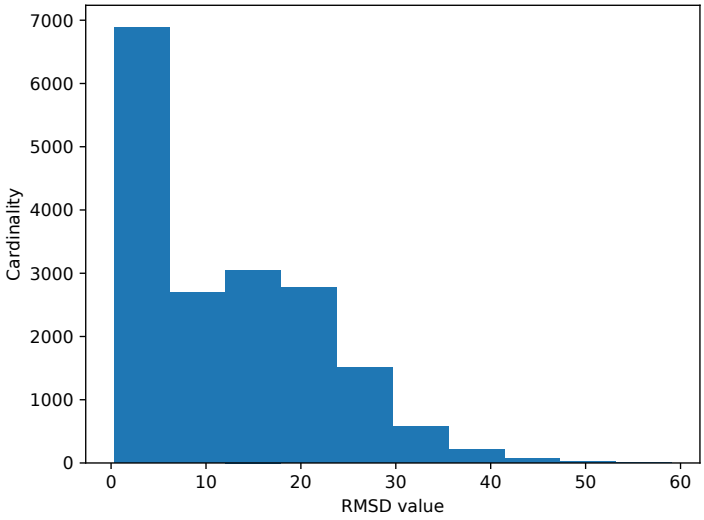


FIGURE 2.3: Distribution of the number of predicted 3D models in the context of their RMSD scores (in range 0-60, with step 5).

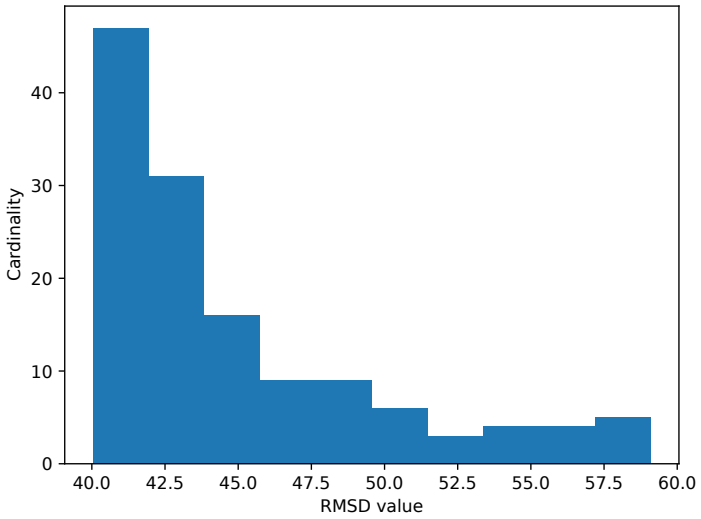


FIGURE 2.4: Distribution of the number of predicted 3D models in the context of their RMSD scores (in range 40-60, with step 2.5).

diversity could enable us to train valuable machine learning models. The majority of the dataset consists of structures with small RMSD values, forming the essential core that faithfully represents the correct fold of RNA structures. Values below 3 Å are considered to be in close proximity to the correct structure.

The RNAQuANet training dataset contains 3D RNA structures of various length, with the intention of covering multiple possible variations of RNA structure folds.

Figure 2.7 depicts a distribution of the RMSD scores (0-60 Å) of predicted 3D models in the

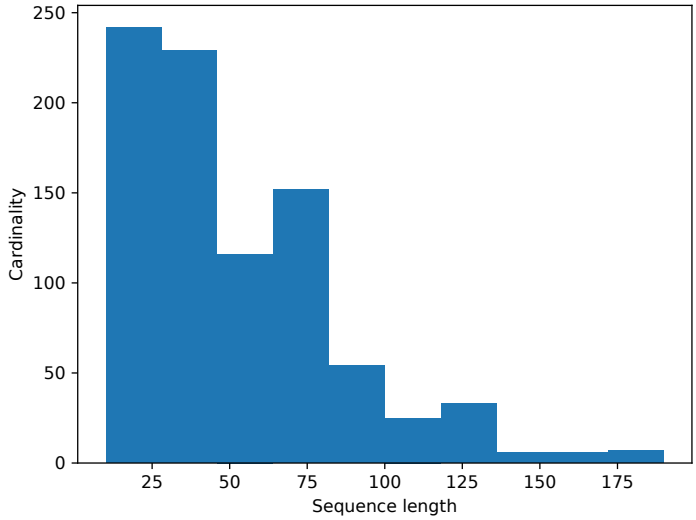


FIGURE 2.5: Distribution of the number of predicted 3D models in the context of their sequence length (in range 10-190, with step 25).

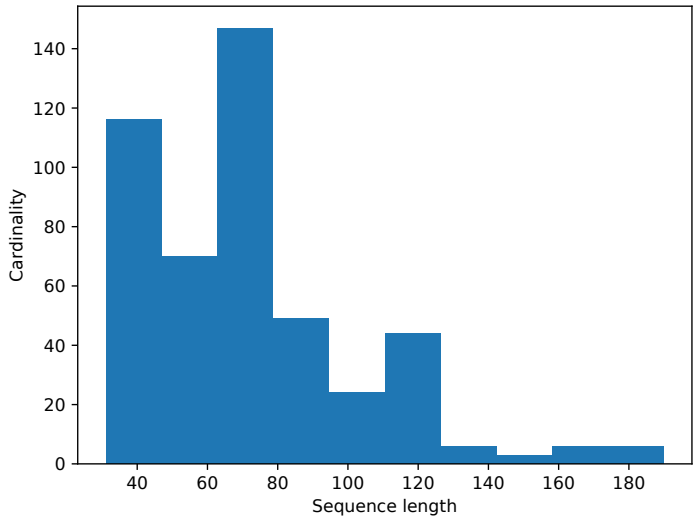


FIGURE 2.6: Distribution of the number of predicted 3D models in the context of their sequence length (in range 40-190, with step 20).

context of their sequence length (10-190).

Additional two side plots on the top and the right side show concentration of dots with the particular sequence length and RMSD score, respectively. The stripes are result of diversity of the predicted 3D models within the context of the corresponding references.

The plot in Figure 2.8 depicts target structure aggregation by their reference structure using resulted RMSD median.

2.3 Training dataset splitting

The training dataset was divided into three subsets: training, validation, test. Training and validation were used during learning process for adjustment and the progress monitoring. Test set

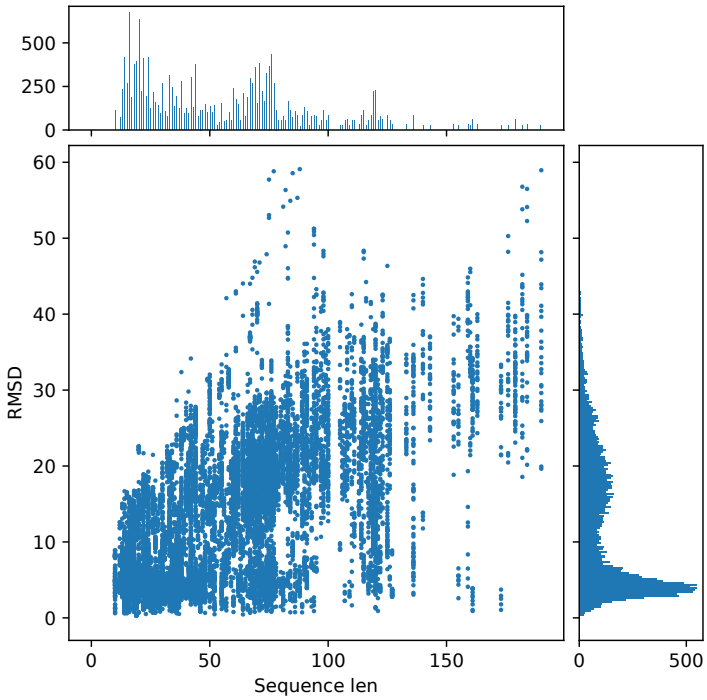


FIGURE 2.7: Distribution of the RMSD scores (0-60 Å) of predicted 3D models in the context of their sequence length (10-190) [plot in the middle]. Plots on top and right side represents concentration of dots with the particular sequence length and RMSD score, respectively.

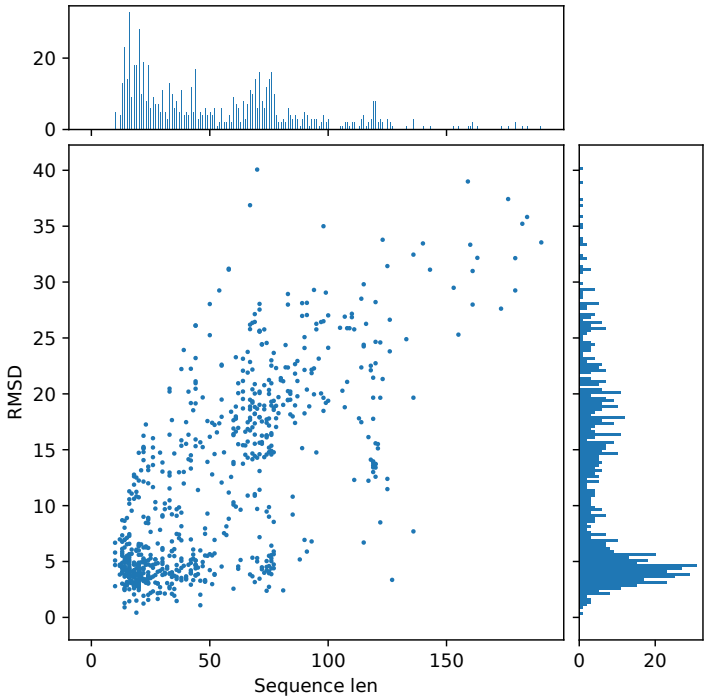


FIGURE 2.8: Distribution of the median RMSD scores (0-40 Å) of predicted 3D models in the context of their sequence length (10-190) [plot in the middle]. Plots on top and right side represents concentration of dots with the particular sequence length and RMSD score, respectively.

was prepared for the final evaluation of the models developed.

Each subset is designed to contain structures that exhibit a wide range of characteristics. The example provided aims to characterize structures based on their sequence length and median RMSD score computed within the context of the reference. These datasets were grouped into twenty five clusters ($k = 25$) using non-supervised *k-means* algorithm (Figure 2.9). This approach facilitated the division of data into groups containing structures with similar characteristics in terms of length and median RMSD. From each cluster, the reference structures were randomly selected to create the final dataset, splitting it into training, validation, and testing sets in a ratio of 6:2:2, respectively.

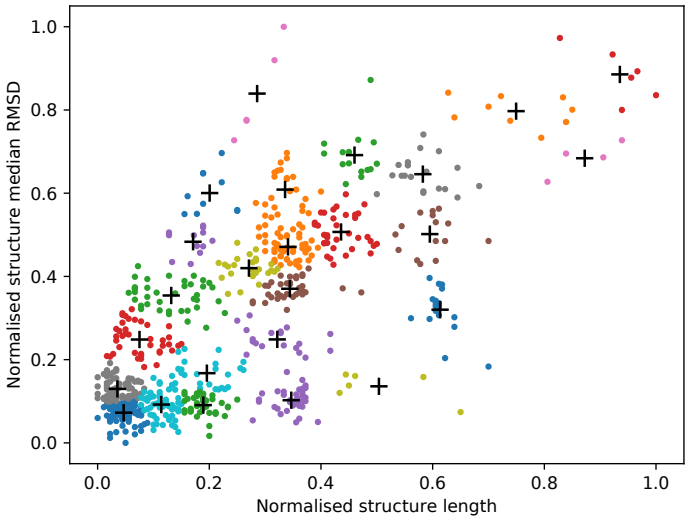


FIGURE 2.9: Clustering results for predicted 3D RNA models using *k-means*.

Table 2.2 provides the final number of groups of reference and the corresponding predicted 3D models for each set: train, validation, and test.

Subset	References	3D models
Train	455 (61.8 %)	10890 (61.2 %)
Validation	161 (21.8 %)	3968 (22.3 %)
Test	120 (16.3 %)	2932 (16.4 %)

TABLE 2.2: Number of references and their corresponding 3D models of each set.

2.4 Data distribution in subsets

We present plots illustrating the distribution of data in considered subsets of the dataset developed, namely the training set (Figure 2.10), validation set (Figure 2.11), and test set (Figure 2.12). Each figure provides insights into a distribution of the median RMSD scores of predicted 3D models in the context of their corresponding sequence length.

Analyzing the distribution of the median RMSD score with respect to structure length in each subset is essential for high-quality training models and ensuring the collection of reliable and representative data for robust model performance.

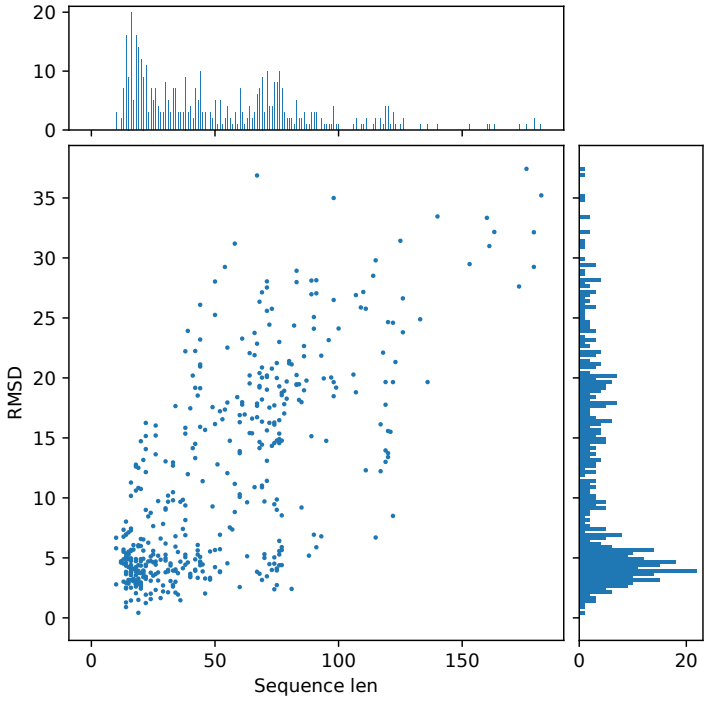


FIGURE 2.10: Distribution of the median RMSD scores (0-35 Å) of predicted 3D models in the context of their sequence length (10-190) in the training set.

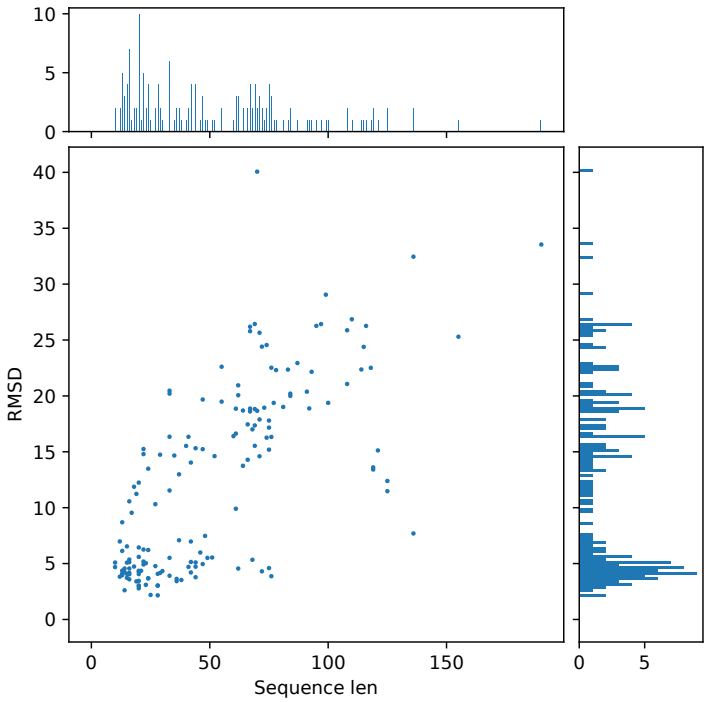


FIGURE 2.11: Distribution of the median RMSD scores (0-40 Å) of predicted 3D models in the context of their sequence length (10-190) in the validation set.

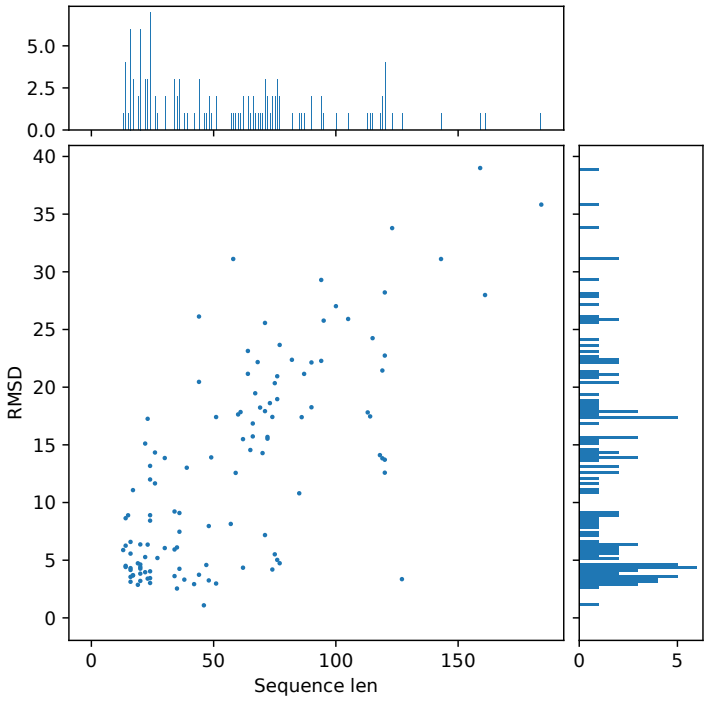


FIGURE 2.12: Distribution of the median RMSD scores (0-40 Å) of predicted 3D models in the context of their sequence length (10-190) in the test set.

2.5 3D RNA structure descriptors

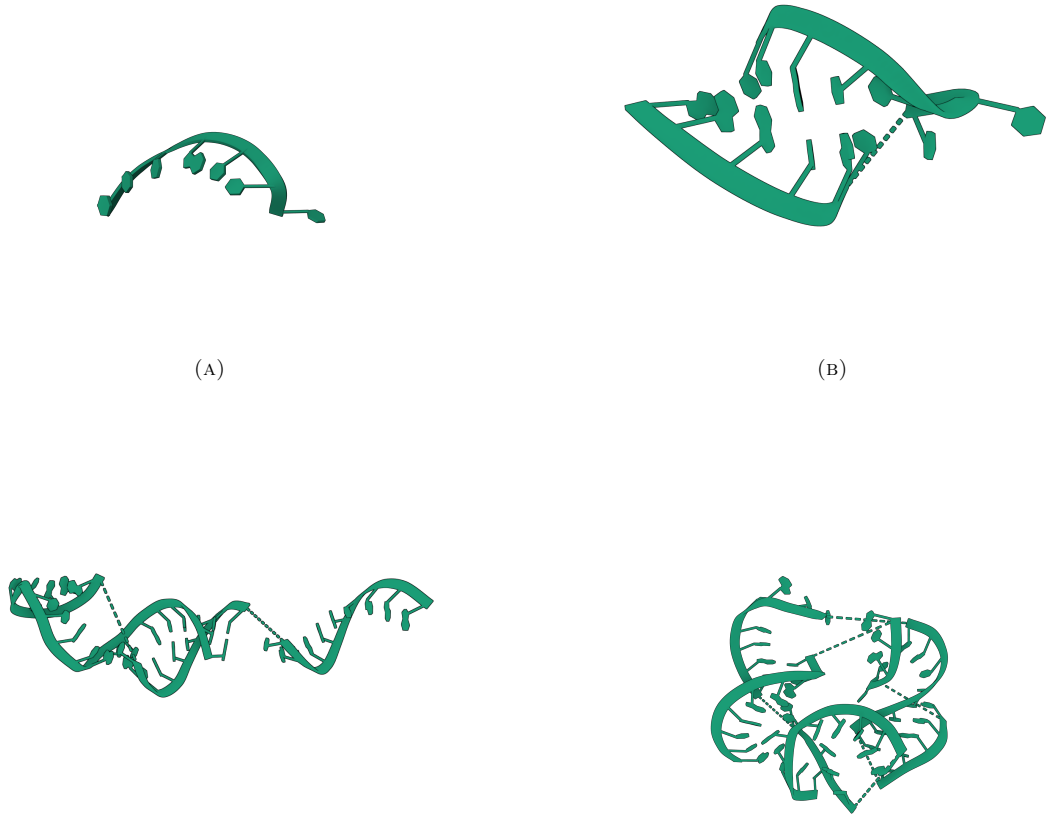
The RNAQuANet descriptors dataset was prepared for scenarios in which the basic training dataset might prove challenging because of insufficient size for a successful application of deep learning. This approach is assumes that the environment surrounding a nucleotide directly affects its properties. The algorithm (`descs-standalone`) [22] processes and generates separate PDB files containing local 3D RNA structure descriptors from the third to the third-from-last nucleotide of the particular RNA molecule. These neighborhoods are defined by in-contact residues located in the spatial proximity represented by the distance between corresponding C5′ atom. Residues that meet this criterion are considered by the descriptor.

This approach can result in multiple segments, which are fragments of the original structure located in the spatial proximity of the central residue.

To prevent information redundancy during learning, it is essential to not only split the dataset into training, validation, and test subsets, but also into subsets containing different number of segments.

The RNAQuANet descriptors dataset was divided into:

- one-segment,
- two-segment descriptors,
- three and more segments ones.



(A) (B)

(C) (D)

FIGURE 2.13: Example of: (A) one-segment descriptor, (B) two-segment descriptor, (C) two-segment descriptor, and (D) seventeen-segment descriptor.

2.5.1 Processing pipeline for extracting descriptors

The RNAQuANet descriptors dataset was created by utilizing 3D RNA structures considered in the RNAQuANet structures dataset.

Reference structure 3D structure of local RNA descriptor representing its spatial proximity was built. A pair of residues are in-contact in 3D space when the distance between their C5′ atoms does not exceed 16 Å. For each descriptor 3D structure from the structures dataset, the algorithm selects the corresponding nucleotides from every 3D model and calculates the RMSD score. This approach enable us to generate a substantial number of the RMSD values are computed.

After filtering outliers, identified as descriptors with RMSD values exceeding 26 Å, the entities in each subset, categorized by the number of segments, were as follows:

- 141,634 (16.27%) one-segment descriptors,
- 370,422 (42.56%) two-segment descriptors,
- 358,311 (41.17%) three and more segments descriptors.

The respective training, validation and test sets follow the same distribution as in the RNAQuANet structure dataset.

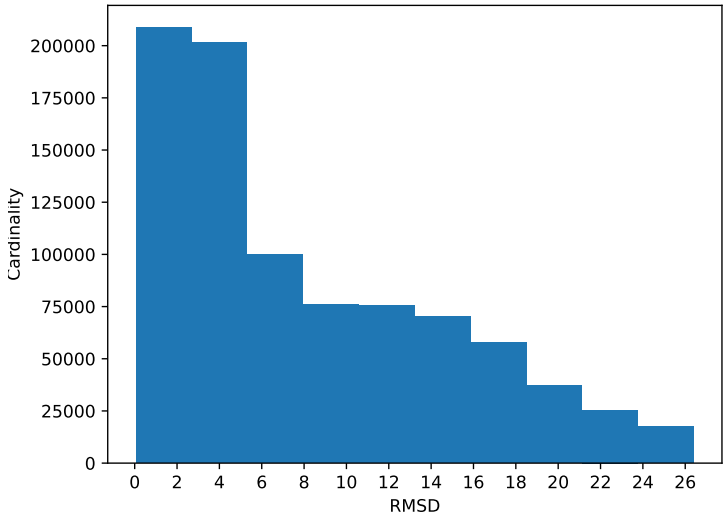


FIGURE 2.14: Distribution of the number of all local 3D structure descriptors in the context of their RMSD score computed within the context of the reference 3D structure descriptor (in range 0-26, divided to 10 bins).

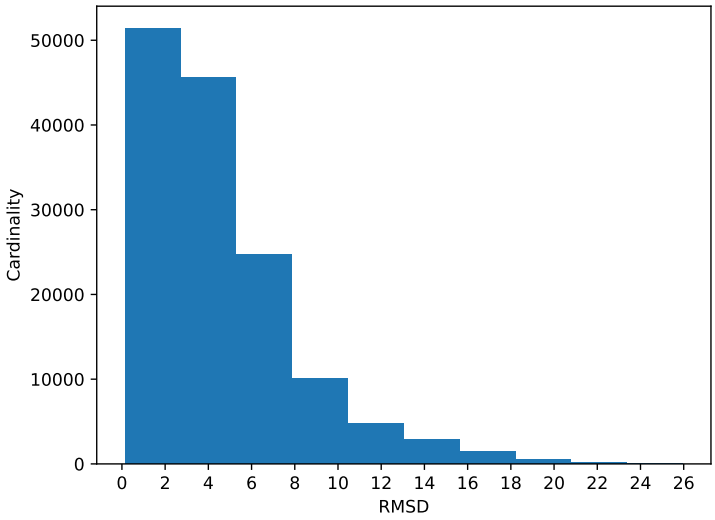


FIGURE 2.15: Distribution of the number of single-segment local 3D structure descriptors in the context of their RMSD score computed within the context of the reference 3D structure descriptor (in range 0-26, histogram divided to 10 bins).

The cardinality of each subset with respect to the number of segments were provided in Table 2.3.

Subset	One segment	Two segments	Three and more segments
Train	90,091 (16.38 %)	229,990 (41.81 %)	229,933 (41.80 %)
Validate	29,276 (16.26 %)	80,218 (44.57 %)	70,457 (39.15 %)
Test	22,267 (15.86 %)	60,214 (42.88 %)	57,921 (41.25 %)

TABLE 2.3: Cardinality of considered subsets of the descriptor-based training dataset.

The plots (Figure 2.14, 2.15, 2.16 and 2.17) illustrate that training needs to be conducted separately for descriptors containing three or more segments.

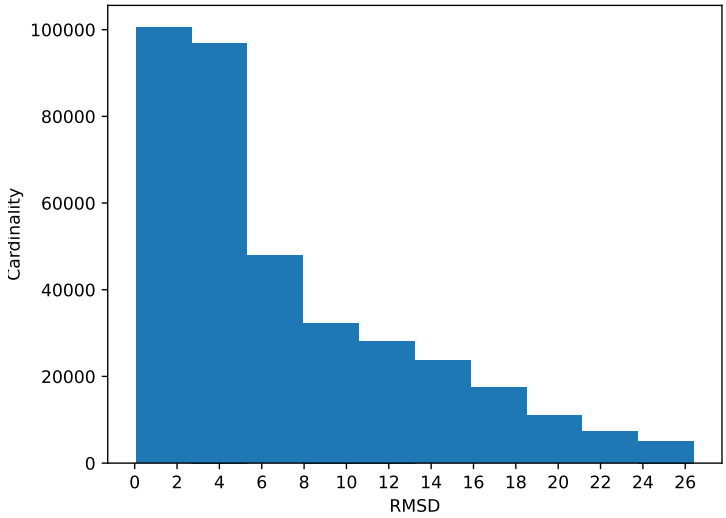


FIGURE 2.16: Distribution of the number of two-segment local 3D structure descriptors in the context of their RMSD score computed within the context of the reference 3D structure descriptor (in range 0-26, histogram divided to 10 bins).

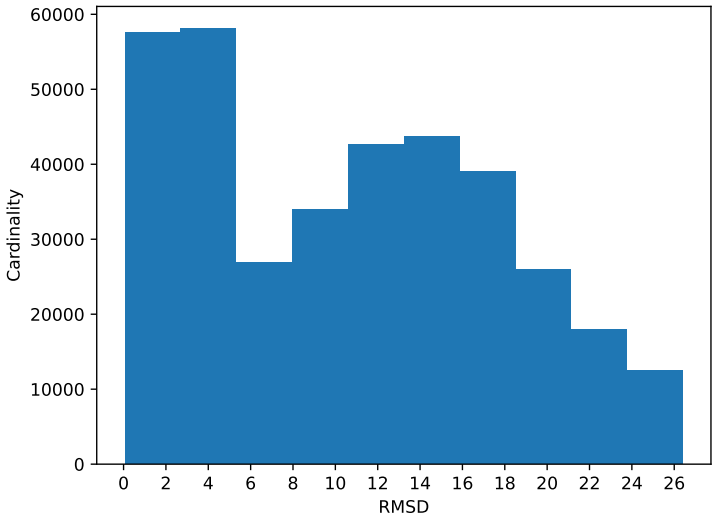


FIGURE 2.17: Distribution of the number of three-segment and more local 3D structure descriptors in the context of their RMSD score computed within the context of the reference 3D structure descriptor (in range 0-26, histogram divided to 10 bins).

2.6 Missing data handling

When considering all possible features related to torsion and bond angles, there may be situations where a specific angle is not defined, resulting in a missing (NaN) value.

This issue is associated with the angle's origin. Some angles rely on certain nucleobases.

Our team decided not to set a default replacement for missing values to avoid limiting the dataset's potential applications in the future.

Chapter 3

Selected features to describe 3D RNA structures

Feature extraction is a crucial step in the field of machine learning, playing a significant role in transforming raw data into a format that is suitable for training and analysis of a model. The essence of the problem lies in identifying and selecting relevant information from the original data, simplifying its representation while retaining essential characteristics and structure of 3D RNA structures.

While considering feature extraction, there are two important points to consider:

- dimensionality reduction, as high-dimensional data with a lot of features poses a challenge known as the curse of dimensionality: as the dimensionality increases, the volume of the data space grows exponentially, making it significantly more difficult for machine learning models to generalize well,
- noise reduction, as raw data often contains irrelevant or redundant features that do not contribute meaningfully to the learning process. Furthermore, these features may introduce noise, making it harder for the model to recognize underlying patterns.

Achieving both dimensionality reduction and noise reduction, while retaining crucial characteristics of the input data, is a delicate balancing act in the realm of feature extraction. It involves making decisions to simplify the dataset for the learning process while ensuring that all critical information about the data in question is retained. Obtaining this tradeoff between mentioned points involves identifying and retaining features that have significant contribution to the learning process, ensuring the extracted subset captures the essential patterns within the data [28].

In graph neural networks, features are represented as node and/or edge attributes. Nodes represent entities, whereas edges represent relationships taking place between entities. Each node and/or edge is associated with a feature vector that encodes information about the entity and/or its adjacent neighborhood.

In *PyTorch Geometric* [24], each graph is represented by a single `Data` object, which in our case is described by four attributes:

- `x` – a tensor representing nodes in the graph and their features. Each row corresponds to a node, and each column corresponds to a feature dimension for that node.
- `edge_index` – a tensor representing edges in the graph. Each column of `edge_index` corresponds to the particular edge, and the pair of values in that column includes the indices of

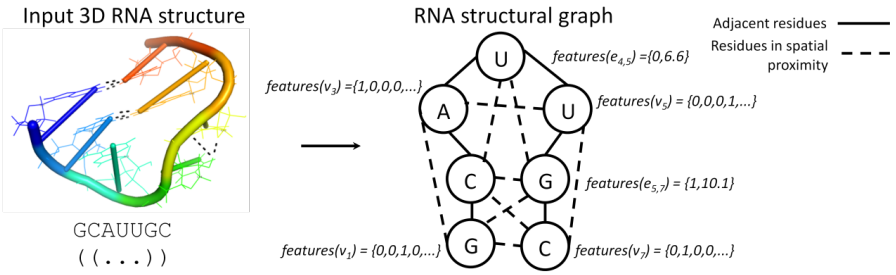


FIGURE 3.1: Example 3D RNA structure and its graph-based representation.

the nodes that form the edge. In *PyTorch Geometric*, all graphs are directed graphs, so if a given graph is undirected, then both $(u, v) \in$ `edge_index` and $(v, u) \in$ `edge_index`, where u, v are adjacent nodes.

- `edge_attr` – a tensor representing features associated with each edge in the graph. Each row corresponds to the particular edge, and each column corresponds to a feature dimension for that edge.
- `y` – a tensor containing graph-level labels (optional). It is used where the goal is to predict a label or property for the entire graph.

In the context of three-dimensional RNA structures, we will consider both node and edge features.

Specifically, each node in the graph corresponds to a single nucleotide in the RNA sequence, and edges are established based on spatial proximity of relevant residues (Figure 3.1). For our label, we will use the expected RMSD value for a given structure. The way in which RMSD scores were computed was described in Chapter 2.

3.1 Node features

In our model, each node represents a single nucleotide. Every residue is depicted by its corresponding distinctive features. They not only distinguish one nucleotide from another, but also provide a solid foundation for application within the learning process.

Node features:

- nucleobase – one of the four nitrogenous bases: either A, C, G, or U, represented using one-hot encoding,
- C1′ atom coordinates,
- all nucleotide bond angles,
- all nucleotide torsion angles,
- base pairings – represented using one-hot encoding.

3.2 Edge features

In our model, edges are formed when two residues are closely located in three-dimensional space. We adopted a convention where an edge is formed between two residues if the Euclidean distance in three-dimensional space does not exceed 16 Å – this means that a pair of residues is in-contact in 3D space. This value can be altered in the configuration file if one prefers to use different distance for considering spatial proximity during the feature extraction process.

Thus, we utilize the following edge features:

- sequential distance: the number of nucleotides located in-between the pair of residues,
- Euclidean distance between the pair of residues in three-dimensional space, up to 16 Å.

All spatial distances were calculated using coordinates of the atom C1′ of considered nucleotides.

3.3 Preprocessing pipeline for extracting structural features

Before running the proper feature extraction process, each PDB file undergoes filtering, so that only lines starting with `ATOM` are considered. This ensures that only relevant information essential for feature extraction is retained, enhancing both time and memory efficiency during further processing.

Next, using *RNAgrowth* [29], we extracted features from filtered PDB files representing predicted 3D models or all local descriptors extracted from them. We use atom C1′ for distance calculation. This value can be altered in the configuration file if one prefers to use different atom for distance and coordinate calculation.

For spatial proximity, we consider residues in the vicinity of 16 Å in the three-dimensional space, using Euclidean distance as a metric (further elaboration on this topic is presented in Section 3.2).

The results are represented by the following files in the tab-separated values format:

- `3dn` – matrix of inter-residue distances in three-dimensional space. Specifically, as the generated filename's suffix is `_16.0.3dn`, that means we consider distance between residues only up to 16 Å.
- `ang` – represents the bond angles of the sugar-phosphate backbone of RNA,
- `atr` – represents all torsion angles;
- `bon` – represents all bond distances of the considered 3D RNA structure,
- `sqn` – represents the immediate predecessor and successor of each residue in the sequence.

Furthermore, using *PDBParser* we extracted coordinates from the filtered PDB file. Next, using *RNApolis* annotator script we extracted secondary structure information from the analyzed 3D structure. It is done by creating a mapping between 2D and 3D structures, and then extracting the 2D structure in dot-bracket notation. Dot-bracket notation is a concise representation of the secondary structure of RNA molecules where dot represents unpaired nucleotides, open bracket represents the opening residue of the base pair, and closed bracket represents the closing residue of the base pair (Listing 3.1). In our model, we consider pseudoknot order by mapping different types of brackets onto separate and distinct categories, using one-hot encoding.

`Data` attribute	Description	Features
`x`	Nodes and their features	Nucleobase
		Coordinates (C1′ atom)
		Bond angles
		Torsion angles
		Base pairings
`edge_index`	Edges	–
`edge_attr`	Edge features	Euclidean distance (C1′ atoms)
		Sequential distance
`y`	*Label*	*RMSD*

TABLE 3.1: Summary of all attributes and features for a single *PyTorch Geometric* `Data` object representing a structure; optional attributes are represented in *italic*.

```
>1B23_1_R
gGCGCGUAACAAAGCGGAUGUAGCGGAUGCAACCGUCUAGUCCGGCGACUCCGGAACGCGCCUCCA
((((((((.......(((((...(((((......))))))...))))))..((....))..)))))))....
((((((((......((((.(((((.....))))).))))....((((((.....))))))))))))))....
((((((((.......((((.((.((((((..........))))).)).))))))..))))))))....
((((((((.......(((((...(((((......))))))...))))).....)))))))....
```

LISTING 3.1: Sequence *1B23_1_R* and its 2D structure in dot-bracket notation. Depicted are the first four lines.

3.4 Final form of the data

The final form of the training data of a single structure is given in Table 3.1.

The data is then saved to a HDF5 file format [30]. HDF5, which stands for *Hierarchical Data Format version 5*, is a file format particularly well-suited for storing and organising large amounts of data. HDF5 organizes data in a hierarchical structure similar to a file system. Its flexibility allows to store data of any type, including complex ones, supporting accurate and succinct representation of our preprocessed training data. Its wide adoption and acceptance in the scientific community, resulting in wide availability of libraries and tools supporting the format was one of primary reasons for choosing HDF5 as our storage format.

After processing all of the structures within a given 3D RNA structures collection, the respective HDF5 files are then concatenated to form a full, distinct dataset fully prepared to be used during the learning process. The structure of such HDF5 file is depicted in Figure 3.2. Performing the full preprocessing pipeline on the input dataset yields three HDF5 files: `train.h5`, `val.h5`, and `test.h5`, containing training, validation, and test set, respectively.

3.5 Assessing the set of features

To summarize our graph representation of a three-dimensional RNA structure, each node represents a nucleotide and has the following features associated with it: nucleobase, its representative coordinates, bond angles, torsion angles, and secondary structure information. Furthermore, we consider an edge between two nodes if and only if the Euclidean distance between the nodes in three-dimensional space is less than a given threshold, which we set at 16 Å.

Considering another approach, the *Atomic Rotationally Equivariant Scorer* (also known as ARES) [17] comprises of larger input space, but specific nodes have much smaller feature dimensionality. This is due to the fact that ARES uses individual atoms to represent nodes of the graph, and considers only atom type and 3D coordinates of neighboring atoms as node features.

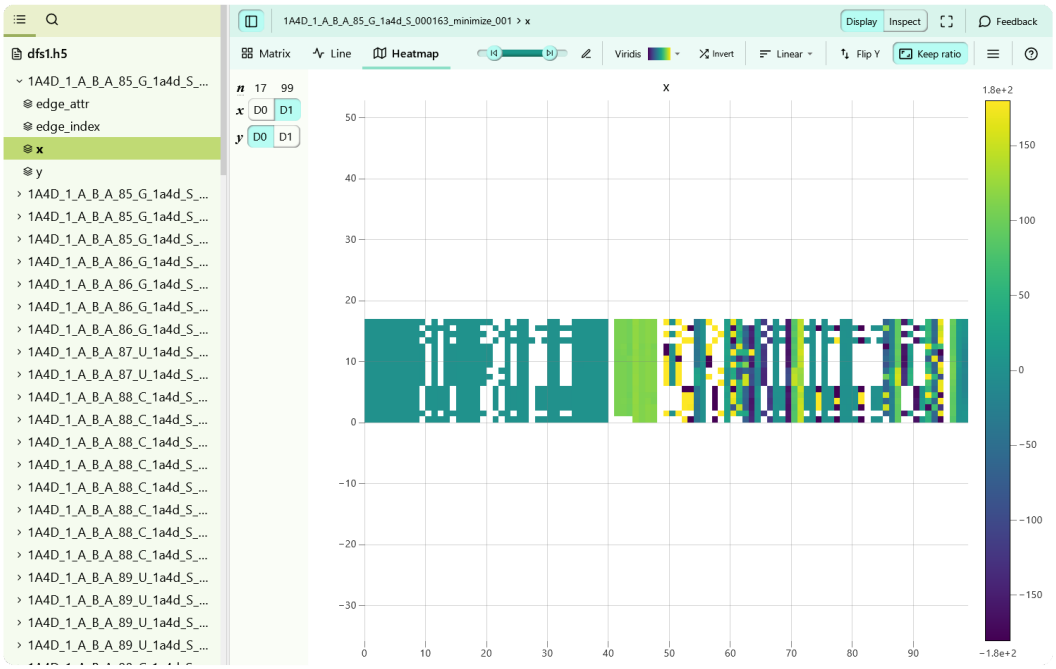


FIGURE 3.2: Visualization of HDF5 file using *myHDF5* tool [31] for exploring and visualizing HDF5 files. The depicted HDF5 file contains multiple structures depicting one-segment descriptors. Each structure has four attributes: `x`, `edge_index`, `edge_attr`, `y`. The matrix shows values of `x` – node features – of structure *1A4D_1_A_B_A_85_G_1a4d_S_000163*.

In our graph representation, we use nucleotides for nodes in the graph. Our approach goes beyond this basic foundation by incorporating additional node features, such as nucleobase composition, planar and torsion angles, and secondary structure information. By integrating these features, our graphs become more compact. This approach not only enhances the complexity of our representation but also has the potential to allow for a more thorough understanding of the underlying biological structure during the learning phase.

Chapter 4

Graph Neural Network-based model architecture

Three-dimensional RNA structures exhibit inherent structural complexities and spatial relationships among individual nucleotides, resembling a graph-like topology. Each nucleotide can be viewed as a node, whereas interactions between nucleotides such as sequential adjacency or spatial proximity form edges, shaping the structural framework of the RNA molecule. This graph-like nature of RNA structures provides a compelling basis for using graph neural networks (GNNs) in the realm of their quality assessment.

Graph neural networks are a class of neural networks designed to work with graph-structured data. Nodes represent entities, whereas edges represent relationships or interactions between entities. GNNs aim to learn meaningful representations of graphs, capturing the underlying structure and relationships within the data.

There are many different problem types associated with GNNs, including but not limited to:

- node classification, in which the goal is to assign a label to each node in the graph: being given the ground-truth labels for a small subset of nodes, the task is to infer the labels for all remaining nodes,
- graph-level classification, in which the objective is to assign discrete class labels to entire graphs based on their structural properties,
- graph-level regression, in which the goal is to predict a single continuous value for the entire graph.

In the context of quality assessment of three-dimensional RNA structures using graph neural networks, the problem clearly fits to graph-level regression. Specifically, we are aiming to predict a single continuous value (i.e., root-mean-square deviation score) for an entire three-dimensional structure represented using a graph. The exact graph representation and properties used to describe 3D RNA structures were described in Chapter 3.

4.1 Graph Neural Network Foundations

Our architecture operates as a graph regression network (GRN) with the objective of predicting a numerical value by leveraging a graph-based representation of the input data. This aligns with the established usage of the terms within the field of deep learning.

In the context of our problem, we are employing three distinct components:

- *GCN convolutions*, using `GCNConv` layers with fixed convolutional kernel to compute node representations [15];
- *GAT convolutions*, applying `GATConv` layers which utilize the concept of attention mechanism to assign varying importance to each node's neighbors [16];
- *GAT transformer convolutions*, using `TransformerConv` layers which incorporates both feature and label propagation and applies a masked label prediction strategy [32].

Of these, especially *GAT* convolutional layers are of importance, as they consider edge features in their input/output space. As mentioned in Chapter 3, edge features are stored using `edge_attr` attribute of *PyTorch Geometric* [24] `Data` object. Whereas both `GCNConv` layer and `TransformerConv` layer consider `x` and `edge_index` as their inputs (representing nodes with their features and edge indices, respectively), `GATConv` layer additionally uses `edge_attr` for its input space besides the former two, allowing to propagate not only information about node features but also edge features.

Additionally within our framework we also make use of fundamental layers used in deep learning, such as *Rectified Linear Unit* [33] to incorporate non-linearity into the model and batch normalization [34] to enhance efficiency and stability throughout the learning process.

4.2 Architecture design and analysis

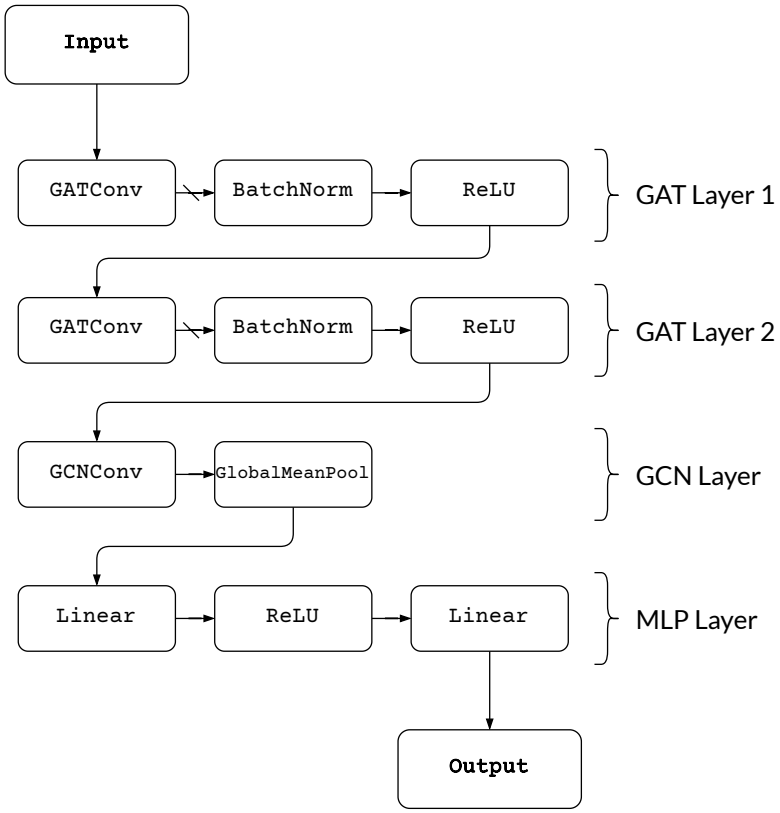


FIGURE 4.1: *GRN* architecture using *GAT* convolutional layers, slash arrows indicate where dropout occurs.

The presented architecture is a graph neural network (*GNN*) model structured within the *PyTorch* [35] framework using the `Sequential` module. This architecture is specifically designed for tasks involving our graph-based representation of three-dimensional RNA structures, especially

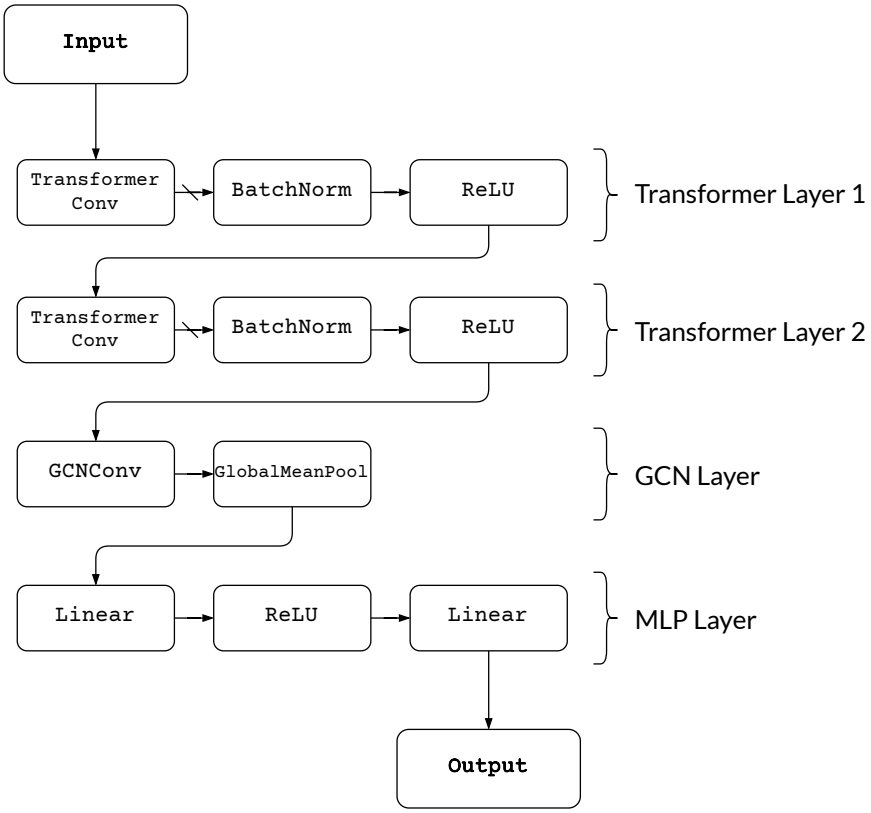


FIGURE 4.2: *GRN* architecture using *Transformer* convolutional layers, slash arrows indicate where dropout occurs.

regarding the quality assessment of such structures, which, as mentioned before, falls into the graph-level regression problems.

In our work, we employed two very similar approaches that vary in the components integrated into the initial layers. One of these methods involves the use of *GAT* convolutional layers (Figure 4.1), enabling the incorporation of edge features by passing `edge_attr` as an input to the convolutional layer. The second approach uses *Transformer* convolutional layers (Figure 4.2). Given the substantial similarity between both approaches, with distinctions confined to these initial layers, we will describe a single architecture, highlighting any noteworthy differences within the context of the problem.

Our input consists of an instance of a *PyTorch Geometric* `Data` object, representing our graph: three-dimensional RNA structure with its encoded features.

GAT/Transformer Layer 1. The first layer in the network is a *GAT*/*Transformer* layer. It initializes with a `GATConv`/`TransformerConv` component with input dimension of 99 and output dimension of 256, using 4 attention heads, and making use of dropout [36] (regularization technique in neural networks involving the temporary exclusion of random neurons during training to prevent overfitting). This layer processes the input graph data consisting of node features (`x`) and edge connections (`edge_index`). Furthermore, when using `GATConv` layer then the input space contains also edge features (`edge_attr`). The output of this layer is then subjected to batch normalization with an input dimension of 1024, followed by *Rectified Linear Unit* (*ReLU*) activation to introduce non-linearity in the model.

GAT/Transformer Layer 2. The subsequent layer is another *GAT*/*Transformer* layer, similar to the first but with different feature dimensionality and parameters. Specifically, the convolutional component has input dimension of 1024 and output dimension of 256 and uses 8 attention heads. The resulting output is again processed through batch normalization and

ReLU activation.

GCN Layer. Following the two initial *GAT / Transformer* layers, a standard *GCN* convolutional layer is introduced. This layer takes the transformed graph data and edge connections as input of dimensionality 2048, applying graph convolution to generate a further refined representation with dimensionality 256. Global average pooling is then applied to aggregate information across all nodes.

MLP Layer. The architecture concludes with a set of fully-connected layers, analogous to a *multi-layer perceptron* (*MLP*). It consists of linear transformation, *ReLU* activation, then at last another linear layer with one output feature for the final prediction.

In conclusion, the architecture incorporates *GAT / Transformer* and *GCN* layers for graph-based feature extraction, batch normalization for stabilization during training, *ReLU* for introducing non-linearity and global average pooling [37] for aggregation. The fully-connected layers at the end enable the model to produce the final prediction based on the processed graph data. This structured approach makes possible for the model to be capable of capture intricate relationships and patterns within the input data.

4.3 Summary

Table 4.1 provides a shorthand summary on our architecture variants applied to our problem. Specifically, it lists all layers with their components, as well as the dimensionality of the input and output channels, applied transformation, and the number of attention heads in the case of `GATConv` or `TransformerConv` components.

Using the described multi-layered approach allows the model to effectively extract features from nodes and edges, leveraging attention mechanisms to focus on relevant structural elements. One strength of our architecture, stemming directly from our approach, is the ability to prevent overfitting. The incorporation of dropout and batch normalization in the *GAT / Transformer* layers ensures the regularization mechanism applied during training, preventing the network from becoming too adjusted to the training data. Batch normalization further contributes to stabilizing the learning process by normalizing inputs within each batch, reducing internal covariate shift and enhancing generalization. Further information about the learning process and hyperparameter optimization is given in Chapter 5.

Additionally, using global mean pooling at the end of graph convolutional layers enables the model to aggregate information, increasing the ability to capture essential structural characteristics and preventing overemphasis on specific local features. This pooling mechanism promotes a more general understanding of the entire graph, contributing to the capacity of quality assessment of diverse 3D RNA structures.

Layer	Component	In	Out	Heads	Transformation
GAT/ Transformer Layer 1	`GATConv`/ `TransformerConv`	99	256	4	(`x`, `edge_index`, `*`) $\rightarrow$ `x`$'$
	`BatchNorm`	1024	1024	–	`x` $\rightarrow$ `x`$'$
	`ReLU`	1024	1024	–	
GAT/ Transformer Layer 2	`GATConv`/ `TransformerConv`	1024	256	8	(`x`, `edge_index`, `*`) $\rightarrow$ `x`$'$
	`BatchNorm`	2048	2048	–	`x` $\rightarrow$ `x`$'$
	`ReLU`	2048	2048	–	
GCN Layer	`GCNConv`	2048	256	–	(`x`, `edge_index`) $\rightarrow$ `x`$'$
	`GlobalMeanPool`	256	256	–	(`x`, `batch`) $\rightarrow$ `x`$'$
MLP Layer	`Linear`	256	64	–	`x` $\rightarrow$ `x`$'$
	`ReLU`	64	64	–	
	`Linear`	64	1	–	

TABLE 4.1: Summary of *GRN* architecture details using *GAT* / *Transformer* convolutional layers. Asterisk `*` represents `edge_attr` when applied to *GAT* convolutional layers.

Chapter 5

Learning process description

In our project, we carried out three learning processes that differed significantly in their characteristics, despite using the same neural network architectures in each of them discussed in the previous chapter. The differences between the learning processes were due to both the variation in the size of the training datasets and the approach used. Our first training used the classical approach, i.e., starting with a model with random weights. It was conducted on relatively small datasets that contained relatively diverse 3D structures in terms of size. Our next approach was to train on a huge dataset that we prepared during the research. The dataset cited contains more than half a million local 3D structure descriptors motivated us to change approach used because of both computational power and overfitting elimination. The final learning process was Transfer Learning [23], which involved using the pre-trained model from the second learning process and training it again on smaller set of larger 3D RNA structures.

5.1 Common stages of the learning process

Independently in every learning process, some common stages were considered. First of all, preprocessing had to be performed on PDB files containing 3D RNA structures. In order to automate the whole process, we introduced a configuration file into our project, which was created for each training dataset considered. In the repository, we provided a guidance on how to create such a file and what the different options are available. In the context of training a neural network, a very important aspect is the splitting between training, validation, and test datasets. The most common data split ratios are 7:2:1 and 8:1:1. In our problem, the situation is not so simple. The datasets contain alternative conformations of the same RNA molecules. In order to prevent an incorrect estimation of the generalization error, it is necessary to make sure that in each subset the distribution of data characteristics is comparable.

For this reason, we decided that the splitting into training, validation, and test collections would already take place at the preprocessing stage. When creating an archive with 3D RNA structure files, it is necessary to divide them into three subdirectories. The datasets we used fulfill this feature and are approximately split in ratio 6:2:2. The preprocessing also results in three H5 files – `train.h5`, `val.h5`, and `test.h5`. Each file contains four tensors: `x`, `edge_index`, `edge_attr`, and `y`, which were described in Chapter 3.

The next common step was to create a special class inheriting from the *PyTorch Geometric* [24] `Dataset`. This class implements how the data will be loaded. Because of the differences in dataset sizes, we implemented two classes. The first simply reads the entire H5 file into RAM memory. The second reads the training data as a stream. The next step is to select the parameters:

- `batch_size` – the number of graphs processed simultaneously,
- `patience` – the value of epochs after which training should be terminated if the network has not reached a lower generalization error,
- `max_epochs` – the maximum number of epochs.

It is also worth mentioning that, despite the graph neural network architecture used in each approach, we used other hyperparameters optimized on the data considered. To deal with overfitting, we mainly used:

- batch normalization – normalizes node features during training, enhancing stability and speeding up convergence [34],
- dropout – regularization method in neural networks where random neurons are temporarily omitted during training [36].

5.2 Preliminary approach

Our first approach was to train our model on the dataset provided by ARES [17], which by default is split into a training set that contains 1000 alternative conformations of 14 experimentally determined 3D RNA structures each and a test set that contains 1000 alternative conformations of 4 experimentally determined 3D RNA structures each. We additionally added a validation set that contained 4 structures from the training set. The ARES neural network also uses GNNs, but differs substantially in the graph representation of RNA structures, as was explained in Chapter 3. The model we trained is comparable to the model provided by ARES (Chapter 7) on ARES test dataset, showing that Feature Extraction and expert knowledge is important especially for neural networks.

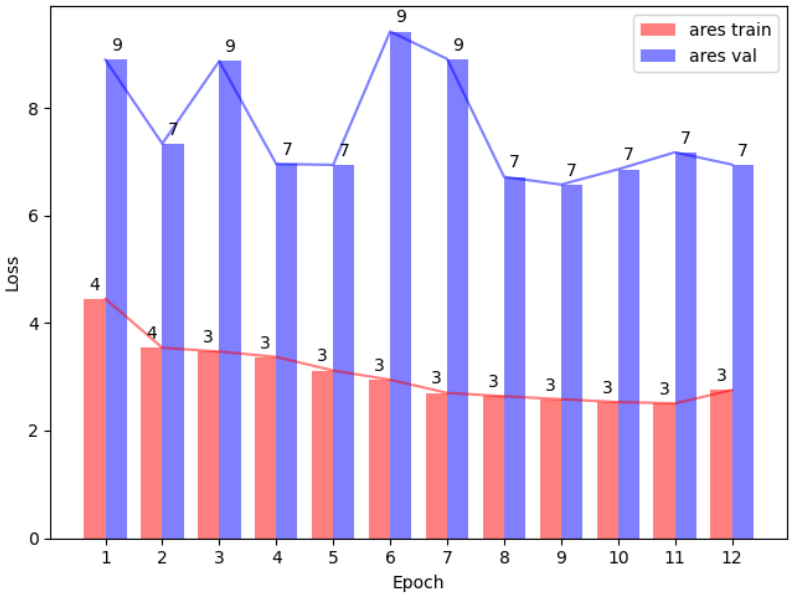


FIGURE 5.1: Epoch-wise Loss (MSE) Analysis our model trained on ARES dataset.

This was a very important observation for us because it proved that our project has potential. We also noticed that the reference 3D RNA structures used by ARES are quite short and do not contain that many nucleotides. Additionally, we noticed that the set of RMSD values is quite

narrow. For this reason, we decided to generate our dataset (Chapter 2) to contain more 3D RNA structures and ensure a more diverse set of the RMSD values. We trained an identical model on the new data. The generalization error measured on the ARES test data was slightly higher than in the our model trained on ARES training dataset, but we noticed a very high error in both training and generalization on our test data. Because of that, we wanted to test the ability of the network to minimize the training error. To do that, we attempted to force overfitting by increasing the complexity of the model. Unfortunately we were not able to get down to a satisfactory value. For this reason, we decided not to share the model and started an approach with training on local 3D RNA structure descriptors. In our opinion, the number of alternative conformations of the reference structures was too low or the complexity of the model architecture used was too low. Our conclusions are argued by the fact that our set had only a few to a dozen alternative conformations while the ARES set had as many as a thousand, and the high value of the loss function on the training set even for architectures with a very large number of parameters.

5.3 Training on large dataset

Our next approach was to train on a large dataset that consists of local 3D RNA structure descriptors extracted from previously predicted 3D RNA models. The argument convincing for this approach was ARES's conclusions that a network trained on small 3D structures performs well on larger 3D structures. These conclusions somewhat follow from the architecture of graph neural networks. A single layer of *Message Passing* aims to transform the feature values of each node into new feature values taking into account both non-linear transformations and the feature values of neighboring nodes. It follows that the graph neural networks are independent of the size of the graph, and that the features they operate on in successive layers are universal. For this reason, we decided to prepare a very large dataset that consists of rather small structures and learn the neural network on it to find general local spatial features that could be generalize for larger 3D RNA structures. In addition, we divided our dataset into one, two, and three or more segmented descriptors and verified how our architecture would perform on each subset and independently how it would perform on a dataset consisting of all 3D structure descriptors. However, each subset contained so many structures that it was impossible to load them into RAM.

For this reason, we implemented a special class inheriting from the Dataset class of the PyTorch library that, instead of loading all the data into memory immediately, loaded them in a streaming way when needed. Our implementation used the *Hierarchical Data Format* (HDF5) [30] and for an Intel i5-6600K processor on a single thread loaded about 600 local 3D RNA structures per second. The whole process was very easy to parallelize, giving us even faster read speed, which was completely sufficient for us. HDF5 is a standard hierarchically organized data format used to store large amounts of information. The data is structurally arranged into groups, each of which can contain data sets or further groups. Each dataset can store a variety of data types, both basic and more advanced – in our case these were *NumPy* arrays, which we converted to tensors from PyTorch during loading. In addition, our features extraction process left some features as missing values due, for example, to the non-existence of certain bond/torsion angles for certain nucleobases. We decided that our datasets should contain null values and that their substitution would only occur at the stage of loading the dataset into memory. Our decision was motivated by the desire to make available a more versatile dataset that everyone could customize for themselves. The HDF5 format supports scalability, enabling the efficient management of large datasets. It is used in a variety of scientific and industrial fields, thanks to its data compression capabilities, support for various programming languages and metadata support.

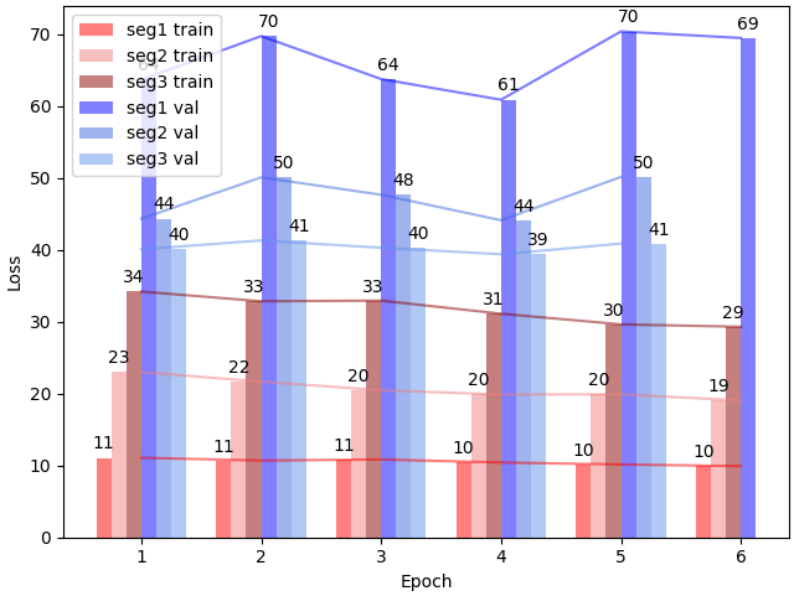


FIGURE 5.2: Epoch-wise Loss (MSE) Analysis of our model trained on all descriptors.

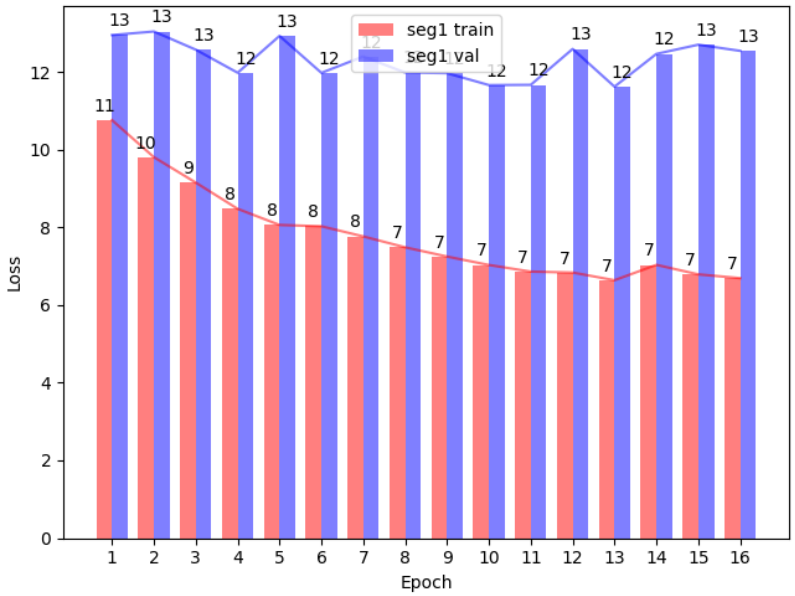


FIGURE 5.3: Epoch-wise Loss (MSE) Analysis of our model trained on only one segment descriptors.

A very important task in training neural networks is to prevent model overfitting as quickly as possible. To do this, a validation set is used which contains the data on which the network has not been learned and after each epoch the generalization error is estimated. As a rule of thumb, an epoch means going over the entire training set and updating the weights after each data batch. The matter gets complicated when we have almost 600,000 samples and a batch size of 32, as this gives us almost 20,000 updates of the model weights. With such a large number of weight updates, detecting overfitting by estimating the error only for every single epoch seems to be a problem, we committed at the very beginning. Fortunately, we quickly realized that this approach was inappropriate. Taking this into account, we implemented additional functionality to our class restricting the training set to random 80,000 samples per epoch (the restriction is passed as an argument to the constructor). In this way, we were able to detect overfitting much faster and train models that generalize better. In comparison, without implementing this approach, it happened

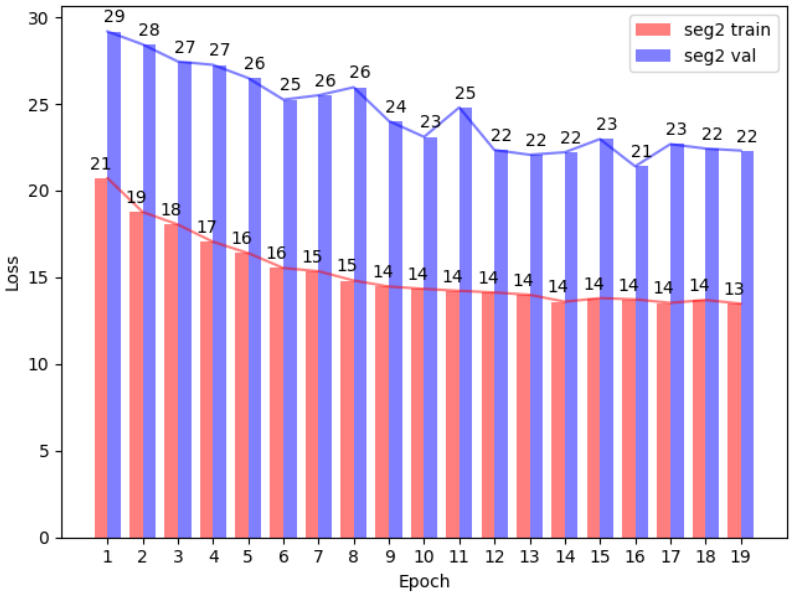


FIGURE 5.4: Epoch-wise Loss (MSE) Analysis of our model trained on only two segment descriptors.

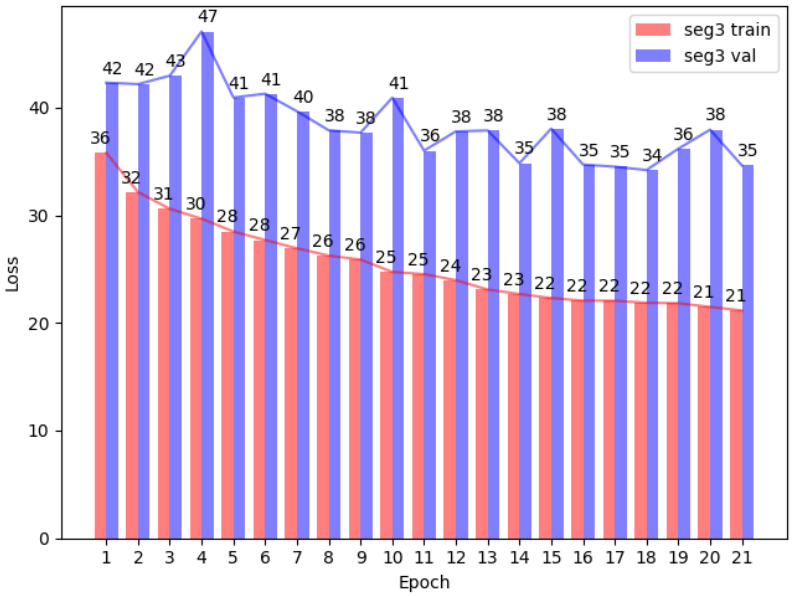


FIGURE 5.5: Epoch-wise Loss (MSE) Analysis of our model trained on only three and more segments descriptors.

that for certain parameters of our architecture, overfitting has already occurred during the first epoch.

Unfortunately, the solution discussed above did not help us with overcoming early overfitting on the entire dataset. Limiting the number of iterations during a single epoch on the training set, only worked well when we applied it to training on three subsets trained independently:

- one segment descriptors (`seg1`),
- two segment descriptors (`seg2`),
- three and more segment descriptors (`seg3`).

The result is represented by three models performing well in different situations. In Figure 5.2 we can observe very fast overfitting and weakness of the model for generalization. It is worth

noting that the training sets of each of our subsets were not equidistant, but during each epoch the model saw exactly 30,000 samples from each subset. In the Figures 5.3, 5.4, and 5.5, we can observe longer learning processes and better generalization on the validation sets of each subset. In Chapter 6, we also present the generalization results of each model on the validation and test sets of each subset.

5.4 Transfer Learning application

Our final approach was to overtrain the `seg2` model from the previous trial to the ARES data. Admittedly, during the previous training, we additionally verified, after each epoch, how our model performed on the data from the ARES, but we did not take this indicator into account as a key one for detecting overfitting and completing the learning, as the ARES set is much smaller than our validation set and we decided that it was not that representative. Nonetheless, we were intent on creating a model that generalizes as well as possible and that would be best for comparison with competing models. For this reason, we froze the weights of the graph layers of our neural network that were responsible for transforming the graph and focused on training only the weights of the layers of the fully-connected network part.

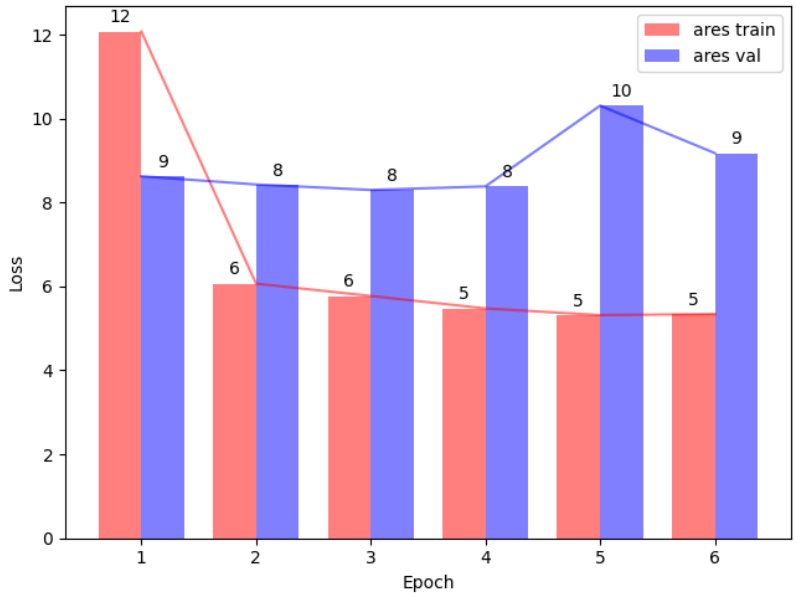


FIGURE 5.6: Epoch-wise Loss (MSE) Analysis (applying transfer learning on seg2 and ARES datasets).

The model adapted very quickly to the new data and achieved low loss function values on the validation set. However, we predicted that a model trained in this way would either be able to achieve the best results on the ARES test set, or, despite its worse performance, maintain satisfactory results on subsets of descriptors.

5.5 Optimization of hyperparameters

The selection of hyperparameters and the proper construction of a neural network are crucial elements in the process of developing machine learning models. It is a process that requires attention, experience, and experimentation, as it significantly influences the quality and effectiveness of the model [38].

The first key element is the proper selection of hyperparameter values, such as the learning rate, regularization coefficients, and optimizer parameters. These parameters affect the dynamics

of the learning process. An excessively high learning rate can result in oscillations in the learning process, making it difficult for the model to converge, while a too low learning rate may lead to a very slow learning process. Proper regularization is crucial in preventing overfitting, which can result in poor generalization to new data.

The size of the neural network, i.e., the number of layers and neurons in each layer, is another essential aspect. A network that is too simple may not be able to learn sufficiently complex patterns in the data, while one that is too complex may lead to overfitting and excessively long training times. Finding the right balance between complexity and generalization ability is crucial.

The proper selection of hyperparameters and the size of the network also impact the consumption of computational resources. Inefficiently chosen parameters may lead to excessive utilization of computational power or excessively long training times. Optimizing these parameters allows for enhancing the model's performance and adapting it to the available computational resources.

In practice, the hyperparameter tuning process often requires multiple experimental iterations. Exploring different parameter combinations, monitoring the learning process, and analyzing validation results are inherent elements of this process. Correctly tuning hyperparameters and the size of the network can lead to the creation of a model that achieves high prediction accuracy, generalizes well to new data, and is efficient in terms of computational resource consumption.

Initially, we created a very rigid architecture in our project and wanted to experiment by testing different combinations of hyperparameters. However, it quickly became apparent that, firstly, this approach was too time- and resource-consuming, as well as simply inefficient. A much better way turned out to be manual experimentation and analysis of the results. By observing how the loss value changes on the training set and the validation set, as well as analyzing the model output, we were able to select parameters and regularize our network much better. For example, for a while we had a pretty good mean generalization error, but it turned out that our network was returning results close to the mean RMSD value of the dataset and additionally with a very small variance. Obviously such a model was useless, but a Grid Search or Random Search strategy can lead to such a situation. It is also worth noting that this problem existed for the ARES dataset and by creating our own dataset, we took care to have a much higher variance on the set of the RMSD values, which offset this problem.

For the reasons described above, our next idea was to initially increase the complexity of the model until we reached a satisfactory result on the training set, and then try to reduce the complexity of the model and apply regularization techniques to get a model that generalizes well. In addition, we experimented with the multiple layers of graph neural networks available in *PyTorch Geometric*. In the end, we were most satisfied with using the `GATConv` and `TransformerConv` layers.

Chapter 6

Evaluation of the developed models

Reliable evaluation of neural network models is a crucial step in the design and implementation of machine learning systems. In short, it involves evaluating the effectiveness and the performance of the models developed in the context of a specific problem. This process is an integral part of the life cycle of a machine learning model, enabling us to observe and deduce from how well the model developed performs in predicting results on totally new, previously unknown data. The purpose of the evaluation is to obtain an objective assessment of the model's performance compared to well-known competitors and our previous models.

We decided to examine two metrics during the evaluation:

- Mean Squared Error (*MSE*) value of each model developed on both validation and test sets,
- the distribution of the predicted values of each model developed on both validation and test sets.

The first metric is commonly during training machine learning models. Its popularity stems from several important features. First, the *MSE* is sensitive to errors, taking into account the squares of differences between predicted and ground-truth values. This makes larger errors, so it has a greater impact on the outcome, which can be important in situations where there are large discrepancies between predictions and reality. Second, the *MSE* is well-defined, making it easier to calculate and interpret. Its definition as the arithmetic mean of the squares of the differences is simple and clear.

We analyzed the distribution of values because in the early stages of our project we noticed that our models often returned very narrow ranges of the RMSD values close to the mean computed within the training set. Such models had fairly low *MSE*, but were obviously useless.

In the process of evaluation, we used four different datasets: the *ARES* training dataset [17], *seg1*, which contains smallest local one segment 3D RNA structure descriptors, *seg2*, which contains descriptors that consist of two-segments, *seg3*, which contains descriptors that have three or more segments.

The our model trained on the *ARES* training data (Figure 6.1) is primarily good on *ARES* validation and test sets. It also performs quite well on the smallest descriptors – one segment. For the other sets, the results are unacceptable.

The model trained on one segment descriptors (Figure 6.2) performs much better on their validation and test sets, and does quite well on the ARES set, although far worse than the previously discussed model. For the other sets, the results are unacceptable.

The model trained on two-segment descriptors (Figure 6.3) is in our opinion the best model if the results for all datasets are compared, but on the data from ARES it is extremely incorrect.

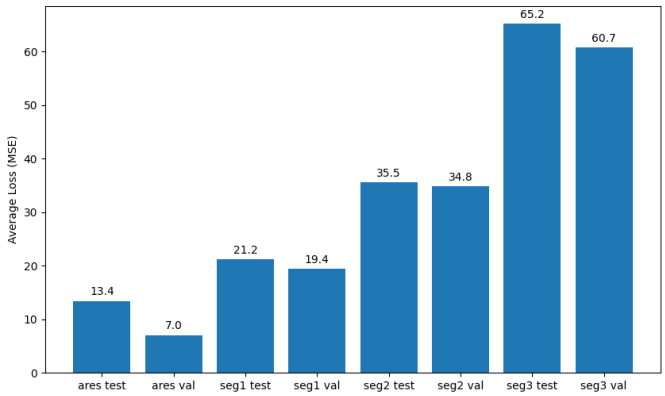


FIGURE 6.1: Average *MSE* across datasets – for the model trained on the *ARES* dataset.

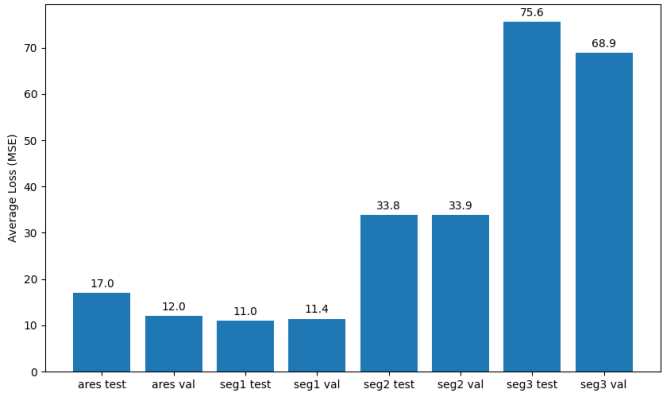


FIGURE 6.2: Average *MSE* across datasets – for the model trained on *seg1* dataset.

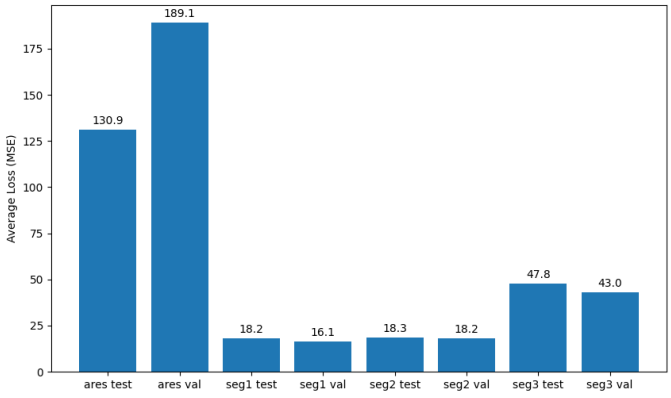


FIGURE 6.3: Average *MSE* across datasets – for the model trained on *seg2* dataset.

For this reason, we performed Transfer Learning (Figure 6.4) to improve the results on the ARES dataset, but the resulting model turned out to be the worst on the all datasets including of descriptors.

For the largest descriptors (Figure 6.5), we failed to get low scores even for the *seg3* validation and test sets.

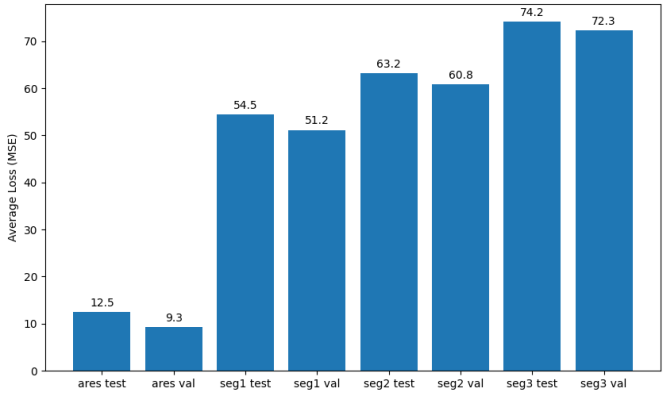


FIGURE 6.4: Average *MSE* across datasets – for the Transfer Learning model trained on both *seg2* and *ARES* datasets.

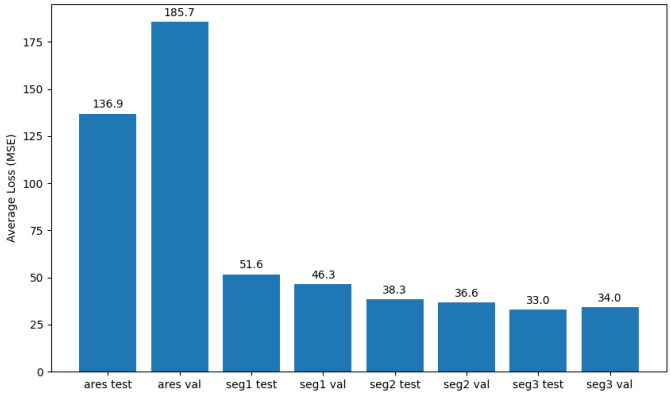


FIGURE 6.5: Average *MSE* across datasets – for the model trained on *seg3* dataset.

Boxplot is a graphical tool for representing the distribution of numerical data. It is designed to show the central tendency, scatter, and identify outliers. The main element is a box, covering 50% of the data between the lower and upper quartiles. The median is indicated by a line inside the box. "Whiskers" extend from the box to the outermost points, helping to see the spread of the data. Boxplots are used for comparisons between groups, making it easier to visualize the distribution of data and identify possible outliers.

It is worth noting that virtually all datasets (except *seg3*) have a lot of outlier observations, which makes the learning process more difficult. The datasets, we generated have wider sets of values and on average more structures with higher RMSD.

Models trained on the *ARES* and *seg1* data predict fairly narrow sets of values, and those trained on *seg2* and *seg3* for the *ARES* data have a strongly shifted median toward higher values, which identifies a problem with the performance of these models on the *ARES* dataset.

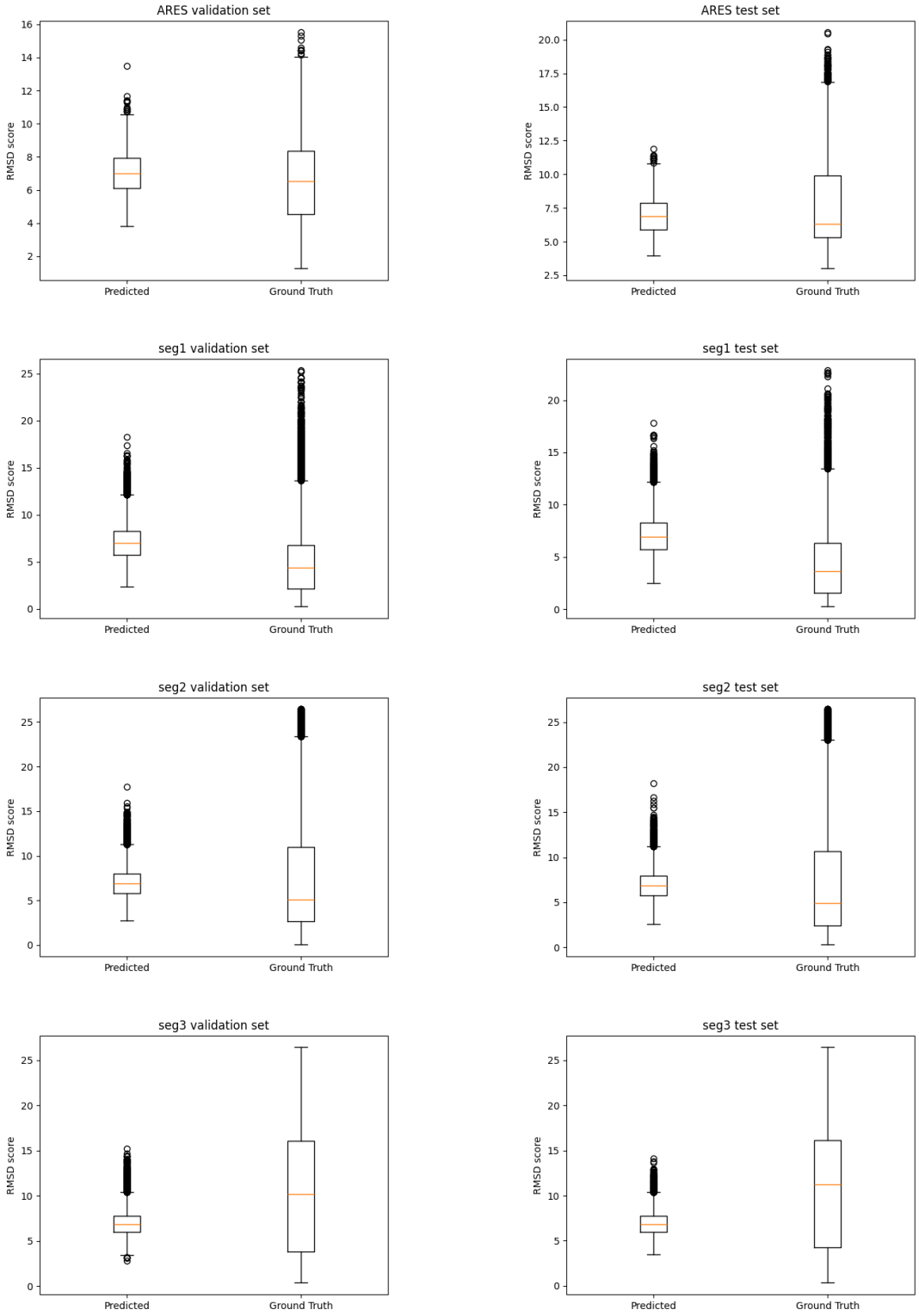


FIGURE 6.6: Distribution of the RMSD values – model trained on the *ARES* dataset.

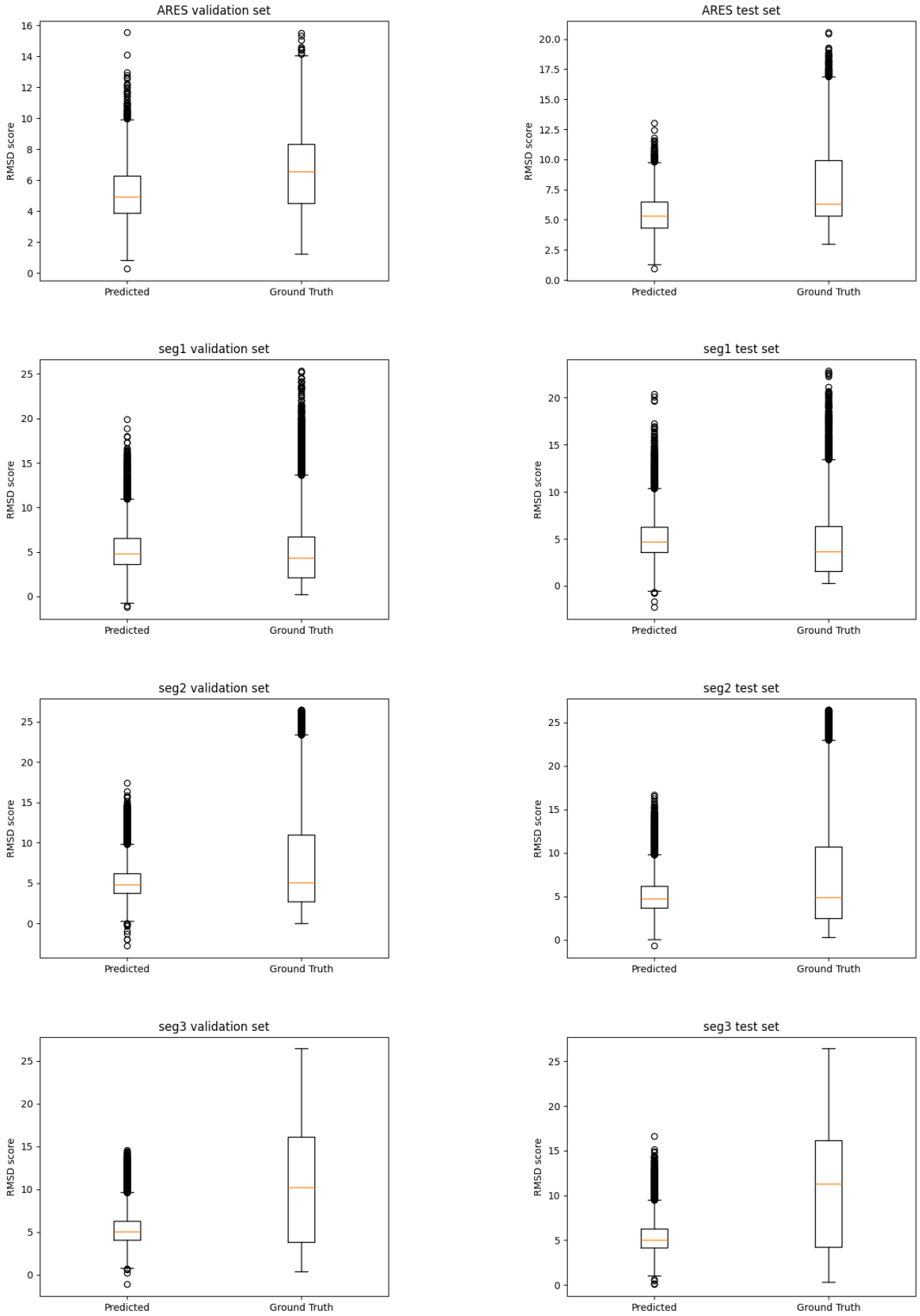


FIGURE 6.7: Distribution of the RMSD values – model trained on the *seg1* dataset.

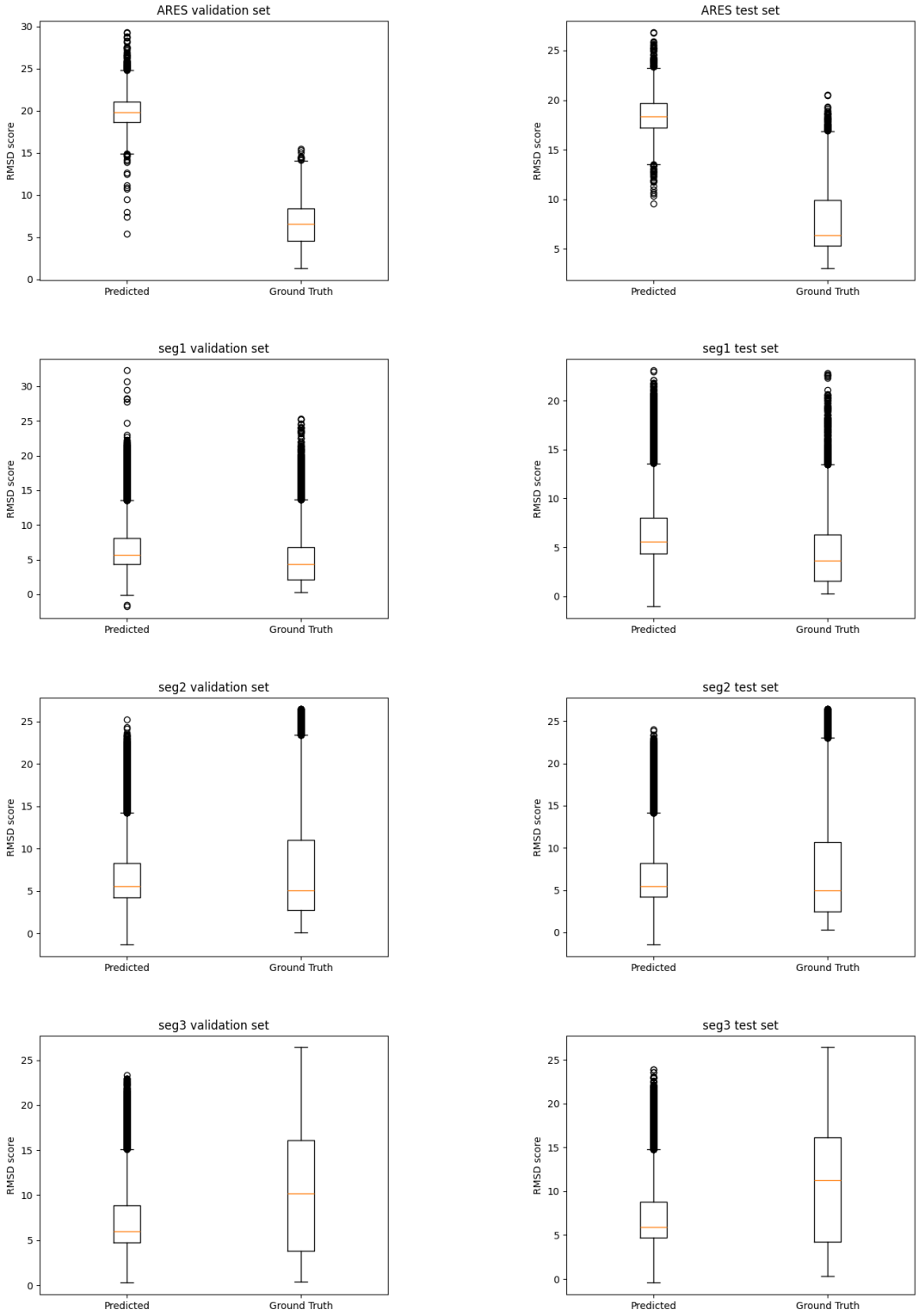


FIGURE 6.8: Distribution of the RMSD values – model trained on the *seg2* dataset.

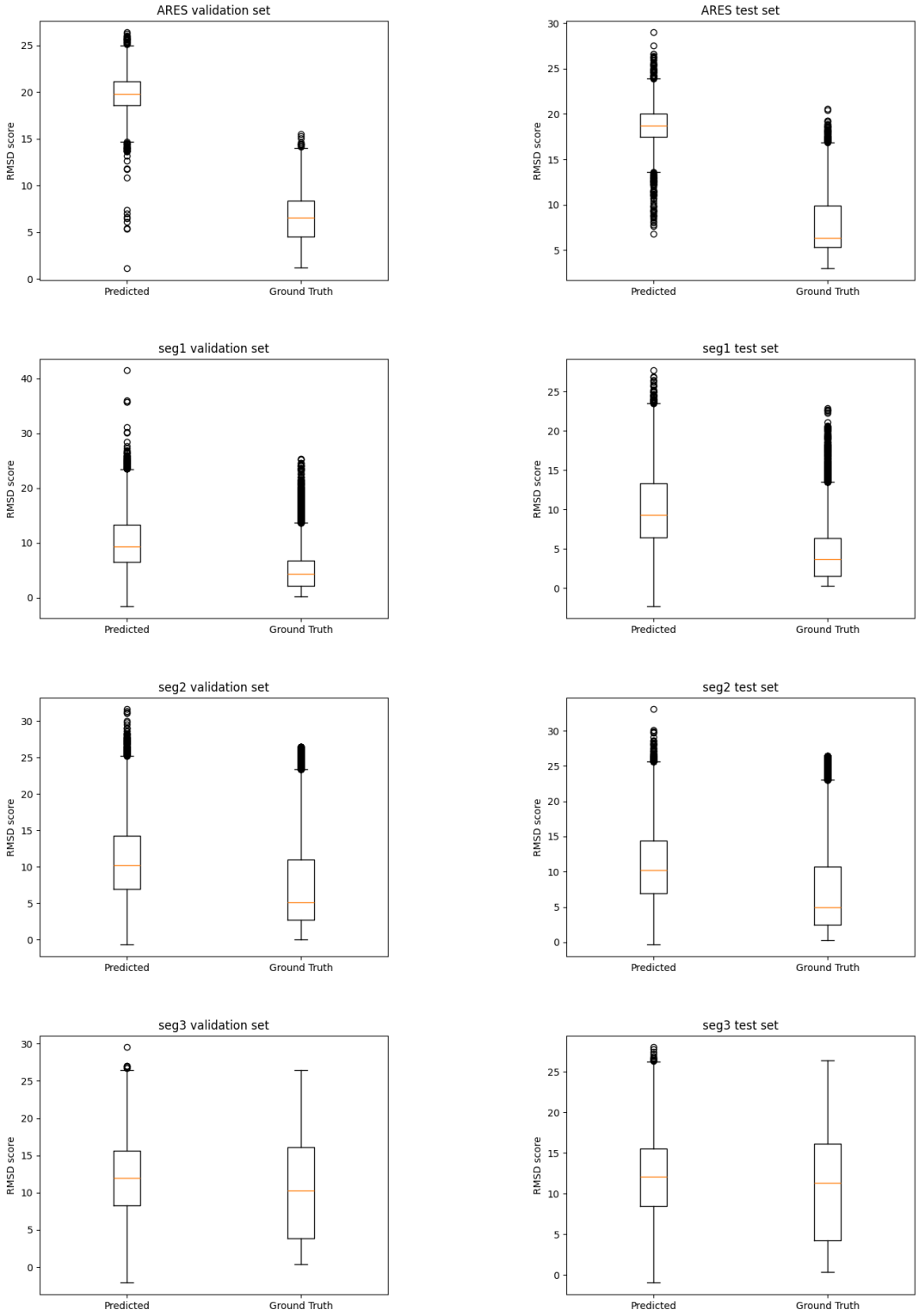


FIGURE 6.9: Distribution of the RMSD values – model trained on the *seg3* dataset.

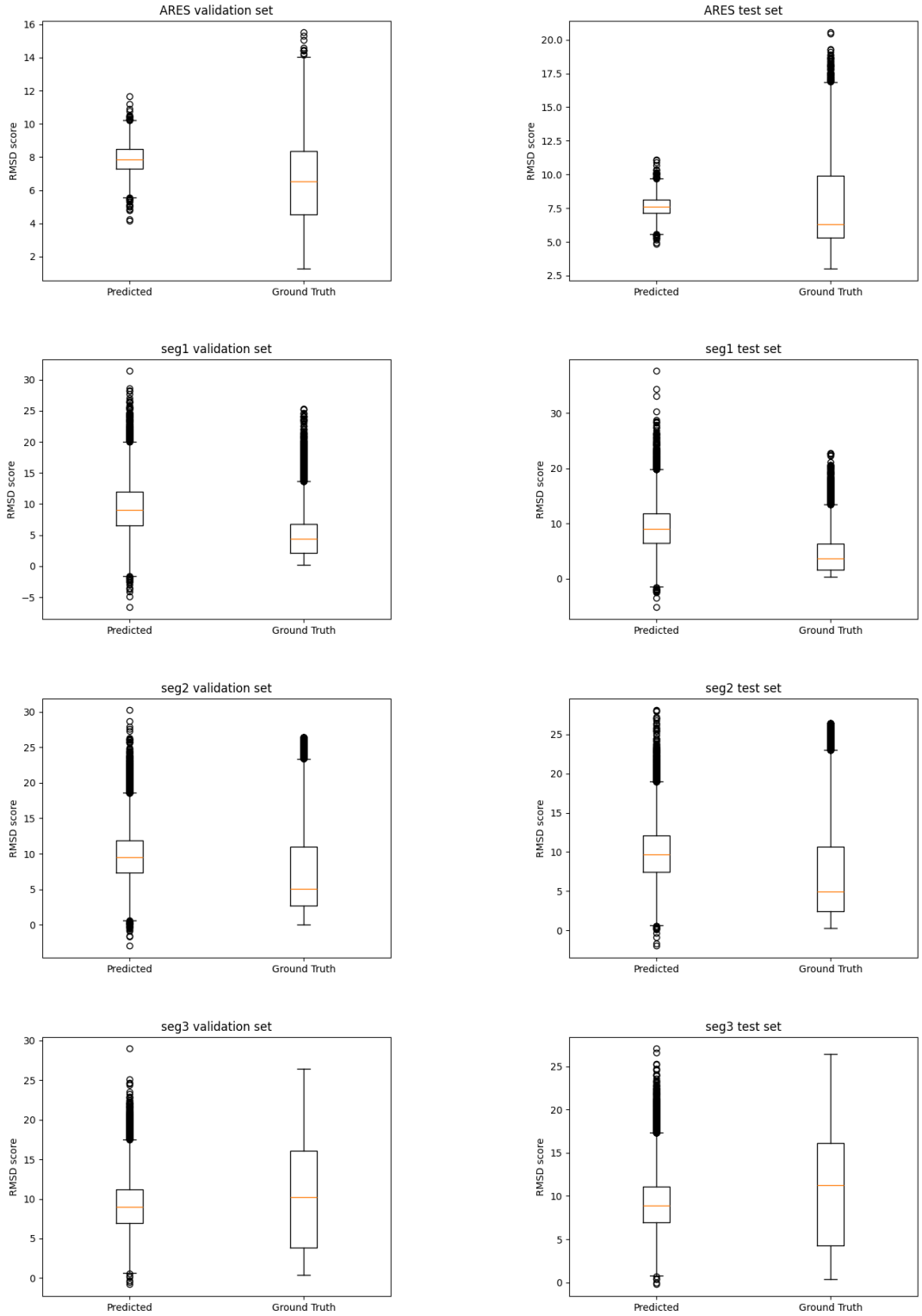


FIGURE 6.10: Distribution of values – Transfer Learning.

Chapter 7

Analysis of the obtained results

One of the methods of assessing the quality of a given machine learning-based model is using comparative analysis. Comparative analysis involves evaluating and comparing two or more ML-based models to identify their strengths, weaknesses, and differences.

In the context of our problem, which involved assessing the quality of three-dimensional RNA structures using graph neural networks, it is essential to identify promising models which can be viably used for such a comparative analysis. We thus, considered solutions using graph convolutional networks and operated on graph representations of three-dimensional RNA structures.

To benchmark our RNAQuANet models, we decided to compare them with ARES model [17]. We also analyzed our dataset construction and selected features within the context of the ARES, thus it is sensible to assess our results with the ones obtained by the available ARES model.

7.1 Test datasets description

To fully understand analysis of the results, it is imperative to investigate the differences between both datasets – ARES and RNAQuANet, as evaluation using different test sets severely influences the model's performance. When comparing two datasets, each representing a different set of samples, employing visual approach often is needed to see the underlying differences in the compared data.

Histograms provide a visual representation of the distribution of certain inherent properties of data – the ones which are of interest to us are the RMSD values. RMSD histograms essentially display the distribution of the RMSD values across the entire datasets, allowing us to identify trends, patterns, detect outliers, and compare datasets using side-by-side visual comparison.

Spearman distance and Spearman footrule distance [39] enable evaluating the rankings of the RMSD scores computed for a set of randomly selected 3D RNA structures. Given a set of N observed values, $Y = \{y_1, \ldots, y_N\}$, a corresponding set of N predictions, $\hat{Y} = \{\hat{y}_1, \ldots, \hat{y}_N\}$, the Spearman distance d_S and Spearman footrule distance d_{SF} are given by the following formulas:

$$d_S(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$d_{SF}(y, \hat{y}) = \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

The structure rankings are computed within the context of the corresponding reference structures.

In this section, we provide the RMSD histograms for test datasets both for RNAQuANet (Figure 7.1) and ARES (Figure 7.2).

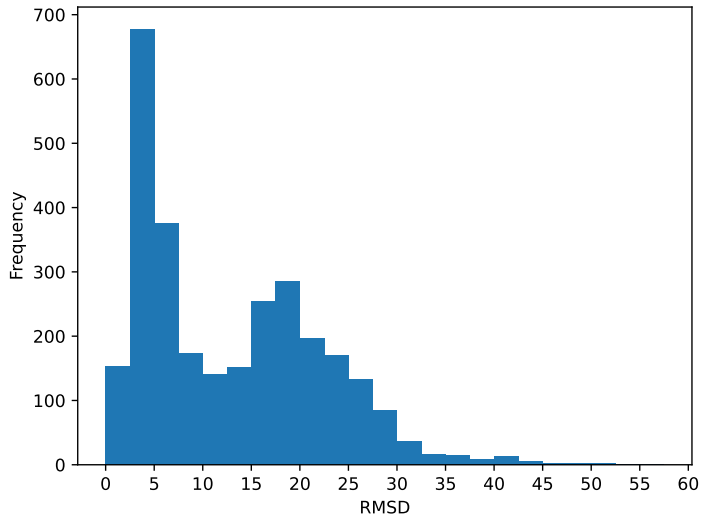


FIGURE 7.1: Histogram of the RMSD scores computed for 3D RNA structures within their corresponding references included in the test subset of RNAQuANet training dataset.

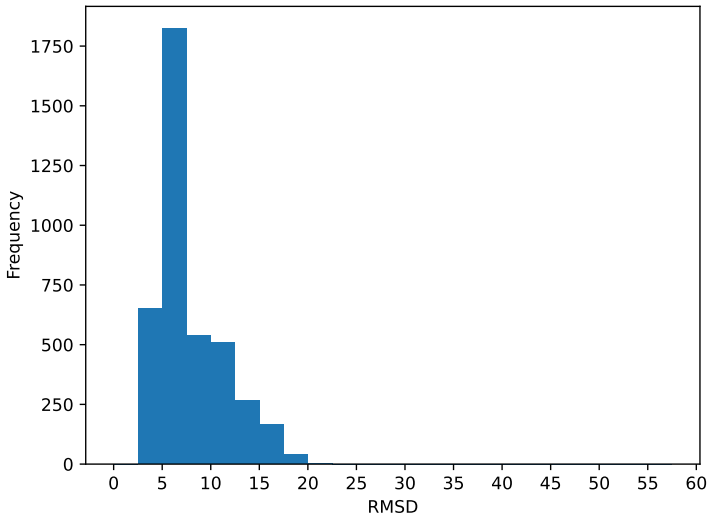


FIGURE 7.2: Histogram of the RMSD scores computed for 3D RNA structures within their corresponding references included in the test subset of ARES training dataset.

7.2 Performance of the developed architecture

In our work, we successfully trained five GNN-based models using four distinct datasets: (1) ARES dataset, containing ARES training, validation, and test sets, (2) seg1, which involves descriptors containing one segment, (3) seg2, which contains two-segment descriptors and (4) seg3, containing descriptors that have three or more segments. The models in question are as follows:

1. trained on seg1 dataset,

2. trained on seg2 dataset,
3. trained on seg3 dataset,
4. trained on ARES dataset,
5. transfer learning on seg2 and the ARES datasets.

Each model involving descriptors (seg1, seg2, and seg3) has been then tested using the dataset in question and the tested on ARES test dataset. In the case of model trained using the ARES's training set, we decided to test it using both the ARES's test set and the RNAQuANet structure test set.

Choosing appropriate and effective quality metrics is crucial in comparative analysis. The selection of metrics directly influences the understanding of a model's performance and its suitability. In our analysis, we chose three such metrics:

- Mean Squared Error (MSE),
- Mean Absolute Error (MAE),
- the absolute difference between reference and predicted the RMSD values.

Of these, the MSE penalizes larger errors influenced by outliers, while the MAE provides a more balanced assessment by considering the average magnitude of errors. The absolute difference between reference and predicted values is an intuitive metric that directly indicates how far off the predictions are from the ground truth.

The results are summarized in Table 7.1.

Table legend:

1s one segment descriptor dataset,

2s two segments descriptor dataset,

3s three and more segments descriptor dataset.

Trained using	Tested on	MAE	MSE
ARES	ARES	2.698	13.367
ARES	RNAQuANet	8.597	127.186
RNAQuANet 1s	ARES	2.941	17.156
RNAQuANet 1s	RNAQuANet	8.556	135.480
RNAQuANet 2s	ARES	10.708	130.714
RNAQuANet 2s	RNAQuANet	7.371	102.806
RNAQuANet 3s	ARES	10.931	136.487
RNAQuANet 3s	RNAQuANet	5.630	56.924
Transfer learning RNAQuANet 2s then ARES	ARES	2.779	12.447
Transfer learning RNAQuANet 2s then ARES	RNAQuANet	9.586	131.802

Table 7.1: Assessment of RNAQuANet's architectures on various test datasets.

7.2.1 Based on the representative 3D RNA descriptors dataset

The histograms below show the distributions of absolute differences between the ground-truth RMSD and the RMSD predicted by the five models trained on the RNAQuANet architecture. The more small differences, the better the model. The graphs show that the models usually

perform better on the ARES test set. In the following sections, we will present analogous plots for models trained on the ARES architecture.

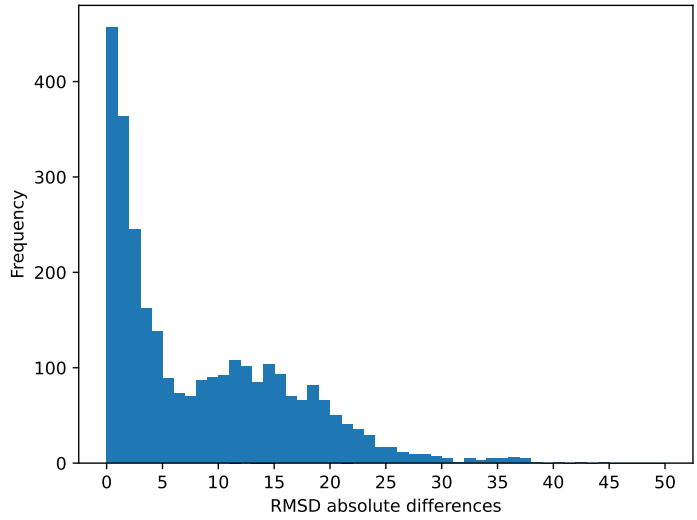


FIGURE 7.3: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg1 training set and tested on RNAQuANet test set.

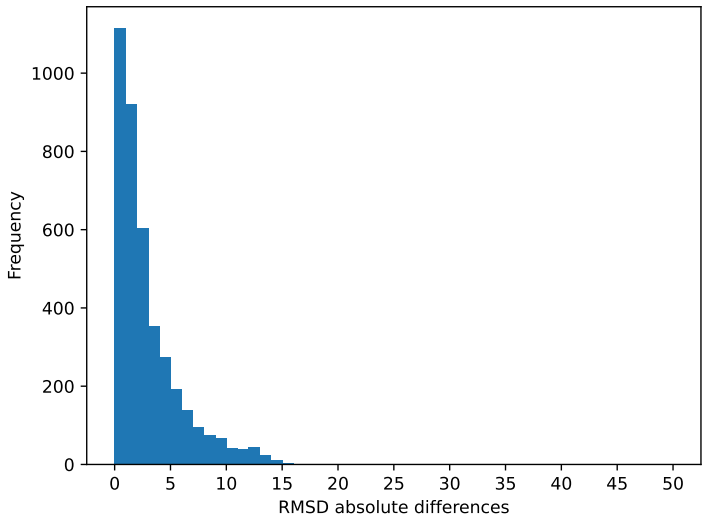


FIGURE 7.4: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg1 training set and tested on ARES test set.

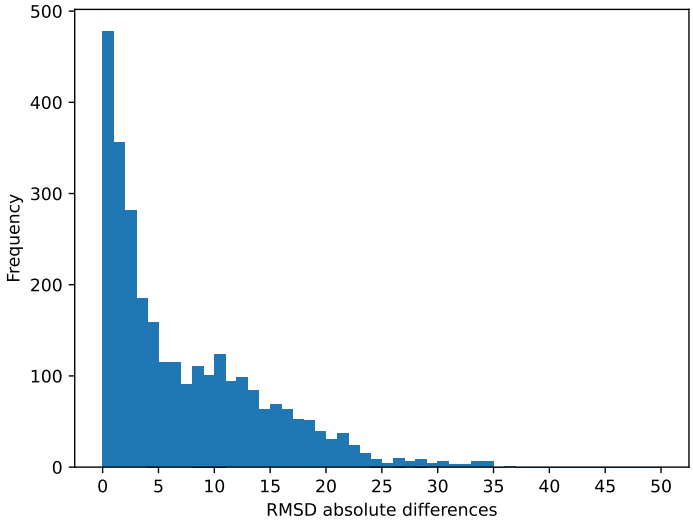


FIGURE 7.5: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg2 training set and tested on RNAQuANet test set.

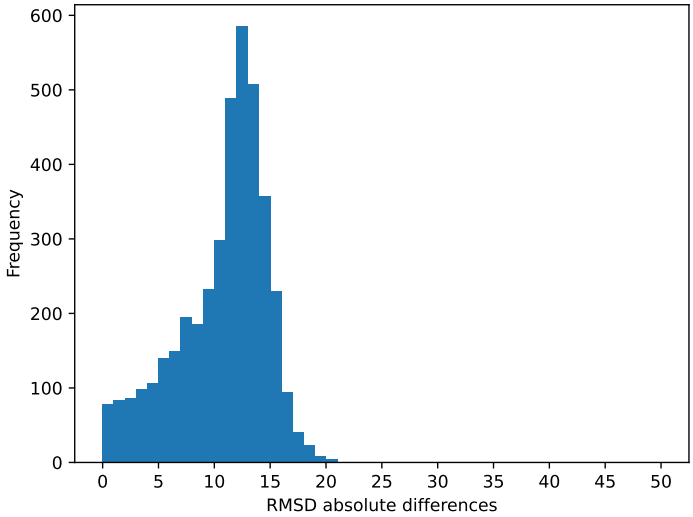


FIGURE 7.6: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg2 training set and tested on ARES test set.

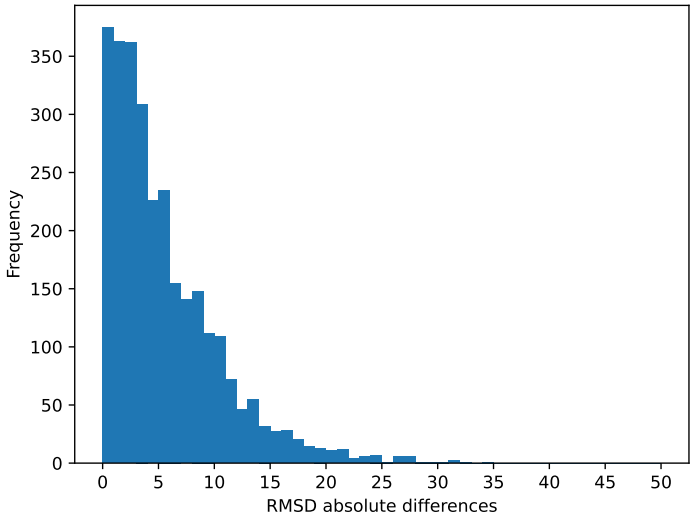


FIGURE 7.7: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg3 training set and tested on RNAQuANet test set.

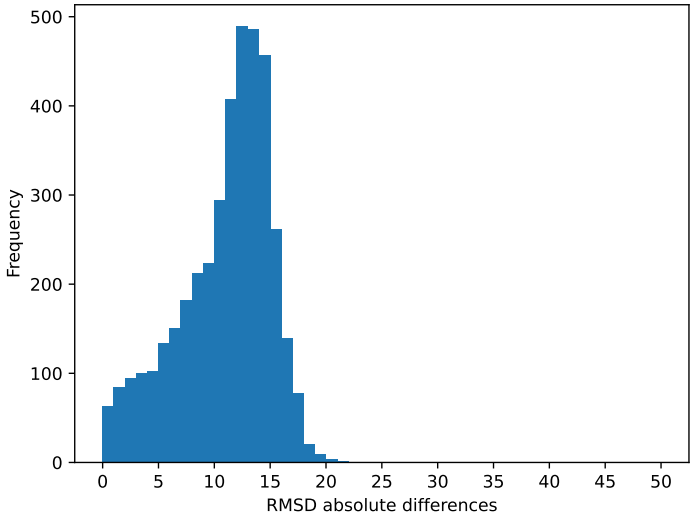


FIGURE 7.8: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on seg3 training set and tested on ARES test set.

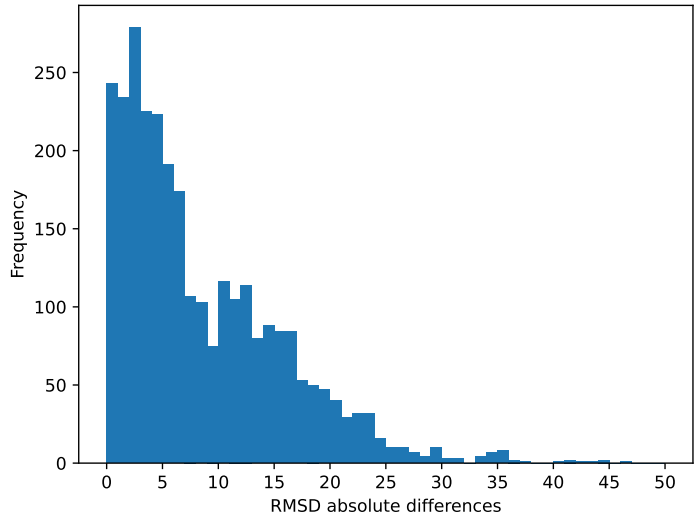


FIGURE 7.9: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on ARES training set and tested on RNAQuANet test set.

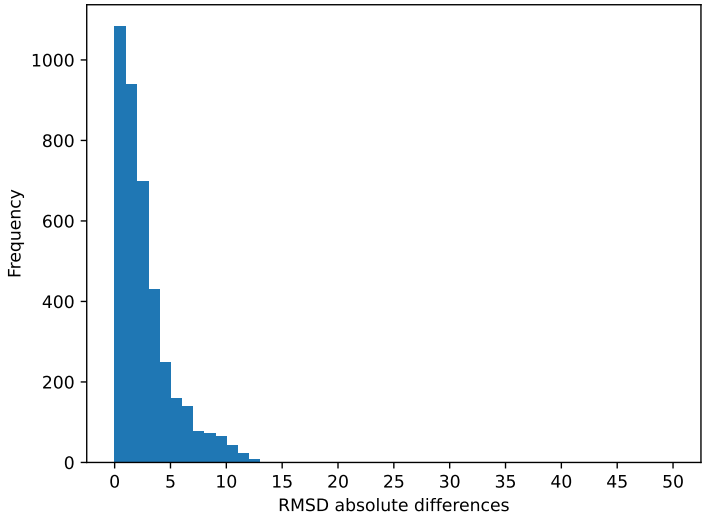


FIGURE 7.10: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on ARES training set and tested on ARES test set.

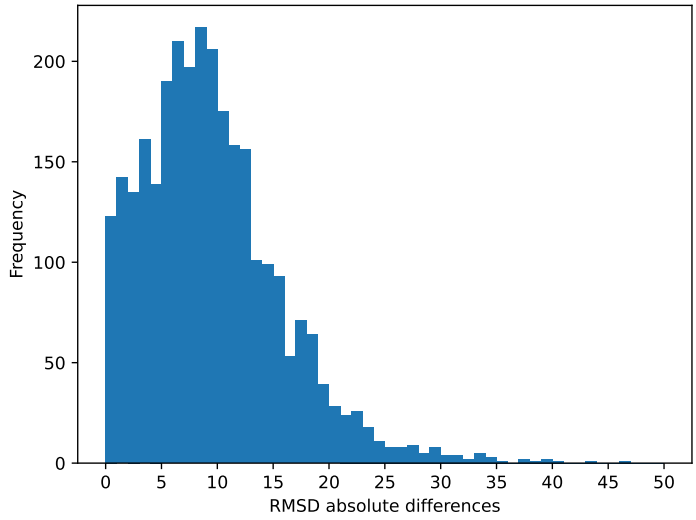


FIGURE 7.11: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on both training sets of RNAQuANet seg2 and ARES one and tested on the RNAQuANet test dataset.

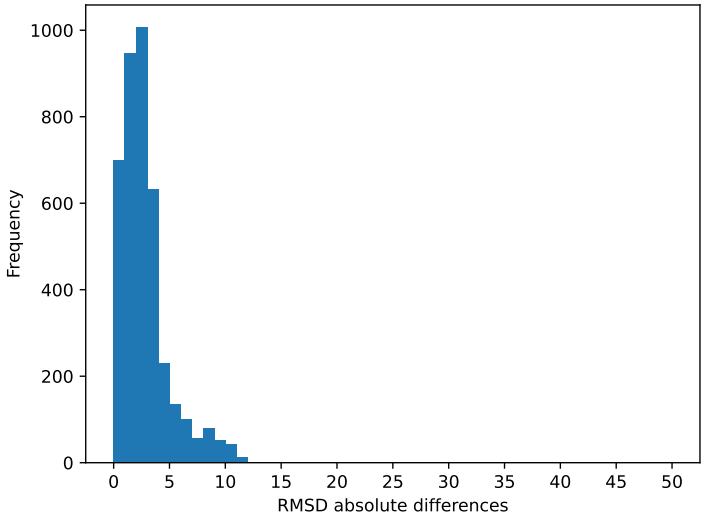


FIGURE 7.12: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the RNAQuANet model trained on both training sets of RNAQuANet seg2 and ARES one and tested on ARES test dataset.

7.3 Performance of ARES's architecture

For completeness of our analysis, we also decided to assess performance of the ARES's architecture. We trained the model using the ARES's architecture and RNAQuANet structure dataset and tested it using both RNAQuANet structure test set and ARES test set. The results are provided in Table 7.2. Relevant plots are included in Figure 7.13 and 7.14.

Tested on	MAE	MSE
RNAQuANet structure	5.02	49.73
ARES	2.76	13.49

Table 7.2: Assessment of the ARES's architecture – trained on RNAQuANet structure datasets.

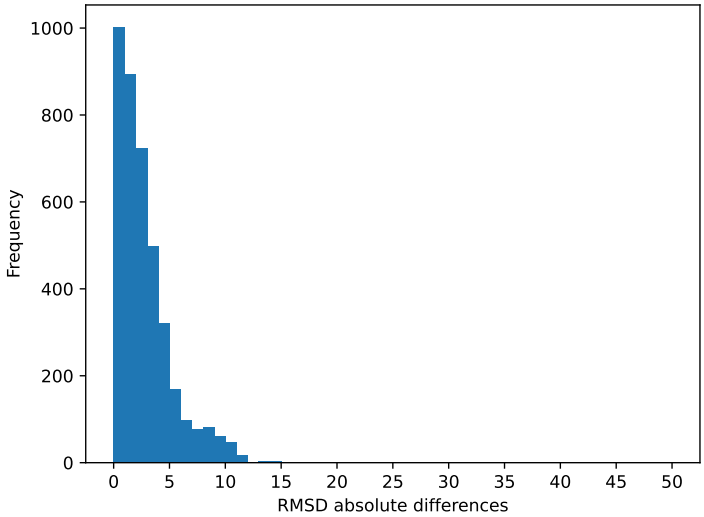


Figure 7.13: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the ARES model trained and tested on the RNAQuANet training and test dataset, respectively.

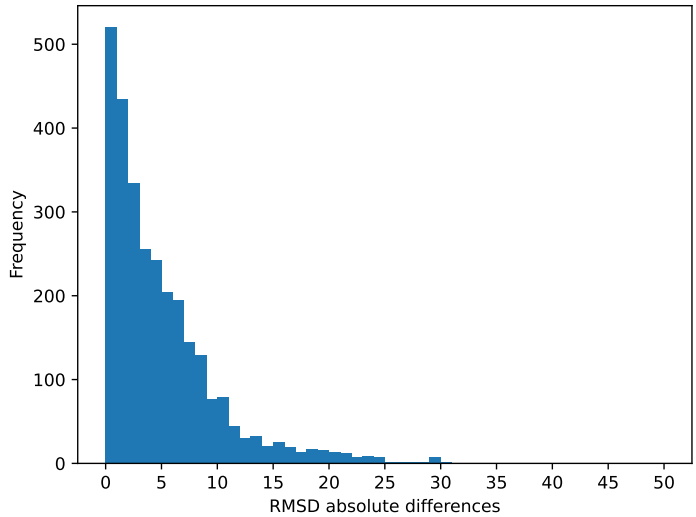


Figure 7.14: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the ARES model trained and tested on the ARES training and test dataset, respectively.

7.3.1 Transfer learning

The application of transfer learning required training ARES's architecture using RNAQuANet structure dataset. After that it was capable to freeze the weights and biases of the trained model to prevent modifications during subsequent training. Finally, the last decoder layers were substituted with new ones and became the subject of the learning process on the ARES dataset.

The results of this approach are provided in Table 7.3.

Tested on	MAE	MSE
RNAQuANet structure	6.786	89.484
ARES	2.449	11.543

TABLE 7.3: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the ARES model trained and tested on both training sets of RNAQuANet structure and the ARES one and the RNAQuANet structure test dataset, respectively.

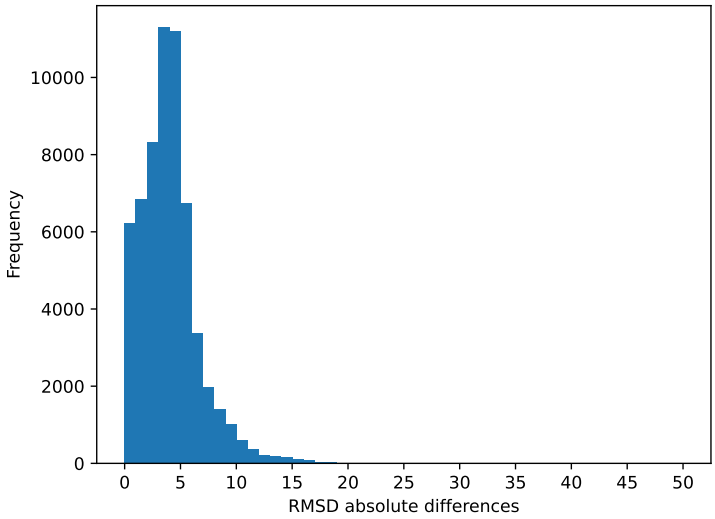


FIGURE 7.15: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the original ARES model tested on the RNAQuANet structure test dataset.

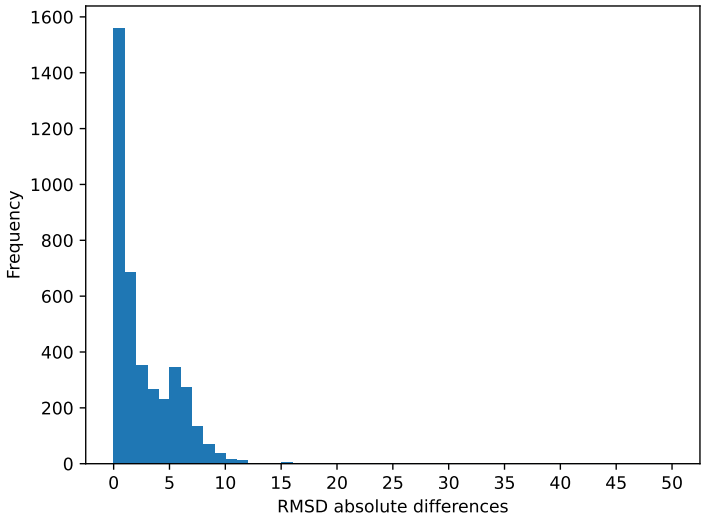


FIGURE 7.16: Histogram of a number of occurences of absolute difference values computed between predicted and ground-truth values for the original ARES model tested on the ARES test dataset.

7.4 ARES model characteristics

Additionally, we decided to test the ARES model which is openly published. We assessed trained ARES model using the RNAQuANet structure dataset and ARES dataset. The results are provided in Table 7.4.

Tested on	MAE	MSE
RNAQuANet structure	7.897	105.54
ARES	3.099	11.39

Table 7.4: ARES trained model assessment.

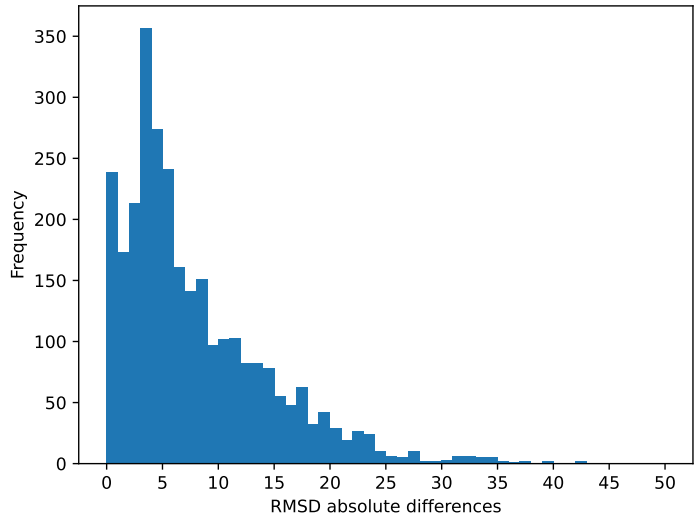


Figure 7.17: Histogram of difference frequency between model prediction and reference value.

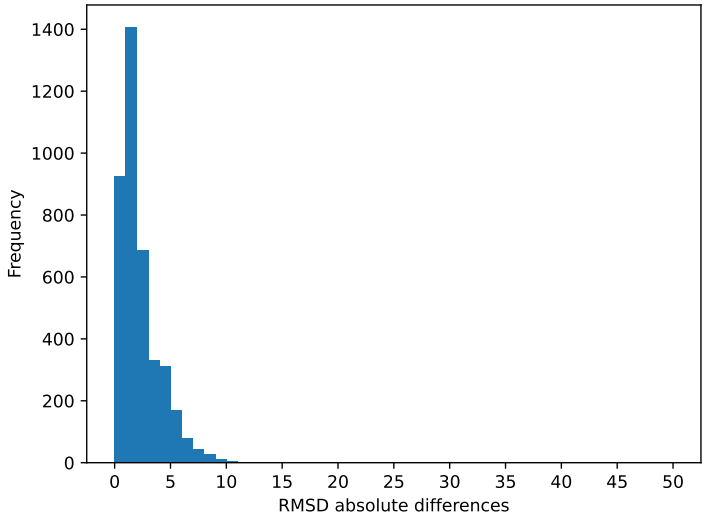


Figure 7.18: Histogram of difference frequency between model prediction and reference value.

7.5 Summary of the developed models comparative analysis

Table legend:

- **1s** one segment descriptors (cardinality: 22267),
- **2s** two segments descriptors (cardinality: 60214),
- **3s** three and more segments descriptors (cardinality: 57921),
- **ARES 1s** one segment descriptors (cardinality: 100),
- **ARES 3s** two segments descriptors (cardinality: 100)
- **ARES 3s** three segments descriptors (cardinality: 75)
- **TL-R2s-A** Transfer learning on the RNAQuANet 2s then the ARES (cardinality: 60214 + 14000)

		RNAQuANet		ARES	
Trained using	Tested on	MAE	MSE	MAE	MSE
ARES	ARES	2.698	13.367	3.099	11.39
ARES	RNAQuANet	8.597	127.186	7.897	105.54
ARES	ARES 1s	2.977	13.597	5.390	33.157
ARES	ARES 2s	2.115	7.399	3.081	12.500
ARES	ARES 3s	2.295	7.626	2.115	6.128
RNAQuANet 1s	ARES	2.941	17.156	3.764	21.691
RNAQuANet 1s	RNAQuANet 1s	2.638	11.023	1.535	4.402
RNAQuANet 1s	ARES 1s	1.471	3.274	1.387	3.229
RNAQuANet 2s	ARES	10.708	130.714	3.846	26.530
RNAQuANet 2s	RNAQuANet 2s	3.233	18.283	2.346	11.641
RNAQuANet 2s	ARES 2s	10.269	197.740	2.205	7.227
RNAQuANet 3s	ARES	10.931	136.487	3.342	17.053
RNAQuANet 3s	RNAQuANet 3s	4.622	32.948	3.449	20.251
RNAQuANet 3s	ARES 3s	11.436	137.037	2.497	8.577
TL-R2s-A	ARES	2.779	12.447	2.655	13.971
TL-R2s-A	RNAQuANet 2s	6.733	62.830	4.014	22.764
TL-R2s-A	ARES 2s	7.078	65.742	2.413	8.914

Table 7.5: Assessment of RNAQuANet's vs the ARES architecture.

7.5.1 Ranking analysis

Spearman distance is a square of Euclidean distance between two rank vectors. The indicator helps with comparing two rankings to assess their diversity. In this example it is crucial to compare structure RMSD variation rankings between prediction and the ground-truth values. The smaller the value of the distance, the better.

In the Table 7.6 Spearman distance is introduced as *SD* and Spearman footrule distance as SFD.

		RNAQuANet		ARES	
Trained using	Tested on	SD	SFD	SD	SFD
ARES	ARES	593,929,966	1,247,416	420,678,896	1,015,956
ARES	RNAQuANet	336,030	24,220	266,736	20,686
ARES	ARES 1s	21,452	1,204	19,912	1,110
ARES	ARES 2s	26,352	1,160	10,876	734
ARES	ARES 3s	71,928	1,994	25,310	1,080
RNAQuANet 1s	ARES	671,629,574	1,344,180	516,132,996	1,144,382
RNAQuANet 1s	RNAQuANet 1s	443,386,124	2,153,034	240,493,076	1,505,108
RNAQuANet 1s	ARES 1s	21,452	1,204	12,756	882
RNAQuANet 2s	ARES	680,090,860	1,354,252	600,927,488	1,253,002
RNAQuANet 2s	RNAQuANet 2s	3,678,229,056	10,073,928	2,211,747,526	7,689,058
RNAQuANet 2s	ARES 2s	26,352	1,160	11,578	718
RNAQuANet 3s	ARES	684,167,340	1,353,968	513,413,986	1,142,826
RNAQuANet 3s	RNAQuANet 3s	6,638,067,492	13,992,846	5,580,207,514	12,514,294
RNAQuANet 3s	ARES 3s	71,928	1,994	31,452	1,142
TL-R2s-A	ARES	628,035,434	1,295,390	485,235,210	1,107,834
TL-R2s-A	RNAQuANet 2s	8,978,950,978	17,451,434	3,161,807,430	9,325,692
TL-R2s-A	ARES 2s	45,958	1,624	12,756	834

Table 7.6: Assessment of RNAQuANet's and ARES' architecture.

The model provided by the ARES performs better when tested on the ARES test set, but its accuracy is only moderately satisfactory when tested on the RNAQuANet structure dataset. Although transfer learning approach applied to ARES's architecture slightly improves performance on the RNAQuANet test dataset, there are no substantial improvements observed using the ARES test set.

Taking into consideration both training sets, ARES demonstrates better learning capabilities, yielding highly relevant results. Based on conducted experiments, we confirmed that nucleotides-based graph representation of 3D RNA structure is too sparse to fully describe interatomic interactions. Future works must be focused on deeper exploration of interatomic representations od 3D RNA structures.

Chapter 8

Technological solutions applied

8.1 Business logic layer

The whole business logic layer of our project is developed using *Python* and consists of two major components: application, which implements the data processing logic and API, which covers data access and communication.

8.1.1 Architecture of the application

The application is comprised of *Python* command line interface `.py` scripts. The scripts use *Python's* `argparse` library [40], which allows to pass command line arguments and provides help messages, describing each config parameter provided. In addition, we use *Docker* [41] to ensure portability and ease of installation.

Furthermore, throughout the whole application numerous options are provided using config files in the *YAML* format. Among them, there are, e.g., name of the dataset used, path to directory storing the input dataset, parameters for feature extraction, and neural network layers configuration.

The neural network is implemented using *PyTorch Geometric* library [24]. Processed input data is stored using HDF5 (*Hierarchical Data Format*) files [30].

8.1.2 Containerization of the application

To enhance the portability and ease of use for our graph neural network models and learning scripts, we included *Docker* as part of our project. Our containerization approach focused on encapsulating the essential components, libraries, and dependencies required for running the models and training processes.

We created a *Docker* image that includes the necessary *Python* dependencies, *PyTorch* [35], *PyTorch Geometric* and other libraries essential for running our framework. This streamlined approach ensures that users can easily replicate our environment without dealing with complicated setup procedures, finding and installation of older, archival versions of libraries, etc. We consider this very important to broadening the openness of science, allowing other researchers to verify, collaborate, and build upon our framework.

This represents an innovative approach in the realm of scientific research, especially in the field of machine learning. Most of the work involving neural networks does not come with tools ensuring ease of reproduction and consistency of environment. This property is especially crucial, as most machine learning libraries, such as *PyTorch*, often introduce breaking changes that do

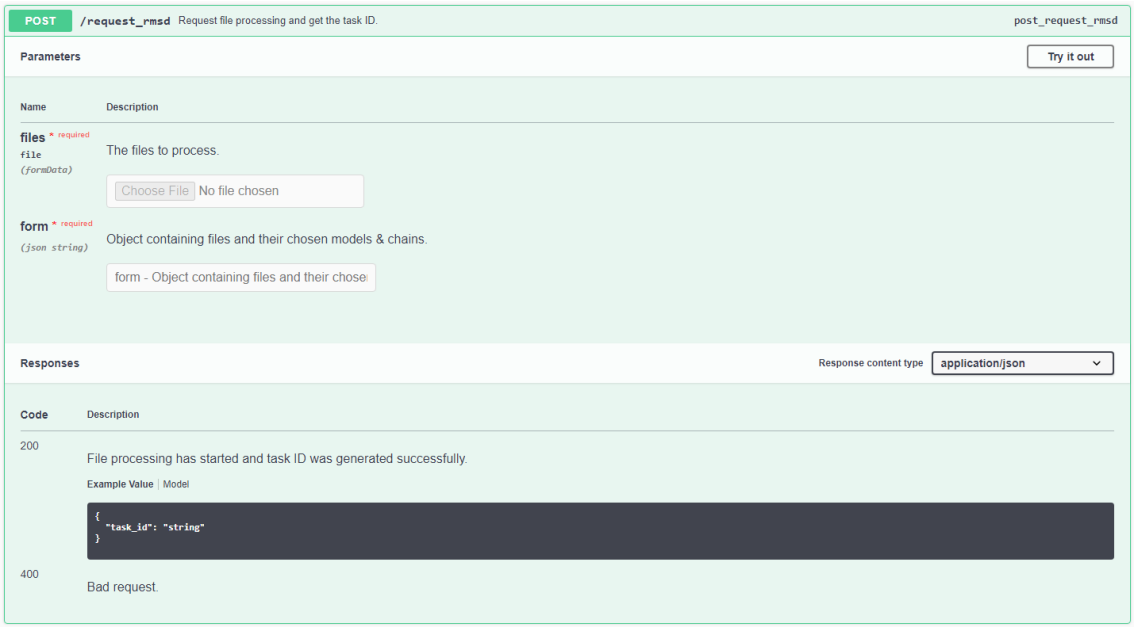


FIGURE 8.1: *Swagger* [47] documentation showing an endpoint for submitting a new task.

not work on older, legacy code. Using a standardized image providing a snapshot of the work environment, further allows to seamlessly deploy the application in different context, such as cloud, local machines, or any system supporting *Docker*, without the need for extensive configuration or modification.

8.1.3 Application programming interface (API)

Our API is built using *Python Flask* [42], a lightweight and versatile web framework that allows us to develop robust and scalable web applications. With Flask, we have integrated several key libraries to enhance functionality, documentation, and interaction with the API:

- *SQLAlchemy* [43] – core *SQL* toolkit and ORM (*Object Relational Mapping*),
- *Redis Queue* [44] – lightweight job queue that supports an efficient and scalable processing of in-background tasks,
- *Werkzeug* [45] – utility library giving access to variety of useful functionalities, such as *FileStorage*, which makes working with files easier, bringing more types to *Python* and creating safe file names,
- *Flasgger* [46] – automatic creation of an interactive API documentation and request testing. Examples of API endpoints can be seen in Figures 8.1 and 8.2.

For the database engine, we chose *SQLite* [48], which is more than enough for our simple needs of keeping users' task data and files in an easily accessible and editable storage. An entity–relationship diagram scheme of the database can be seen in Figure 8.3.

8.2 User interface layer

Our frontend is built using a modern stack of libraries and tools to ensure smooth, user-friendly, and responsive user interface. It is built on *React* [49] with *Vite* [50] as its development server

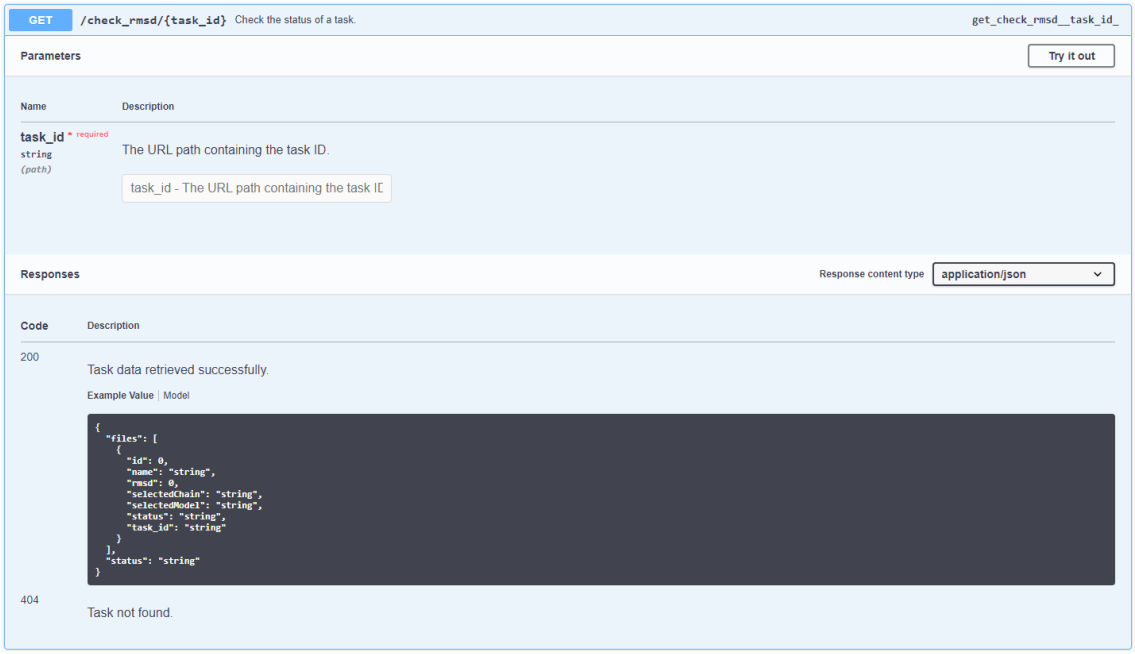


FIGURE 8.2: *Swagger* Documentation showing an endpoint that checks the current processing status for a given task.

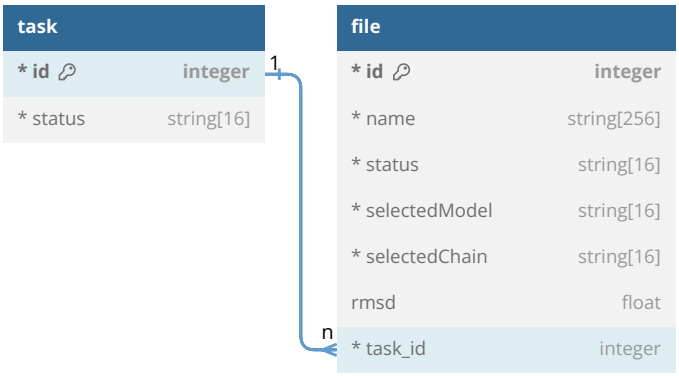


FIGURE 8.3: Entity–relationship diagram of the database.

and *TypeScript* [51] to add support for static typing to the codebase. *Material UI* [52] is our main supplier of components, icons, hooks, and other styling tools. More about the frontend, including example images, can be seen in Section 9.1.

8.3 Computational infrastructure

The virtual machine utilized for the preprocessing, training, and development of the machine learning models played a significant role in achieving results in as fast as possible time by providing the necessary computational power. The most important components of the VM we used are:

- **Processor.** The server was equipped with a high-performance *Intel Haswell* architecture processor, with 32 cores, allowing for parallelized computation during various stages of our models' development.
- **Caches.** The server's caching system was designed to enhance computational efficiency by minimizing data retrieval times. It featured multiple levels of cache, with capacities

distributed across the various cache types, such as:

  - L1 Data Cache: 1 MiB (32 instances),
  - L1 Instruction Cache: 1 MiB (32 instances),
  - L2 Cache: 128 MiB (32 instances),
  - L3 Cache: 512 MiB (32 instances).

- **Memory.** To handle large datasets and complex machine learning algorithms, the server was equipped with a substantial amount of *Random Access Memory* (RAM). The memory configuration includes a total of 96 GB, distributed across six modules of 16 GB each.

Chapter 9

User interface

9.1 Web application presentation

Apart from our main goal of creating a quality assessment model for 3D RNA structures, we also developed a web interface operated on the main application through the API. The website allows anyone to quickly upload their own files containing 3D RNA structures and start the processing on our server. The main focus while designing the application was to ensure user-friendly interface, which we achieved by splitting it into multiple smaller components. An example can be seen in Figure 9.1, which displays the main page of the website.

We allow users to choose the files to evaluate using two different methods, which can be used separately or both at once:

- by their PDB ID (Figure 9.2) – we included a preselected list of valid structure identifiers as well as a search bar that allows users to process structures directly from Protein Data Bank [53]. The entered ID is first validated by its length. If successful, a request is made to RNAsolo [20] to verify whether it is a valid 3D RNA structure,
- by uploading files from local drive (Figure 9.3) – uploaded files undergo extension verification before being sent to the backend for a quick check for easily noticeable errors.

The interface supports upload of multiple structures at a time, which then join a queue and wait until there are enough free resources on the server. After the task is dequeued, the processing starts. Each structure is evaluated by our model and the results represented by predicted RMSD values are saved in the database. On the initial request, user is also given a unique task identifier, which is used to create a corresponding URL link that allows one to access results of the submitted task. The task id can also be remembered and introduced by the user to a dedicated field shown in Figure 9.4. The results page consists of two stages. During the resource processing phase, the current status is displayed: step-by-step breakdown of already completed operations and future ones. An example can be seen in Figure 9.5. The last step is the showcase of generated results.

Once the whole process of evaluation of uploaded 3D RNA structures is completed, users can employ the previously generated task id to access the results. In case of any errors that occurred during the whole process, such as improper file content, invalid structure of uploaded data, an internal server error, or any other problem, a descriptive message will be displayed. An example of the results page can be seen in Figure 9.6.

All the submitted data and the task results are removed after a week from the task completion time to save server's resources.

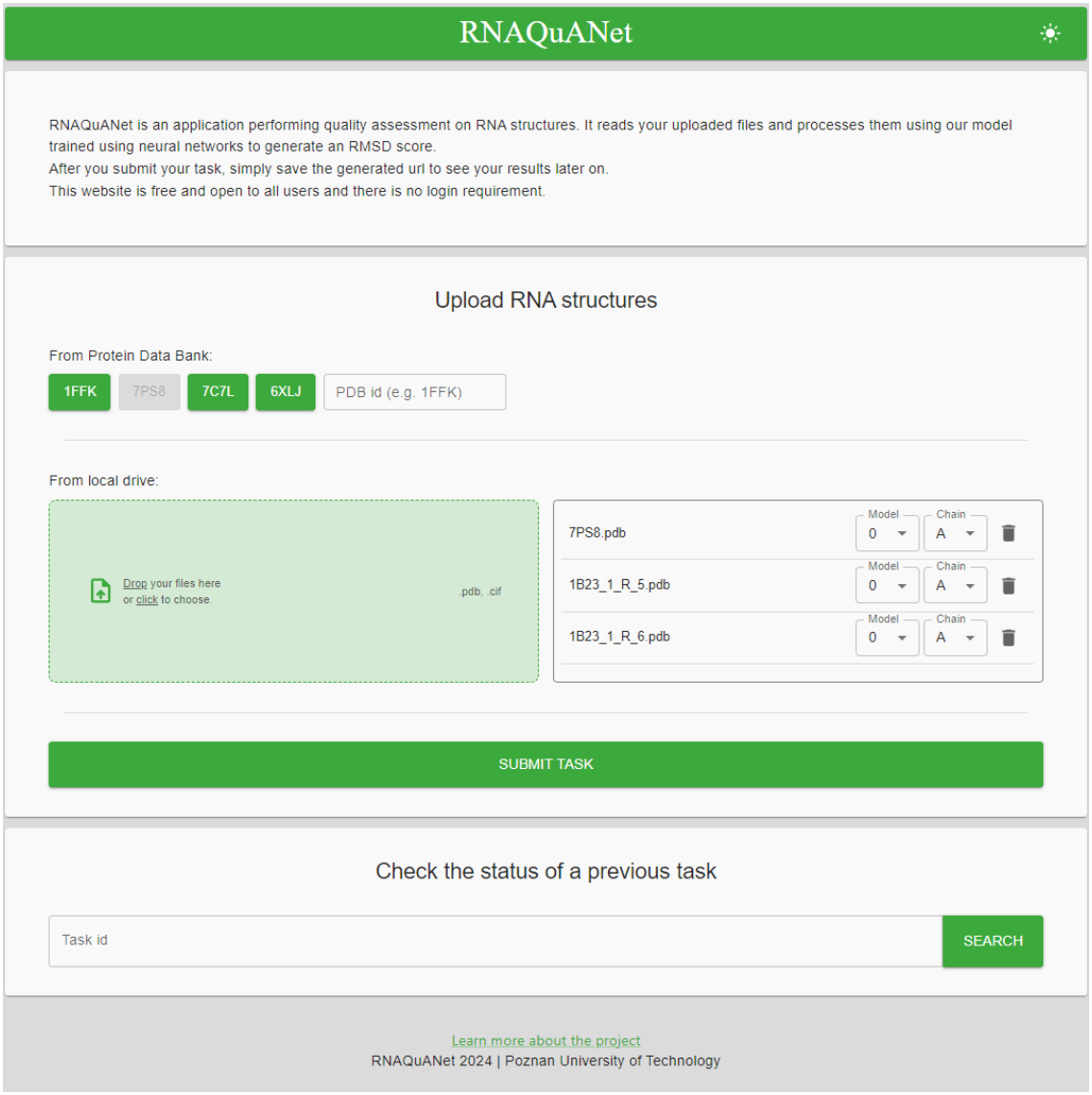


FIGURE 9.1: Main page of the developed RNAQuANet web application.

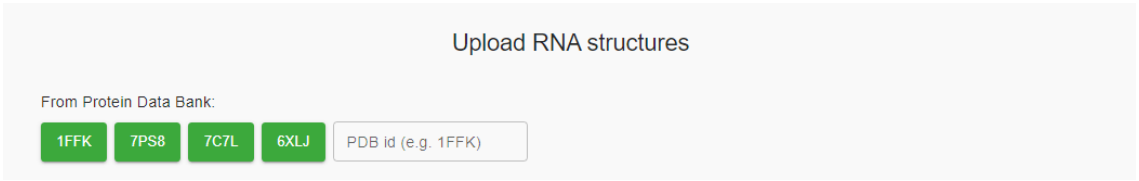


FIGURE 9.2: A predefined examples of 3D RNA structures from Protein Data Bank API and a custom search PDB ID field.

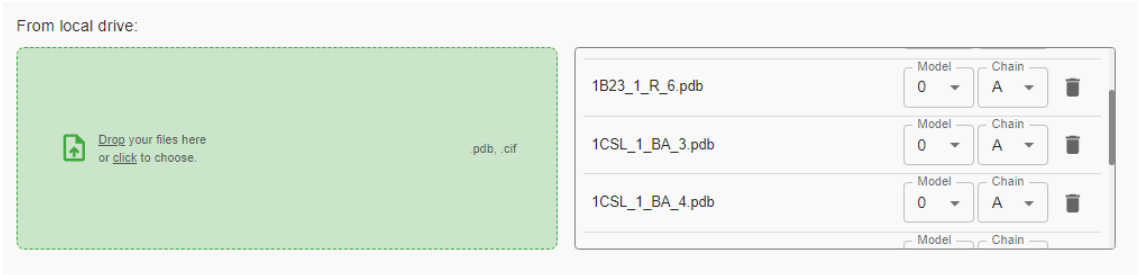


FIGURE 9.3: A drag and drop component allowing users to upload 3D RNA structures from local drive.

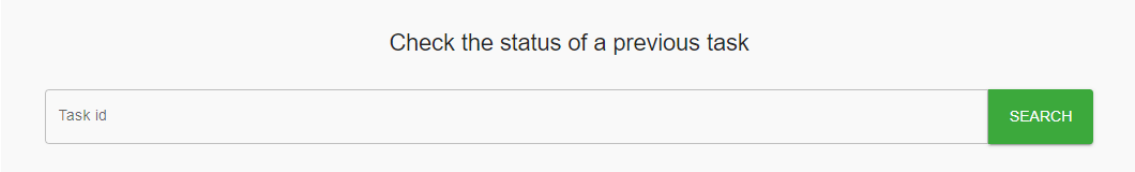


FIGURE 9.4: Component that allows users to search for the results of the task.

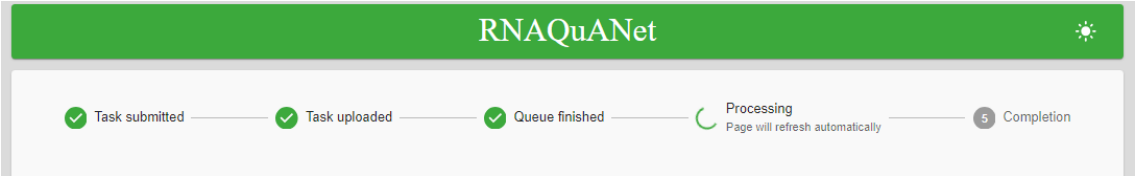


FIGURE 9.5: User preview of the current status of the task in form of steps.

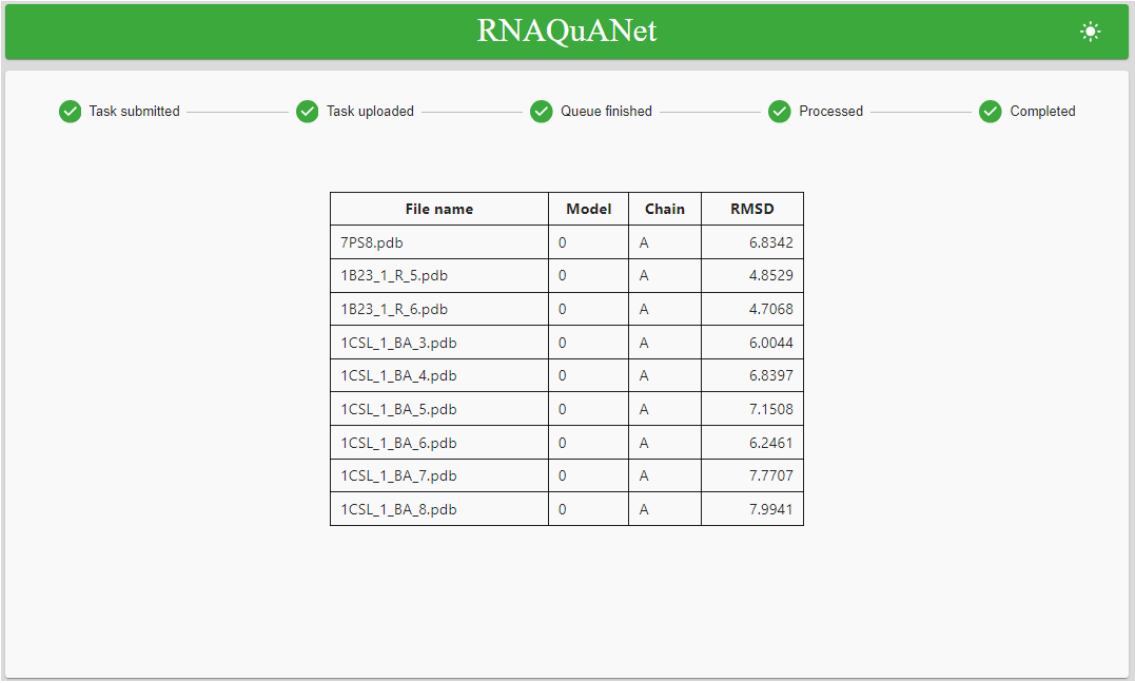


FIGURE 9.6: Task's result page.

9.2 Command line interface description

Another method of interacting with the application is the CLI (*Command Line Interface*). It enables customization of the processing by providing additional options when executing the scripts. The CLI also allows for a deeper examination of every part of the processing, observing file inputs and outputs, as well as running the process on a local machine. There are six standalone scripts available:

- `download_preprocessed_data` – download already preprocessed, i.e. HDF5 files containing *PyTorch Geometric* `Data` objects representing specified train, validation, and test sets,
- `download_raw_data` – download a raw dataset containing files requiring preprocessing before being suitable for training,
- `preprocess_data` – preprocess a given dataset into a format suitable for training, as described in Chapter 3,

- `train_network` – initiate training on a provided dataset that has already been preprocessed. As a result, a trained and complete model is returned,
- `eval_network` – evaluates a specified model, providing relevant statistics, i.e. average mean squared error over RMSD scores,
- `get_rmsd` – utilizes the trained model to compute the RMSD score for a given 3D RNA structure stored in the PDB [54] file.

All of the scripts mentioned above are configurable through the use of optional flags, as well as a configuration file. An example of the configuration file can be seen in Listing 9.1. In the file, we configure many parameters, such as a URL where the dataset should be downloaded from, names and paths to files created during the processing, additional options used during the feature extraction, GAT (*Graph Attention Network*) dropout rate, batch size, number of workers, and many more.

```yaml
name: 'rnaquadataset' # dataset name
tools_path: '/app/tools' # path to directory containing external tools used in the
    pipeline
verbose: True
data:
  path: '/app/data' # path to directory where data should be saved
  download:
    url: 'https://www.dropbox.com/scl/fi/ajnl5fhsdj2hjjvtnutyj/rnaquadataset.tar.gz
        ?rlkey=7uxlfhr2gigdxavwbgkg86o1x&dl=1' # URL to download the dataset from
    archive_ext: 'tar.gz' # archive type (tar / tar.gz)
    train_folder: 'train' # archive path to directory with a train dataset
    val_folder: 'val' # archive path to directory with a validation dataset
    test_folder: 'test' # archive path to directory with a test dataset
    train_csv: 'train.csv' # path to a CSV file containing RMSD values for each RNA
         structure from the train dataset
    val_csv: 'val.csv' # path to a CSV file containing RMSD values for each RNA
        structure from the validation dataset
    test_csv: 'test.csv' # path to a CSV file containing RMSD values for each RNA
        structure from the test dataset
    csv_delimiter: ',' # separator used in CSV files (usually a comma)
    csv_rmsd_column_name: 'rmsd' # name of a column containing RMSD values inside
        the CSV file
    csv_structure_column_name: 'description' # name of a column containing a names
        of files referencing the RMSD inside the CSV file
  download_preprocessed:
    train_url: '' # URL to download a H5 file containing train dataset
    val_url: '' # URL to download a H5 file containing validation dataset
    test_url: '' # URL to download a H5 file containing test dataset
  features:
    atom_for_distance_calculations: "C1'"
    max_euclidean_distance: '16.0'
    regenerate_features_when_exists: True # if True: if feature extraction file
        already exists, remove it and create it again
network:
  model_output_path: '/app/models' # path to save the trained model
  hidden_dim: 128
  layer_type: 3
  num_of_heads: 1
  num_of_layers: 4
  num_of_node_features: 96
  batch_norm: False
  gat_dropout: 0.5
  lr: 0.001
  weight_decay: 0.3
  scheduler_step_size: 10
  scheduler_gamma: 0.5
  batch_size: 1000
  num_workers: 1
  shuffle_train: True
  shuffle_val: False
  shuffle_test: False
  max_epochs: 50
```

Listing 9.1: An example of a configuration file.

Chapter 10

Summary

Our work successfully achieved all aims and objectives set in advancing the field of structural bioinformatics. The primary goal was to explore the application of graph neural networks for assessing the quality of three-dimensional RNA structures. Moreover, we prepared a representative, non-redundant and diverse training set of 3D RNA structures and developed the pipeline for its easy and efficient extending when new RNA molecules occur. Lack of high-quality training sets is currently a critical obstacle in applying machine learning techniques for 3D RNA structure analysis and modeling. All outlined objectives have been met and the proposed framework has demonstrated its potential for the task at hand.

All of our work has been done using *Python* scripts and command line environment, due to portability and resource efficiency. We focused on assembling a very diverse set of tools and data, designing top-rank *GRN* architectures and training our models to our best ability. As our work focused on quality assessment of three-dimensional RNA structures, we wanted to provide an easy way to evaluate a given 3D RNA structure using developed models. Considering that, we decided to implement an user-friendly web interface that allows the user to do just that.

10.1 Repository

To facilitate further research and collaboration, all scripts, datasets, and related resources developed during the thesis have been published in an open repository on GitHub. Using Docker containers, we ensured replicability of our experiments, allowing researchers to build upon the established framework.

- **Project repository:** `https://github.com/maciejbilinski/rnaquanet`
- **ARES image:** `https://hub.docker.com/r/adamczykb/ares_qa`

10.2 Future works

Possible extensions to our framework include:

1. Developing more sophisticated architectures that can capture even finer details of the intricate connectivity patterns within three-dimensional RNA structures.
2. Enhancing feature extraction to focus more on atoms instead of nucleotides.
3. Integration of experimental data to enhance accuracy of quality assessments.

4. Further development of user-friendly tools and interfaces, including integration of visualization tools, to help researchers interpret and validate the results more effectively.

5. Exploring potential biases in the datasets used for training and evaluations, as well as methods useful in mitigating biases and ensuring fair predictions.

Bibliography

[1] Mariusz Popenda, Marta Szachniuk, Maciej Antczak, Katarzyna J. Purzycka, Piotr Lukasiak, Natalia Bartol, Jacek Blazewicz, and Ryszard W. Adamiak. Automated 3D structure composition for large RNAs. *Nucleic Acids Research*, 40(14):e112–e112, April 2012.

[2] Maciej Antczak, Mariusz Popenda, Tomasz Zok, Joanna Sarzynska, Tomasz Ratajczak, Katarzyna Tomczyk, Ryszard Walenty Adamiak, and Marta Szachniuk. New functionality of RNAComposer: application to shape the axis of miR160 precursor structure. *Acta Biochimica Polonica*, 63(4), March 2017.

[3] Joanna Sarzynska, Mariusz Popenda, Maciej Antczak, and Marta Szachniuk. RNA tertiary structure prediction using RNAComposer in CASP15. *Proteins: Structure, Function, and Bioinformatics*, 91(12):1790–1799, August 2023.

[4] Zhichao Miao, Ryszard W. Adamiak, Maciej Antczak, Robert T. Batey, Alexander J. Becka, Marcin Biesiada, Michał J. Boniecki, Janusz M. Bujnicki, Shi-Jie Chen, Clarence Yu Cheng, Fang-Chieh Chou, Adrian R. Ferré-D'Amaré, Rhiju Das, Wayne K. Dawson, Feng Ding, Nikolay V. Dokholyan, Stanisław Dunin-Horkawicz, Caleb Geniesse, Kalli Kappel, Wipapat Kladwang, Andrey Krokhotin, Grzegorz E. Łach, François Major, Thomas H. Mann, Marcin Magnus, Katarzyna Pachulska-Wieczorek, Dinshaw J. Patel, Joseph A. Piccirilli, Mariusz Popenda, Katarzyna J. Purzycka, Aiming Ren, Greggory M. Rice, John Santalucia, Joanna Sarzynska, Marta Szachniuk, Arpit Tandon, Jeremiah J. Trausch, Siqi Tian, Jian Wang, Kevin M. Weeks, Benfeard Williams, Yi Xiao, Xiaojun Xu, Dong Zhang, Tomasz Zok, and Eric Westhof. RNA-Puzzles Round III: 3D RNA structure prediction of five riboswitches and one ribozyme. *RNA*, 23(5):655–672, January 2017.

[5] Zhichao Miao, Ryszard W. Adamiak, Maciej Antczak, Michał J. Boniecki, Janusz Bujnicki, Shi-Jie Chen, Clarence Yu Cheng, Yi Cheng, Fang-Chieh Chou, Rhiju Das, Nikolay V. Dokholyan, Feng Ding, Caleb Geniesse, Yangwei Jiang, Astha Joshi, Andrey Krokhotin, Marcin Magnus, Olivier Mailhot, Francois Major, Thomas H. Mann, Paweł Piątkowski, Radoslaw Pluta, Mariusz Popenda, Joanna Sarzynska, Lizhen Sun, Marta Szachniuk, Siqi Tian, Jian Wang, Jun Wang, Andrew M. Watkins, Jakub Wiedemann, Yi Xiao, Xiaojun Xu, Joseph D. Yesselman, Dong Zhang, Yi Zhang, Zhenzhen Zhang, Chenhan Zhao, Peinan Zhao, Yuanzhe Zhou, Tomasz Zok, Adriana Żyła, Aiming Ren, Robert T. Batey, Barbara L. Golden, Lin Huang, David M. Lilley, Yijin Liu, Dinshaw J. Patel, and Eric Westhof. RNA-Puzzles Round IV: 3D structure predictions of four ribozymes and two aptamers. *RNA*, 26(8):982–995, May 2020.

[6] Andriy Kryshtafovych, Maciej Antczak, Marta Szachniuk, Tomasz Zok, Rachael C. Kretsch, Ramya Rangan, Phillip Pham, Rhiju Das, Xavier Robin, Gabriel Studer, Janani Durairaj, Jerome Eberhardt, Aaron Sweeney, Maya Topf, Torsten Schwede, Krzysztof Fidelis, and John Moult. New prediction categories in CASP15. *Proteins: Structure, Function, and Bioinformatics*, 91(12):1550–1557, June 2023.

[7] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, September 1976.

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens

Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, July 2021.

[9] Joana Pereira, Adam J. Simpkin, Marcus D. Hartmann, Daniel J. Rigden, Ronan M. Keegan, and Andrei N. Lupas. High-accuracy protein structure prediction in CASP14. *Proteins: Structure, Function, and Bioinformatics*, 89(12):1687–1699, July 2021.

[10] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. *Computing Research Repository (arXiv)*, November 2015.

[11] Jun Li, Wei Zhu, Jun Wang, Wenfei Li, Sheng Gong, Jian Zhang, and Wei Wang. RNA3DCNN: Local and global quality assessments of RNA 3D structures using 3D deep convolutional neural networks. *PLOS Computational Biology*, 14(11):e1006514, November 2018.

[12] Gokul Yenduri, Ramalingam M, Chemmalar Selvi G, Supriya Y, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj G, Rutvij H Jhaveri, Prabadevi B, Weizheng Wang, Athanasios V. Vasilakos, and Thippa Reddy Gadekallu. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *arXiv*, May 2023.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv*, June 2014.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv*, December 2013.

[15] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv*, September 2016.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *arXiv*, October 2017.

[17] Raphael J. L. Townshend, Stephan Eismann, Andrew M. Watkins, Ramya Rangan, Maria Karelina, Rhiju Das, and Ron O. Dror. Geometric deep learning of RNA structure. *Science*, 373(6558):1047–1051, August 2021.

[18] David Sehnal, Alexander Rose, Jaroslav Koca, Stephen Burley, and Sameer Velankar. Mol*: Towards a Common Library and Tools for Web Molecular Graphics, 2018.

[19] H. M. Berman. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[20] Bartosz Adamczyk, Maciej Antczak, and Marta Szachniuk. RNAsolo: a repository of cleaned PDB-derived RNA 3D structures. *Bioinformatics*, 38(14):3668–3670, June 2022.

[21] Neocles B. Leontis and Craig L. Zirbel. *Nonredundant 3D Structure Datasets for RNA Knowledge Extraction and Benchmarking*, pages 281–298. Springer Berlin Heidelberg, 2012.

[22] Maciej Antczak, Marta Kasprzak, Piotr Lukasiak, and Jacek Blazewicz. Structural alignment of protein descriptors – a combinatorial model. *BMC Bioinformatics*, 17(1), September 2016.

[23] Asmaul Hosna, Ethel Merry, Jigmey Gyalmo, Zulfikar Alom, Zeyar Aung, and Mohammad Abdul Azim. Transfer learning: a friendly introduction. *Journal of Big Data*, 9(1), October 2022.

[24] PyTorch Geometric. `https://pytorch-geometric.readthedocs.io`.

[25] Tomasz Puton, Lukasz P. Kozlowski, Kristian M. Rother, and Janusz M. Bujnicki. CompaRNA: a server for continuous benchmarking of automated methods for RNA secondary structure prediction. *Nucleic Acids Research*, 41(7):4307–4323, February 2013.

[26] Marcin Magnus, Maciej Antczak, Tomasz Zok, Jakub Wiedemann, Piotr Lukasiak, Yang Cao, Janusz M Bujnicki, Eric Westhof, Marta Szachniuk, and Zhichao Miao. RNA-Puzzles toolkit: a computational resource of RNA 3D structure benchmark datasets, structure manipulation, and evaluation tools. *Nucleic Acids Research*, December 2019.

[27] José Almeida Cruz, Marc-Frédérick Blanchet, Michal Boniecki, Janusz M. Bujnicki, Shi-Jie Chen, Song Cao, Rhiju Das, Feng Ding, Nikolay V. Dokholyan, Samuel Coulbourn Flores, Lili Huang, Christopher A. Lavender, Véronique Lisi, François Major, Katarzyna Mikolajczak, Dinshaw J. Patel, Anna Philips, Tomasz Puton, John Santalucia, Fredrick Sijenyi, Thomas Hermann, Kristian Rother, Magdalena Rother, Alexander Serganov, Marcin Skorupski, Tomasz Soltysinski, Parin Sripakdeevong, Irina Tuszynska, Kevin M. Weeks, Christina Waldsich, Michael Wildauer, Neocles B. Leontis, and Eric Westhof. RNA-Puzzles: A CASP-like evaluation of RNA three-dimensional structure prediction. *RNA*, 18(4):610–625, February 2012.

[28] Dunja Mladenić. *Feature Selection for Dimensionality Reduction*, pages 84–102. Springer Berlin Heidelberg, 2006.

[29] Maciej Antczak, Tomasz Zok, Maciej Osowiecki, Mariusz Popenda, Ryszard W. Adamiak, and Marta Szachniuk. RNAfitme: a webserver for modeling nucleobase and nucleoside residue conformation in fixed-backbone RNA structures. *BMC Bioinformatics*, 19(1), August 2018.

[30] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, EDBT/ICDT '11. ACM, March 2011.

[31] myHDF5: Explore & Visualize HDF5 Files. `https://myhdf5.hdfgroup.org`.

[32] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. *arXiv*, September 2020.

[33] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, September 2022.

[34] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning – Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[35] PyTorch. `https://pytorch.org`.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, January 2014.

[37] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding Pooling in Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2022.

[38] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, November 2020.

[39] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10. ACM, April 2010.

[40] `argparse` – Python 3.11 documentation. `https://docs.python.org/3.11/library/argparse.html`.

[41] Docker. https://www.docker.com.

[42] Flask. https://flask.palletsprojects.com/en/3.0.x.

[43] SQLAlchemy. https://www.sqlalchemy.org.

[44] Redis Queue. https://redis.com/glossary/redis-queue.

[45] Werkzeug. https://werkzeug.palletsprojects.com/en/3.0.x/.

[46] Flasgger. https://github.com/flasgger/flasgger.

[47] Swagger. https://swagger.io.

[48] SQLite. https://www.sqlite.org.

[49] React. https://react.dev.

[50] Vite. https://vitejs.dev.

[51] TypeScript. JavaScript with syntax for types. https://www.typescriptlang.org.

[52] Material UI. https://mui.com/.

[53] Stephen K Burley, Helen M Berman, Charmi Bhikadiya, Chunxiao Bi, Li Chen, Luigi Di Costanzo, Cole Christie, Jose M Duarte, Shuchismita Dutta, Zukang Feng, Sutapa Ghosh, David S Goodsell, Rachel Kramer Green, Vladimir Guranovic, Dmytro Guzenko, Brian P Hudson, Yuhe Liang, Robert Lowe, Ezra Peisach, Irina Periskova, Chris Randle, Alexander Rose, Monica Sekharan, Chenghua Shao, Yi-Ping Tao, Yana Valasatava, Maria Voigt, John Westbrook, Jasmine Young, Christine Zardecki, Marina Zhuravleva, Genji Kurisu, Haruki Nakamura, Yumiko Kengaku, Hasumi Cho, Junko Sato, Ju Yaen Kim, Yasuyo Ikegawa, Atsushi Nakagawa, Reiko Yamashita, Takahiro Kudou, Gert-Jan Bekker, Hirofumi Suzuki, Takeshi Iwata, Masashi Yokochi, Naohiro Kobayashi, Toshimichi Fujiwara, Sameer Velankar, Gerard J Kleywegt, Stephen Anyango, David R Armstrong, John M Berrisford, Matthew J Conroy, Jose M Dana, Mandar Deshpande, Paul Gane, Romana Gáborová, Deepti Gupta, Aleksandras Gutmanas, Jaroslav Koča, Lora Mak, Saqib Mir, Abhik Mukhopadhyay, Nurul Nadzirin, Sreenath Nair, Ardan Patwardhan, Typhaine Paysan-Lafosse, Lukas Pravda, Osman Salih, David Sehnal, Mihaly Varadi, Radka Vařeková, John L Markley, Jeffrey C Hoch, Pedro R Romero, Kumaran Baskaran, Dimitri Maziuk, Eldon L Ulrich, Jonathan R Wedell, Hongyang Yao, Miron Livny, and Yannis E Ioannidis. Protein Data Bank: the single global archive for 3D macromolecular structure data. *Nucleic Acids Research*, 47(D1):D520–D528, October 2018.

[54] PDB file format.
https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/introduction.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.