Albert -Ludwig University of Freiburg

Department of Bio-informatics

Benchmarking Genomic Foundation models on predicting RNA-binding protein sites.

Shivani Sharma

Supervisor: **Michael Uhl**

Master of Science in Computer Science

May 31, 2025

Declaration

I, Shivani Sharma, of the Department of Computer Science, University of Freiburg, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

Shivani Sharma

Abstract

The rapid advancement in AI has revolutionised the world of genomics. In recent years, we have seen the heavy use of transformers in genomics, further advancing foundational DNA or RNA language models. This paper presents a benchmarking study of six foundational models, which include DNABERT [Ji et al. (2021)], DNABERT-2 [Zhou et al. (2024)], RNA-FM [Chen et al. (2022)], Caduceus [Schiff et al. (2024)], GeneMask [Roy et al. (2023)], and RNAErnie [Yin et al. (2024)]. The benchmarking task for these models was to find the RNA-binding protein site. We used datasets of different proteins to analyse the models using the same dataset. The models were benchmarked on various factors, including preliminary factors such as F1 score, accuracy, and loss. Every model has its working architecture and tokenisation. We utilised the pre-trained models with their weights and created fine-tuning scripts to get the results for our tasks. Our observation provides a platform to choose and optimise the DNA and RNA foundational models, assisting and guiding researchers in implementing these tools while getting a better understanding.

Github link: https://github.com/SharmaShivani12/Benchmarking-Genomic-Foundation-models-on-predicting-RNA-binding-protein-sites/tree/main

Contents

Chapter 1

Introduction

The last two decades have been enriching for bioinformatics. Advances in natural language processing have helped the foundation language model decode information from genetic data and make it available for further use. As with pre-training, the large language models (LLM) are categorized into language, vision, graph and multimodal. Language models like GPT, Llama, and Mistral have proven successful in the current era and have revolutionised the information world. Their application in the other world has also proven to be remarkable. The systematic categorisation of the models helps researchers select the most appropriate model based on the specific bioinformatic task. This advancement in the bioinformatics field helps scientists carry out complex tasks more efficiently and provides advantages such as large-scale analysis, better predictive capabilities, and robust design capabilities.

The foundation of life, DNA (deoxyribonucleic acid), is expressed in four letters or, to be more specific, in four bases: adenine(A), guanine(G), thymine(T) and cytosine(C). DNA and RNA are essential building blocks of life on this earth. Just like DNA, RNA is also very crucial in carrying out cell functions. In RNA, thymine is replaced by uracil (U), about 5% of the RNA is mrna (coding RNA), and the remaining serves as ncrna (noncoding RNA). These ncRNAs help in cell signalling, post-transcriptional regulation, and gene expression.

The critical aspect of DNA and RNA foundation models is the sequencing embeddings. When we feed the sequence to a model (a transformer-based model like BERT, Nucleotide Transformer), we get hidden states, usually representing the tokens. The two popular strategies are sentence-level summary token and mean token embedding. This embedding is crucial in showing the model's understanding of the sequence data, which can further assist in different downstream tasks such as sequence classification and structure prediction. In our study, we added a few RNA-foundation models because the dataset we are targeting is RNA-binding protein sites, and it would be interesting to see how the RNA-foundation models perform on these datasets. This study demonstrates an unbiased and comprehensive analysis of the state-of-the-art foundation models. We have targeted multiple models associated with different architectures, and we collected different datasets of proteins to benchmark the model's performance and examine its overall generalizability. Our study provides a comparative analysis to understand the factors that influence performance, the strengths and limitations of the model, and the model design.

1.1 DNA Foundation Models

We selected models with different architectures to benchmark foundation models. We have trained and evaluated six models on each dataset. In the following, we briefly describe the models:

- **DNABERT-1**: The BERT principle is a transformer-based contextualised language representation model. DNABERT's [Ji et al. (2021)] pretraining process was similar to the original BERT model in removing the next sentence prediction part, as stated in the paper. The model size is 86 M trainable parameters. This model takes the collection of sequences tokenised into k-mers, including the CLS (a tag to represent the meaning of the entire sequence), SEP (sentence separator), and MASK (a representation of masked K-mers) tokens. We used the pre-trained model DNBERT6 in our tasks, which takes a K-mers value of six. As shown in the figure below, the input sequence is first passed through the embedding layer and then the transformer blocks of 12 layers. The first output of the last hidden state is used for the sentence-level classification, and the rest is for the token-level classification.

  For our downstream task, predicting RNA binding protein sites, we adjusted the DNABERT model's fine-tune script for binary classification. To use the script, we prepared the dataset as K-mers. The custom scripts were created to transform the raw data into an acceptable model format. The data was then split into the train, test, and validation sets with a ratio of 60 (train) and 40 (test and validation) ratio. Once the training and validation were done, the results of the test set were stored in a CSV file for later comparative analysis.

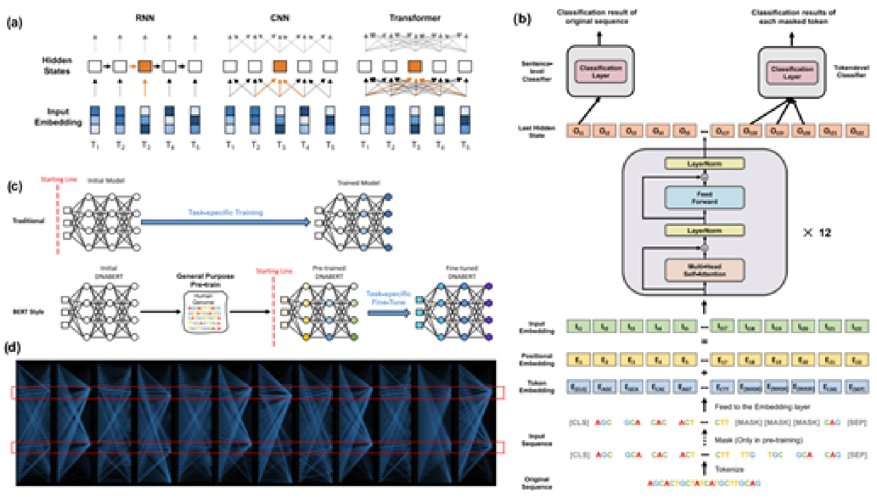

Figure 1.1: Architecture and characteristics of the DNABERT model. From "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in the genome" by Yanrong Ji et.al, Volume 37, Issue 15, August 2021, Pages 2112–2120.

- **DNABERT-2**: A multi-species genome foundation model replaces K-mer tokenisation with byte pair encoding (BPE). BPE [Sennrich et al. (2016)] is a data compression technique in which the common consecutive byte of data is replaced by the byte that has not occurred in the data. DNABERT-2 incorporates SentencePiece [Kudo and Richardson (2018)] with BPE to tokenise the DNA sequences. DNABERT-2 [Zhou et al. (2024)] adopts attention with linear biases ALiBi instead of fixed-sized learned positional embeddings. This helps the model overcome the input length limitation associated with traditional transformers. DNABERT-2 has 117 million trainable parameters and 768 output embedding dimensions. To avoid memory and computation issues related to standard attention mechanisms, it utilises FlashAttention [Dao et al. (2022)] algorithms and low-precision layer normalisation by employing low-rank adaptation (LoRa) [Hu et al. (2021)]. LoRA helps the model avoid overfitting, save memory, and provide faster training. Instead of training full weight matrices, it adds trainable matrices to the original weights. We used the finetune script on GitHub for our RNA-binding protein site prediction tasks. We downloaded the pre-trained model and its weights and adjusted the parameters according to our task. Ultimately, we saved the results for further inference-related tasks. Initially, a few errors were faced while running the model, as the scripts provided in the repository require some customization. We used the same data as that used in DNABERT-1. Thus, the same custom data preparation scripts were used here.

- **GENEMASK**: Large-language models like DNABERT-1 do not use gene-based semantics. In other words, they use a small, fixed-size window like a k-mer (subsequence of length k). This basic tokenisation technique doesn't emphasise the functional information or the biological significance associated with that subsequence, such as whether the subsequence is a coding region or a promoter. Unlike other large-language models, GENEMASK [Roy et al. (2023)] picks a random position or mask centre (if m=9, then mask-centres are MC1. . . MC9), as shown in step 1 (see the figure below). Then it searches for nucleotides around those MC. For each MC, the sequence is tokenised into k-mers. The score of Normalised Point-wise Mutual Information (how informative or surprising the sequence is) is calculated for these k-mers. Each MC, as per their local region, is then mapped to a new candidate based on the MPMI score (Maximum Point-wise Mutual Information). We rank the MC centres based on their MPMI scores and divide them into high-set and low-set. As shown in step 5, in the high-set, the model masks the contiguous 11 tokens centred around the MC; on the other hand, it masks short spans for low sets as low as six tokens. In addition to the above details, the overall trainable parameters are nearly 110 M. We downloaded the pretrained model from GitHub and then fine-tuned it on our dataset for our task.

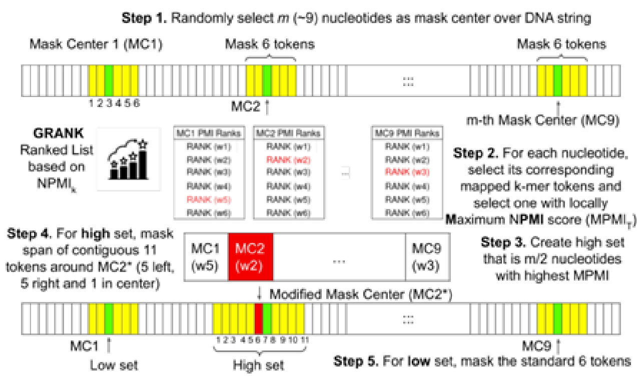
Step 1. Randomly select *m* (~9) nucleotides as mask center over DNA string
Mask Center 1 (MC1)
Mask 6 tokens
Mask 6 tokens
1 2 3 4 5 6
MC2
m-th Mask Center (MC9)
GRANK
Ranked List based on NPMI$_k$
MC1 PMI Ranks
RANK (w1)
RANK (w2)
RANK (w3)
RANK (w4)
RANK (w5)
RANK (w6)
MC2 PMI Ranks
RANK (w1)
RANK (w2)
RANK (w3)
RANK (w4)
RANK (w5)
RANK (w6)
MC9 PMI Ranks
RANK (w1)
RANK (w2)
RANK (w3)
RANK (w4)
RANK (w5)
RANK (w6)
Step 2. For each nucleotide, select its corresponding mapped k-mer tokens and select one with locally Maximum **NPMI** score (MPMI$_T$)
Step 4. For **high** set, mask span of contiguous 11 tokens around MC2* (5 left, 5 right and 1 in center)
MC1 (w5)
MC2 (w2)
...
MC9 (w3)
Step 3. Create high set that is m/2 nucleotides with highest MPMI
Modified Mask Center (MC2*)
1 2 3 4 5 6 7 8 9 10 11
MC1
Low set
High set
MC9
Step 5. For **low** set, mask the standard 6 tokens

Figure 1.2: Steps involved in the GENEMASK model. From "GENEMASK: Fast Pretraining of Gene Sequences to Enable Few-Shot Learning" by Soumyadeep Roy et.al, 29th July 2023.

1.2 CADUCEUS : Mamba Block Architecture Model

The Caduceus model uses the Mamba architecture [Gu and Dao (2023)] for the analysis, an improved version of the state-space model. The Mamba model incorporates an extra selection mechanism, which helps parameterize the state-space model's parameters based on the input. This selection mechanism allows the model to filter out irrelevant information and focus on relevant information for the task and the future. Because of the selection mechanism's addition, they are termed a selective state-space model. Selective SSMs [Soydan et al. (2024)] are fully recurrent models that provide high-quality language and genomics performance. Also, Caduceus uses cuBLAS-based matrix operations and is subject to a non-deterministic nature.

Caduceus [Schiff et al. (2024)] is the first RC (Reverse Complement) bidirectional long-range DNA model, which means it models both strands of DNA. The model incorporates long-range Mamba Block, thus handling hundreds of thousands of nucleotides. It extends 'BiMamba' to handle RC equivariance, resulting in 'MambaDNA' block. Overall, the Caduceus model size is approximately 7.73 M. The figure below shows that the input sequence (one-hot encoded sequence X1: T) first goes through the token-embedding layer. This model incorporates two 'BiMamba' blocks, each using state-space models and convolutions. The channels are split and flipped to maintain the forward and reverse complement symmetry. The input is divided into two as it goes through the channel dimensions. One split goes through the reverse complement (RC) operation, and the parameters are shared for forward and reverse complement sequence processing through the module. The reverse sequence goes through RC operation before being concatenated with the forward output. Ultimately, the LM-head linear projection layer provides the Y1: T logits as the predictions.

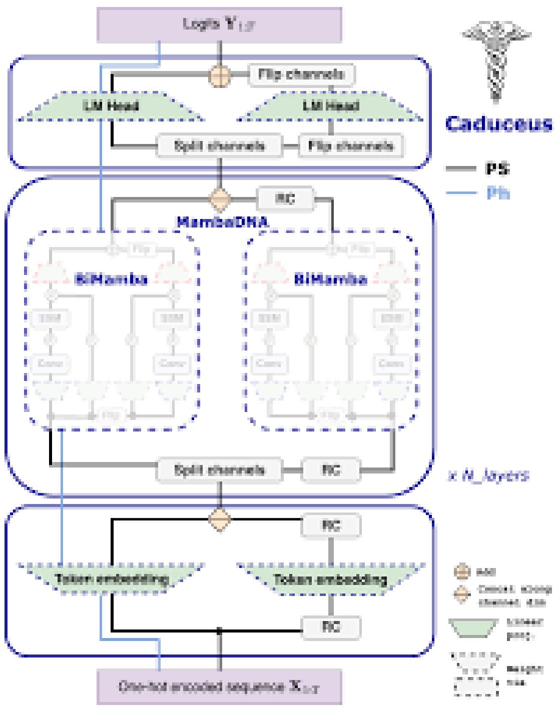

Figure 1.3: Caduceus Architecture. From "Caduceus: Bi-Directional Equivariant Long-Range DNA Sequence Modelling" by Yair Schiff et.al, 5th March 2024.

1.3 RNA Foundation Models

- **RNAErnie**: A pretrained RNA language model incorporating the Enhanced Representation through Knowledge Integration (ERNIE) [Yin et al. (2024)] framework. Previously, we discussed RNA-FM, which uses the standard masking technique as a pretraining strategy. This can result in losing high-density biological information hidden in contiguous RNA sequences. The RNAErnie model handles this challenge by introducing a motif-aware pretraining strategy, recognising motifs as critical units and using targeted masking. Motif-aware masking involves base-level masking, subsequence-level masking, and motif-level random masking. This type of masking helps the RNA model to enrich its RNA sequence representation with subsequence and motif-level knowledge. The model size of RNAErine is approximately 86.7 M.

  Furthermore, it supports type-guided fine-tuning with stacking, which means a stacking-based architecture utilises the predicted RNA type to guide the fine-tuning. A raw RNA sequence goes through a model, and the pretrained block generates the output embedding. Based on the output embedding, it predicts the coarse-grained RNA types. Afterwards, the sequence and the predicted RNA type are fed to the RNAErnie block and task-specific head.

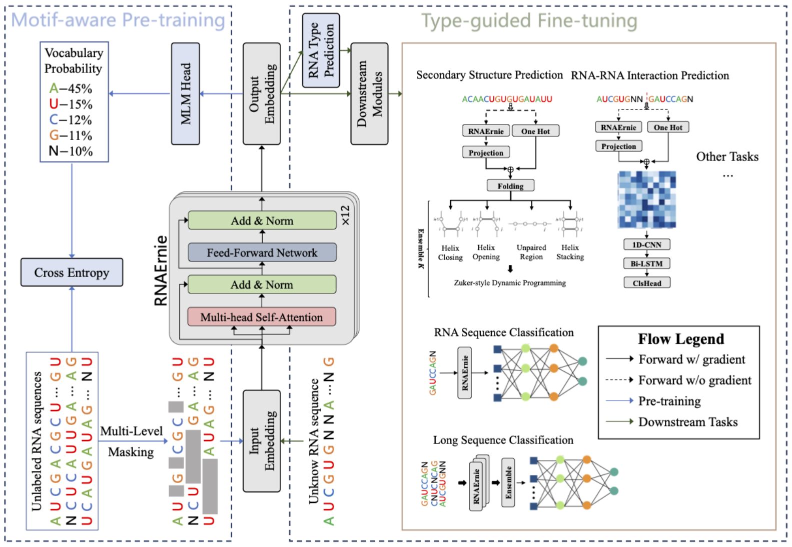

Figure 1.4: Architecture and characteristics of the RNAErnie model. From "Multi-purpose RNA language modelling with motif-aware pretraining and type-guided fine-tuning" by Ning Wang et.al, 01 Jan 2024.

- **RNA-FM**:The RNA-FM [Chen et al. (2022)] stands for RNA foundation model, a large pre-trained model explicitly designed for RNA sequences. It works on the self-supervised learning principle, meaning it does not need labels. This model is trained on 23 million RNA sequences, thus covering more than 800,000 species. The model uses a similar masking strategy to the BERT model, masking 15% of tokens. With the help of RNA-FM, you can perform two types of structure-related tasks, such as secondary structure prediction or other structure-related and function-related tasks. The RNA-FM model has 12 transformer layers with multi-head self-attention, which means it simultaneously looks at different parts of an input sequence and a feedforward network. The model size of the RNA-FM model is approximately 99 M. After training, it produces a (L $\times$ 640) embedding matrix for each RNA sequence. In our RNA binding site prediction task, we used the pre-trained model from Google Drive, which the paper's authors shared on GitHub, and developed a fine-tuning script for our classification task.

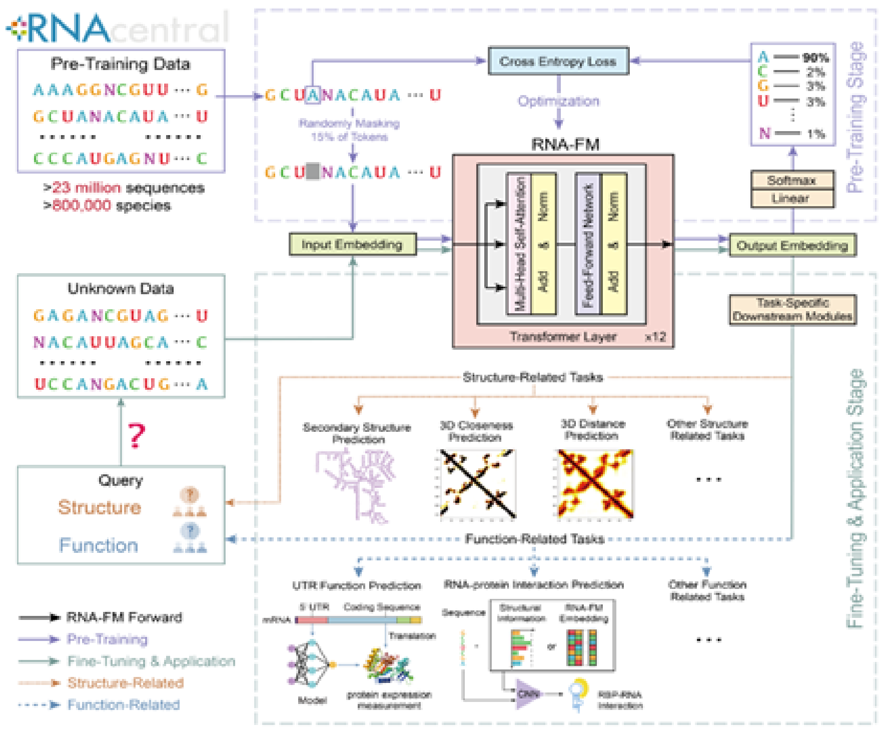

Figure 1.5: Architecture and characteristics of the RNA-FM model. From "Interpretable RNA Foundation Model from Unannotated Data for Highly Accurate RNA Structure and Function Predictions" by Jiayang Chen et.al, 1st April 2022.

Chapter 2

Methodology

2.1 Software-Environment and Dataset

For our analysis, we used the de.NBI cloud machine. The de.NBI cloud is a big data platform with training modules, tools, and services.It provides storage and computation resources for the bioinformatics projects. It provides access to the remote desktop with a graphical interface in addition to the command line, which can be accessed from your local machine and allows you to do high-end research from anywhere. This lets you choose a specific GPU or CPU according to your task or project. Once you have set up the configuration on the machine, you can access your remote folders on your local machine and even sync with your code editor via SSH connection. As shown in the figure below, we created a separate environment for each model after cloning the repository to avoid a version conflict. We used the respective model's requirement.txt files or environment.yml, which are available in the GitHub repository. This also helps streamline the task and analysis. Afterwards, the datasets are loaded, and fine-tuning scripts are created for each model.

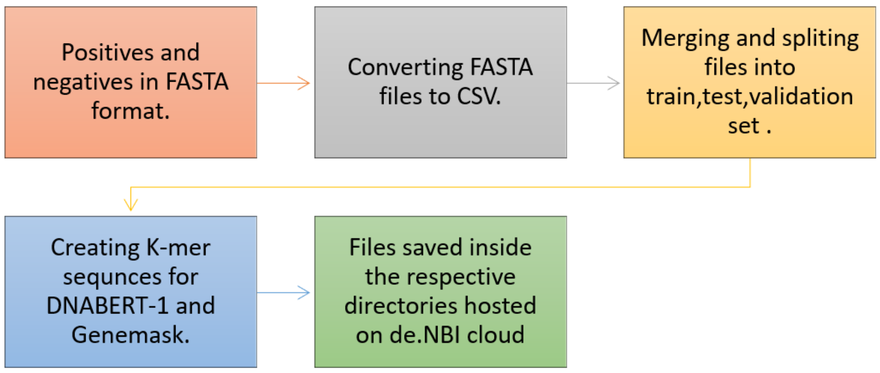
Positives and negatives in FASTA format.
Converting FASTA files to CSV.
Merging and spliting files into train,test,validation set .
Creating K-mer sequnces for DNABERT-1 and Genemask.
Files saved inside the respective directories hosted on de.NBI cloud

Figure 2.1: Datas-Preparation Flow-chart.

We collected a different set of protein data with their positives and negatives in FASTA format. A FASTA file is a text-based format representing nucleotides and amino acid sequences. We converted our FASTA files to CSV format to use them with our analysis. For our task, we created custom scripts in Python using libraries like Bio and Seqio. Later, we merged the CSV files and split the dataset into a ratio of 60% (train) and 40% (test and validation). The creation of the K-mer step is explicitly dedicated to models which use K-mer tokenisation, such as DNABERT-1 and Genemask. Finally, we saved the dataset in their respective folder, where we named the folders by the protein names. We collected different sets of proteins;

below is each file's description.

- **HNRNPM, HNRNAPL**: Heterogeneous nuclear ribonucleoprotein.
- **MATR3** : Matrin 3 helps maintain the structure of the chromatin.
- **PTBP1**: Polypyrimidine tract binding protein 1. It plays a vital role in RNA metabolism
- **QKI**: QKI, KH domain containing RNA binding. It belongs to the STAR protein family.
- **ILF3** :Interleukin enhancer binding factor 3. It is a double-stranded RNA-binding protein.
- **KHSRP**: KH-type splicing regulatory protein.

2.2 Fine-tuning

Fine-tuning is specific to each model; we cannot use the same script for all the models. Every model has a different internal architecture, and this internal architecture defines the various parts of the scripts, which include the following sections:

Input format: This is the input for the model, like token IDS in our case. However, it can vary from model to model. For example, BERT architecture models require masks, position embeddings, and token types. On the other hand, models depend on the Mamba architecture, a state-space model that needs fewer parameters than transformer-based encoder models utilise sequential embeddings.

Output heads: this block sits atop the base model, converting the embedding to predictions. In the transformer-based models, you get a big attention map and extract from the middle.

Model Configuration: The number of layers, the activation function, the weights, and the network. In our demonstration, we used models with the BERT and Mamba architectures. For all the models, the running parameters were kept the same, like the number of epochs: 70, but we implemented early-stopping with the patience count of 3, learning rate: 3e-5, batch size: 16 and 8, and weight decay: 0.01.

1. **Loading Pre-trained model** : We begin the fine-tuning script by loading the pretrained model and libraries required for the script to analyse. The loading part can be achieved using auto classes (like AutoModel, AutoTokenizer, etc.) from the HuggingFace transformers library. Sometimes the model is unavailable on Hugging Face; checking the GitHub repository is better. We have provided links to all the GitHub and Huggiface repositories in our references section.

2. **Dataset Pre-processing and Dataloader setup**: After loading the model, we need to set the data path and create a function that applies the tokenisation, as shown in the example below. The function's max-length defines the sequence length after which truncation should be done. In this step, we define the data loader using the Huggingface DataCollatorWithPadding for batch collation, dynamic padding, and wrapping the tokenised data to PyTorch loaders.

3. **Trainer Setup and Evaluation:** Trainers provide an optimized loop to transformer models. Our project used different training and evaluation metrics based on the pretrained model and the machine type. The first way is the HuggingFace Trainer class, which provides an API for complete training in the Pytorch package, providing built-in

compute metrics for our task. Evaluate the model after the prediction is done on the full dataset. The trainer calls the compute metrics and passes the prediction. Inside the compute metric function, we extract the ground truth and prediction. Later, we calculate the metrics (accuracy, precision, F1-score, recall and loss) and return the values as a dictionary, such as an example's finetuning script of RNAErnie. In Pytorch native Trainer, which uses the torchmetrics library to compute the metrics, we can get the evaluation metrics in the middle of the batch or epoch, based on the training loop. The integration style is the only difference between the PyTorch and HuggingFace API trainers. In the Pytorch native trainer, you must manually specify the custom training loop and function, such as training, evaluation, and prediction functions. For reference, you can check the DNABERT-1 and DNABERT-2 fine-tuning scripts, because there is no built-in class.

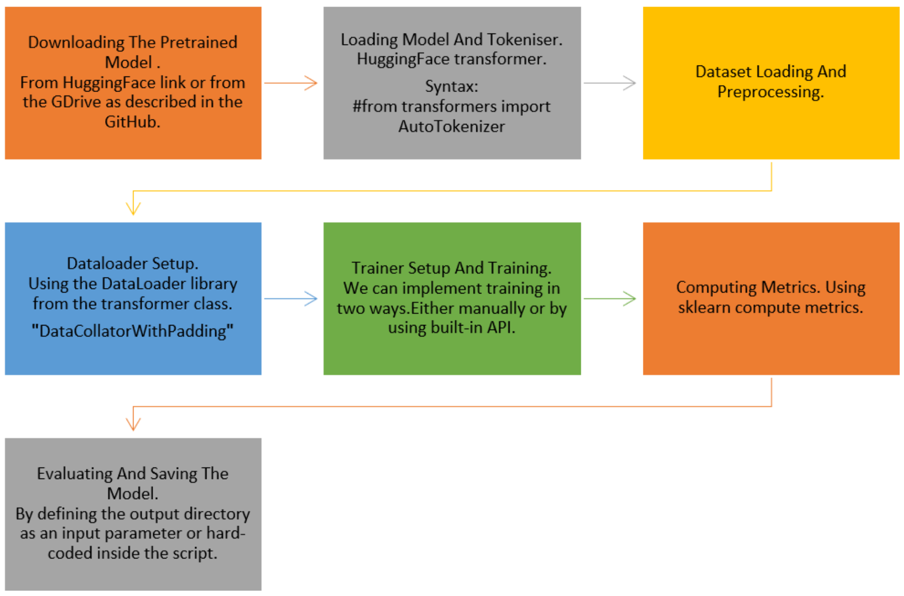
Downloading The Pretrained Model .
From HuggingFace link or from the GDrive as described in the GitHub.
Loading Model And Tokeniser. HuggingFace transformer.
Syntax:
#from transformers import AutoTokenizer
Dataset Loading And Preprocessing.
Dataloader Setup.
Using the DataLoader library from the transformer class.
"DataCollatorWithPadding"
Trainer Setup And Training.
We can implement training in two ways.Either manually or by using built-in API.
Computing Metrics. Using sklearn compute metrics.
Evaluating And Saving The Model.
By defining the output directory as an input parameter or hard-coded inside the script.

Figure 2.2: Pipeline for Fine-tuning Script.

2.3 Evaluation Metrics

All the parameters used for benchmarking depend on the confusion matrix, which evaluates the model's predictive capabilities by identifying the amount of time the model predicted True Positives, True Negatives, False Positives, and False Negatives.

For our task, we kept the datasets the same for all the foundation models and benchmarked them on three parameters: Accuracy, Loss, and F1-score. Accuracy is a metric which tells how often a model predicts the correct output. Furthermore, we used the F1 score, which provides a more comprehensive evaluation of the tasks. It takes into account both false positives and negatives. The F1 score is evaluated based on precision and recall metrics. It assesses the model based on class-wise performance, which is critical to tasks like classification. Then we have a loss function for inference; we used the log loss function. Loss functions, sometimes called error functions, quantify the difference between the predicted output and the target value. For the inference, we have used log-loss.

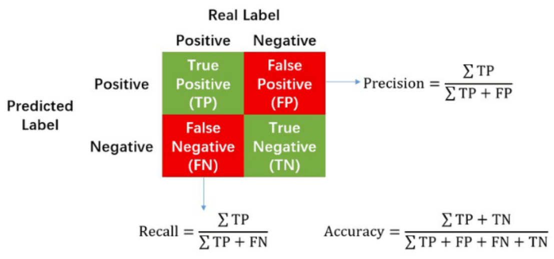
Real Label
Positive
Negative
Predicted Label
Positive
Negative
True Positive (TP)
False Positive (FP)
False Negative (FN)
True Negative (TN)
Precision = $\frac{\sum TP}{\sum TP + FP}$
Recall = $\frac{\sum TP}{\sum TP + FN}$
Accuracy = $\frac{\sum TP + TN}{\sum TP + FP + FN + TN}$

Figure 2.3: Confusion matrix and Accuracy Calculation.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 2.4: F1-score Formulation.

$$Logloss = \frac{1}{N}\sum_{i=1}^{N} logloss_i$$

$$Logloss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

Figure 2.5: Log-loss function: i = obersvation ; y = true value; p = prediction probability and ln = natural logarithm.

Chapter 3

Results

3.1 Observations :

The tables below describe the performance of all the models for the RNA-binding site prediction tasks (Binary Classification). As the table indicates, DNABERT-2 [Zhou et al. (2024)]has outperformed all the foundation models. Its accuracy remains far better than the RNA foundation models, considering at times nearly 94% for some of the data. Although DNABERT-2 has performed better in all the metrics, its performance is slightly better than that of DNABERT-1, Caduceus (a long-range Mamba model) [Schiff et al. (2024)], and Genemask [Roy et al. (2023)] for some datasets.Another significant observation was the number of epochs for models like DNABERT-1 and Caduceus, which learned quickly compared to other models. In most cases, the early stop occurred before reaching epoch 4. The RNA-FM model ran for more epochs (in most cases, nearly 20) than all the other models. At one instant, it ran for 56 epochs before early stopping occurred. This signifies the superiority of the Byte Pair Encoding (BPE) based method over another type of tokenisation, like overlapping K-mer tokenisation.

We evaluated the models using fixed parameters like the number of epochs, learning rate, batch size, and weight decay. Considering the effect of our fixed parameter, DNABERT-2 has performed well, but DNABERT-1 [Ji et al. (2021)] and Genemask have shown promising results. The stats of the RNA-foundation model [Chen et al. (2022)] [Yin et al. (2024)] shows that further fine-tuning can also achieve promising results because every model has an architecture sensitive to specific parameters, which might not be the case for others. (Discussed in the section Conclusion).

Table 3.1: Accuracy (%)

Model/Dataset	GTF2F1	HNRNAPL	HNRNPM	PTBP1	ILF3	MATR3	KHSRP	QKI
DNABERT-1	87.64	87.87	88.39	92.78	74.89	88.54	83.47	90.61
CADUCEUS	85.16	81.59	85	91.52	70.27	86.74	74.11	80.57
GENEMASK	88.2	87.4	87.49	92.55	76.09	88.85	81.76	93.67
DNABERT-2	89.56	85.79	86.91	93.1	75.49	89.11	83.37	93.89
RNA-FM	86.07	81.98	80.55	89.3	65.3	83.54	66.98	70.96
RNAERNIE	82.73	72.35	81.68	89.75	60.67	83.54	64.37	76.2

Table 3.2: F1-Scores

Model/Dataset	GTF2F1	HNRNAPL	HNRNPM	PTBP1	ILF3	MATR3	KHSRP	QKI
DNABERT-1	0.8785	0.877	0.8881	0.9249	0.7463	0.888	0.8354	0.8986
CADUCEUS	0.8524	0.8164	0.8553	0.912	0.7333	0.8701	0.7606	0.8187
GENEMASK	0.8803	0.8678	0.8841	0.9251	0.7466	0.8931	0.8307	0.9368
DNABERT-2	0.8905	0.8493	0.8705	0.9284	0.7317	0.8968	0.826	0.9386
RNA-FM	0.8556	0.8145	0.8179	0.8899	0.6809	0.8461	0.6686	0.7297
RNAERNIE	0.8187	0.7475	0.8298	0.8927	0.597	0.8437	0.7187	0.7753

Table 3.3: Loss-Function values

Model/Dataset	GTF2F1	HNRNAPL	HNRNPM	PTBP1	ILF3	MATR3	KHSRP	QKI
DNABERT-1	4.4552	4.3717	4.1859	2.6028	9.0495	4.1288	5.9588	3.384
CADUCEUS	5.3499	6.634	5.408	3.0559	10.7174	4.7798	9.332	7.0041
GENEMASK	4.2544	4.5398	4.509	2.6841	8.6171	4.0172	6.5752	2.2822
DNABERT-2	3.7614	5.1207	4.7197	2.4866	8.8333	3.9243	5.993	2.2035
RNA-FM	5.0213	6.4964	7.0093	3.8577	12.5087	5.9329	11.9004	10.4668
RNAERNIE	6.2264	9.9663	6.6019	3.695	14.1766	5.9329	12.8422	8.5781

Chapter 4

Discussion

Our comprehensive evaluation of DNA, RNA and MambaDNA foundation models provides a better understanding of the technicality and architecture of different models. In this experiment, we analysed the performance of the pre-trained model by fine-tuning it for a classification task and setting the fixed value of the hyperparameter. The stats show that further improvement may yield more promising results for DNA foundation models and Caduceus, such as pre-training the model instead of using a pre-trained model. Observation of the RNA foundation models signifies that certain areas need tweaking based on the model architecture and training, because what's working for one model might not work for another.

The caduceus model did not establish a deterministic nature, as it uses cuBLAS. In such a scenario, the Mamba model can implement further integration or adaptation to add determinism to the model. Increasing or decreasing the learning rate, or another hyperparameter, may yield better results. Considering the objective of the task, it is pretty evident that additional improvisation can be incorporated, like integrating the hyperparameter optimization (optuna [Akiba et al. (2019)] or Raytune), light-weight adapters (Lora) [Hu et al. (2021)] or incorporating the smart tokenisation such as SentencePiece. After fine-tuning the model in our task, we observed a significant difference in the performance of the DNA foundation and RNA foundation models since DNA foundation models are trained on the entire human genome with labeled tasks. However, RNA foundation models are trained on a limited dataset, leading to limited performance. In addition, RNA modeling is very hard and complex, adding more limitations to the performance of the models. In conclusion, our study demonstrates the areas that can be further explored to overcome the models' limitations. It also guides the model specification, helping other researchers perform specific genomic tasks.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M. (2019), 'Optuna: A next-generation hyperparameter optimization framework', https://arxiv.org/abs/1907.10902. arXiv preprint arXiv:1907.10902.

Chen, J., Hu, Z., Sun, S., Tan, Q., Wang, Y., Yu, Q., Zong, L., Hong, L., Xiao, J., Shen, T., King, I. and Li, Y. (2022), 'Interpretable rna foundation model from unannotated data for highly accurate rna structure and function predictions', https://arxiv.org/abs/2204.00300.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A. and Ré, C. (2022), 'Flashattention: Fast and memory-efficient exact attention with io-awareness', https://arxiv.org/abs/2205.14135. arXiv preprint arXiv:2205.14135.

Gu, A. and Dao, T. (2023), 'Mamba: Linear-time sequence modeling with selective state spaces', https://arxiv.org/abs/2312.00752. arXiv preprint arXiv:2312.00752, submitted on 1 Dec 2023, revised on 31 May 2024.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W. (2021), 'Lora: Low-rank adaptation of large language models', https://arxiv.org/abs/2106.09685.

Ji, Y., Zhou, Z., Liu, H. and Davuluri, R. V. (2021), 'DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome', *Bioinformatics* **37**.

Kudo, T. and Richardson, J. (2018), Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, *in* 'Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations', pp. 66–71.

Roy, S., Wallat, J., Sundaram, S. S., Nejdl, W. and Ganguly, N. (2023), 'Genemask: Fast pretraining of gene sequences to enable few-shot learning', https://arxiv.org/abs/2307.15933.

Schiff, Y., Kao, C.-H., Gokaslan, A., Dao, T., Gu, A. and Kuleshov, V. (2024), 'Caduceus: Bi-directional equivariant long-range dna sequence modeling', https://arxiv.org/abs/2403.03234.

Sennrich, R., Haddow, B. and Birch, A. (2016), 'Neural machine translation of rare words with subword units', https://arxiv.org/abs/1508.07909. arXiv preprint arXiv:1508.07909.

Soydan, T., Zubić, N., Messikommer, N., Mishra, S. and Scaramuzza, D. (2024), 'S7: Selective and simplified state space layers for sequence modeling', https://arxiv.org/abs/2410.02056. arXiv preprint arXiv:2410.02056.

Yin, W., Zhang, Z., He, L., Jiang, R., Zhang, S., Liu, G., Zhang, X., Qin, T. and Xie, Z. (2024), 'Ernie-rna: An rna language model with structure-enhanced representations', https://doi.org/10.1101/2024.03.17.585376.

Zhou, Z., Ji, Y., Li, W., Dutta, P., Davuluri, R. and Liu, H. (2024), 'Dnabert-2: Efficient foundation model and benchmark for multi-species genome', https://arxiv.org/abs/2306.15006.