T.C.

DOKUZ EYLÜL ÜNIVERSITESI

IZMIR INTERNATIONAL BIOMEDICINE AND GENOME INSTITUTE

DETECTION OF PRECURSOR AND MATURE miRNAS BY EMPLOYING DEEP LEARNING METHODS: DeepMirFinder

SEYEDEHSADAF ASFA

MOLECULAR BIOLOGY AND GENETICS

MASTER OF SCIENCE THESIS

IZMIR-2023

THESIS CODE: DEU.IBG.MSc-2020850027

T.C.

DOKUZ EYLÜL ÜNIVERSITESI

IZMIR INTERNATIONAL BIOMEDICINE AND GENOME INSTITUTE

DETECTION OF PRECURSOR AND MATURE miRNAS BY EMPLOYING DEEP LEARNING METHODS: DeepMirFinder

SEYEDEHSADAF ASFA

MOLECULAR BIOLOGY AND GENETICS

MASTER OF SCIENCE THESIS

IZMIR-2023

THESIS CODE: DEU.IBG.MSc-2020850027

T.C.

DOKUZ EYLÜL ÜNIVERSITESI

IZMIR INTERNATIONAL BIOMEDICINE AND GENOME INSTITUTE

DETECTION OF PRECURSOR AND MATURE miRNAS BY EMPLOYING DEEP LEARNING METHODS: DeepMirFinder

MOLECULAR BIOLOGY AND GENETICS

MASTER OF SCIENCE THESIS

SEYEDEHSADAF ASFA

Supervisor: Asst. Prof. ATHANASIA PAVLOPOULOU

THESIS CODE: DEU.IBG.MSc-2020850027

Dokuz Eylül University Izmir International Biomedicine and Genome Institute Department of Genomics and Molecular Biotechnology, Molecular Biology and Genetics graduate program Master of Science student Seyedehsadaf Asfa has successfully completed her Master of Science thesis titled **"Detection of Precursor and Mature miRNAs by employing Deep Learning methods: DeepMirFinder"** on the date of 12/06/2023.

HEAD OF THE COMMITTEE

Asst. Prof. Dr. Athanasia PAVLOPOULOU

MEMBER

Asst. Prof. Dr. Hani ALOTAİBİ

MEMBER

Prof. Dr. Ralph MEUWİSSEN

YÜKSEK LİSANS TEZ SAVUNMA SINAVI TUTANAĞI

Dokuz Eylül Üniversitesi İzmir Uluslararası Biyotıp ve Genom Enstitüsü Genom Bilimleri ve Moleküler Biyoteknoloji Anabilim Dalı, Moleküler Biyoloji ve Genetik Yüksek Lisans programı öğrencisi Seyedehsadaf Asfa **"Derin Öğrenme yöntemleri kullanılarak Öncü ve Olgun miRNA'ların Tespiti: DeepMirFinder"** konulu Yüksek Lisans tezini 12/06/2023 tarihinde başarılı olarak tamamlamıştır.

BAŞKAN
Dr. Öğr. Üyesi Athanasia PAVLOPOULOU

JÜRİ ÜYESİ
Dr. Öğr. Gör. Hani ALOTAİBİ

JÜRİ ÜYESİ
Prof. Dr. Ralph MEUWİSSEN

TABLE OF CONTENTS

TABLE OF CONTENTS i
INDEX OF FIGURES iii
LIST OF ABBREVIATIONS iv
ACKNOWLEDGEMENTS v
ÖZET 1
ABSTRACT 2
1. INTRODUCTION AND AIM 3
1.1. Statement and Importance of the Problem 3
1.2. Aim of the the Study 4
1.3. Hypothesis of the Study 4
2. General Information 5
2.1.1. miRNA biogenesis 5
2.1.2. Biological function of miRNAs 7
2.2. miRNA sequence prediction 8
2.2.1. Pre-miRNA prediction 8
2.2.2. Mature miRNA prediction 9
2.3. Deep Learning for miRNA prediction 10
2.3.1. CNN for miRNA prediction 10
2.3.2. CNN architecture 10
2.3.3. RNN architecture 11
2.3.3.1 Long Short-Term Memory 11
2.4. Introduction to Orthologs 11
2.5. Evolutionary Conservation of miRNA Target Sites 12
2.6. Regulation of Orthologous Genes by Common miRNAs 13
2.7. Comparative Analysis of miRNA-target Interactions 13
3. MATERIALS AND METHODOLOGY 14
3.1. Type of the Study 14
3.2. Time and Place of the Study 14
3.3. Materials and Methods of the Study 14
3.4. Variables of the Study 14
3.5. Tools for Data Collection 14
3.6. Study Plan and Calendar 15
3.7. Acquisition of sequences 16
3.7.1. Acquisition of miRNA sequences 16

3.7.2. Acquisition of non-miRNA hairpin sequences.....18
3.8. CCN model building.....**18**
3.8.1. Dataset preprocessing.....18
3.8.2. Vectorization.....23
3.8.3. Training the CNN model.....25
3.9. Identification of mature miRNAs.....**28**
3.11. Detaction of pre-miRNAs across species.....**47**
4. Results.....**53**
4.1. DeepMirFinder performance.....**53**
4.1.1. Model performance of predicting mature miRNAs.....53
4.1.2. Model performance of predicting miRNA hairpins.....56
4.2. DeepMirFinder description.....**58**
4.3. Identification of orthologous pre-miRNA sequences.....**60**
5. DISCUSSION.....**62**
6. CONCLUSIONS AND FUTURE ASPECTS.....**64**
7. REFERENCES.....**65**
8. APPENDIX.....**69**
8.1. Ethics Committee Approval.....**69**
8.2. Curriculum Vitae.....**70**

INDEX OF FIGURES

Figure 1. Illustrative depiction of the process of miRNA biogenesis and miRNA-mediated gene regulation in metazoa 6
Figure 2. Mechanisms underlying the release of microRNAs into the extracellular environment and their subsequent uptake by recipient cells 8
Figure 3. Illustration of the CNN architecture 10
Figure 4. Sample generated secondary structure for each miRNA sequence.....17
Figure 5. Sample dataset of pre-miRNA entries used for training 18
Figure 6. Sample dataset of negative data used for training 21
Figure 7. Sample dataset including positive and negative samples. 23
Figure 8. Receiver operating characteristic (ROC) curve.....32
Figure 9. Sample screensot of the Excel spreadsheet representing a detailed comparison of various pairs of miRNAs and their respective similarity scores, identity percentages, and similarity percentages. 50
Figure 10. Training and validation loss curve of DeepMirFinder for mature miRNAs. 53
Figure 11. ROC curve and AUC statistics illustrating the discriminatory ability of DeepMirFinder to classify true and false mature miRNAs.....55
Figure 12. Training versus validation loss curve of DeepMirFinder for precursor miRNAs 57
Figure 13. AUC-ROC curve representing the capability of DeepMirFinder of distinguishing between true and false pre-miRNAs. 58
Figure 14. DeepMirFinder screenshot showing the prediction output of candidate mature miRNA sequences 59
Figure 15. DeepMirFinder screenshot showing the output of candidate pre-miRNA sequences 60
Figure 16. Pairwise sequence alignments of orthologous miRNAs 61

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
HGNC	HUGO Gene Nomenclature Committee
LSTM	Long Short-Term Memory
miRNA	microRNA
pre-miRNA	precursor microRNA
RNN	Recurrent Neural Network
UTR	Untranslated Region

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my research supervisor, Asst. Prof. Athanasia Pavlopoulou for offering me the opportunity to carry out my research and providing me with invaluable guidance throughout my studies.

I am also grateful to my committee members Asst. Prof. Dr. Hani Alotaibi and Prof. Dr. Ralph Meuwissen for their valuable time and advice.

Finally I am grateful to my family, who has been supporting me emotionally and financially.

Derin Öğrenme yöntemleri kullanılarak Öncü ve Olgun miRNA'ların Tespiti: DeepMirFinder

Seyedehsadaf ASFA, İzmir Uluslarası Biyotıp ve Genom Merkezi, Dokuz Eylül Üniversitesi Sağlık Kampüsü, Balçova 35340 – İzmir / TÜRKİYE

ÖZET

Derin öğrenme, çeşitli makine öğrenme yaklaşımlarını geride bırakan bir tekniktir ve birçok biyolojik görevde yaygın olarak kullanılmaktadır. Bu çalışmada, aday prekürsör mikroRNA'ların (pre-miRNA'ların) hassas ve doğru bir şekilde tespiti için derin öğrenme yöntemi geliştirilmiştir. Ayrıca, çeşitli genomlardaki homolog pre-miRNA dizilerinin belirlenmesi için bir sinir ağı temelli yaklaşım da uygulanmıştır. Bu amaçla aşağıdaki adımlar gerçekleştirilmiştir: (1) veri toplama, hazırlama ve bölümleme, (2) insan prekürsör ve olgun miRNA'ları tahmin etmek için algoritma tasarımı (evrişimli sinir ağları ve tekrarlayan sinir ağları), (3) olası miRNA'ları etiketli pre-miRNA'lar ve miRNA olmayan yapılarla test ederek metodoloji değerlendirmesi, (4) taksonlar arasında (örneğin, ortologlar) ortolog miRNA'ların belirlenmesine yönelik bir kelime gömme tekniği geliştirme. Bu araştırma tezi bağlamında, hızlı ve doğru prekürsör ve olgun miRNA tahmini için bağımsız ve kullanıcı dostu bir uygulama geliştirilmiştir. Ayrıca, bu çalışmada geliştirilen derin öğrenme yöntemi, filogenetik olarak korunmuş prekürsör miRNA'ların keşfedilmesine olanak sağlar.

Anahtar kelimeler: prekürsör mikroRNA, olgun miRNA, derin öğrenme algoritmaları, konvolüsyonel sinir ağları, uzun-kısa süreli bellek, ortologlar

Detection of Precursor and Mature miRNAs by employing Deep Learning methods: DeepMirFinder

Seyedehsadaf ASFA, Izmir International Biomedicine and Genome Center, Dokuz Eylül University Health Campus, Balçova 35340 – Izmir / TURKEY

ABSTRACT

Deep learning, which outperforms various machine learning approaches, is widely used in multiple biological tasks. In this study, a deep learning method was developed for the sensitive and accurate identification of candidate precursor microRNAs (pre-miRNAs), as well as mature miRNAs. A neural network-based approach was also applied for the identification of homologous pre-miRNA sequences in diverse genomes. To this end, the following steps were carried out: (1) data collection, preparation and partition, (2) algorithm design (convolutional neural networks and recursive neural networks) to predict human precursor and mature miRNAs, (3) methodology evaluation by testing the putative miRNAs against annotated pre-miRNAs and non-miRNA structures, (4) development of a word embedding technique towards the identification of orthologous miRNAs across taxa (i.e. orthologs). In the context of this research thesis, a standalone and user-friendly application was developed for the fast and accurate prediction of *bona fide* precursor and mature miRNAs. In addition, a deep learning method developed in this study allows the discovery of phylogenetically conserved precursor miRNAs.

Keywords: precursor micoRNA, mature miRNA, deep learning algorithms, convolutional neural networks, long short-term memory, orthologs

1. INTRODUCTION AND AIM

1.1. Statement and Importance of the Problem

MicroRNAs, which are short non-coding RNA molecules, have a significant impact on gene regulation by targeting messenger RNAs (mRNAs) to either degrade them or inhibit their translation (O'Brien et al., 2018). Since their initial identification in the 1990s, miRNAs have been extensively studied, and their significance in diverse biological processes such as development, differentiation, and disease has gained increasing recognition (Tüfekci et al., 2014).

The discovery of new miRNAs and their corresponding target mRNAs is crucial for gaining insights into the molecular mechanisms involved in cellular processes and diseases. Nonetheless, experimental identification of miRNAs is a time-consuming and expensive process, often constrained by the availability of appropriate samples. As a result, computational techniques for predicting miRNA sequences have been devised as a valuable complement to experimental methods (Hamzeiy et al., 2014)

Existing approaches for miRNA prediction currently depend on detecting conserved patterns within the primary sequence or secondary structure of the associated miRNA precursors (pre-miRNAs). These pre-miRNAs are RNA hairpin molecules that undergo processing to become mature miRNAs (Takashima et al., 2022). While these methods have demonstrated effectiveness in identifying established miRNAs, they have certain limitations when it comes to predicting new miRNAs and recognizing non-conserved miRNAs.

In recent times, the progress made in deep learning techniques, such as convolutional neural networks (CNNs) (Vaz & Balaji, 2021) and long short-term memory (LSTM) algorithms, has exhibited significant potential across diverse domains, including bioinformatics. These methods enable the identification and prediction of intricate patterns and structures within extensive datasets. In the realm of miRNA prediction, deep learning approaches hold promise in enhancing accuracy and sensitivity by capturing a broader range of complex and diverse features present in miRNA sequences (Zheng et al., 2020).

Based on the aforementioned considerations, the objective of my research is to create and assess a deep learning methodology for accurately predicting novel,

authentic precursor and mature miRNA sequences. The primary aim is to overcome the drawbacks associated with existing approaches and enhance the precision and sensitivity of miRNA prediction. This investigation concentrates on the development and evaluation of CNN and LSTM models, which have demonstrated efficacy in diverse bioinformatics endeavors, including the analysis of DNA and protein sequences (Wang et al., 2021) .

The outcomes of this research possess the capacity to propel the domain of miRNA prediction forward and make significant contributions to the advancement of more precise and effective techniques for detecting new miRNAs and their corresponding target mRNAs (Zheng et al., 2020). Moreover, the implications of this study extend beyond the specific field, as it has the potential to impact the utilization of deep learning methods in the realms of bioinformatics and computational biology at large.

1.2. Aim of the the Study

The aim of the present study is to predict candidate precursor and mature miRNA sequences (Ryu et al., 2011) by employing a deep learning method (DeepMirFinder).

1.3. Hypothesis of the Study

It is hypothesized that precursor and mature miRNA sequences can be predicted with high confidence by employing deep learning methods.

2. General Information

2.1. Introduction to miRNAs

2.1.1. miRNA biogenesis

MicroRNAs are small RNA molecules that do not code for proteins and exert control over gene expression post-transcriptionally. The formation of miRNAs involves a regulated and intricate process (as depicted in Figure 1). It commences with the transcription of miRNA genes by RNA polymerase II, resulting in the production of long primary transcripts (pri-miRNAs) (Y. Lee et al., 2004). Typically, pri-miRNAs comprise one or more hairpin structures that undergo processing in the nucleus by the RNase III enzyme Drosha and its cofactor DGCR8 to generate pre-miRNA. The pri-miRNA is processed in two steps: firstly, Drosha cleaves the hairpin structure at its base, yielding a ~70 nucleotide pre-miRNA with a two-nucleotide overhang at the 3' end, and secondly, the overhang is eliminated by the exonuclease activity of a protein known as Pasha (Dexheimer & Cochella, 2020).

Following its formation, the pre-miRNA is transported to the cytoplasm through the aid of Exportin-5. Once in the cytoplasm, it undergoes further processing facilitated by the RNase III enzyme Dicer, resulting in the formation of a mature miRNA duplex that spans approximately 22 nucleotides in length. Among the two strands of the mature miRNA duplex, namely the guide strand and the passenger strand, the guide strand is incorporated into the RNA-induced silencing complex (RISC), while the passenger strand is degraded. The RISC-miRNA complex subsequently binds to the 3' untranslated region (UTR) of target mRNAs, leading to post-transcriptional gene silencing through translational repression or mRNA degradation (van den Berg et al., 2008).

The biogenesis of miRNAs is a tightly controlled process that necessitates the involvement of numerous protein factors and accessory molecules. For instance, the RNA-binding protein TRBP plays a role in stabilizing the Dicer complex, while the protein Ars2 assists in the processing of specific pre-miRNAs. Disruptions in the regulation of miRNA biogenesis have been linked to a range of diseases, including cancer, cardiovascular disorders, and neurological conditions. Consequently, comprehending the mechanisms underlying miRNA formation and regulation is of

utmost importance in gaining insights into disease pathogenesis and devising therapeutic approaches (Libri et al., 2013).

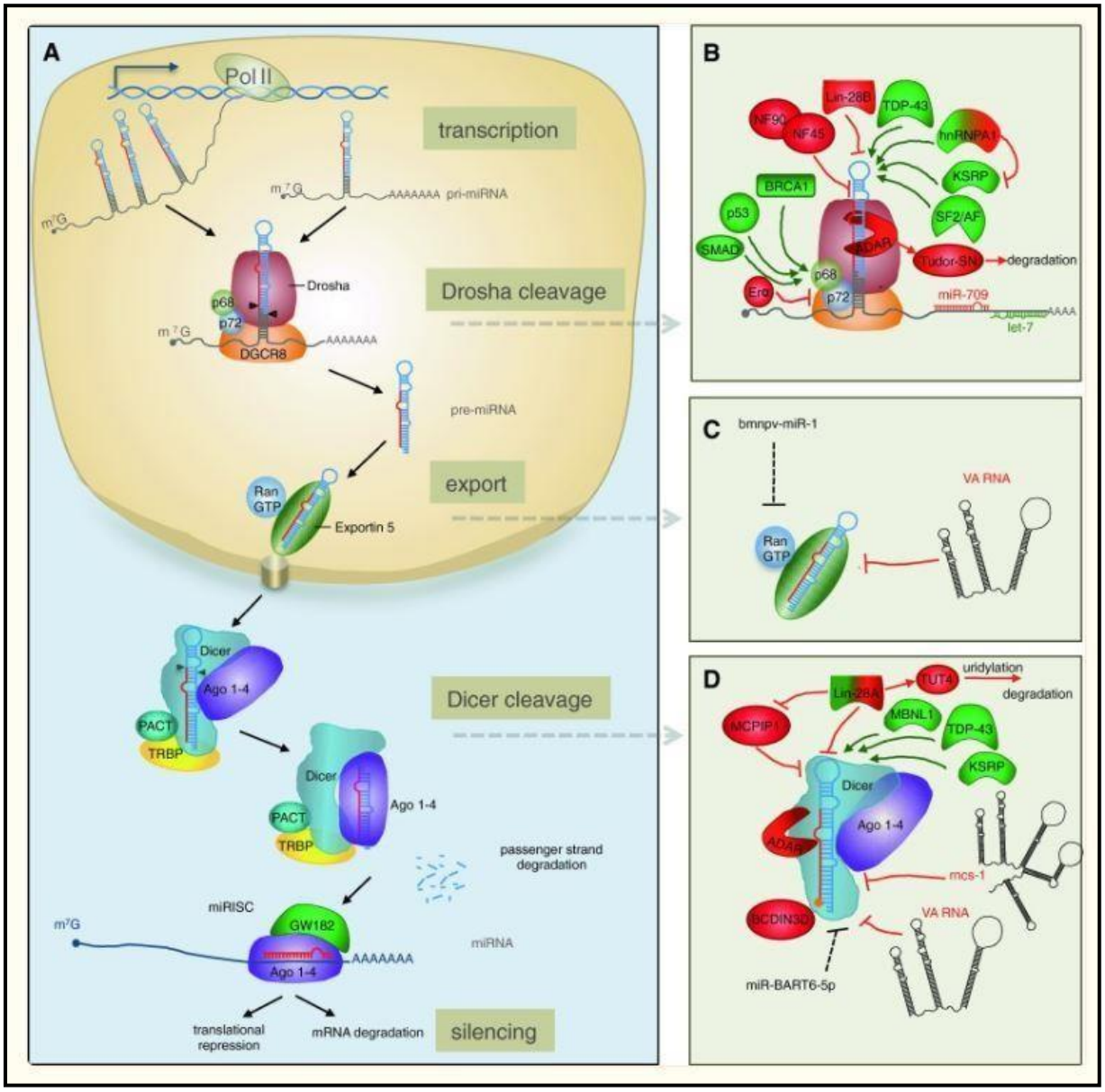


Figure 1. Illustrative depiction of the process of miRNA biogenesis and miRNA-mediated gene regulation in metazoa (adapted from Libri et al., 2013). A) The canonical biogenesis pathway involves the transcription of pri-miRNAs in the nucleus by RNA polymerase II, with a 7-methylguanosine-cap (m7G) and poly(A) tail. The pri-miRNA can encompass a single pre-miRNA or a cluster of pre-miRNAs, with the sequence corresponding to the mature miRNA highlighted in red. The microprocessor complex, composed of Drosha and the cofactor DGCR8, catalyzes the cleavage of the pri-miRNA within the nucleus. This cleavage process involves direct interactions with the DEAD-box RNA helicases p68 and p72. Exportin-5 binds to the pre-miRNA in the nucleus under high RanGTP levels and transports it to the cytoplasm through the

nuclear pore complex. In the cytoplasm, Dicer, assisted by the RNA-binding proteins TRBP and PACT, cleaves the stem of the pre-miRNA. The resulting transient miRNA/miRNA* duplex is then loaded onto an Argonaute (AGO) protein within the RNA-induced silencing complex (RISC), with one strand, known as the guide strand, being preferentially retained for gene targeting. B) The processing efficiency of pri-miRNAs by Drosha can be influenced positively (indicated in green) or negatively (indicated in red) by various proteins, either through direct interactions with the pri-miRNA or through interactions with auxiliary proteins p68 and p72 (represented by arrows). Additionally, mature miRNAs can also modulate pri-miRNA processing through downstream interactions of the stem-loop. C) The export of pre-miRNA can be regulated by two viral non-coding RNAs. D) Dicer is responsible for the cleavage of pre-miRNA and is subject to regulation.

2.1.2. Biological function of miRNAs

MiRNAs have a crucial role in regulating gene expression and have been implicated in a broad spectrum of biological processes, such as development, differentiation, metabolism, and immune response. Typically, miRNAs bind to the 3' UTR of target mRNAs through base-pairing interactions, leading to the inhibition of translation and/or degradation of the mRNA. The specificity of miRNA-mRNA interactions primarily relies on the level of complementarity between the miRNA seed region (nucleotides 2-8) and the target mRNA sequence (O'Brien et al., 2018).

By concurrently regulating the expression of multiple genes, miRNAs exhibit significant control over intricate biological processes. Disruptions in miRNA expression or function have been associated with various diseases, including cancer, cardiovascular disorders, and neurological conditions (Tüfekci et al., 2014).

Apart from their involvement in post-transcriptional gene regulation, miRNAs have emerged as key players in intercellular communication. Newer research has revealed that miRNAs can be released from cells into the extracellular environment. Once there, they can be internalized by recipient cells, influencing gene expression patterns (Ortiz-Quintero, 2020) (see Figure 2). This discovery has paved the way for the utilization of miRNA-based biomarkers in disease diagnosis and prognosis. Moreover, it has prompted investigations into the potential of miRNA-based therapeutics for treating various diseases.

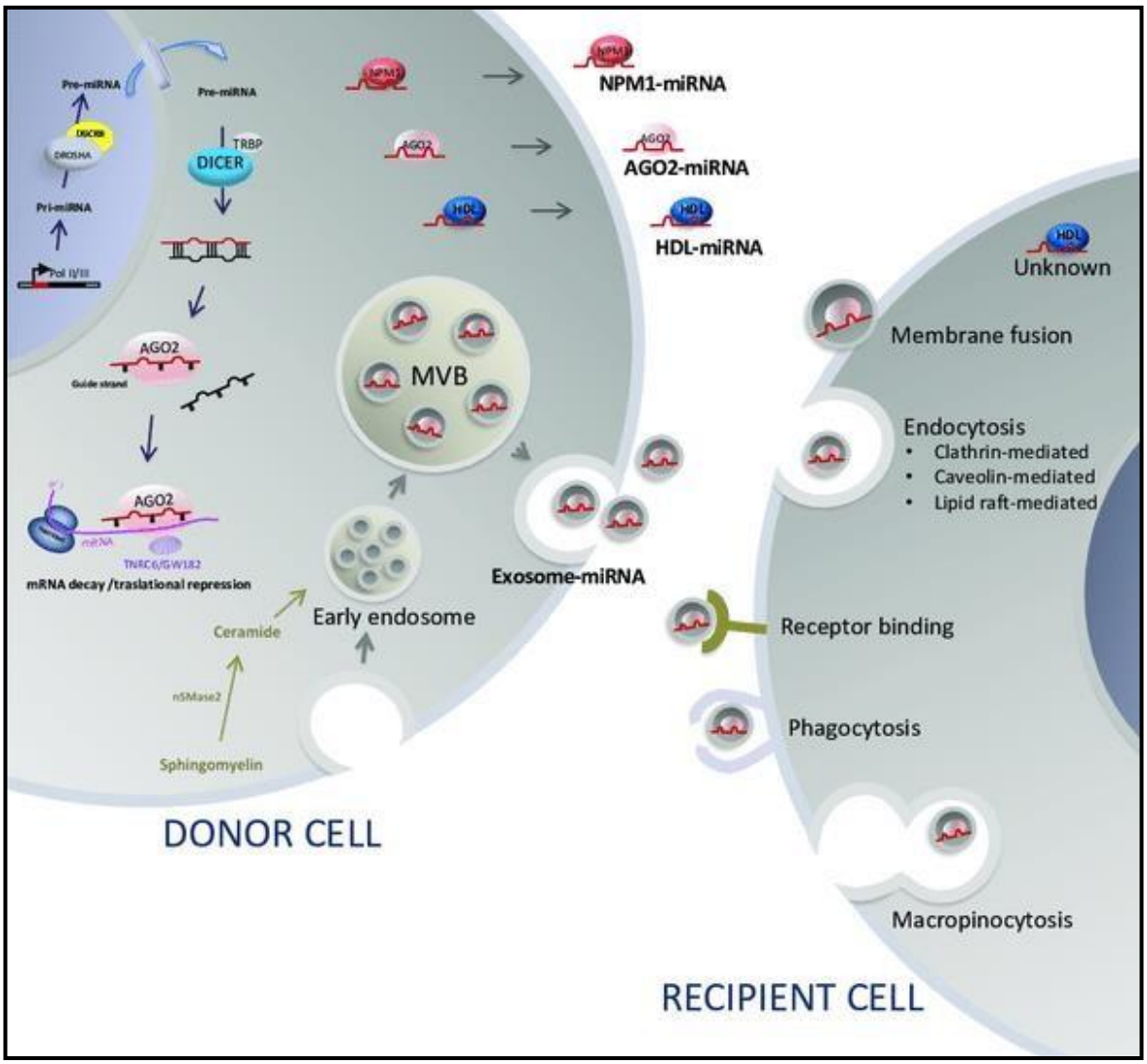


Figure 2. Mechanisms underlying the release of microRNAs into the extracellular environment and their subsequent uptake by recipient cells (adapted from Ortiz-Quintero, 2020). Donor cells secrete miRNAs through exosomes or extracellular vesicles, or as protein complexes. Following their release into the extracellular milieu, miRNAs are transported into recipient cells by exosomes through processes such as membrane fusion, endocytosis, receptor binding, phagocytosis, and macropinocytosis (Ortiz-Quintero, 2020)

2.2. miRNA sequence prediction

2.2.1. Pre-miRNA prediction

During the process of miRNA biogenesis, hairpin-like structures called pre-miRNAs are formed. The prediction of pre-miRNAs is a critical step in the discovery of novel miRNAs and the understanding of their biological functions.

In general, computational methods for pre-miRNA prediction rely on the utilization of machine learning algorithms to differentiate pre-miRNAs from other RNA sequences. These algorithms are trained on a dataset consisting of known pre-miRNAs and non-hairpin precursors. Subsequently, they are applied to predict novel pre-miRNAs based on sequence and/or structural characteristics (Fu et al., 2019).

Various software tools are available for pre-miRNA prediction, including miRDeep2 (Friedländer et al., 2012), miREval (Gao et al., 2013), PREdictor, and miRLocator (Cui et al., 2015). These tools employ diverse machine learning algorithms such as support vector machines, random forests, and artificial neural networks to forecast pre-miRNAs. Some tools also incorporate additional features such as conservation, thermodynamic stability, and sequence length to enhance the accuracy of prediction (Kleftogiannis et al., 2012).

While computational approaches for pre-miRNA prediction have demonstrated encouraging outcomes, they encounter certain obstacles, including a notable false positive rate and difficulty in accurately identifying pre-miRNAs with unconventional characteristics. Consequently, it is imperative to make additional advancements in prediction algorithms and create novel computational tools to augment the accuracy of pre-miRNA prediction and expedite the exploration of undiscovered.

2.2.2. Mature miRNA prediction

Mature miRNAs are short RNA molecules generated through the processing of pre-miRNAs, and they serve as crucial regulators of gene expression at the post-transcriptional level. The disruption of their functionality has been associated with a range of diseases, encompassing cancer and neurodegenerative disorders.

Mature miRNAs are pivotal regulators of gene expression, exerting their influence by binding to the 3' UTRs of messenger RNAs (mRNAs) and modulating translation or facilitating degradation processes, thus orchestrating gene expression. These molecules have been implicated in a wide array of biological processes, such as development, differentiation, cell proliferation, apoptosis, and immune responses. The dysregulation of mature miRNAs has been associated with numerous diseases, including cancer, cardiovascular diseases, and neurodegenerative disorders, underscoring the significance of accurate mature miRNA prediction (Ha, 2011).

2.3. Deep Learning for miRNA prediction

2.3.1. CNN for miRNA prediction

Deep learning, a potent methodology that has gained traction in various domains such as image recognition, natural language processing, and genomics, has garnered attention in recent years. Among the deep learning architectures, convolutional neural networks (CNNs) have demonstrated their efficacy in solving numerous bioinformatics challenges, encompassing gene expression prediction, protein classification, and genomic variant identification. In the realm of miRNA prediction, CNNs have exhibited promising outcomes in the identification of pre-miRNA and mature miRNA sequences, offering valuable insights (Zheng et al., 2020).

2.3.2. CNN architecture

The architecture utilized in miRNA prediction using CNN encompasses various layers such as convolutional layers, pooling layers, and fully connected layers (as shown in Figure 3). Convolutional layers employ filters to extract relevant features from the input sequence, while pooling layers decrease the output's dimensionality. Finally, fully connected layers are responsible for classifying the output by mapping it to a predefined set of classes (e.g., mature miRNA or non-miRNA sequences).

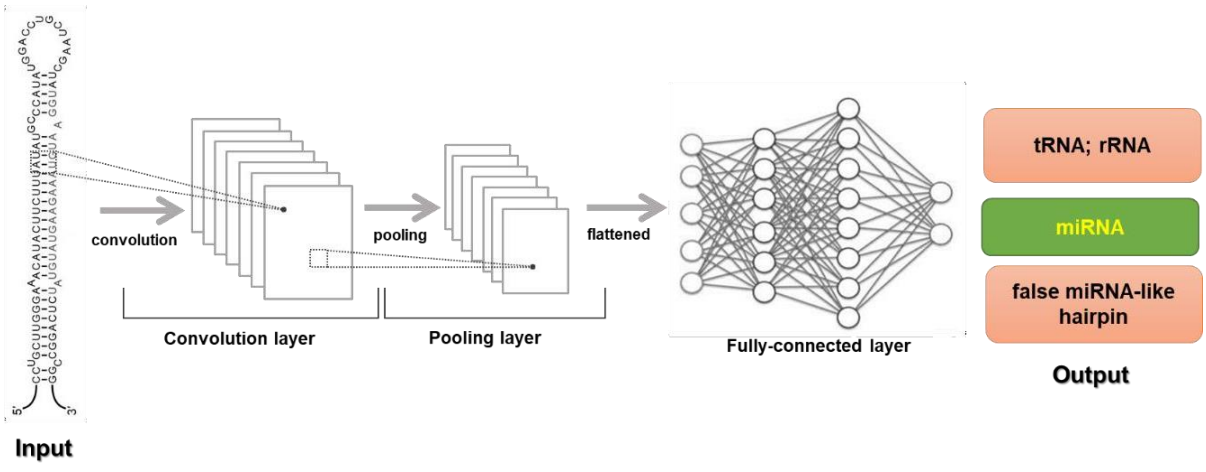


Figure 3. Illustration of the CNN architecture.

2.3.3. RNN architecture

Recurrent neural networks (RNNs) have demonstrated their effectiveness in modeling sequential data, rendering them highly suitable for miRNA prediction tasks. The utilization of RNN architectures has been extensive in capturing the intricate connections and temporal patterns inherent in miRNA sequences, thereby enhancing the precision of miRNA prediction models (Sudarshan, 2023).

2.3.3.1 Long Short-Term Memory

An extensively employed variation of RNNs in miRNA prediction is the long short-term memory (LSTM) network. LSTMs tackle the issue of vanishing gradients by integrating memory cells and gating mechanisms. These memory cells empower LSTMs to capture prolonged dependencies in sequences, thus making them highly suitable for modeling miRNA sequences that frequently encompass crucial motifs and structural characteristics (Tasdelen & Sen, 2021).

2.4. Introduction to Orthologs

Orthologs are a fundamental and widely recognized concept in the biological sciences, especially within the realms of evolutionary biology and comparative genomics. They refer to genes that are present in multiple species and can be traced back to a shared ancestral gene through the process of speciation. Although orthologous genes may have undergone genetic changes due to evolutionary pressures and environmental adaptations, they frequently retain comparable functions across species. This conservation of function signifies the vital roles these genes play in upholding essential biological processes (Gabaldón & Koonin, 2013).

The investigation of orthologs offers valuable knowledge regarding the evolutionary origins of genes and the preservation of their functions across various organisms. Through the comparison of orthologous genes in diverse species, scientists can uncover the intricacies of genetic variation, gene regulation, and molecular evolution. Additionally, orthologs serve as a basis for deducing the ancestral state of genes, facilitating the reconstruction of ancestral genomes and the identification of genetic modifications that transpired during speciation events (Steenwyk et al., 2022).

Gaining insight into orthologs is essential for comprehending the interconnections among species and the preservation of biological functions. Through the examination of shared and distinct characteristics among orthologous genes, researchers can enhance their understanding of the genetic foundations that underlie the diverse phenotypes observed across various organisms. Orthologs offer a valuable framework for investigating the genetic elements that influence the development of specific traits, adaptation to different environments, and the onset of diseases (Quintana-Murci, 2016).

Additionally, the analysis of orthology holds substantial importance in gene annotation and functional inference. With the rapid sequencing of genomes, a considerable number of genes lack experimentally determined functions. However, by leveraging orthology relationships with extensively studied genes in model organisms, researchers can transfer functional annotations and hypotheses to related genes in newly sequenced genomes. This facilitates the interpretation of the biological roles of genes, enables the prediction of their functions, and provides guidance for subsequent experimental investigations (Kristensen et al., 2011).

The recognition and examination of orthologs necessitate the integration of experimental and computational methodologies. In the realm of large-scale orthology prediction, computational approaches have become essential due to their effectiveness and scalability. These approaches employ techniques such as sequence similarity, phylogenetic analyses, calculations of evolutionary distances, and other features to infer orthology relationships among genes. By harnessing these computational methods, scientists can efficiently identify orthologs across a wide range of genomes, facilitating comprehensive studies in comparative genomics (Altenhoff & Dessimoz, 2009).

2.5. Evolutionary Conservation of miRNA Target Sites

A significant element of miRNA targeting involves the examination of conserved miRNA target sites within orthologous mRNAs. A miRNA target site pertains to a specific region in the 3' untranslated region (UTR) of an mRNA where a miRNA binds and interacts. Since orthologous genes stem from a shared ancestor, the preservation of miRNA target sites can be observed across different species. This conservation

indicates that the regulatory mechanisms facilitated by miRNAs have been evolutionarily conserved to regulate gene expression (Lee et al., 2007).

2.6. Regulation of Orthologous Genes by Common miRNAs

An intriguing element involves the control of orthologous genes through shared miRNAs. Given their sequence preservation, orthologous genes may exhibit comparable miRNA binding sites. As a result, they can be regulated by the same miRNAs, establishing a cohesive regulatory network that transcends species boundaries (Wu & Song, 2011).

2.7. Comparative Analysis of miRNA-target Interactions

The utilization of comparative genomics and computational methods is highly advantageous in the identification and examination of miRNA-target interactions within orthologous genes. Through the comparison of genomic sequences and miRNA target predictions across different species, scientists can pinpoint conserved miRNA-target interactions, draw functional conclusions, and investigate the evolutionary aspects of miRNA-mediated gene regulation (Riffo-Campos et al., 2016).

3. MATERIALS AND METHODOLOGY

3.1. Type of the Study

The entire study was performed by employing computational methods and resources.

3.2. Time and Place of the Study

This study was conducted in Izmir Biomedicine and Genome Institute from March 2021 to January 2023.

3.3. Materials and Methods of the Study

In this study, precursor miRNAs and their corresponding mature miRNAs, as well as non hairpin precursors were used for training and evaluating a deep learning model. The CNN model was used to predict pre-miRNAs. The CNN and LSTM models were used for the prediction of mature miRNAS.

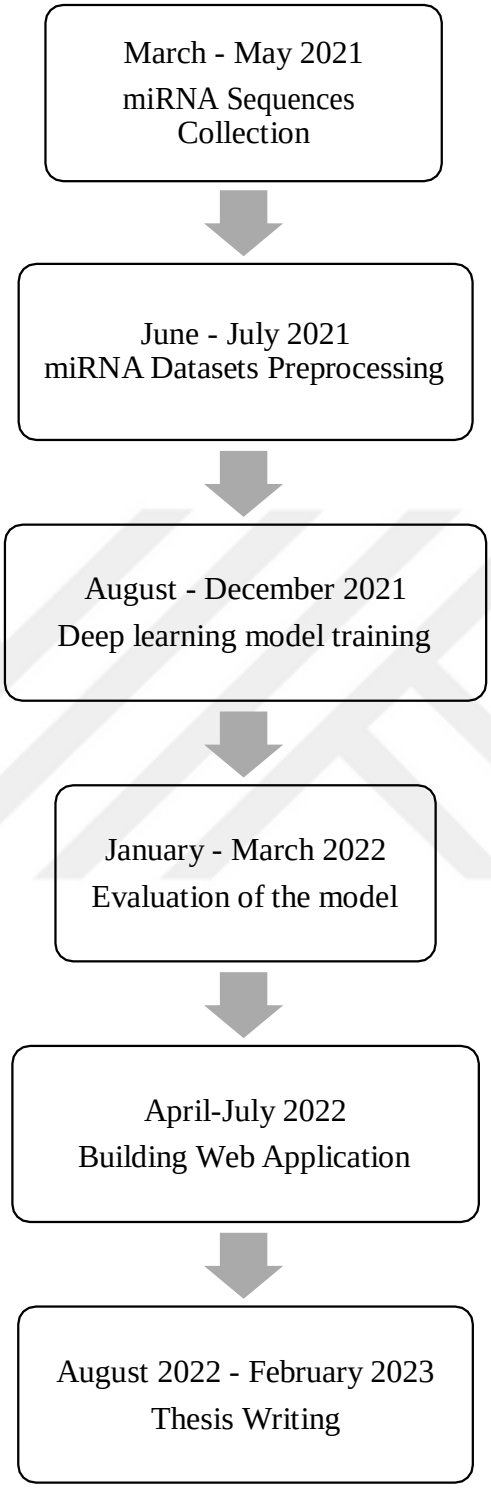
3.4. Variables of the Study

The variables of the study are the ‘positive dataset’ of precursor miRNA and mature miRNA sequences and the ‘negative dataset’ consisting of non-miRNA hairpins and false mature miRNAs.

3.5. Tools for Data Collection

The Python programming language version 3.9 (https://www.python.org/), and Linux (GNU Free Software Foundation) /GNU Bash version 3.2.48 platform were used to conduct all analyses. Gradio was used to develop a user-friendly machine learning application.

3.6. Study Plan and Calendar



3.7. Acquisition of sequences

3.7.1. Acquisition of miRNA sequences

In this study, miRNA sequences of the species *Homo sapiens*, *Macaca mulatta*, *Mus musculus*, *Rattus norvegicus*, *Equus caballus*, *Sus scrofa*, *Monodelphis domestica*, *Ornithorhynchus anatinus*, *Gallus gallus*, *Danio rerio*, *Ciona intestinalis*, *Nematostella vectensis*, *Amphimedon queenslandica*, *Caenorhabditis elegans*, and *Drosophila melanogaster* were obtained from miRBase (Kozomara et al., 2019), a comprehensive database of miRNA sequences and annotation information. To this end, two files, “Hairpin_high_conf.fa.zip” and “Mature_high_conf.fa.zip”, were downloaded from miRBase.

The acquired files were converted from FASTA format to CSV format for both hairpin and mature sequences (i.e. ‘positive dataset’). This was done using a Python script that parsed the FASTA files and extracted the relevant information, such as the miRNA ID, sequence, and organism.

Notably, only those miRNA sequences annotated as ‘high confidence’ in miRBase were used in this study. These sequences are known to be reliable and accurate, which is important for ensuring the validity of the predictions of the deep learning model developed in this study.

After downloading the miRNA sequences from miRBase, the next step was to check the files for specific species. In order to verify the correctness of the downloaded files, the miRBase species page for *Caenorhabditis elegans* was manually browsed and the number of high confidence stem loops and mature sequences was compared to those in the downloaded files. The browsing page indicated that there were 113 high confidence stem loops and 223 mature sequences for *Caenorhabditis elegans*. After checking the downloaded files, it was confirmed that the numbers were consistent with those in the browsing page.

After downloading the high confidence mature and hairpin sequences from miRBase, duplicate miRNA sequences with >90% identity were removed, using CD-HIT-EST (http://www.bioinformatics.org/cd-hit/), a web application for clustering and comparing protein or nucleotide sequences. After removing the duplicates, the number of hairpin miRNA sequences decreased from 3320 to 1821, and accordingly the

number of mature miRNA was reduced from 5564 to 3829. Next, the resulting FASTA file was converted to a TXT file and then to an XLSX file. A shuffled dataset was generated by shuffling 1423 human pre-miRNAs sequences obtained from miRBase.

To obtain the secondary structure of the pre-miRNA amd mature miRNA sequences for all species under investigation, the first step involved extracting the miRNA sequences and storing them in a TEXT file. Next, the ViennaRNA package 2.4.14 (https://gensoft.pasteur.fr/docs/ViennaRNA/2.4.14/index.html) was used with the command "RNAfold --noLP justSeq.txt > miR_seq_sec90.txt" to generate the secondary structure for each miRNA sequence. The resulting output file contained both the miRNA sequence and its corresponding secondary structure (Figure 4). To separate the miRNA sequence and secondary structure, the EXCEL formula "=IF(ISODD(ROW(A3)),A3,"")" was used to identify the sequence and remove the secondary structure from the same row. The resulting data were then further cleaned by removing the blank spaces between the sequence and structure.



Figure 4. Sample generated secondary structure for each miRNA sequence.

3.7.2. Acquisition of non-miRNA hairpin sequences

A total of 8494 human pseudo miRNA hairpins were collected from NCBI's RefSeq (https://www.ncbi.nlm.nih.gov/refseq/), 387 functional ncRNAs (negatives) from Rfam 7.0 (http://pfam.xfam.org/) (Kozomara et al., 2019), and non-redundant non-coding RNAs, including tRNA, siRNA, snRNA, and snoRNA, were randomly chosen from the exonic regions of protein-coding genes of the NCBI's Genome resources (http://www.ncbi.nlm.nih.gov/genome). The noncoding RNAs were retrieved from the fRNAdb (Kin et al., 2007), NONCODE (He et al., 2008), and snoRNA-LBME-db (Lestrade & Weber, 2006) databases. All of the retrieved sequences were <=180 nt. A hairpin-like structure in each selected sequence was predicted using ViennaRNA package (Tav et al., 2016).

3.8. CCN model building

3.8.1. Dataset preprocessing

The training of the CNN and LSTM model commenced by importing the necessary libraries and data. The 'positive dataset' (Figure 5), containing the 'filtered' pre-miRNA sequences from 3.7.1., was read from the EXCEL file using the pandas library. Specifically, the "positive.xlsx" file provides the input file for training a hybrid CNN-LSTM model, consequently enabling the exploration and prediction of pre-miRNAs across various species.

	Accession	HairpinSequence	RNAFolds
0	cel-let-7	UACACUGUGGAUCCGGUGAGGUAGUAGGUUGUAUAGUUUGGAAUAU...	....((((...(((((((((((((.(((((((((((.....(...
1	cel-lin-4	AUGCUUCCGGCCUGUUCCCUGAGACCUCAAGUGUGAGUGUACUAUU...	.(((((((((.((((..(((.((((.((((.(((((((.(((.....
2	cel-mir-1	AAAGUGACCGUACCGAGCUGCAUACUUCCUUACAUGCCCAUACUAU...	.....((((..(((..(((((((((((.((((((..(((((((((...
3	cel-mir-2	UAAACAGUAUACAGAAAGCCAUCAAAGCGGUGGUUGAUGUGUUGCA...	...(((((..((((..(((((((((((((((.((.((((((...((...
4	cel-mir-34	CGGACAAUGCUCGAGAGGCAGUGUGGUUAGCUGGUUGCAUAUUUCC...	.((((.(((.(((...(((((((.((.((((((.(((((.....
...	...	...	...
1818	mdo-mir-7375c-4	UAGCCCUUACCCCCCCAACUAGUAGUAUUAAUCUGUUAGGGAGGGG...	..(((.((.((((((((((.((.....)))))).))).)))))...
1819	mdo-mir-7373f	AAGCCCUUUUAUUAAUGAUUGAGAUGUUGCGAUAUUUUCCUCAGGC...	((((((((((.((((.((..((((.....))))..).)...
1820	mdo-mir-7262i	AAAGUCUUUGUGUAGUUUUAAGCUAUUAAGGCUUAAACUGUACUAA...	((((((((.((((((((.((((((.....)))))))))))))).))...
1821	mdo-mir-12366b	CCUGCAGCUCUGAGAUUCUAUGAUUCUGCUAUGAUGUGAUCAUAGA...	(((.((((.((((.((((((((((..((.....))))))))))....
1822	mdo-mir-7385g-2	UAGUCUGAUAUUCCAUGUUUCUAGGGUAUCAAGAGACUUGGAGCAU...	((((((((((((.((((((.((((.....))))))))))....

1823 rows × 3 columns

Figure 5. Sample dataset of pre-miRNA entries used for training.

The positive dataset underwent processing to extract and retain specific columns while adding a new column for classification (Script 1). The pandas DataFrame object, labeled as "hsa," was manipulated to retain only three relevant columns: "Accession," "HairpinSequence," and "RNAFolds." By utilizing the .loc indexer, the DataFrame was filtered to include only these columns. Additionally, a new column named "Classification" was added to the "all_species" DataFrame. This column serves as a marker to indicate the positive classification for the pre-miRNAs. The value "TRUE" was assigned to all instances in this column, signifying that the corresponding pre-miRNAs in the dataset are classified as positive examples.

Script 1

```
#process the positive data. remove other columns and keep 3 columns,
also add classification column

hsa = hsa.loc[:,["Accession","HairpinSequence","RNAFolds"]]
all_species['Classification'] = "TRUE"
```

The *merge_loci* function was defined to merge two columns of a DataFrame. It takes two input columns, "df_column1" and "df_column2" and creates a new column called "new_column". It iterates over each row of the input columns and concatenates the corresponding values from "df_column1" and "df_column2", separating them by a space character. The merged values are stored in the "new_column" list, and the function returns this list as the output (Script 2).

Script 2

```
#define a function to merge columns

def merge_loci(df_column1,df_column2):
    new_column = []
    for i in range(len(df_column1)):
        new_item = ""
        for j in range(len(df_column1[i])):
            new_item += df_column1[i][j]
            new_item += df_column2[i][j]
            new_item += " "
        new_column.append(new_item)
    return new_column
```

Next, the "HairpinSequence" and "RNAFolds" columns in the "all_species" DataFrame were merged into a new column named "species_merge_column" using

the *merge_loci()* function. Subsequently, the merged column was transformed into a NumPy array using *np.array()*, and the resulting content was printed. To enable separate access to each miRNA sequence and its corresponding structure, the merged column was converted into a pandas Series using *pd.Series()*. Finally, the merged data were assigned to a new column, "seq_struc" in the "all_species" DataFrame, effectively associating the merged sequence and structure information with the respective miRNA entry (Script 3).

Script 3

```
# merge HairpinSequence and RNAFolds in hsa
species_merge_column =
merge_loci(all_species["HairpinSequence"],all_species["RNAFolds"])

species_merge_column = np.array(species_merge_column)
print(sepecies_merge_column)

#convert to series to have seperately for each miR
species_merge_column = pd.Series(species_merge_column)

#put the merge data under the new column called se1_struc
all_species["seq_struc"] = species_merge_column
```

The next stage involved processing the negative data, which included non-coding RNAs and mature miRNAs. The negative data was read from the "negative.xlsx" file. The relevant columns "Accession," "HairpinSequence," and "RNAFolds" were selected from the negative dataset using the .loc indexer. This step aimed to extract essential information for further analysis.

In order to match the number of samples in the positive dataset, the number of rows in the positive dataset was determined using the variable *all_species_new_row*. This was achieved by calculating the length of the "all_species_new" DataFrame.

To introduce randomness in the selection of samples, the negative dataset (Figure 6) was shuffled using the *shuffle* function (Script 4) from the appropriate library. This step ensured a random distribution of samples within the negative dataset, promoting unbiased analysis. The resulting negative dataset, after shuffling, was printed using print (negative). This provided an overview of the shuffled negative samples, which encompassed non-coding RNAs and mature miRNAs.

These stages and codes reflect the passive form and describe the processing steps for the negative dataset, including data reading, column selection, matching sample numbers, and introducing randomness for subsequent analysis.

Script 4

```
#process the negative data
negative =
pd.read_excel("/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/negative.xlsx")

negative = negative.loc[:,["Accession","HairpinSequence","RNAFolds"]]

# get the same number as postive dataset
all_species_new_row = len(all_species_new)
print(all_species_new_row)

# to have random selection
negative = shuffle(negative,random_state = 0)
print(negative)
```

	Accession	HairpinSequence	RNAFolds
11565	random_seq_from_cds__NO_61424	CAGGCGGCGCGCGGGCACCGCGGCUCCUCCGAGACCUCGGGUGCCG...	..(((.((((((((((.((((((((... ((((....)))))..))))....
12272	random_seq_from_cds__NO_42540	CUUUCUGCUCCUUGCCACCGUCUUCUCCUGCAUCUGGAAGAGCCAC...	.....((((.((.(((((((((((((...(((((.....
19580	dme-miR-252-3p	UCCUGCUGCCCAAGUGCUUAUU	....(((.....))).....
5567	ENST00000384600.1 ncrna chromosome:GRCh38:13:2...	GUGCUAGCUUUGGCAGCACAUAUACUAAAACUGAAACAAUACAGAG...	(((((.((....)))))).....(((.....)))....
807	URS0000100F50_9606 Homo sapiens (human) non-pr...	UAAAUUUAAUAACAUCAUGCAUGGAAAGUGCCCAGCACAAUUCCUG...	.....((((.((((..((((.((((...)))).))))..))))....
...	...	...	...
5415	ENST00000384242.1 ncrna chromosome:GRCh38:14:1...	GUGCUGGCUUUGGCAGCACAUAUACUAAAAUUGGAAUGAUACAGAG...	((((((.....))))))..((((((((((....((..(...
20017	gga-miR-1329-3p	CCUCGUAGCUUGAUCACGAUAU	..((((.....))))....
13528	random_seq_from_cds__NO_63257	AGGAGGUGGCGAGAAGGCCAGCAGUGCCCCGUUGUGGUCCACGCGG...	.((((((...((((..(((..(..(((((((((. (....)).)))).))....
17228	mmu-miR-301a-5p	GCUCUGACUUUAUUGCACUACU	((.....))....
10764	random_seq_from_cds__NO_5281	AUGCAGUAGGAGGCCAUGAUGGAUCCUCUUAUCUCAAUAGUGUUGA...	.(((.(((((.(((((((.((.(...((((...((.....(((.....

1823 rows × 3 columns

Figure 6. Sample dataset of negative data used for training.

To ensure a proper index alignment, the "negative" DataFrame was reset using the *reset_index()* function with the parameter drop=True. This resulted in a new index for the DataFrame without preserving the previous index values.

To differentiate the negative samples from the positive ones, a new column named "Classification" was added to the "negative" DataFrame. All entries in this column were assigned the value "FALSE," indicating their classification as negative examples.

Similar to the previous step, the "HairpinSequence" and "RNAFolds" columns in the "negative" DataFrame were merged into a new column named "negative_merge_column" using the *merge_loci() function*.

The merged column was then converted into a NumPy array using *np.array()*, and subsequently transformed into a pandas Series using *pd.Series().* This conversion allowed for separate access to each merged sequence and structure within the negative dataset.

Finally, the merged data was assigned to a new column named "seq_struc" in the "negative" DataFrame, linking the merged sequence and structure information to the respective negative sample (Script 5).

Script 5

```
negative = negative.reset_index(drop=True)
negative['Classification'] = "FALSE"

# merge HairpinSequence and RNAFolds in Negative
negative_merge_column =
merge_loci(negative["HairpinSequence"],negative["RNAFolds"])
negative_merge_column = np.array(negative_merge_column)

negative_merge_column = pd.Series(negative_merge_column)
negative['seq_struc'] = negative_merge_column
```

The positive and negative datasets were merged into a single dataset, referred to as the "mixed_dataset," using the *pd.concat() function*. This operation combined the "all_species_new" DataFrame, representing the positive data, with the "negative_new" DataFrame, containing the negative data.

By passing both DataFrames as arguments to *pd.concat()*, the positive and negative samples were vertically stacked, resulting in a combined dataset that encompasses both types of samples. This facilitated the integration of the positive and negative data for subsequent analysis.

After merging, the rows in the "mixed_dataset" were shuffled to ensure a random distribution of samples. The *shuffle()* function was applied to the "mixed_dataset" DataFrame, with the parameter random_state=2 specified to ensure reproducibility of the shuffling process (Script 6).

Script 6

```
# merge the positive and negative data into a dataset
mixed_dataset = pd.concat([all_species_new,negative_new])

mixed_dataset = shuffle(mixed_dataset,random_state = 2)
mixed_dataset
```

The above steps enabled the creation of a comprehensive dataset containing both positive and negative samples (Figure 7).

	Accession	seq_struc	Classification
1205	bmo-mir-2774a-4	A. C. A. A(C(A. A(C(A(U. U. C. G(G(A(C...	TRUE
1089	mml-mir-934	A(G(G(A(A(U(A(A(G(G(C(U(U(C(U(G...	TRUE
822	random_seq_from_cds__NO_25799	G(G(G(A. G. A(A(A(C(C(U(C(A. U. G. G...	FALSE
858	ENST00000365249.1 ncrna chromosome:GRCh38:6:15...	G(U(G(C(G. U(G(C(C. U. C. G. G) C) A) G...	FALSE
338	rno-mir-291a	C(C(G. G(U(G(U(A(G(U(A. G(C(C(A(U...	TRUE
...	...	...	...
1783	random_seq_from_cds__NO_54588	U(A(G(U(A(U(G. G. A. C(A(U. C(C. G. G...	FALSE
1608	mdo-mir-7246	U(A(U(C(C(A(A(C(C(U(U(A(U(U(U(C...	TRUE
718	URS0000D572B1_9606 Homo sapiens (human) non-pr...	A. U. G(G(A(U(G(G(G. C(U(C. C. G. A. G...	FALSE
752	random_seq_from_cds__NO_10788	U. G. G. A. G. G. G(G(U(G(G. U(C(G(C. A...	FALSE
1417	random_seq_from_cds__NO_15842	U. C. U. C(U(G(G(A(U. U. U. A. G. U. G. U...	FALSE

3646 rows × 3 columns

Figure 7. Sample dataset including positive and negative samples.

3.8.2. Vectorization

A dictionary named "x_cast" was defined to map each nucleotide and structural symbol to its corresponding one-hot encoded vector representation. Each symbol is associated with a specific binary vector.

Next, a function named "vectorize_seq_struc" is implemented to transform the values in the "df_column" into vectors. The function iterates over each line in the "df_column" and processes it by stripping any leading or trailing whitespace and splitting it based on spaces. The resulting line is then converted into a list.

For each item in the line, the corresponding one-hot encoded vector from the "x_cast" dictionary is appended to a temporary list. If the line is shorter than a

predefined sequence length (SEQ_LEN), padding is performed by appending a vector of zeros. This ensures that all sequences have the same length.

The temporary list is then appended to the "x_dataset" list, which holds the vectorized representations of the sequences. Once all lines have been processed, the function returns the "x_dataset" as a numpy array (Script 7).

Script 7

```
x_cast = {"A.":[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],"U.":[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],\
          "G.":[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],"C.":[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],\
          "A(":[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],"U(":[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],\
          "G(":[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],"C(":[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],\
          "A)":[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],"U)":[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],\
          "G)":[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],"C)":[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
}
```

The function *vectorize_seq_struc* takes the "seq_struc" column from the "mixed_dataset" DataFrame as input (Script 8). It iterates over each line in the column, processing and transforming the values into vectors. For each line, the corresponding one-hot encoded vectors are retrieved from the "x_cast" dictionary and appended to a temporary list. Padding is performed if necessary to ensure a consistent sequence length. Finally, the function returns the "x_dataset" as a numpy array. A dictionary named "y_cast" is defined to map the classification values ('FALSE' and 'TRUE') to their respective binary vectors.

The function *vectorize_TF* takes the "Classification" column from the "mixed_dataset" DataFrame as input. It iterates over each line in the column and retrieves the corresponding binary vector representation from the "y_cast" dictionary. The resulting vectors are appended to the "y_dataset". The "TF_data_vec" variable is then converted into a numpy array using *np.array()*. Similarly, the "mixed_data_vec" variable, representing the vectorized sequences, is converted into a numpy array (Script 8).

Script 8

```
#Transform the seq_struct data and classification values into vectors
#using one-hot encoding. All the sequences are padded into 180 lengths.
#The returned x_dataset,y_dataset are numpy arrays.
def vectorize_seq_struc(df_column):
    SEQ_LEN = 180
    x_dataset = []
    for line in df_column:
        line = line.strip()
        line = line.split(" ")
        temp_list = []
        for item in line:
            temp_list.append(x_cast[item])
        for i in range(SEQ_LEN-len(line)):
            temp_list.append([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])
        x_dataset.append(temp_list)
    return x_dataset
```

3.8.3. Training the CNN model

The *Keras* library, along with the required modules and classes such as regularizers, Sequential, Dense, Activation, Dropout, Flatten, Conv1D, and MaxPooling1D, was imported to facilitate the construction and configuration of the deep learning model (Script 9). Additionally, the *train_test_split* function from the sklearn library's model_selection module was imported to perform the splitting of the data into training and testing sets. This import is part of the preparatory steps for subsequent model training and evaluation.

The data was divided into "X_train" and "X_test" for input features, as well as "y_train" and "y_test" for target labels. This division of the data was accomplished by utilizing a test size of 0.25, indicating that 25% of the data was allocated for testing. The random state value of 101 was set to ensure the reproducibility of the results (Script 9).

By performing this data splitting, the subsequent steps of model development and training were executed. The testing set, which consisted of a separate portion of the data, was used to evaluate the performance of the trained model on unseen data.

Script 9

```
import keras
from keras  import  regularizers
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout,Flatten
from keras.layers import Conv1D,MaxPooling1D
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(mixed_data_vec,
TF_data_vec , test_size= 0.25, random_state=101)
```

The model was initialized as a sequential model using *the Sequential()* function (Script 10).

The first layer of the model consisted of a one-dimensional convolutional layer (Conv1D) with 16 filters, a kernel size of 4, an activation function set to 'relu', padding set to 'same', and an input shape of (180,12,1). This layer was followed by a max-pooling layer (MaxPooling1D) with a pool size of 2.

The second layer of the model included another one-dimensional convolutional layer with 32 filters, a kernel size of 4, 'relu' activation, and 'same' padding. This was followed by another max-pooling layer.

Similarly, the third and fourth layers of the model were defined as one-dimensional convolutional layers with 64 and 128 filters, respectively. The kernel sizes were 4 and 4, and 'relu' activation and 'same' padding were applied to both layers. Each convolutional layer was followed by a max-pooling layer.

To prepare the data for the dense layers, the model was flattened using *Flatten()*. A dropout layer (Dropout) with a dropout rate of 0.5 was added to reduce overfitting.

The next step involved adding a dense layer (Dense) with 256 units, 'relu' activation, and a kernel regularizer (kernel_regularizer) with an L2 regularization parameter of 0.001. Another dropout layer with a dropout rate of 0.5 was added before the final dense layer, which had 2 units and 'softmax' activation.

The model was then compiled using *model.compile()*, specifying the loss function as 'categorical_crossentropy', the optimizer as 'adam', and the metric to track accuracy as 'accuracy'.

To examine the model architecture and parameters, the *model.summary()* function was called, and the summary was printed (Script 10).

Script 10

```
model = Sequential ()
input_shape=(180, 12, 1)
#first layer
model.add(Conv1D(16,4,activation = 'relu',padding = 'same',input_shape =
input_shape ))
model.add(MaxPooling1D(pool_size = 2))

#second layer
model.add(Conv1D(32,4,activation = 'relu',padding = 'same'))
model.add(MaxPooling1D(pool_size = 2))
#third layer
model.add(Conv1D(64,4,activation = 'relu',padding = 'same'))
model.add(MaxPooling1D(pool_size = 2))
#fourth layer of convolution and max-pooling
model.add(Conv1D(128,4,activation = 'relu',padding = 'same'))
model.add(MaxPooling1D(pool_size = 2))
_model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(256,activation = 'relu',kernel_regularizer =
regularizers.l2(0.001)))
model.add(Dropout(0.5))

model.add(Dense(2,activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics =
['accuracy'])
print(model.summary())
```

The EarlyStopping callback was imported from tensorflow.keras.callbacks to monitor the validation loss during model training (Script 11). If the loss did not improve for a defined number of epochs, the training process was halted.

The model was trained using the *model.fit()* function, where training was conducted for 400 epochs with a batch size of 32. The validation data, comprising the testing set (X_test and y_test), was incorporated using the validation_data parameter. The early_stop callback was included as a callback function to oversee the validation loss and execute early stopping when necessary.

Following the model training, its performance was evaluated. The model.metrics_names attribute was utilized to retrieve the names of the metrics employed for evaluation, which were 'loss' and 'accuracy'.

A dataframe named "losses" was created to store the loss values recorded during training. The *model.evaluate() function* was executed to assess the model's loss and accuracy on the testing set (X_test and y_test).

Script 11

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',patience=30)

model.fit(X_train, y_train, epochs = 400,batch_size =
32,validation_data=(X_test,y_test),callbacks=[early_stop])

#Evaluate the model
model.metrics_names

losses = pd.DataFrame(model.history.history)
losses[['val_loss','loss']].plot()

print(model.metrics_names)
print(model.evaluate(X_test,y_test,verbose=0))
```

3.9. Identification of mature miRNAs

3.9.1. Dataset preparation

The positive mature dataset includes high confidence mature miRNAs from miRBase. Negative examples were generated from the respective precursor sequences based on the observation that known miRNA precursors do not produce multiple overlapping mature miRNAs. For each verified mature miRNA, a same-size sliding window was used and all possible sequences were selected which could be created by sliding 1 base pair towards either direction from the verified mature miRNA over the precursor sequence, excluding any hairpin loops (Script 12).

Script 12

```
import pandas as pd

def generate_negative_examples(verified_miRNAs, precursor_sequences,
window_size):
    negative_examples = []

    for miRNA in verified_miRNAs:
        precursor_sequence = precursor_sequences[miRNA] # Get precursor
sequence for the miRNA

        for i in range(len(precursor_sequence) - window_size + 1):
            window_sequence = precursor_sequence[i:i+window_size]  # Extract
window sequence

            # Exclude hairpin loops by checking for any parentheses within
the window sequence
            if '(' not in window_sequence and ')' not in window_sequence:
                negative_examples.append(window_sequence)

    return negative_examples

# Load precursor sequences from Excel
precursor_df =
pd.read_excel('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/mature/ha
irEx.xlsx')

# Convert precursor sequences to a dictionary
precursor_sequences = dict(zip(precursor_df['miR'],
precursor_df['Sequence']))

# Extract miRNA names from the Excel file
verified_miRNAs = precursor_df['miR'].tolist()

window_size = 22  # Sliding window size

negative_examples = generate_negative_examples(verified_miRNAs,
precursor_sequences, window_size)

# Save negative examples to a new Excel file
negative_df = pd.DataFrame({'Negative Sequence': negative_examples})
negative_df.to_excel('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/ma
ture/negative_examples.xlsx', index=False)
```

3.9.2. CNN and LSTM model building for mature miRNA prediction

Script 13 was used for building the CNN and LSTM models.

Script 13

```
# One-Hot Encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
_max_sequence_length = 26

#Build the CNN-LSTM model
model = keras.models.Sequential()

model.add(keras.layers.Conv1D(filters=32, kernel_size=4, activation='relu',
input_shape=(max_sequence_length , 1)))
model.add(keras.layers.MaxPooling1D(pool_size=2))

model.add(keras.layers.Conv1D(filters=64, kernel_size=4, activation='relu'))
model.add(keras.layers.MaxPooling1D(pool_size=2))


# LSTM layer
model.add(keras.layers.LSTM(128, return_sequences=True))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(256, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001)))

model.add(keras.layers.Dense(2, activation='softmax'))

early_stop = EarlyStopping(monitor='val_loss',patience=30)

model.fit(X_train, y_train, epochs = 100,batch_size = 32,
validation_data=(X_test,y_test),callbacks=[early_stop])
```

3.10.1. Receiver operating characteristic curve for evaluating the precursor and mature miRNA models

The receiver operating characteristic (ROC) curve (Figure 8) is a graphical representation of the performance of a given binary classifier. It plots the true positive rate (TPR) on the y-axis against the false positive rate (FPR) on the x-axis. TPR, also known as sensitivity or recall, measures the proportion of actual positive instances correctly classified as positive. FPR represents the proportion of actual negative instances incorrectly classified as positive.

A perfect classifier would have a ROC curve that reaches the top-left corner of the plot, indicating a TPR of 1 and an FPR of 0. This means that the classifier correctly identifies all positive instances while making no false positive errors. A perfect classifier's ROC curve would enclose an area under the curve (AUC) of 1.

In contrast, a random classifier would have an ROC curve that closely follows the diagonal line from the bottom-left to the top-right of the plot. This indicates that the classifier's TPR and FPR are roughly equal, suggesting random guessing. A random classifier's ROC curve would enclose an AUC close to 0.5.

The terms "better" and "worse" in the context of the ROC curve refer to the relative performance of different classifiers or different models of the same classifier. A classifier with a higher AUC and a ROC curve that is closer to the top-left corner is considered better because it has a higher TPR for the same or lower FPR. On the other hand, a classifier with a lower AUC and a ROC curve that deviates further from the diagonal line is considered worse because it has a lower TPR for the same or higher FPR.

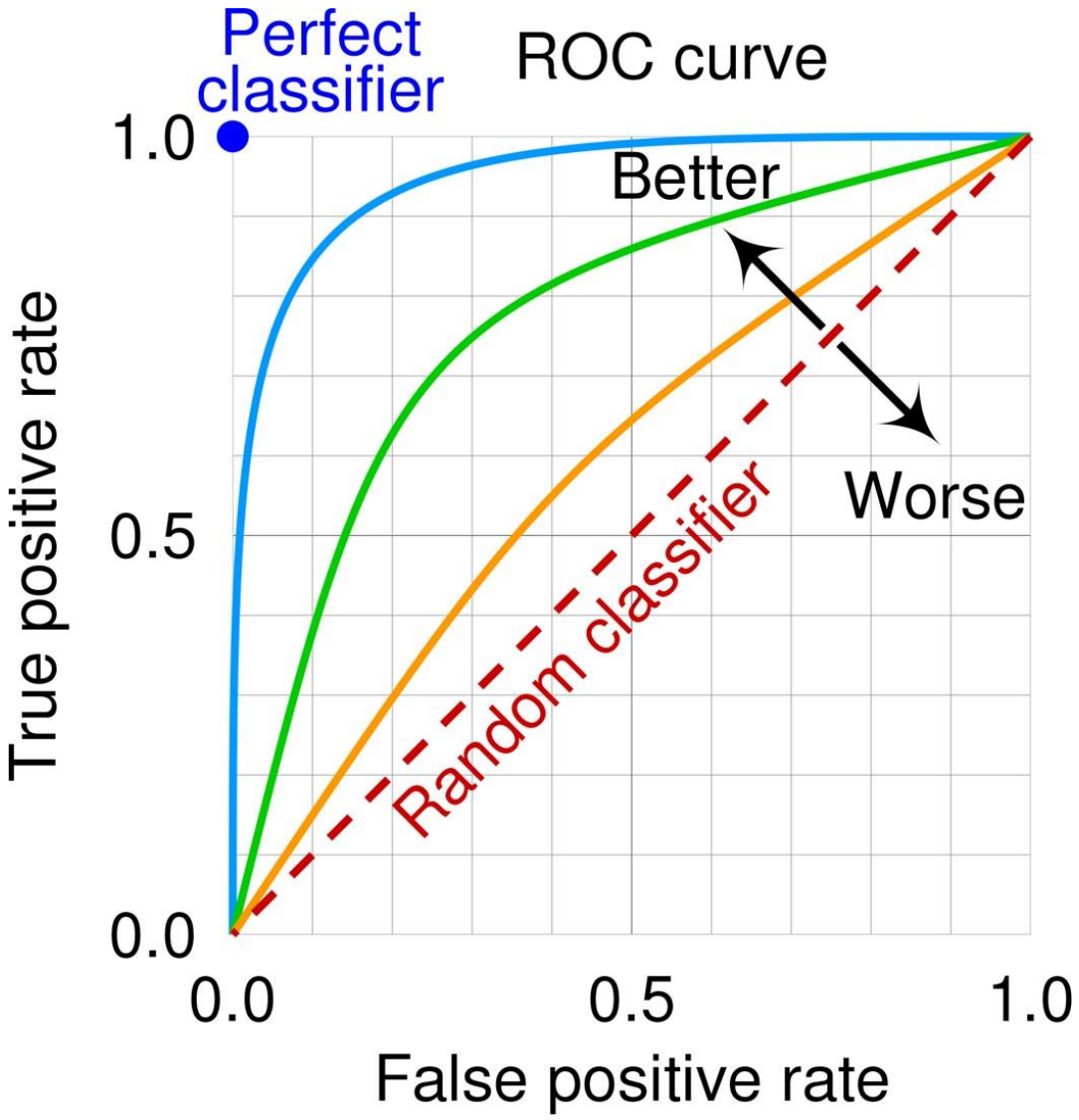


Figure 8. Receiver operating characteristic (ROC) curve.

.

Script 14 was used to create the ROC curve.

Script 14.

```
import tensorflow as tf
#print(tf._version_)

from numpy import loadtxt
from keras.models import load_model
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.preprocessing import label_binarize
# Load the saved modelmodel =
load_model("/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/mir_deep_model_pre
_final.h5")

# Load the data
data = np.load('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/train_test_data.npz')
X_train = data['X_train']
X_test = data['X_test']
y_train = data['y_train']
y_test = data['y_test']
# Convert multilabel indicators to binary labels
y_test_bin = label_binarize(y_test, classes=[0, 1])  # Replace [0, 1] with your classes

# Predict probabilities for test set
y_pred_prob = model.predict(X_test)

# Calculate the AUC-ROC score
auc_roc = roc_auc_score(y_test_bin, y_pred_prob)
# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test_bin.ravel(), y_pred_prob.ravel())

# Print the AUC-ROC score
print("AUC-ROC:", auc_roc)
# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve')
plt.text(0.6, 0.3, 'AUC =' % auc_roc, ha='center', fontsize=12, fontweight='bold')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

3.10.2. Cross-validation to evaluate mature and pre-miRNA models

Cross-validation (10-fold) was used to estimate the generalization performance of the developed models (Script 15).

Script 15.1

```
from keras.preprocessing.text import Tokenizer
import pandas as pd
import numpy as np
import itertools
from keras.utils.data_utils import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt

# Step 1: Load the data
positive_data =
pd.read_excel('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/mature/pre.xls
x')
negative_data =
pd.read_excel('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/mature/negativ
e.xlsx')

# Step 2: Preprocessing
positive_data['label'] = 1
negative_data['label'] = 0
data = pd.concat([positive_data, negative_data], ignore_index=True)

# step 3: preparation the dataset (similar to previous sections)

# Step 4: Prepare the data for cross-validation

# Define 10-fold cross validation
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cv_scores = []
history_list = []

for train, test in kfold.split(X, data['label']):
    X_train, X_test = X[train], X[test]
    y_train, y_test = y[train], y[test]

    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

    # One-Hot Encoding
    y_train = tf.keras.utils.to_categorical(y_train)
    y_test = tf.keras.utils.to_categorical(y_test)
```

```
#Step 5: Build the CNN-LSTM model
    model = keras.models.Sequential()
    model.add(keras.layers.Conv1D(filters=32, kernel_size=4, activation='relu',
input_shape=(max_sequence_length, 1)))
    model.add(keras.layers.MaxPooling1D(pool_size=2))
    model.add(keras.layers.Conv1D(filters=64, kernel_size=4, activation='relu'))
    model.add(keras.layers.MaxPooling1D(pool_size=2))
    model.add(keras.layers.LSTM(128, return_sequences=True)) # LSTM layer
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dropout(0.5))
    model.add(keras.layers.Dense(256, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.Dense(2, activation='softmax'))  # 2 for binary
classification

    # Compiling the CNN-LSTM model
    model.compile(optimizer=keras.optimizers.Adam(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy', tf.keras.metrics.Precision(),
tf.keras.metrics.Recall()])

    # Early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(patience=30, monitor='val_loss')

    # Training the CNN-LSTM on the Training set and evaluating it on the Test set
    history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test), callbacks=[early_stopping])

    # Evaluate the model
    score = model.evaluate(X_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
    print(model.metrics_names)
    print(model.evaluate(X_test,y_test,verbose=0))

    # Plotting the training and validation losses
    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Test')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    cv_scores.append(score[1])  # Assuming that score[1] is accuracy
    history_list.append(history)

print("Average accuracy: %.2f%% (+/- %.2f%%)" % (np.mean(cv_scores)*100,
np.std(cv_scores)*100))
```

3.10.3. Accuracy, precision and recall evaluation metrics

The evaluation metrics accuracy, precision and recall were used to assess the performance of the proposed model (Script 16). Accuracy measures the overall correctness of the model's predictions by calculating the proportion of correctly classified samples over the total number of samples. Recall, also known as sensitivity or true positive rate, estimates the proportion of true positive predictions among the actual positives. Precision estimates the proportion of true positive predictions among the total predicted positives.

Script 16.1

```
import tensorflow as tf
from keras.models import load_model
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import accuracy_score

# Load the saved model
model =
load_model("/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/mir_deep
_model_pre.h5")

# Load the data
data =
np.load('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/train_test_
data.npz')
X_train = data['X_train']
X_test = data['X_test']
y_train = data['y_train']
y_test = data['y_test']

# Convert true/false labels to multilabel format
y_train_binary = label_binarize(y_train, classes=['False', 'True'])
y_test_binary = label_binarize(y_test, classes=['False', 'True'])

# Predict probabilities for training and test sets
y_train_pred_prob = model.predict(X_train)
y_test_pred_prob = model.predict(X_test)

# Set a threshold to convert probabilities to binary class predictions
threshold = 0.5
y_train_pred = (y_train_pred_prob > threshold).astype(int)
y_test_pred = (y_test_pred_prob > threshold).astype(int)
```

```
# Calculate sensitivity and specificity
from sklearn.metrics import confusion_matrix, recall_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Convert predictions and ground truth labels from one-hot encoding to
original form
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Calculate sensitivity/recall
sensitivity = recall_score(y_test_classes, y_pred_classes)
print('Sensitivity:', sensitivity)

# Calculate specificity
confusion = confusion_matrix(y_test_classes, y_pred_classes)
TN = confusion[0, 0]
FP = confusion[0, 1]
specificity = TN / (TN + FP)
print('Specificity:', specificity)

# Calculate precision, recall, and F1-score for training data
train_precision = precision_score(y_train_binary, y_train_pred,
average='weighted')
train_recall = recall_score(y_train_binary, y_train_pred,
average='weighted')
train_f1 = f1_score(y_train_binary, y_train_pred, average='weighted')
train_f1 = f1_score(y_train_binary, y_train_pred, average='weighted')
train_accuracy = accuracy_score(y_train_binary, y_train_pred)

# Calculate precision, recall, and F1-score for test data
test_precision = precision_score(y_test_binary, y_test_pred,
average='weighted')
test_recall = recall_score(y_test_binary, y_test_pred,
average='weighted')
test_f1 = f1_score(y_test_binary, y_test_pred, average='weighted')
test_accuracy = accuracy_score(y_test_binary, y_test_pred)
```

Script 16.2

```
import tensorflow as tf
from keras.models import load_model
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import accuracy_score

# Load the saved model
model =
load_model("/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/mir_deep_
model_pre.h5")

# Load the data
data =
np.load('/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/model/train_test_d
ata.npz')
X_train = data['X_train']
X_test = data['X_test']
y_train = data['y_train']
y_test = data['y_test']

# Convert true/false labels to multilabel format
y_train_binary = label_binarize(y_train, classes=['False', 'True'])
y_test_binary = label_binarize(y_test, classes=['False', 'True'])

# Predict probabilities for training and test sets
y_train_pred_prob = model.predict(X_train)
y_test_pred_prob = model.predict(X_test)

# Set a threshold to convert probabilities to binary class predictions
threshold = 0.5
y_train_pred = (y_train_pred_prob > threshold).astype(int)
y_test_pred = (y_test_pred_prob > threshold).astype(int)

# Calculate sensitivity and specificity
from sklearn.metrics import confusion_matrix, recall_score

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Convert predictions and ground truth labels from one-hot encoding to
original form
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Calculate precision, recall, and F1-score for training data
train_precision = precision_score(y_train_binary, y_train_pred,
average='weighted')
train_recall = recall_score(y_train_binary, y_train_pred,
average='weighted')
train_accuracy = accuracy_score(y_train_binary, y_train_pred)



# Calculate precision, recall, and F1-score for test data
test_precision = precision_score(y_test_binary, y_test_pred,
average='weighted')
test_recall = recall_score(y_test_binary, y_test_pred, average='weighted')
test_accuracy = accuracy_score(y_test_binary, y_test_pred)


# Print the performance metrics for training and test data
print("Training Data:")
print("Precision:", train_precision)
print("Recall:", train_recall)
print("Test Data:")
print("Precision:", test_precision)
print("Recall:", test_recall)
```

3.10.4. Creation of web-based interfaces for the models

Gradio is a Python library that is open-source and provides a convenient and efficient method to present machine learning and deep learning models using a web-based interface. This tool simplifies the usage of these models, enabling people from any location to easily access and utilize them.The web-based application developed in this thesis has been deployed locally for demonstration purposes. By leveraging Gradio (Script 17), this research project aims to create interactive and intuitive demos and web applications that showcase the capabilities of deep learning and data science techniques.

Script 17.1

```
model = load_model('/mir_deep_model_pre.h5')

taking data from user and processing on it
def seq_process(seq):
    # remove line break
    seq = seq.strip('\n')
    seq = seq.replace('\n','')
    # remove sapce
    seq =seq.replace(' ', '')
    # onvert the string to all uppercase
    seq = seq.upper()
    # print(seq)
    # check correctness of the RNA sequence
    seq = seq.replace('T','U')
    return seq

x_cast =
{"A.":[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],"U.":[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
,\
          "G.":[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],"C.":[0,0,0,1,0,0,0,0,0,0,0,0
,0,0,0,0],\
          "A(":[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],"U(":[0,0,0,0,0,1,0,0,0,0,0,0
,0,0,0,0],\
          "G(":[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],"C(":[0,0,0,0,0,0,0,1,0,0,0,0
,0,0,0,0],\
          "A)":[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],"U)":[0,0,0,0,0,0,0,0,0,1,0,0
,0,0,0,0],\
          "G)":[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],"C)":[0,0,0,0,0,0,0,0,0,0,0,1
,0,0,0,0],}
```

Script 17.2

```
def vectorize_seq_struc(df_column):
    SEQ_LEN = 4743
    x_dataset = []
    for line in df_column:
        line = line.strip()
        line = line.split(" ")
        temp_list = []
        for item in line:
            temp_list.append(x_cast[item])
        for i in range(SEQ_LEN-len(line)):
            temp_list.append([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])
        x_dataset.append(temp_list)
    return x_dataset

def merge_input(sample,sec):
    new_list = [i+j for i, j in zip(sample, sec)]
    x = ' '.join(new_list)
    return x

from numpy import argmax
def predict_result(sample):
    predictSample = model.predict(sample)
    if argmax(predictSample) == 0:
        return 'It is a pre-miRNA'
    else:
        return 'It is NOT a pre-miRNA'



def get_input_give_result(Sequence, Secondary_Structure):
    Sequence = seq_process(Sequence)
    merged_seq_RNAfold = merge_input(Sequence,Secondary_Structure)
    merged_seq_RNAfold_series = pd.Series(merged_seq_RNAfold)
    merged_seq_RNAfold_vector =
vectorize_seq_struc(merged_seq_RNAfold_series)
    predicted_one = predict_result(merged_seq_RNAfold_vector)
    mypredict=f"{predicted_one}"
    return mypredict
```

Script 17.3

```
css = """
        .container {
            max-width: 750px;
            margin: auto;
            padding-top: 1.5rem;
            text-align: center;
        }

        .gr-button {
            color: white;
            border-color: #04a96d;
            background: #04a96d;
            padding: 10px 20px;
            text-align: center;
            display: inline-block;
            font-size: 16px;
            margin: 6px 4px;
            transition-duration: 0.4s;
            cursor: pointer;
            position: relative;

        }

        .gr-button:hover {
          background-color: #4CAF50;
          color: white;
          }

        .footer {
            margin-bottom: 45px;
            margin-top: 35px;
            text-align: center;
            border-bottom: 1px solid #e5e5e5;
        }
        .footer>p {
            font-size: .8rem;
            display: inline-block;
            padding: 0 10px;
            transform: translateY(10px);
            background: white;
        }
        .dark .footer {
            border-color: #303030;
        }
        .dark .footer>p {
            background: #0b0f19;
        }

"""
```

```
block = gr.Blocks(css=css)
examples=[
        ['''GGCCAGCUGUGAGUGUUUCUUUGGCAGUGUCUUA
        GCUGGUUGUUGUGAGCAAUAGUAAGGAAGCAAUCAGCAA
        GUAUACUGCCCUAGAAGUGCUGCACGUUGUGGGGCCC''','((((.(((((((((((((..(((((((
((((.((((((((((....((....)).....))))))))))))).)))))))..)))))))).))))..)).)))
).'],
        ['''GCAGGCAGGGGUGAGGAGCGCAGCGGGCGGCGAGCG
               GGAGGCACUGGCCUCCAGAGCCCGUGGCCAAGGCGGGCCUC
               GCGGGCGGCGACGGAGCCGGGAUCGGUGCCUCAGCGUUCGG
               GCUGGAGACGAGGCCAGGUCUCCAGCUGGGGUGGACGUGCC
          CACCAGCUGCCGAAGGCCAAGACGCCAGGUCCGGUG''','.........((((((..(((... (((
.(((.((.((((((....))))))...)).))).)))...)))..))))))...((((.....))))(((((((((.(
(..((.((((((((((((((.......)))))))))...(((((.....)))))).....)))))).)).)).)))).
)))))....']]
with block:
    gr.HTML(
        """
          <div class='container';style="max-width: 750px;margin: 0 auto;">
            <head>
              <meta name="viewport" content="width=device-width, initial-
scale=1.0">
            </head>
              <div
                style="
                  display: inline-flex;
                  align-items: center;
                  gap: 0.8rem;
                  font-size: 1.75rem;
                  "
              >
                <h1 style=" font-weight: 900; margin-bottom: 7px;">
                  Welcome to PMP: Pre-miRNA Predictor
                </h1>
              </div>
              <p style="text-align: left ;margin-bottom: 10px; font-size: 94%">
              PMP is a state of the art tool which predicts whether or not your
sequence is pre-miRNA.<br>Paste your plain sequence and its secondary structure
in the boxes below and click the predict button.
    <br>To get the secondary structure you can use
                <a
                  href="http://rna.tbi.univie.ac.at/cgi-
bin/RNAWebSuite/RNAfold.cgi"
                  style="text-decoration: underline;"
                  target="_blank"
                  >RNAfold WebServer.</a
                >
              </p>
            </div>
        """
    )
```

Script 17.4

```
with gr.Group():
        with gr.Box():
                text1 = gr.Textbox(
                    label="Sequence",
                    max_lines=4,

                ).style(
                    #border=(True, False, True, True),
                    #rounded=(True, False, False, True),
                    container=True

                )
                text2 = gr.Textbox(
                    label="Secondary Structure",
                    max_lines=4,

                ).style(
                    #border=(True, False, True, True),
                    #rounded=(True, False, False, True),
                    container=True

                )

                btn = gr.Button("Predict").style(
                        rounded=(True, True, True, True)
                        )
                outputs= gr.Textbox(label="Result")




    ex = gr.Examples(examples=examples, fn=get_input_give_result,
label='Examples: hsa-mir-34a & lncRNA H19' ,inputs=[text1,text2],
outputs=outputs)


#text.submit(infer, inputs=[text, samples, steps, scale, seed],
outputs=gallery)
    btn.click(get_input_give_result, inputs=[text1, text2], outputs=outputs)

    gr.HTML(
            """
                <div class="footer">
                    </p>
                </div>
                """
                )

block.queue().launch()
```

Script 17.5

```
import gradio as gr
import numpy as np
import tensorflow as tf
from keras.models import load_model
from keras.utils.data_utils import pad_sequences
import itertools
# Load the saved model
model = load_model("/my_model.h5")
# Function to preprocess the input sequence
def preprocess_sequence(sequence):
    sequence = sequence.upper()  # Convert to uppercase
    sequence = sequence.replace('T', 'U') # Replace 'T' with 'U'
    return sequence
# Function to predict the sequence
def predict_sequence(sequence_with_name):
    name, sequence = sequence_with_name.split(": ")
    preprocessed_sequence = preprocess_sequence(sequence)
    k = 3
    max_sequence_length = 28

    nucleotides = ['A', 'C', 'G', 'U']
    kmers = [''.join(p) for p in itertools.product(nucleotides, repeat=k)]
    kmer_to_index = {kmer: i for i, kmer in enumerate(kmers)}

    def sequence_to_kmer(sequence):
        kmer_vector = []
        for i in range(len(sequence)-k+1):
            kmer = sequence[i:i+k]
            if kmer in kmers:
                index = kmers.index(kmer)
                kmer_vector.append(index)
        return kmer_vector
      sequence_vector = sequence_to_kmer(preprocessed_sequence)
    padded_sequence = pad_sequences([sequence_vector], maxlen=max_sequence_length)
      # Check if the sequence is too short
    if len(preprocessed_sequence) < 10:
        return "Normal Sentence"
       # Make prediction
    prediction = model.predict(np.expand_dims(padded_sequence, axis=2))
    if prediction[0][0] > prediction[0][1]:
        result = "Not a Mature･ω･"
    else:
        result = "Mature miRNA ゾ "
    return result
# Create an input box for the sequence
input_text = gr.inputs.Textbox(label="Enter Sequence")
# Create an output label for the prediction result
output_label = gr.outputs.Label(label="Mature Prediction")
```

Script 17.6

```
examples = [
    ['hsa_miR_146a: UGAGAACUGAAUUCCAUGGGUU'],
    ['sRNA: UGGUUUGUUGGCCUCCUAUGCCUAUGGGAAUGGAAUAUAAAGGAUAUGG']
]
# existing code..

with gr.Blocks() as demo:

    with gr.Row():
        with gr.Column(scale=1):
            text1 = gr.Textbox()
            text2 = gr.Textbox()
        with gr.Column(scale=4):
            btn1 = gr.Button("clear sequence")
            btn2 = gr.Button("Predict")

# If you want to link these buttons to a function, you can use the .click()
method. For example:
# btn1.click(my_function, inputs=[text1, text2])

# You can continue your code here, add more HTML or other components...

# Then you need to launch your interface:
# Create the interface
iface = gr.Interface(fn=predict_sequence,
                     inputs=input_text,
                     outputs=output_label,
                     title="Mature miRNA Predictor",
                     description="Predict whether a given sequence is a mature
miRNA.",
                     examples=examples,
                     example_title='Enter Sequences',
                     allow_flagging="manual",
                     show_translation=False)
submit_btn_id = next(i for i,k in iface.blocks.items() if getattr(k, "value",
None) == "Submit")
iface.blocks[submit_btn_id].value = 'Predict'

18882422100
import gradio as gr

io = gr.Interface(lambda x: x, "textbox", "textbox")
io.launch()

# Launch the interface

iface.launch()
```

3.11. Detaction of pre-miRNAs across species

In this study, the identification of orthologous miRNAs was accomplished using a Python implementation. Several essential libraries were employed to facilitate the analysis. The *pandas* library was utilized to read an EXCEL file containing miRNA sequences (Script 18). The columns of interest, namely 'ID' and 'Sequence', were extracted, representing the unique identifier and nucleotide sequence of each miRNA, respectively.

Script 18

```
import numpy as np
import pandas as pd
all_species =
pd.read_excel("/Users/sadaf/Desktop/IBG/sadafThesis/data_for_model/po
sitive.xlsx")
```

For the training of a Word2Vec model, the gensim library was employed (Script 19).

Script 19

```
import pandas as pd
import numpy as np

pip install gensim
from gensim.models import Word2Vec

df =
pd.read_excel(r'H:\unifiles\SadafProposalMaster\miRNA_seq_100.xlsx',
usecols=["ID", "Sequence"])
```

K-mers, subsequences with a length of k, were derived from the miRNA sequences using a sliding window approach. In this study, k was set to 4. The generated k-mers were used as input to train the Word2Vec model, which learned vector representations for each k-mer (Script 20).

Script 20

```
k = 4
kmers = [seq[i:i+k] for seq in df['Sequence'] for i in range(len(seq)
- k + 1)]

model = Word2Vec(sentences=[kmers], min_count=1)
# Get the word vector for a specific k-mer
vector = model.wv['ACUG']

# Convert sequences to vectors using Word2Vec model
vectors = []
for seq in df['Sequence']:
    seq_vectors = []
    for i in range(len(seq) - k + 1):
        kmer = seq[i:i+k]
        if kmer in model.wv.key_to_index:
            seq_vectors.append(model.wv[kmer])
    if len(seq_vectors) > 0:
        vectors.append(np.mean(seq_vectors, axis=0))

}
```

The pairwise cosine similarity between the miRNA sequence vectors was calculated using the *sklearn.metrics.pairwise.cosine_similarity* function (Script 21). This calculation resulted in a similarity matrix, where each element represented the similarity between two miRNA sequences. To identify orthologous miRNAs, a similarity threshold of 0.9 was established.

Script 21

```
# Calculate pairwise cosine similarity between vectors
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrix = cosine_similarity(vectors)


# Find orthologs and syntenologous miRNAs based on similarity
threshold
threshold = 0.9
ortho_pairs = [(df.iloc[i]['ID'], df.iloc[j]['ID']) for i in
range(len(df)) for j in range(i+1, len(df)) if
similarity_matrix[i][j] >= threshold]
```

Orthologous miRNA pairs were determined based on the similarity matrix and the defined threshold (Script 22). The miRNA IDs corresponding to each pair were retrieved from the DataFrame. Additionally, functions were implemented to calculate the alignment identity and similarity between the aligned sequences of the miRNA pairs.

Script 22

```
def alignment_similarity(seq1, seq2):
    """Calculates the percentage of similarity between two aligned
sequences."""
    similarity_count = 0
    for i in range(min(len(seq1), len(seq2))):
        if seq1[i] == seq2[i]:
            similarity_count += 1
        elif seq1[i] == '-' or seq2[i] == '-':
            similarity_count += 0.5
    return similarity_count / max(len(seq1), len(seq2))

threshold = 0.9
ortho_pairs = []
for i in range(len(df)):
    for j in range(i+1, len(df)):
        if similarity_matrix[i][j] >= threshold:
            score = similarity_matrix[i][j]
            identity = alignment_identity(df.iloc[i]['Sequence'],
df.iloc[j]['Sequence'])
            similarity = alignment_similarity(df.iloc[i]['Sequence'],
df.iloc[j]['Sequence'])
            ortho_pairs.append((df.iloc[i]['ID'], df.iloc[j]['ID'],
score, identity, similarity))

for pair in ortho_pairs:
    print(f"{pair[0]} and {pair[1]} have a similarity score of
{pair[2]}, identity of {pair[3]:.2%}, and similarity of
{pair[4]:.2%}.")
```

	ID1	ID2	Similarity_Score	Identity	Similarity
2	cel-mir-35	cel-mir-37	0.9999911	0.551020408	0.551020408
3	hsa-mir-99a	hsa-mir-100	0.9999929	0.543209877	0.543209877
4	hsa-let-7d	hsa-let-7f-2	0.99999076	0.482758621	0.482758621
5	hsa-mir-92a-1	hsa-mir-92a-	0.9999896	0.461538462	0.461538462
6	cel-mir-46	cel-mir-47	0.9999935	0.443298969	0.443298969
7	cel-mir-46	cel-mir-74	0.99998385	0.443298969	0.443298969
8	hsa-let-7a-1	hsa-let-7b	0.9999924	0.43373494	0.43373494
9	cel-mir-36	cel-mir-37	0.9999913	0.428571429	0.428571429
10	cel-mir-78	hsa-mir-29b-	0.99998766	0.427083333	0.427083333
11	cel-mir-44	cel-mir-45	0.99999386	0.416666667	0.416666667
12	cel-mir-35	cel-mir-36	0.9999873	0.412371134	0.412371134
13	cel-mir-38	cel-mir-40	0.9999891	0.410526316	0.410526316
14	hsa-let-7e	hsa-mir-27a	0.99998504	0.405063291	0.405063291
15	cel-mir-84	hsa-let-7a-1	0.9999893	0.4	0.4
16	cel-mir-37	cel-mir-55	0.99998623	0.397959184	0.397959184
17	hsa-let-7e	hsa-let-7f-2	0.99998635	0.397590361	0.397590361
18	hsa-mir-31	hsa-mir-32	0.99998283	0.394366197	0.394366197
19	cel-mir-66	cel-mir-73	0.99998826	0.393939394	0.393939394
20	cel-mir-72	cel-mir-83	0.99998933	0.393939394	0.393939394

Figure 9. Sample screensot of the Excel spreadsheet representing a detailed comparison of various pairs of miRNAs and their respective similarity scores, identity percentages, and similarity percentages.

In the context of Script 22, the similarity score (Figure 9) represents a quantification of how similar two sequences are based on the similarity matrix. The similarity matrix is often used in bioinformatics to measure the similarity between sequences of DNA, RNA, or protein. Each cell in the similarity matrix represents the degree of similarity between two sequences based on their alignment. If the similarity score is close to 1, this suggests that the two sequences are highly similar.

Identity Score: Sequence identity is the measure of the exact nucleotide matches between two sequences. This is different from similarity, as identity requires exact matches, whereas similarity allows for matches that are functionally similar but not identical. For instance, in the case of protein sequences, two amino acids might be considered similar if they are both hydrophobic, even if they are not the exact same amino acid. In the context of this code, it counts how many characters match exactly in the same position between the two sequences.

Similarity (in function): The function alignment_similarity is a bit different from the similarity score. It still measures the degree of similarity between two sequences but in a slightly more relaxed way than identity. It not only increments the count for exact matches (as in identity) but also increments half for gaps ('-'). This could be useful when dealing with sequence alignments where insertions or deletions might occur.

To summarize, the similarity score represents how similar two sequences are based on the similarity matrix. The identity score quantifies the percentage of exact sequence character matches between the two sequences. The similarity (function) computes similarity in a more relaxed way than identity by also considering gaps.

In the context of finding orthologous pre-miRNAs, high values in these scores would suggest that the two miRNAs might have evolved from a common ancestor and possibly carry out similar functions across different species.

Script 23 was used to generate a visual representation of the sequence alignment for the top two pairs of orthologous miRNAs.

Script 23

```
# Select the top 2 miRNA pairs
top2_pairs = df_sorted.head(2).values
# Create a figure with subplots for sequence alignments
fig, axs = plt.subplots(len(top2_pairs), 1, figsize=(10, 3 *
len(top2_pairs)))
spacing = 6
for i, pair in enumerate(top2_pairs):
    # Retrieve the miRNA sequences and IDs
    miRNA1_seq = df[df['ID'] == pair[0]]['Sequence'].values[0]
    miRNA2_seq = df[df['ID'] == pair[1]]['Sequence'].values[0]
    # Set the positions for text display
    ypos = [0, 0.5]
    ylabels = [pair[0], pair[1]]
   # Plot the sequence alignment
   axs[i].set_ylim(-0.5, 1)
    axs[i].set_xlim(0, spacing * max(len(miRNA1_seq), len(miRNA2_seq)))  #
```

```
#Adjust xlim according to spacing
    axs[i].set_yticks(ypos)
    axs[i].set_yticklabels(ylabels, fontsize=12)
    axs[i].set_xticks([])
    axs[i].spines['left'].set_visible(False)
    axs[i].spines['right'].set_visible(False)
    axs[i].spines['top'].set_visible(False)
    axs[i].spines['bottom'].set_visible(False)
    axs[i].grid(axis='x', linestyle='--')

    for j in range(len(miRNA1_seq)):
        char1 = miRNA1_seq[j]
        char2 = miRNA2_seq[j] if j < len(miRNA2_seq) else ' '

        if char1 == char2:
            color = 'blue'
            # Add a line between similar nucleotides
            axs[i].plot([j * spacing, j * spacing], [ypos[0] + 0.07,
ypos[1] - 0], color='blue', linewidth=0.5)  # Adjusted y-coordinates

        else:
            color = 'red'

        axs[i].text(j * spacing, ypos[0], char1, fontsize=10, ha='center',
color=color)  # Adjust x-position according to spacing
        axs[i].text(j * spacing, ypos[1], char2, fontsize=10, ha='center',
color=color)  # Adjust x-position according to spacing

plt.suptitle('Sample Sequence Alignment of Top Two Orthologous miRNA
Pairs')
plt.tight_layout(rect=[0, 0, 1, 1])  # Adjust the rect to give space for
the suptitle
plt.show()
```

4. Results

4.1. DeepMirFinder performance

4.1.1. Model performance of predicting mature miRNAs

The performance of the proposed model on discriminating between true and false mature miRNAs was evaluated on the test dataset using the evaluation metrics ‘loss’, ‘accuracy’, ‘precision’, and ‘recall’. Validation and training loss is used to assess the error of the model on the training and test set, respectively. In Figure 10 the 'val_loss' and 'loss' curves illustrate the model's performance on the validation and training data, respectively. According to the results (Figure 10), DeepMirFinder is a well-fit model, i.e., the ‘validation loss’ and the ‘training loss’ curves are tracking close together throughout training, indicating that the model has a good ability to generalize by fitting well both the training and new (blind) data. During the training process, training loss also got minimized

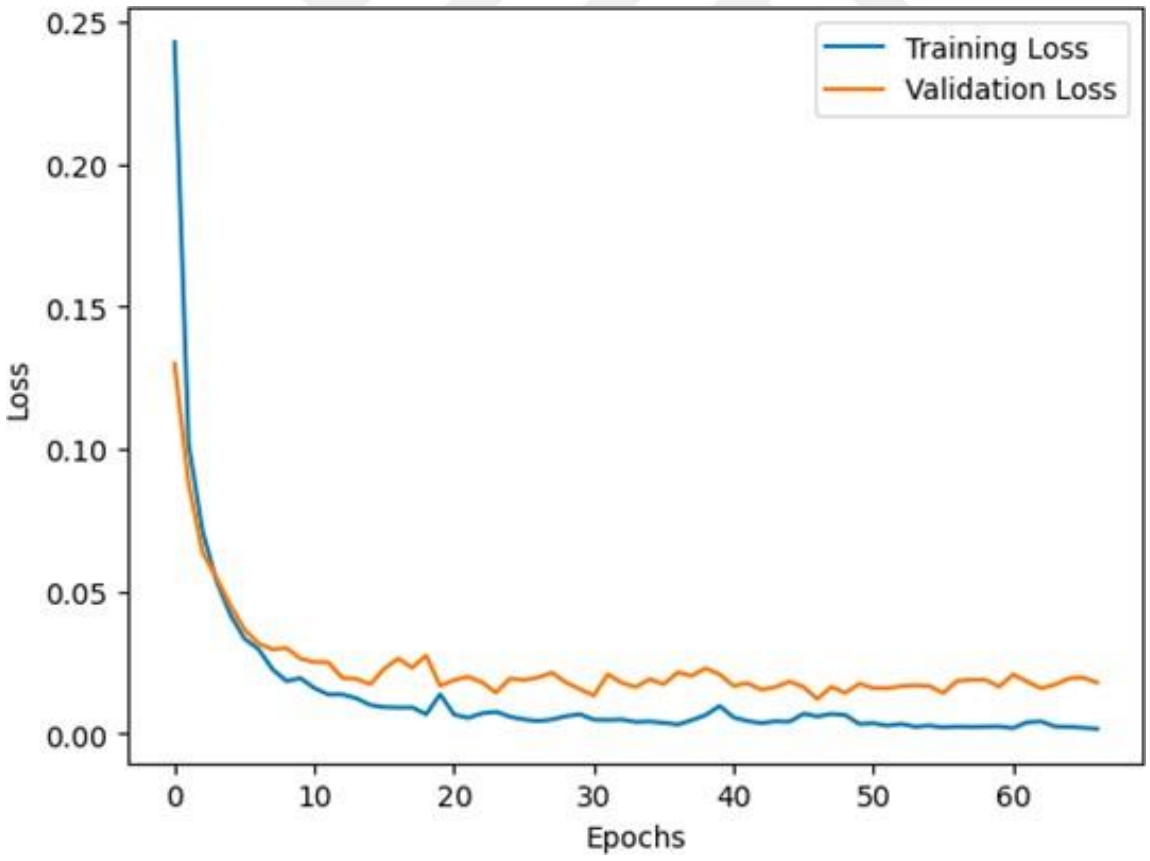


Figure 10. Training and validation loss curve of DeepMirFinder for mature miRNAs.

The proposed model achieved a test accuracy above 97% with a validation loss of 0.1853. The model's precision (specificity) and recall (sensitivity) are very high, approximatelly 98%, suggesting that the model is highly effective in classifying the given data.

Accuracy	Loss	Recall	Precision
0.9728	0.18530	0.98177	0.97643

The model performed very well both on the training and test data with an accuracy score of 0.9881 and 0.9728, respectively. The training accuracy score indicates that the model has learned the training data very effectively, correctly predicted the positive class, making very few false-positive predictions, and maintaining a good balance between precision and recall. On the other hand, the test accuracy score is slightly lower than the corresponding training accuracy score, which is expected as the model has not directly learned from the test data but the training data. The consistency between these metrics for the training and testing datasets suggests that the proposed model is not overfit or underfit, but rather is a well-fit model that should perform well on new data.

The performance of the model was also evaluated based on AUC-ROC (Figure 11) across all possible classification thresholds. The ROC curve is a graphical illustration of the true positive rate *versus* the false positive rate. The closer the ROC curve is to the top-left corner of the graph, the higher the accuracy of the model because the true positive rate (specificity) is equal to 1.

Cross validation is used to evaluate how well machine learning models would generalize on new data. A high average accuracy in cross validation generally suggests that the model will perform similarly on new data. An average accuracy of 97.23% (+/- 0.44%) was achieved after 10-fold cross validation. This indicates that, on average, the proposed model correctly predicted 97.23% of the samples in each of the 10 different test sets. The small standard deviation of +/- 0.44% means that the accuracy did not vary much across different folds, suggesting a high level of consistency and robustness of the model.

Figure 11. ROC curve and AUC statistics illustrating the discriminatory ability of DeepMirFinder to classify true and false mature miRNAs. AUC score of 1.0: perfect classifier; AUC score of 0.5: the classifier that is no better than random.

The AUC score of 0.97 (Figure 11) indicates that the model has a remarkably high degree of discriminative power, being able to accurately differentiate between the positive and negative classes in our dataset.

In conclusion, the high AUC score in conjunction with the high accuracy, sensitivity and specificity, suggests that the model is performing very well. It is apparently able to generalize effectively on blind data, as well as to predict both precursors and mature miRNAs, making it a reliable model for this classification task.

4.1.2. Model performance of predicting miRNA hairpins

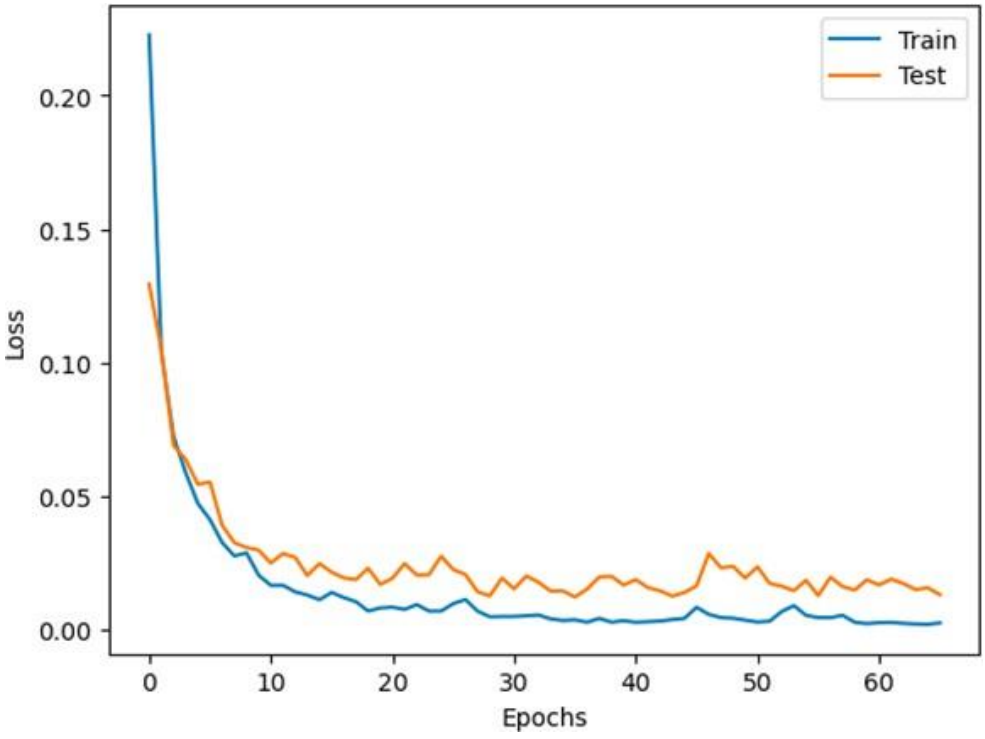
The performance of DeepMirFinder was also assessed on distinguishing between pre-miRNAs and non-miRNA hairpin structures using the ‘loss’, ‘accuracy’, ‘precision’, and ‘recall’ evaluation metrics.

The training loss (‘train’) and validation loss (‘test’) curves (Figure 12) are close to each other (i.e. the model is performing similarly on both the training data and the validation data), and decrease consistently over time, indicating that the model not only has learned well on training data but can also generalize effectively its predictions to new, unseen data. This behavior suggests that the model is not overfit and is able to capture the underlying patterns in the training data.

The consistent decrease in both training and validation loss curves over time indicates that the model is improving as it goes through the training iterations. This trend suggests that the model is learning from the training data, by adjusting its parameters, and gradually reducing its prediction errors.

Furthermore, similar patterns are observed for both 60 (Figure 12A) and 400 (Figure 12B) epochs. Overall, this finding suggests that the model's learning process has converged after the 60th epoch and further training beyond 60 epochs does not result in substantial improvements.

(A)



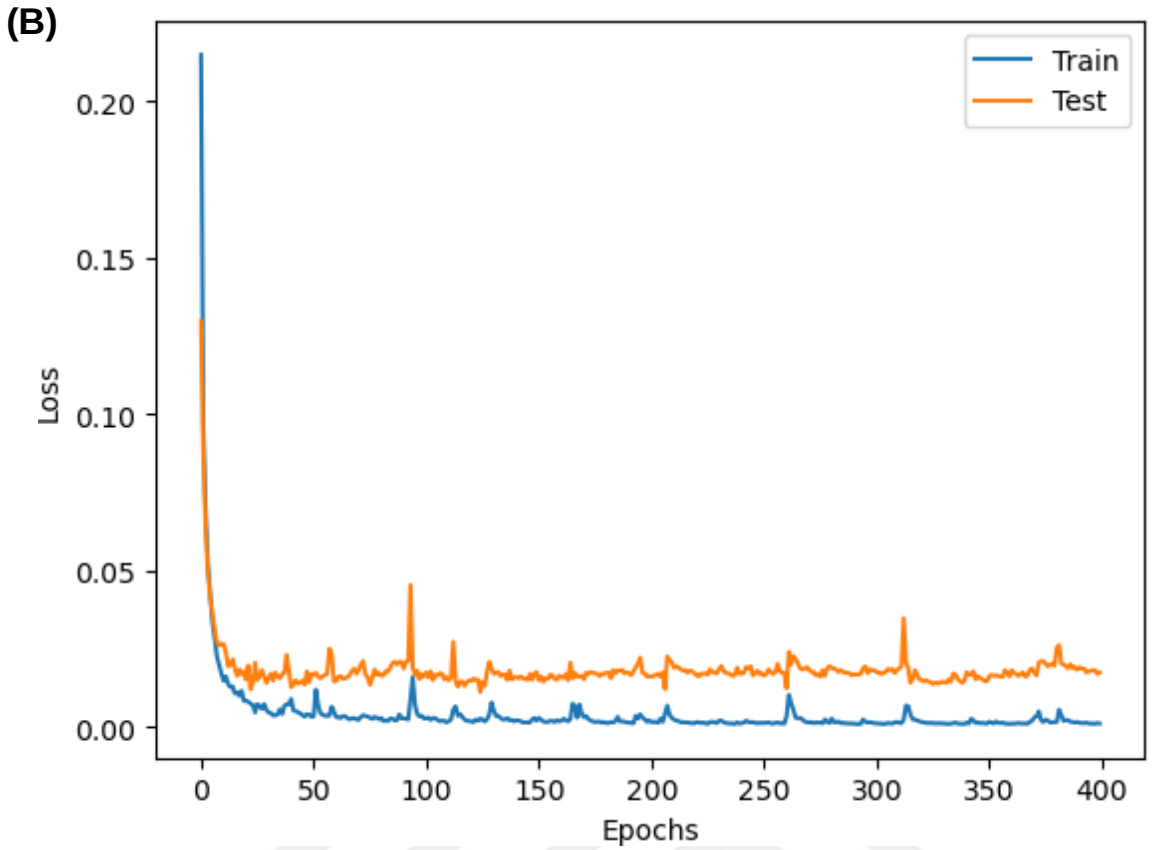


Figure 12. Training versus validation loss curve of DeepMirFinder for precursor miRNAs. A) 60 epochs B) 400 epochs.

The estimated accuracy of the model is approximately 95% with a loss of 0.2465. The precision and recall scores are relatively high, as well, suggestive of the model's classification effectiveness.

Accuracy	Loss	Recall	Precision
0.9440	0.2465	0.9503	0.9376

Moreover, the proposed model showed high performance both on the training and test data of pre-miRNAs with an accuracy score of 0.9889 and 0.9440, respectively. This suggests that DeepMirFİnder is well-fit model and can perform well on unseen data.

According to the AUC of 0.95 results, the model has a 95% probability of correctly distinguishing between positive and negative classes.

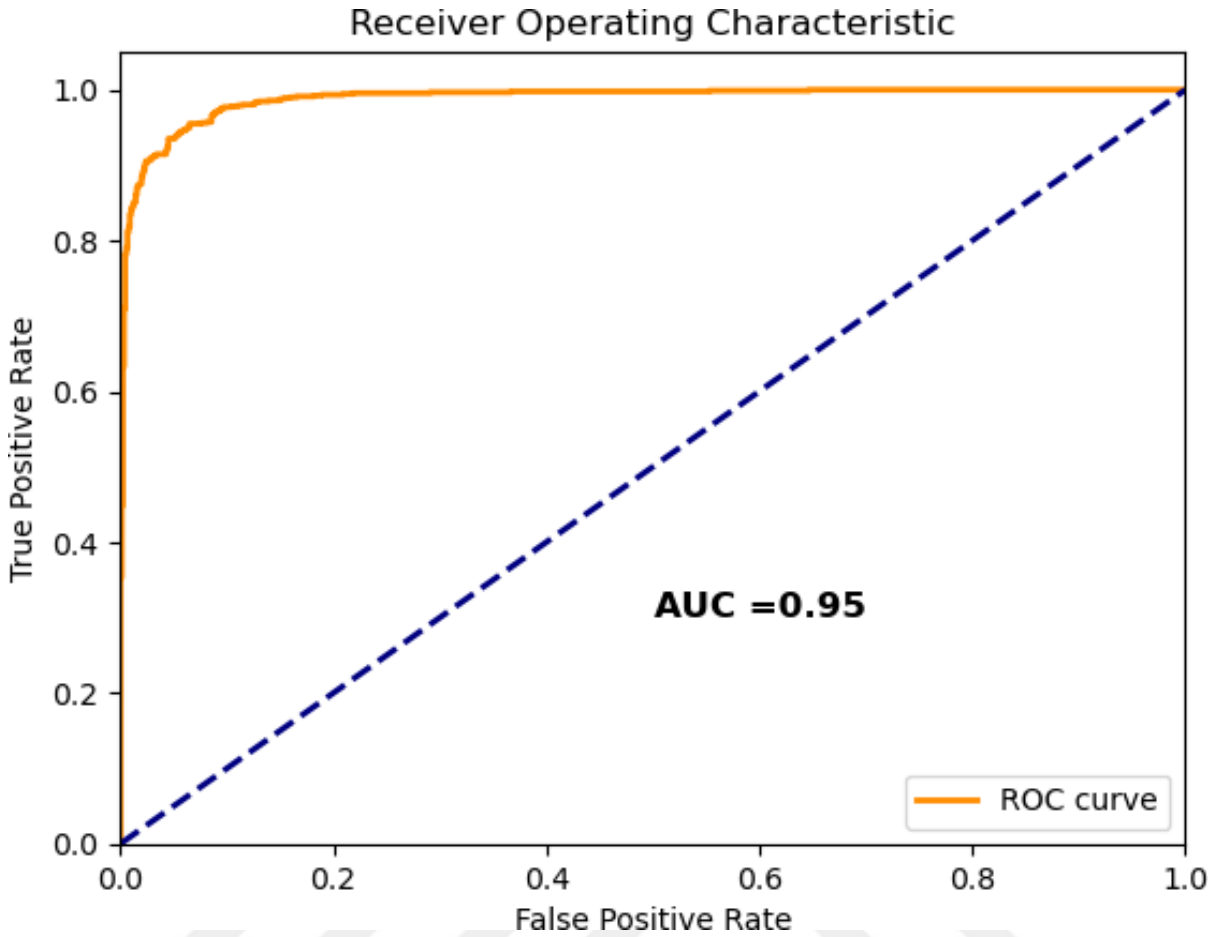


Figure 13. AUC-ROC curve representing the capability of DeepMirFinder of distinguishing between true and false pre-miRNAs.

4.2. DeepMirFinder description

DeepMirFinder provides a user-friendly interface, thus allowing a broad range of users to benefit from this model. It consists of two separate modes: i) the “Mature miRNA Predictor” to predict whether the given sequence is a mature miRNA, and ii) the “PMP (Pre-miRNA Predictor)” mode for predicting whether the input sequence is a *bona fide* pre-miRNA or not. The input includes the query sequence provided in plain sequence format and its corresponding secondary sequence in dot-parenthesis notation (Figure 14 and 15). The output includes the prediction whether the input sequence is or it is not a mature miRNA/pre-miRNA (Figure 14 and 15).

(A)

(B)



Figure 14. DeepMirFinder screenshot showing the prediction output of candidate mature miRNA sequences. The input sequence is predicted to be a true (top) or false (down) mature miRNA.

(A)



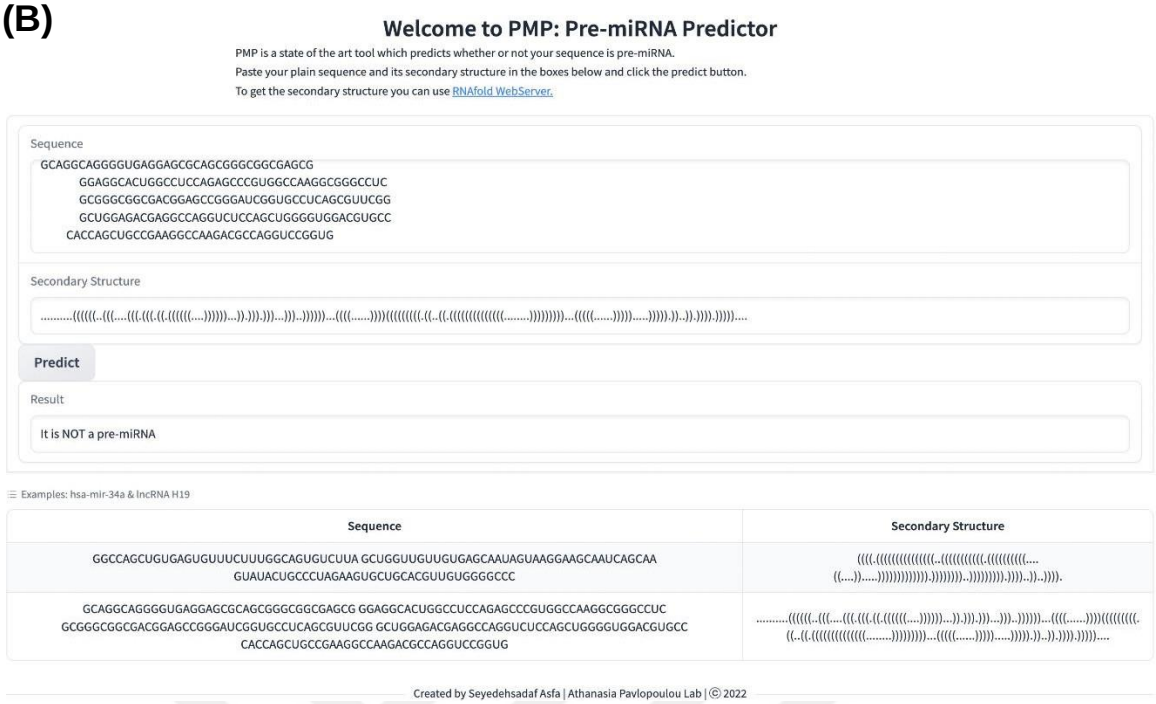


Figure 15. DeepMirFinder screenshot showing the output of candidate pre-miRNA sequences. The input sequence is predicted to be a true mature miRNA hairpin (top) or false (down) pre-miRNA hairpin.

4.3. Identification of orthologous pre-miRNA sequences

A visual representation of the sequence alignment for the top two pairs of orthologous miRNAs is shown in Figure 16. This color-coded visual representation enables a quick and comprehensive overview of the sequence alignments, highlighting regions of similarity (blue) and divergence (red) between each pair of miRNAs**.**

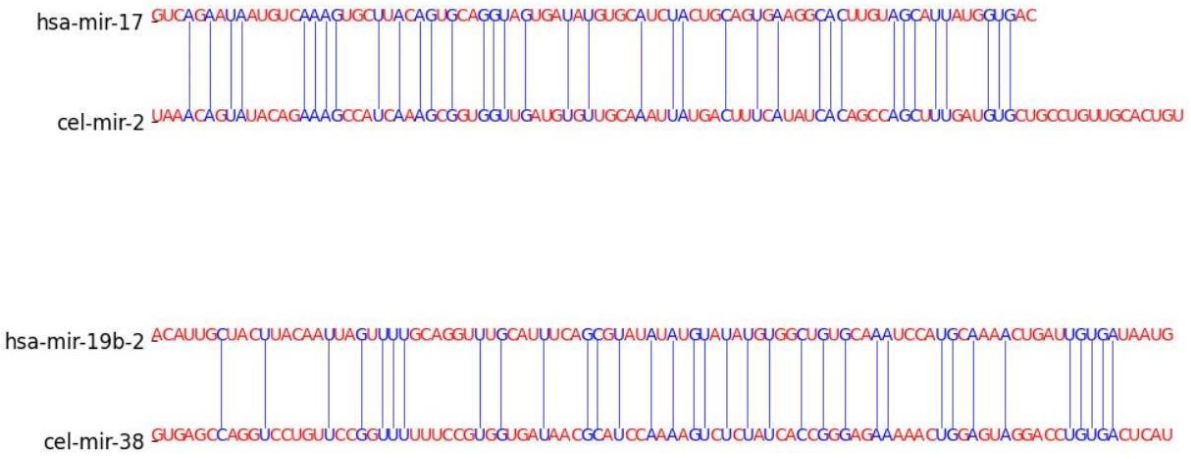


Figure 16. Pairwise sequence alignments of orthologous miRNAs. Human (hsa) pre-miRNA sequences were aligned to nematode (cel) sequences. Each pair of sequences is displayed on a separate subplot. The nucleotides of each sequence are plotted along the x-axis, with the y-axis divided into two halves for each pair, denoting each miRNA within the pair. When the miRNAs share the same nucleotide at a specific position, the nucleotide is highlighted in blue, and a thin vertical blue line connects them, indicating a match. Conversely, mismatches are depicted in red.

5. DISCUSSION

In the context of this thesis, a neural deep learning-based model, DeepMirFinder, was developed for predicting potential mature miRNAs and pre-miRNAs. The modern computational methods for the *ab initio* discovery of miRNA hairpin structures apply machine learning approaches, such as support vector machines, depend on manually selected features of pre-miRNAs by experts in the field (Gerlach et al., 2009; Rahman et al., 2012). DeepMirFinder utilizes neural networks, CNN and LSTM, to learn to automatically extract features from the precursor and mature miRNAs; CNN extracts local spatial features and LSTM can model sequential data. In this study, a CNN and a hybrid CNN-LSTM model were designed, trained and evaluated towards the identification of precursor and mature miRNAs, respectively. A similar model developed by Zheng and colleagues (2020) (Zheng et al., 2020), dnnMiRPre, utilized CNNs and RNNs to detect human pre-miRNAs. However, this model was trained on a relatively small 'positive' dataset of only 88 human pre-miRNAs collected from the miRBase. This dataset included both 'high' and 'low' confidence entries, introducing in this way bias into the model. In my study, in order to avoid inaccuracies and enhance the predicitve robustness of the model, a 'positive' dataset compiled exclusively of a relatively large number of pre-miRNAs and mature miRNAs annotated as 'high confidence' in miRBase was used to train the neural network model; those entries with 'high' confidence annotation have more than 100 sequencing reads mapping across the pre-miRNA locus, and less than 30% of reads mapping to the more abundant arm of the hairpin precursor have the same 5' end (Kozomara et al., 2019). Moreover, duplicate sequences were removed and secondary structures were predicted.

To ensure comprehensiveness, pre-miRNAs and their corresponding mature miRNAs from fifteen well-annotated genomes of both vertebrate and invertebrate species, representing diverse taxonomic divisions, were included in the 'positive' dataset.

For training and evaluating splitting, DeepMirFinder was trained on the training set (75% of the merged positive and negative dataset) with an adequate number of epochs, and evaluated on the remaining 25% of the testing set. The model could predict mature miRNAs and miRNA hairpin structures with exceptionally high accuracy in the testing dataset.

MiRNAs represent critical components of the eukaryotic gene regulatory machinery, and therefore there is evolutionary pressure to retain their function across eukaryotic species. Independent of sequence divergence, evolutionarily related pre-miRNAs in different organisms (i.e. orthologs) may exhibit similar function (Pang et al., 2006; Tarver et al., 2012).

The detection of orthologous miRNA genes is hindered by their lack of strong primary sequence conservation, as opposed to protein-coding genes. Therefore, miRNA orthologs cannot be detected easily using conventional sequence similarity search methods like BLAST, which is implemented in several methods for screening miRNA candidates across species (Bryzghalov et al., 2020; Gerlach et al., 2009).

DeepMirFinder was designed to identify both known and novel pre-miRNA orthologs across eukaryotic species, even with poor sequence conservation. In this study, it was made effective use of popular Python libraries for efficient data processing, sequence modeling, and similarity calculations. These libraries offered reliable and well-established implementations of the necessary functionalities, enabling the successful execution of the ortholog miRNA prediction analysis.

In order to identify evolutionarily divergent miRNA sequences across species (i.e., sequences most similar to each other as compared to similar sequences in a given genome), pairwise sequence similarities were performed by applying Word2Vec, a neural network-based technique for generating word embeddings (i.e. representations of a word in vector space), used for natural language processing (NLP) (Mikolov et al., 2013). Word2Vec has been apllied for generating cancerous mutations representations (Kim et al., 2018), comparison of DNA sequences (Ren et al., 2022), accurate classification of COVID-19 viral sequences (Adjuik & Ananey-Obiri, 2022), prediction of drug-target Interactions (Zhang et al., 2019). The pre-miRNA sequences are divided into k-mers (substrings of length k), and are represented by vectors. Word2Vector creates word embeddings of various k-mers and processes them like words of NLP; this approach allows capturing local dependencies between nucleotides in the sequence.

6. CONCLUSIONS AND FUTURE ASPECTS

The discovery of precursor and mature microRNAs poses a major challenge to computer scientists which is confronted with the advent of deep learning techniques in the last decade. In this study, neural network-based approaches were applied for the fast, accurate and sensitive prediction of *bona fide* hairpin precursors and mature microRNAs, as well as the identification of orthologous precursor miRNAs across species. Herein, an application with a user-friendly interface, DeepMirFinder, was developed which allows users from different disciplines without any computing skills to predict potential mature miRNAs and miRNA precursors. Several evaluation metrics were used for assessing the overall performance of the proposed model. According to these metrics, DeepMirFinder is a well-fit model which can discriminate between true and false precursor/mature miRNAs with high accuracy. Moreover, DeepMirFinder demonstrated a high level of consistency and could generalize its predictions effectively to unseen data. In addition, this model could be implemented in databases that collect and curate miRNAs in order to ensure the correct classification of their entries.

7. REFERENCES

Adjuik, T. A., & Ananey-Obiri, D. (2022). Word2vec neural model-based technique to generate protein vectors for combating COVID-19: A machine learning approach. *International Journal of Information Technology: An Official Journal of Bharati Vidyapeeth's Institute of Computer Applications and Management*, *14*(7), 3291–3299. https://doi.org/10.1007/s41870-022-00949-2

Altenhoff, A. M., & Dessimoz, C. (2009). Phylogenetic and Functional Assessment of Orthologs Inference Projects and Methods. *PLOS Computational Biology*, *5*(1), e1000262. https://doi.org/10.1371/journal.pcbi.1000262

Bryzghalov, O., Szcześniak, M. W., & Makałowska, I. (2020). SyntDB: Defining orthologues of human long noncoding RNAs across primates. *Nucleic Acids Research*, *48*(D1), D238–D245. https://doi.org/10.1093/nar/gkz941

Cui, H., Zhai, J., & Ma, C. (2015). miRLocator: Machine Learning-Based Prediction of Mature MicroRNAs within Plant Pre-miRNA Sequences. *PLOS ONE*, *10*(11), e0142753. https://doi.org/10.1371/journal.pone.0142753

Dexheimer, P. J., & Cochella, L. (2020). MicroRNAs: From Mechanism to Organism. *Frontiers in Cell and Developmental Biology*, *8*. https://www.frontiersin.org/articles/10.3389/fcell.2020.00409

Friedländer, M. R., Mackowiak, S. D., Li, N., Chen, W., & Rajewsky, N. (2012). MiRDeep2 accurately identifies known and hundreds of novel microRNA genes in seven animal clades. *Nucleic Acids Research*, *40*(1), 37–52. https://doi.org/10.1093/nar/gkr688

Fu, X., Zhu, W., Cai, L., Liao, B., Peng, L., Chen, Y., & Yang, J. (2019). Improved Pre-miRNAs Identification Through Mutual Information of Pre-miRNA Sequences and Structures. *Frontiers in Genetics*, *10*, 119. https://doi.org/10.3389/fgene.2019.00119

Gabaldón, T., & Koonin, E. V. (2013). Functional and evolutionary implications of gene orthology. *Nature Reviews Genetics*, *14*(5), Article 5. https://doi.org/10.1038/nrg3456

Gao, D., Middleton, R., Rasko, J. E. J., & Ritchie, W. (2013). miREval 2.0: A web tool for simple microRNA prediction in genome sequences. *Bioinformatics*, *29*(24), 3225–3226. https://doi.org/10.1093/bioinformatics/btt545

Gerlach, D., Kriventseva, E. V., Rahman, N., Vejnar, C. E., & Zdobnov, E. M. (2009). miROrtho: Computational survey of microRNA genes. *Nucleic Acids Research*, *37*(Database issue), D111-117. https://doi.org/10.1093/nar/gkn707

Ha, T.-Y. (2011). MicroRNAs in Human Diseases: From Cancer to Cardiovascular Disease. *Immune Network*, *11*(3), 135–154. https://doi.org/10.4110/in.2011.11.3.135

Hamzeiy, H., Allmer, J., & Yousef, M. (2014). Computational methods for microRNA target prediction. *Methods in Molecular Biology (Clifton, N.J.)*, *1107*, 207–221. https://doi.org/10.1007/978-1-62703-748-8_12

He, S., Liu, C., Skogerbø, G., Zhao, H., Wang, J., Liu, T., Bai, B., Zhao, Y., & Chen, R. (2008). NONCODE v2.0: Decoding the non-coding. *Nucleic Acids Research*, *36*(Database issue), D170-172. https://doi.org/10.1093/nar/gkm1011

Kim, S., Lee, H., Kim, K., & Kang, J. (2018). Mut2Vec: Distributed representation of cancerous mutations. *BMC Medical Genomics*, *11*(Suppl 2), 33. https://doi.org/10.1186/s12920-018-0349-7

Kin, T., Yamada, K., Terai, G., Okida, H., Yoshinari, Y., Ono, Y., Kojima, A., Kimura, Y., Komori, T., & Asai, K. (2007). fRNAdb: A platform for mining/annotating functional RNA candidates from non-coding RNA sequences. *Nucleic Acids Research*, *35*(Database issue), D145-148. https://doi.org/10.1093/nar/gkl837

Kleftogiannis, D., Theofilatos, K., Papadimitriou, S., Tsakalidis, A., Likothanassis, S., & Mavroudi, S. (2012). *ncRNA-class Web Tool: Non-coding RNA feature extraction and pre-miRNA classification web tool*. *382*, 632–641. https://doi.org/10.1007/978-3-642-33412-2_65

Kozomara, A., Birgaoanu, M., & Griffiths-Jones, S. (2019). miRBase: From microRNA sequences to function. *Nucleic Acids Res.*, *47*(D1), D155–D162.

Kristensen, D. M., Wolf, Y. I., Mushegian, A. R., & Koonin, E. V. (2011). Computational methods for Gene Orthology inference. *Briefings in Bioinformatics*, *12*(5), 379–391. https://doi.org/10.1093/bib/bbr030

Lee, C.-T., Risom, T., & Strauss, W. (2007). Evolutionary Conservation of MicroRNA Regulatory Circuits: An Examination of MicroRNA Gene Complexity and Conserved MicroRNA-Target Interactions through Metazoan Phylogeny. *DNA and Cell Biology*, *26*, 209–218. https://doi.org/10.1089/dna.2006.0545

Lee, Y., Kim, M., Han, J., Yeom, K.-H., Lee, S., Baek, S. H., & Kim, V. N. (2004). MicroRNA genes are transcribed by RNA polymerase II. *The EMBO Journal*, *23*(20), 4051–4060. https://doi.org/10.1038/sj.emboj.7600385

Lestrade, L., & Weber, M. J. (2006). SnoRNA-LBME-db, a comprehensive database of human H/ACA and C/D box snoRNAs. *Nucleic Acids Research*, *34*(Database issue), D158-162. https://doi.org/10.1093/nar/gkj002

Libri, V., Miesen, P., van Rij, R. P., & Buck, A. H. (2013). Regulation of microRNA biogenesis and turnover by animals and their viruses. *Cellular and Molecular Life Sciences*, *70*(19), 3525–3544. https://doi.org/10.1007/s00018-012-1257-1

Mikolov, T., Chen, K., Corrado, G. s, & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at ICLR*, *2013*.

O'Brien, J., Hayder, H., Zayed, Y., & Peng, C. (2018). Overview of MicroRNA Biogenesis, Mechanisms of Actions, and Circulation. *Frontiers in*

Endocrinology, *9*. https://www.frontiersin.org/articles/10.3389/fendo.2018.00402

Ortiz-Quintero, B. (2020). Extracellular MicroRNAs as Intercellular Mediators and Noninvasive Biomarkers of Cancer. *Cancers*, *12*(11), 3455. https://doi.org/10.3390/cancers12113455

Pang, K. C., Frith, M. C., & Mattick, J. S. (2006). Rapid evolution of noncoding RNAs: Lack of conservation does not mean lack of function. *Trends in Genetics: TIG*, *22*(1), 1–5. https://doi.org/10.1016/j.tig.2005.10.003

Quintana-Murci, L. (2016). Understanding rare and common diseases in the context of human evolution. *Genome Biology*, *17*(1), 225. https://doi.org/10.1186/s13059-016-1093-y

Rahman, M. E., Islam, R., Islam, S., Mondal, S. I., & Amin, M. R. (2012). MiRANN: A reliable approach for improved classification of precursor microRNA using Artificial Neural Network model. *Genomics*, *99*(4), 189–194. https://doi.org/10.1016/j.ygeno.2012.02.001

Ren, R., Yin, C., & S-T Yau, S. (2022). kmer2vec: A Novel Method for Comparing DNA Sequences by word2vec Embedding. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, *29*(9), 1001–1021. https://doi.org/10.1089/cmb.2021.0536

Riffo-Campos, Á. L., Riquelme, I., & Brebi-Mieville, P. (2016). Tools for Sequence-Based miRNA Target Prediction: What to Choose? *International Journal of Molecular Sciences*, *17*(12), 1987. https://doi.org/10.3390/ijms17121987

Ryu, S., Joshi, N., McDonnell, K., Woo, J., Choi, H., Gao, D., McCombie, W. R., & Mittal, V. (2011). Discovery of Novel Human Breast Cancer MicroRNAs from Deep Sequencing Data by Analysis of Pri-MicroRNA Secondary Structures. *PLoS ONE*, *6*(2), e16403. https://doi.org/10.1371/journal.pone.0016403

Sudarshan, S. (2023, May 30). RECURRENT NEURAL NETWORK (RNN). *MLearning.Ai*. https://medium.com/mlearning-ai/recurrent-neural-network-rnn-20a619190586

Steenwyk, J. L., Phillips, M. A., Yang, F., Date, S. S., Graham, T. R., Berman, J., Hittinger, C. T., & Rokas, A. (2022). An orthologous gene coevolution network provides insight into eukaryotic cellular and genomic structure and function. *Science Advances*, *8*(18), eabn0105. https://doi.org/10.1126/sciadv.abn0105

Takashima, Y., Murata, A., Iida, K., Sugai, A., Hagiwara, M., & Nakatani, K. (2022). Method for Identifying Sequence Motifs in Pre-miRNAs for Small-Molecule Binding. *ACS Chemical Biology*, *17*(10), 2817–2827. https://doi.org/10.1021/acschembio.2c00452

Tarver, J. E., Donoghue, P. C. J., & Peterson, K. J. (2012). Do miRNAs have a deep evolutionary history? *BioEssays: News and Reviews in Molecular, Cellular and Developmental Biology*, *34*(10), 857–866. https://doi.org/10.1002/bies.201200055

Tasdelen, A., & Sen, B. (2021). A hybrid CNN-LSTM model for pre-miRNA classification. *Scientific Reports*, *11*(1), Article 1. https://doi.org/10.1038/s41598-021-93656-0

Tav, C., Tempel, S., Poligny, L., & Tahi, F. (2016). miRNAFold: A web server for fast miRNA precursor prediction in genomes. *Nucleic Acids Research*, *44*(W1), W181-184. https://doi.org/10.1093/nar/gkw459

Tüfekci, K. U., Meuwissen, R. L. J., & Genç, S. (2014). The role of microRNAs in biological processes. *Methods in Molecular Biology (Clifton, N.J.)*, *1107*, 15–31. https://doi.org/10.1007/978-1-62703-748-8_2

van den Berg, A., Mols, J., & Han, J. (2008). RISC-Target Interaction: Cleavage and Translational Suppression. *Biochimica et Biophysica Acta*, *1779*(11), 668–677. https://doi.org/10.1016/j.bbagrm.2008.07.005

Vaz, J. M., & Balaji, S. (2021). Convolutional neural networks (CNNs): Concepts and applications in pharmacogenomics. *Molecular Diversity*, *25*(3), 1569–1584. https://doi.org/10.1007/s11030-021-10225-3

Wang, Y., Ma, H., Alifu, K., & Lv, Y. (2021). Remote sensing image description based on word embedding and end-to-end deep learning. *Scientific Reports*, *11*(1), 3162. https://doi.org/10.1038/s41598-021-82704-4

Wu, X., & Song, Y. (2011). Preferential regulation of miRNA targets by environmental chemicals in the human genome. *BMC Genomics*, *12*(1), 244. https://doi.org/10.1186/1471-2164-12-244

Zhang, Y.-F., Wang, X., Kaushik, A. C., Chu, Y., Shan, X., Zhao, M.-Z., Xu, Q., & Wei, D.-Q. (2019). SPVec: A Word2vec-Inspired Feature Representation Method for Drug-Target Interaction Prediction. *Frontiers in Chemistry*, *7*, 895. https://doi.org/10.3389/fchem.2019.00895

Zheng, X., Fu, X., Wang, K., & Wang, M. (2020). Deep neural networks for human microRNA precursor detection. *BMC Bioinformatics*, *21*(1), 17. https://doi.org/10.1186/s12859-020-3339-7

8. APPENDIX

8.1. Ethics Committee Approval

İZMİR BİYOTIP VE GENOM MERKEZİ
GİRİŞİMSEL OLMAYAN ARAŞTIRMALAR ETİK KURULU (İBG-GOEK)
KARARI

Toplantı Tarihi	: 24/09/2021	**Toplantı Günü**	: Cuma
Toplantı Sayısı	: 10	**Toplantı Saati**	: 10:30

Sayın None Athanasia PAVLOPOULOU,

2021-032 Protokol No'lu; sorumlusu olduğunuz "Derin Öğrenme yöntemleri kullanılarak Öncü ve Olgun miRNA'ların Tespiti: DeepMirFinder" başlıklı araştırmanın uygulanmasında etik açıdan sakınca olmadığına oy birliği ile karar verilmiştir.

Bilgilerinizi ve gereğini rica ederiz.

Prof. Dr. H. Alper BAĞRIYANIK **Başkan** (Katılmadı)	**Prof. Dr. Hilmi ORHAN** **Başkan Yardımcısı** (Katılmadı)
Prof. Dr. Sedef AKGÜNGÖR **Üye**	**Prof. Dr. İnci ALACACIOĞLU** **Üye**
Prof. Dr. Gülgün OKTAY **Üye** (Katılmadı)	**Dr. Serap ERKEK ÖZHAN** **Üye**
Prof. Dr. Şermin GENÇ **Üye** (Katılmadı)	**Doç. Dr. Canan Aslı YILDIRIM** **Üye**
Doç. Dr. Can KÜÇÜK **Üye**	**Dr. Öğr. Üyesi Yavuz OKTAY** **Üye**

8.2. Curriculum Vitae

Seyedehsadaf Asfa

Address: İzmir Uluslararası Biyotıp ve Genom Enstitüsü (IBG-İzmir) Balçova 35340 İzmir

e-mail: Gmail.com

University	Faculty/Institute	Field	Degree	Graduation Year
Dokuz Eylul University, Turkey	İzmir International Biomedicine and Genome Institute	Molecular Biology and Genetics	M.Sc.	2023
Azad University of Falavarjan, Iran	Faculty of Cellular and Molecular Biology	Cellular and Molecular Biology	B.Sc.	2012

Skills

Bash scripting
Cytoscape
Machine Learning
Python
R programming language

Languages

Persian (native)
English (Fluent)

Research Interests

Bioinformatics
Systems Biology
Deep Learning (CNN, RNN)
Cancer Computational Biology

Publications in ISI Journals

1) Asfa S, Toy HI, Arshinchi Bonab R, Chrousos GP, Pavlopoulou A, Geronikolou SA (2023) “Soft Tissue Ewing Sarcoma Cell Drug Resistance Revisited: A Systems Biology approach.” *International Journal of Environmental Research and Public Health* 20:6288.

2) Özbek M, Toy HI, Takan I, Asfa S, Arshinchi Bonab R, Karakülah G, Kontou PI, Geronikolou SA, Pavlopoulou A (2022) “A Counterintuitive Neutrophil-Mediated Pattern in COVID-19 Patients Revealed through Transcriptomics Analysis” *Viruses* 15:104
3) Sagkrioti E, Biz GM, Takan I, Asfa S, Nikitaki Z, Zanni V, Kars RH, Hellweg CE, Azzam EI, Logotheti S, Pavlopoulou A, Georgakilas AG (2022) “Radiation Type- and Dose-Specific Transcriptional Responses across Healthy and Diseased Mammalian Tissues” *Antioxidants* 11:2286
4) Arshinchi Bonab R, Asfa S, Kontou P, Karakülah,G, Pavlopoulou A (2022) “Identification of neoplasm-specific signatures of miRNA interactions by employing a systems biology approach” *PeerJ* 10:e14149
5) Pavlopoulou A, Asfa S, Gioukakis E, Mavragani IV, Nikitaki Z, Takan I, Pouget J-P, Harrison L, Georgakilas AG (2021) “In Silico Investigation of the Biological Implications of Complex DNA Damage with Emphasis in Cancer Radiotherapy through a Systems Biology Approach” *Molecules* 26:7602