MACHINE LEARNING BASED LANGUAGE MODELS ON NUCLEOTIDE SEQUENCES OF HUMAN GENES

by

Musa Nuri İhtiyar

B.S., Computer Engineering, Boğaziçi University, 2022

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2023

MACHINE LEARNING BASED LANGUAGE MODELS ON NUCLEOTIDE SEQUENCES OF HUMAN GENES

APPROVED BY:

Assoc. Prof. Arzucan Özgür ....
(Thesis Supervisor)

Prof. Tunga Güngör ....

Prof. Olcay Taner Yıldız ....

DATE OF APPROVAL: 23.05.2023

ACKNOWLEDGEMENTS

I want to thank a lot to my family for their support. Also, I want to say a big thank you to my advisor Assoc. Prof. Arzucan Özgür due to all of her efforts like guiding and encouraging me about the thesis. Moreover, thanks a lot to Prof. Tunga Güngör and Prof. Olcay Taner Yıldız for being on my thesis defense jury. Lastly, this work is supported by Boğaziçi University Research Fund under the Grant Number 16903.

ABSTRACT

MACHINE LEARNING BASED LANGUAGE MODELS ON NUCLEOTIDE SEQUENCES OF HUMAN GENES

The use of computers for different fields of science has provided tremendous benefits. This phenomenon is expected to be more common as the speed of computers and the amount of data available for different kinds of scientific problems increase. This study focuses on genomics, one of the most exciting areas of science. We have applied several techniques to obtain a model for nucleotide sequences of genes that are found in human beings so that the model can learn the general pattern in these nucleotide sequences and predict how likely it is that an unseen sequence is a gene that belongs to human beings. They can even generate new nucleotide sequences.

All of the methods used are examples of machine learning, where the programs are designed to learn from data for a specific task, rather than explicitly programming what to do at each step. Traditional approaches such as N-grams and more recent deep learning-based techniques such as recurrent neural networks and transformer architecture language models are used. In addition to the classical metrics, the strength of the methods is measured using a real-world task from the field of genomics.

Finally, the results show an interesting comparison of how all these models perform on a task that is inherently different from classical natural language processing tasks, and how sometimes simple models like N-grams can be as good as, if not better than, more sophisticated techniques such as transformer for solving certain types of problems. Furthermore, the significance of evaluating obtained models on real-life tasks is seen because the transformer model was superior to the N-gram model according to perplexity, although it performed worse on real-world task.

ÖZET

İNSAN GENLERİ NÜKLEOTİT DİZİLERİNİN MAKİNE ÖĞRENMESİ İLE MODELLENMESİ

Bilgisayarların kullanımı bilimin değişik alanlarındaki birçok çalışmada çeşitli faydalar sağlamıştır. Bilgisayarların hızındaki ve elimizdeki verinin miktarındaki artış dikkate alındığında bu durumun daha da yaygınlaşacağı beklentisi kuvvet kazanmaktadır. Bu çalışma bilgisayarları bilimin en ilginç alanlarından birisi olan Genetik'te kullanmak üzerinedir. İnsan genlerini oluşturan nükleotit dizilerindeki yapıyı modellemek için birçok farklı tekniğin kullanıldığı bu çalışmada elde edilen modeller görülmemiş bir nükleotit dizisinin bir insan genine ait olup olmadığını tahmin edebilme özelliğini sahiptir. Ayrıca yeni nükleotit serilerini üretme görevini de yerine getirebilmektedirler.

Kullanılan tüm metodlar Makine Öğrenmesi isimli bir yaklaşım tabanlı olup hedef bilgisayara her aşamada ne yapması gerektiğini tek tek izah etmek yerine veriyi kullanarak öğrenmesini sağlamak şeklinde açıklanabilir. Eskiden beri kullanılan N-gram tarzı tekniklerin yanında son zamanlarda çok popüler hale gelen Derin Öğrenme tabanlı Recurrent Neural Networks ve Transformer dil modellerinden de faydalanılmıştır. Geliştirilen sistemler klasik başarı ölçütlerinin yanında gerçek hayata daha yakın olan ve Genetik ile alakalı bazı görevlerdeki başarıları ile de değerlendirilmiştir.

Sonuçlar Doğal Dil ile bazı farkları barındıran bir problemde farklı tekniklerin kıyaslanması adına ilginçtir. Ayrıca N-gram tarzı basit modellerin bazı problemleri çözmede Transformer gibi karmaşık modellerden daha iyi olmalarının mümkün olduğu görülmüş olmuştur. Son olarak modelleri ölçerken gerçek görevlerde testin çok mühim olduğu öğrenilmiştir çünkü transformer modeli perplexity dikkate alındığında N-gram'dan daha iyi iken gerçek testte daha kötü sonuç vermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$< s >$	Beginning of the sentence
$< /s >$	End of the sentence
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
$max(a, b)$	Maximum of a and b
p	Probability
ReLU	Rectified Linear Unit
Tanh	Hyperbolic Tangent

LIST OF ACRONYMS/ABBREVIATIONS

3D	Three Dimensional
AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
DNA	Deoxyribonucleic Acid
FC	Fully Connected
GAN	Generative Adversarial Network
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
NCBI	The National Center for Biotechnology Information
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
RNA	Ribonucleic acid Acid
RNN	Recurrent Neural Network
SVM	Support Vector Machine

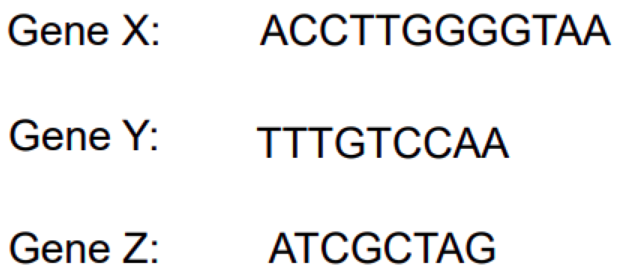
1. INTRODUCTION

Understanding the world we live in is one of science's essential aims. The magnitude of the examined topic differs significantly between different branches of science. For instance, while quantum physics works with tiny subatomic particles, astronomy deals with enormous planets and stars. Nevertheless, the goal of determining the essential concepts, building blocks, and laws explaining the relationship between the fundamental concepts remains the same.

Biology and medicine are two fascinating fields among scientific disciplines because they focus directly on human beings. They are also useful for learning more about our bodies and getting ideas for curing diseases. While there are many different structures of varying sizes in our bodies, all of which are critically important, deoxyribonucleic acid (DNA) seems to have a special significance.

DNA is a significant building block responsible for carrying fundamental data about various factors affecting the lives of the creatures it belongs to, such as human beings, animals, plants, and others. DNA consists of nucleotides, yet some sequences have unique functionalities and importance; therefore, they are called genes.

The actual structure of genes looks simple. Four different nucleotide types are Adenine, Thymine, Cytosine, and Guanine. All genes can be considered a sequence of these four nucleotides, but the sequence length may vary from 100s to 10.000s. Some symbolic genes are given in Figure 1.1. Consequently, despite looking simple, the number of possible nucleotide sequences for a chosen size like 10.000 is vast (about 10^{6000}). Yet, only a few exist in human beings, animals, and others. If we specifically focus on genes existing in human beings, the number decreases to about 70.000.



Gene X:	ACCTTGGGGTAA
Gene Y:	TTTGTCCAA
Gene Z:	ATCGCTAG

Figure 1.1. Nucleotide Sequences For Symbolic Genes.

Artificial intelligence and machine learning are also among the most popular and exciting fields of our time. These fields involve helping computers or robots perform operations usually done by human beings, such as recognizing objects in an image, understanding natural language in the form of text, playing games like chess, or driving a car. However, they can also be used for more sophisticated goals, like helping humans make new scientific discoveries.

Because DNA nucleotide sequences resemble the text format of natural languages, methods commonly used for natural language processing could theoretically be used to solve various genomics tasks. Although the use of natural language understanding techniques, which involve understanding a particular sentence, is an option, the use of natural language generation methods, which is about modeling language using given sentences, is also possible in the field of genomics.

In this study, we investigated the problem of modeling nucleotide sequences for genes associated with humans. In other words, we looked at the performance of different machine learning models in the task of looking at nucleotide sequences for some of the genes to obtain a model so that they can predict how likely it is that unseen sequences belong to a human gene. They could also generate new nucleotide sequences for candidate genes.

Language models, primarily transformer-based ones [1], obtained colossal success in natural language understanding, for instance, BERT [2], and natural language generation, for example, GPT-3 [3], when natural language processing is considered. If the DNA-related bioinformatics domain is concerned, discriminative models, like DNABert [4], exist. Yet, the generative side of the coin is mainly unexplored to the best of our knowledge. Nonetheless, DNA sequences are very close to natural language in structure, so we focused on developing an autoregressive generative language model like GPT-3 for DNA sequences.

Because working with whole DNA sequences is challenging without substantial computational resources, we decided to carry out our study on a smaller scale, focusing on nucleotide sequences of human genes, unique parts in DNA with specific functionalities, instead of the whole DNA. The expected behavior is illustrated in Figure 1.2.

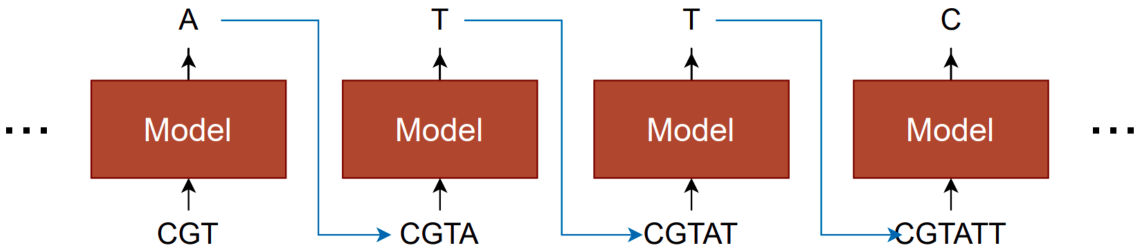


Figure 1.2. An Autoregressive Generative Language Model For Genes.

Generative models have proven their success in different domains, most obviously with ChatGPT [5]. Theoretically, they can carry out almost all of the tasks done by a discriminative model, so they are superior in that sense. Especially combining different domains such as natural language and programming yielded incredible outcomes like OpenAI Codex [6], which can write code by only getting natural language as input. DALL-E 2 [7] is another nice example from a different field. One of its most exciting achievements is replacing an object in an image with another thing without changing the background, called in-painting. Similar work can be done for the combination of any domain and natural language. Although the current version of our study only focuses on genes, the same idea could be applied to whole DNA sequences, and a similar combination with natural language could provide huge benefits. To give concrete examples, we might show a mutated DNA sequence causing disease and prompt it to convert this DNA sequence into the healthy version. Another option could be giving a sequence carrying a specific property like being resistant as input and wanting the program to describe how to integrate that feature into another organism's DNA so that it has the same property as well. In fact, theoretical limits for what can be done may be very close to the limits of what genomics could do at all. When all these possibilities are taken into account, we had better think carefully about ethical considerations too.

In Chapter 2, we discuss some introductory concepts to familiarize readers with several related areas. In addition, we also mention some related works for each area. That is, we give some concrete and specific examples on related areas to better understand the current research direction. Then, in Chapter 3, we describe how the dataset we worked on was obtained and give various statistics about the dataset. After describing in Chapters 4 and 5 the different machine learning techniques and the evaluation metrics used from a more theoretical perspective, in Chapter 6 we explain in detail how the experiments were conducted and the results obtained. Finally, conclusions and future work are discussed in Chapters 7 and 8.

2. RELATED WORK

2.1. Artificial Intelligence

Understanding the world in which we live has been an interesting topic for humankind throughout history. It led to the formation of different sciences like mathematics, physics, biology, and others. In addition to understanding the laws of our universe, human beings put a lot of effort into inventing machines that will be useful in different areas by using the discovered rules, which can be called engineering and technology.

Although many period and location pairs can be called golden ages for science and technology, if we focus on the one that caused the last leap, which has been happening since Renaissance and Industrial Revolution, it is easy to see that lots of critical developments happened in fields requiring physical power coming from human beings. Despite this crucial point, developing machines that can carry out tasks requiring some form of intelligence started emerging less than a century ago with far less speed. Furthermore, our knowledge about understanding the concept of intelligence in humans and animals still needs to be improved, which might explain the failure of developing artificial intelligence.

The actual reason for not being able to reach artificial intelligence is not related to the lack of effort or financial resources because lots of time and money was actually spent on this topic after the invention of computers, which opened the door for proper research in the field of artificial intelligence. Developing human-like and general artificial intelligence, also called artificial general intelligence, would open many doors in almost all the areas of academia and industry, such as computers trying to discover drugs for illnesses. Nevertheless, the possible negative impacts of such a system had better not be forgotten since that side of the coin has enormous potential too.

When we think about ourselves from an intelligence point of view, some of the concepts coming to our minds are planning, decision-making, and reasoning. Computers were relatively successful in these areas under certain assumptions [8–10]. Yet, the difficulty came from tasks related to areas like vision and language, which are computer vision and natural language processing in computer science terms. Owing to this circumstance, each of these areas has been studied heavily by several researchers for a couple of decades. There are even systems trying to combine them [11–14].

2.2. Natural Language Processing

Language is an essential part of our lives as human beings. Consequently, having an adequate level of performance in language-related tasks was a necessity for developing human-like artificial intelligence.

If we think more deeply about the subject, it consists of two subtasks. The first is natural language understanding, which refers to grasping the message in a sentence or text. The second part is called natural language generation, which refers to generating meaningful and logical texts. In other words, natural language understanding is about understanding the input correctly, and natural language generation is about generating a correct output when we express the problem using the vocabulary of computer science.

Natural language understanding tasks are relatively more manageable than natural language generation, although it also depends on specific tasks chosen from each. However, children start to understand what is said before saying similar words could be an excellent explanatory example for giving some intuition. Still, both of them are vital for artificial intelligence research. Improvements in these fields contribute to the artificial general intelligence effort in the big picture and provide the infrastructure for many practical, real-world applications, such as Web search [15] and chatbots [16].

Finally, almost any field that has a language-like structure can benefit from natural language processing techniques, even if the field is called natural language processing. Various biological phenomena like protein and DNA sequences are nice examples.

2.3. Machine Learning

The primary strategy for artificial intelligence was based on the development of rules that explain structure in the domains studied, such as vision or language. In addition, some logical approaches were used, where the systems attempted to extract new information from already known information using the classical rules of logic. The Prolog language [17] is a nice example of this. Nevertheless, both rule-based and logical strategies failed to achieve the desired success. This was due to factors such as the inability to deal with uncertainty in many different areas of life.

An alternative method was machine learning, which allowed computers to learn from data related to the target task instead of explicitly telling the rules and what to do at each step. The concept of machine learning refers to a family of approaches having common properties. Still, currently, the most successful and popular one among them is a method called deep learning [18], which tries to map the raw features given in the problem directly to the desired outputs without requiring intermediate human intervention. Even though it is possible to make more detailed explanations about these topics, mentioned points look sufficient since this study's goal is not to survey these areas, and necessary matters will be discussed in Chapter 4.

2.4. Language Models

One of the most important concepts for natural language processing is language models, as they can be useful for various tasks. Simply put, a language model is a model that can make probabilistic predictions about two tasks. The first is the probability that a given word comes after a sequence of words, and the second is the measure of how likely it is that a sequence of consecutive words will occur as a sentence.

After the invention of a special kind of architecture in deep learning called transformers [1], which will be explained in more detail in Chapter 4, these language models became extremely popular not only for natural language understanding tasks [19] but also for natural language generation tasks [20].

If we focus more on natural language understanding tasks, it wouldn't be an exaggeration to say that at the moment, the default strategy for most of the text classification type problems, such as sentiment analysis, is choosing the right pre-trained large language model like BERT [2] and fine-tuning it on the target dataset.

Looking at the other side of the coin, some of the most influential and exciting language generation tools, such as GPT-3 [3] and ChatGPT [5], use the same building blocks and methodology but for natural language generation tasks instead of natural language understanding tasks. Because of all these points, they look very promising for future studies.

2.5. Generative Adversarial Networks for Natural Language Processing

Generative Adversarial Networks [21] is a viral deep learning-based generative model. Although they were used more commonly for vision-related tasks such as image generation [22–27] at the beginning, they can also be used for text generation [28]. Furthermore, using transformer architecture is not necessary for these models, so they ought to be remembered when the issue of natural language generation is discussed; however, they aren't as successful as they are in computer vision.

The underlying principle of how Generative Adversarial Networks work is fascinating. Initially, a dataset is collected consisting of samples from the target group to be generated. For example, pictures of ships. Then there are two different models, the generator and the discriminator, that compete against each other. While the former tries to generate results that are similar to the existing dataset, the latter tries to discriminate between the generated and the actual. Through competition, both get better and better, eventually leading to the generator producing realistic samples from the target domain.

2.6. Bioinformatics

The invention of computers hasn't affected only well-known areas such as digital sectors. It was very crucial for other fields like science in general as well. Due to the power of fast and reliable calculations they provided, computers were beneficial for lots of different scientific and technological processes.

Biology benefits from these machines with great potential. Various biological building blocks could be studied much more robustly and quickly with these solid computing machines [29, 30]. Such efforts are sometimes referred to as bioinformatics or computational biology.

Proteins are one of the most fundamental building blocks when it comes to biology. Both discriminative and generative models are used for these critical elements. While McDowall and Hunter [31] provide a tool for automatic classification of proteins using models such as hidden Markov models, Madani et al. [32] attempt to develop a generative model for proteins based on big data that can be used for various application domains like medicine.

Since proteins can have different 3D structures, working with them requires special care. In addition, there are studies that focus on the interactions between proteins and various chemicals, as these are of great importance for topics such as drug development. For example, Nagamine and Sakakibara [33] use Support Vector Machines, a conventional machine learning algorithm, to predict the probability of interaction for a given protein and chemical pair. On the other hand, Cobanoglu et al. [34] use probabilistic matrix factorization, which is an alternative method for a closely related task.

DNA and RNA sequences have a massive importance in our lives; consequently, applications related to them are prevalent. The area is not new, even though various facilities in the past were limited. For example, Wang et al. [35] devised two algorithms, one unsupervised, for classifying DNA. Nonetheless, Nguyen et al. [36] prefer using convolutional neural networks for classifying DNA even though convolutional neural networks are not commonly used for classifying sequence-like inputs. Furthermore, Qi et al. [37] use an unsupervised clustering method on RNA sequences in addition to supervised applications.

Some studies used classical artificial intelligence methods to solve bioinformatics problems without using machine learning. Altschul et al. [38] use such an approach to measure the similarity of two sequences, as this is a crucial problem for tasks like search. Burge and Karlin [39] use a probabilistic approach to solve tasks such as gene identification. Lowe and Eddy [40] use the same technique for RNA-related work with a very low error rate. Also, Trapnell et al. [41] develop a robust method for gene expression by taking advantage of RNA sequences. Despite the existence of such studies, the integration of machine learning techniques in one form or another is extremely popular nowadays.

Traditional machine learning methods are a serious option for tasks with limited data, as deep learning approaches, their main competitors, need more data to perform better. Rannala and Yang [42] pursue a statistical method for estimating different species using DNA. Choi et al. [43] investigate the success of the classical logistic regression algorithm in RNA sequence studies.

As in other fields, deep learning is inevitably very common in bioinformatics because it is very successful when enough data is available for a given task. Lan et al. [44] mention several use cases for this, which is expected since deep learning is extremely popular in this field. As an interesting example, Zeng et al. [45] deep learning to make predictions about RNA and protein pairs to better understand these structures.

Lastly, using language models is also a popular topic. For instance, DNABert [4] uses an approach similar to the classical BERT, where data comes from human DNA for training and various DNA sources for testing instead of natural language in order to solve various tasks related to genomics. While DNABert worked on datasets where samples were DNA sequences, our study was on a smaller scale since our samples were nucleotide sequences of genes.

In addition, Luo et al. [46] try to do a similar study for the generative part of the coin, yet for biomedical text instead of directly using biological units. However, a successful study which trained a gigantic generative transformer model on structures like DNA or proteins is not available, to the best of our knowledge.

Like natural language processing, biomedical domains try to use language models as much as possible due to their vast potential and proven success in several applications. This circumstance is also called biological language models. It is possible to find very different use cases, ranging from discriminative to generative when the subject is examined in depth [30, 47–51].

As a result, bioinformatics is a very hot field, and our study focuses directly on DNA instead of other molecules such as RNA or proteins or interactions between pairs. In addition, currently, popular machine learning-based approaches, especially deep learning, are employed in our work. Also, in spite of the fact that DNABert is one of the closely related studies to our interests, the task is selected as a generative problem, not a discriminative one, since the latter was studied more extensively and the former has, arguably, more importance. Lastly, our work is relatively small sized in terms of datasets and models owing to factors like limited resources. However, it is still work on an exciting and unexplored problem.

3. DATASET

The aim of preparing the dataset was to use the nucleotide sequences for genes existing in human beings. The National Center for Biotechnology Information (NCBI) [52], a government institution, provides different databases such as Genome, Gene, and Nucleotide. From the Gene Database, we filtered genes such that only the ones existing in human beings remained. Then we got information like the commonly used name, id used in the database, and nucleotide sequence for all these genes. Nucleotide sequences for a specific gene can be accessed using the Nucleotide database of NCBI, and the dataset we obtained is online [53]. Some general statistics about the dataset are given in Table 3.1 to provide a general overview of what kind of data we worked on.

Table 3.1. General statistics about dataset

	Whole Dataset
Number of Samples	70475
Min Value	2
Max Value	2473603
Mean Value	26417.25
Median Value	3566.0

Next, we had to filter out genes whose sequence length exceeded a certain threshold, since transformer-based models assume a certain maximum length. This threshold was set at 1000, since larger values led to problems in fitting the models to the memory of the computers. Then we used 80% of it for training and 20% of the dataset for testing purposes. The split was done randomly. Some general statistics on the datasets can be found in Table 3.2. The datasets are available online [53].

Table 3.2. General statistics about train and test datasets

	Actual Dataset	**Training Set**	**Test Set**
Number of Samples	23779	19023	4756
Min Value	2	2	17
Max Value	1000	1000	1000
Mean Value	358.57	358.00	360.85
Median Value	295.0	295.0	295.0

In addition to these operations, the use of a validation set for models requiring hyperparameter optimization was necessary. For this reason, the training set was further split into 75% and 25%, where the former corresponds to the training set without validation and the latter is the validation set. Table 3.3 also contains statistics for this split. Consequently, the split was 60%, 20%, and 20% for the training, validation, and test sets, respectively.

Table 3.3. General statistics about train and validation datasets

	Training Set	**Training Set (Split)**	**Validation Set**
Number of Samples	19023	14267	4756
Min Value	2	2	7
Max Value	1000	1000	999
Mean Value	358.00	358.49	356.54
Median Value	295.0	295.0	295.0

The nucleotide sequence was the essential information for each gene, yet other details such as NCBI gene id, symbol, description, and gene type information were also given. Lastly, < and > symbols were used to represent the beginning and end of the genes on the nucleotide sequence column of the corresponding files, and processing was done accordingly in the experiments.

4. METHODS

Language models have several uses, as they can generate new samples and measure the probability of a given sequence. They can do this by predicting two distinct but related points about languages. Firstly, they can predict $p(w_i|w_1^{i-1})$, which is the probability of the next word given previous words. Also, they can predict the probability of a sequence as a whole. This time it corresponds to $p(w_1^N)$. Nevertheless, these two problems are very closely related in the following way $p(w_1^N) = \prod_{i=1}^{N} p(w_i|w_1^{i-1})$. Thus, the solution of the first problem already enables the solution of the second problem. There are several strategies to have models that allow the computation of both objectives.

4.1. N-gram Language Models

The basic idea for these models is that looking at only the immediate past could give sufficiently good approximations, also called Markov property. Consequently, for calculating $p(w_i|w_1^{i-1})$, an N-gram model takes only previous (N-1) many words into account; that is, $p(w_i|w_1^{i-1}) = p(w_i|w_{i-N+1}^{i-1})$ is assumed. Depending on the N value chosen, special names like unigrams and bigrams are used for these models.

The simplest way to calculate the probabilities is using

$$p(w_i|w_{i-N+1}^{i-1}) = count(w_{i-N+1}^{i})/count(w_{i-N+1}^{i-1}) \tag{4.1}$$

on the given training corpus for calculating desired probabilities. Actually, this is a very reasonable idea, and it corresponds to Maximum Likelihood Estimation; in other words, the calculated values maximize the probability of sentences in the training corpus [54].

Lastly, it is significant to note that when N is greater than one, $< s >$ symbols are used at the beginning of sentences, and $< /s >$ symbols are used at the end of sentences in order to be consistent with the formula used.

4.1.1. Laplace Smoothing

Although N-gram language models are intuitive and pleasant, they have a serious drawback. When they encounter an unseen pattern, they predict its probability as zero, since they only check the count. This can lead to problems because representing all possible sequences in languages is impossible owing to the huge datasets required to do so. In practice, this problem results in sentences with even a single unseen pattern being assigned a probability of zero. Therefore, a simple solution to this problem is to increment all possible counts by one to avoid zero probabilities, which is called Laplace smoothing [55]. Of course, the probabilities obtained for cases with zero counts are still very low, but not zero.

4.2. Deep Learning

Deep learning is a branch in the field of machine learning that is one of the approaches proposed on the way to artificial intelligence. The main difference between machine learning and the other techniques used in artificial intelligence is that programs are meant to learn about a specific task based on data, rather than being explicitly told what to do at each step of execution. There are many points that make deep learning unique in the machine learning field. However, one of the most important is that it provides end-to-end solutions, meaning it only looks at the input and output pairs for the task at hand. It then attempts to develop a model that maps the given inputs to the outputs as best as possible. However, classical machine learning approaches use hand-designed features before the actual learning algorithm is used, which is a major burden when working with many different tasks, each with its own character.

An examination of what deep learning models do in terms of mathematics could be very useful in understanding them. The most basic idea is that most problems in the world can be represented as mathematical functions because most of these problems have inputs, such as an image composed of pixel values, a sentence composed of words, and a board game with a specific board configuration at each time step, all of which can be represented in one way or another with numbers. The specific goal of these tasks, such as classifying the given input into a category or choosing the best move in the game, can also be represented with numbers. These two arguments imply that all these problems are actually the problem of finding the right mathematical function for the input and output pairs.

However, there are a few critical points to be examined to clarify the explanation. First, we don't have access to all possible input and output pairs for most of the problems. Still, the critical assumption in deep learning is that we will have access to a sufficiently large subset of these actual pairs so that we can develop a model with a pretty good generalization capability; that is, although we don't see all possible cases for a specific task, we can still have an idea about the general structure in the problem. Based on this, we can solve the problem with relative success, even in unseen cases. Of course, using this approach for situations where mistakes cannot be tolerated because of reasons like the possibility of the loss of lives is not a good idea. However, it is possible to find many different tasks in different application areas, such as computer vision, natural language processing, and robotics, where using this approach is meaningful.

Another important problem is that an infinite number of functions can be chosen as solutions. So the crucial question is how to choose an appropriate function. The proposed solution to this problem is to use a loss calculation mechanism with the existing dataset to measure the performance of a particular function, and then apply optimization techniques to the function and loss to achieve a desired function.

Finding which function to use is a challenging task. Consequently, a friendly approach combines some simple mathematical operations to obtain more complex functions if needed. The mathematical functions used in deep learning consist of consecutive substructures called layers. Each layer is a mathematical operation, but they are called layers since deep learning also tries to make an analogy to the human brain. This explains why the mathematical functions in deep learning are called artificial neural networks. Although there are some similarities between deep learning methods and how the human brain works, it isn't very reliable to assume a correspondence between these two concepts beyond a certain extent.

When different kinds of possible layers are inspected, it isn't wrong to say that the number of possible layers is infinite. Despite having an unlimited option for layers, only about a dozen of them are actually used in real life.

The most popular layer is called the fully connected layer. This corresponds to matrix multiplication, so it's a linear operation.

Another critical layer type needed is non-linear operations. The reason is that combining multiple linear layers is not different from using a single linear layer in terms of expressiveness. Because of this, some non-linear operations are element-wise used in particular neural network layers. Some famous examples are

$$ReLU(x) = max(0, x) \tag{4.2}$$

$$Sigmoid(x) = 1/(1 + e^{-x}) \tag{4.3}$$

$$Tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \tag{4.4}$$

These functions are also called activation functions. ReLU is especially common because its derivative can be easily calculated [56].

While some layers have learnable parameters like the matrix to be used in multiplication in the fully connected case, some do not such as activation functions.

For loss functions, different options are available. Still, only a few of them are used in practice since most of the real-world problems are either classification or regression, whose aim is finding a real-valued number. For classification problems, the default structure choice uses a layer called softmax, which takes predictions for each class from the previous layer, applies exponentiation operation, and normalizes the results for prediction so that they follow the rules of probability. When calculating loss, however, the negative of the natural logarithm of the prediction for the actual class is used.

On the optimization part, gradient-based approaches like gradient descent are used. The basic recipe for the gradient descent algorithm is, at any point, calculating the gradient of the loss function with respect to learnable parameters in the artificial neural network and then changing the values of learnable parameters in the exact opposite direction of the gradient according to a predetermined step size. Furthermore, a dynamic programming-based algorithm called Backpropagation [57] is used to calculate the gradient more efficiently in terms of calculation. In addition to the classical gradient descent technique, a modified version called Adam optimizer [58], which adjusts step size dynamically and considers the trend existing in the previous steps, is also popular.

Lastly, there are some design decisions about the model. These are called hyperparameters. Some examples are the number of layers to use and how to initialize the learnable parameters at the beginning of the training. The primary solution for this circumstance is separating a portion of the training set as a validation set. Training several models using different possible hyperparameter values and choosing the best one according to the performance on the unseen data / validation set since the actual property wanted is generalization.

4.2.1. Recurrent Neural Networks

When the task of language modeling is concerned, where predicting the next word given the previous sequence is essential, building a deep learning model with the components described is possible. Nevertheless, they don't provide the flexibility to work with sequences with arbitrary lengths; a specific sequence length must be assumed so that components can be chosen accordingly.

A nice solution to this circumstance is using a model called recurrent neural networks, which are like neural networks containing a loop. The explanation is instead of having a flow which can be described as feedforward where we apply operations of each layer consecutively, recurrent neural networks use the same block to all the elements of a sequence one by one to update information kept in the block, which functions like memory, where the building block contains not only linear operations but also non-linear activations.

In addition to sounding intuitive, it also decreases the number of parameters used in the model, but more computation is necessary for calculating the gradient. Moreover, it has the downside of forgetting distant information quickly. An improved version of this classical recurrent neural network is proposed to solve the problem. Long Short-Term Memory [59] is also a type of recurrent neural network, yet it divides the initial memory inside the block into two. While one of them is capable of keeping track of distant history by being able to forget unnecessary information, the other one focuses more on recent history. Also, recurrent neural networks can be combined with different model types of deep learning.

4.2.2. Transformer Model

Despite the fact that Long Short-Term Memory (LSTM) is a very powerful model, it has two crucial downsides. The first of them is that it takes a long time to train because of the recurrent nature of it. Moreover, it doesn't have the capability to remember the past at the desired level although it is far better than vanilla recurrent neural networks.

Transformer model was proposed to solve both of these problems. For each word in a sequence, it describes the word as a combination of the representations of the previous words and itself. The critical point is that it uses a mechanism called attention which looks at how vital a previous word is concerning the current word instead of looking at how close they are in the sentence. It is also able to encode the position information. Using this approach eliminates recurrence problems because all of the computation for each word is independent of each other in terms of algorithm flow.

Also, they can capture strong relationships between distant word pairs.

Similar to other structures used in deep learning, blocks used in this approach also contain non-linear elements; because of this, it has sufficient expressive power in that sense. Even though the strategy is relatively new compared to its rivals such as recurrent neural networks, it has proven its success in different studies such as BERT and GPT-3.

5. EVALUATION METRICS

Evaluating the success of different techniques is essential for both comparing different methods and measuring progress. Perplexity is one of the most commonly used metrics for language models [60].

5.1. Perplexity

When a language model is available for a certain task, one of the most common ways to measure its performance is using a dataset (different from the training set) from the target domain in order to see how probable the dataset looks from the language model's point of view.

Different from most metrics used in machine learning such as accuracy, the lower the value, the better the model in the case of Perplexity since the probability of sequence is in the denominator. The exact formula for Perplexity is

$$\sqrt[N]{\frac{1}{p(w_1^N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} p(w_i|w_1^{i-1})}} \tag{5.1}$$

where N is the length of the sequence. Taking the Nth root is like normalization since it would favor shorter sequences otherwise due to the fact that all probabilities are multiplied.

The theoretical limit for the lower bound is one since the probabilities used are, at most, 1. On the other hand, this metric measures the number of choices at each step for the next word prediction on average for the model. Usually, this metric is calculated on the whole test set to have an idea about overall performance, which is the case in our uses.

5.2. Real Life Tasks

Evaluating the success of the methods is not trivial, so a real-life task could be very useful. For this reason, we tried to find a real-life problem where modeling nucleotide sequences of human genes could be helpful. Distinguishing between real nucleotide sequences of a real gene and a mutated version seemed like an exciting task, since the ability to do such a task well requires some level of understanding of the data we were working on.

Then the strategy was to obtain pairs such that each pair consists of the nucleotide sequence of an actual human gene and its mutated version. Then obtained models would assign a probability or perplexity to both of them and predict which one is real and which one is mutated based on these values. If it uses probability, the one with a higher predicted probability would be selected as the real one. If the model uses perplexity, the gene with a lower value will be chosen as the real one since the lower, the better for the perplexity metric. Then the accuracy for predictions made will be checked to see the success of a certain model. After deciding on this strategy, the only part that needed to be obtained was obtaining a dataset of real and mutated gene pairs.

5.2.1. Synthetic Mutation Dataset

One way to obtain a dataset consisting of real and mutated pairs is basically generating the mutated samples from the normal ones with the help of a computer by using a random process.

The method followed was like the following. Firstly, some hyperparameters such as the number of samples to obtain for the dataset and the maximum number of changes to be applied to a selected real gene, are determined. We selected the first as 1000. For the second, we experimented with three different values, which are 1, 5, and 10 to see how the performance changes according to this parameter. Lastly, since the process contains random number generation, a random seed was used to obtain reproducible results. For that part, we, again, used three different seeds to be able to use the general behavior.

After determining hyperparameters, for each selected sample from the test set, the number of changes to apply was selected, which is between 1 and the maximum number of changes (both inclusive) with equal probability. Then for each change, an option among four different candidates was chosen with equal probability. These options were adding a new randomly chosen nucleotide to a random position, deleting an arbitrary position in the sequence, changing the nucleotide on a randomly selected position to another one, and swapping randomly chosen two adjacent nucleotides which do not have the same type. The described operations are visualized in Figure 5.1 and the mentioned algorithm is also described in Figure 5.2.

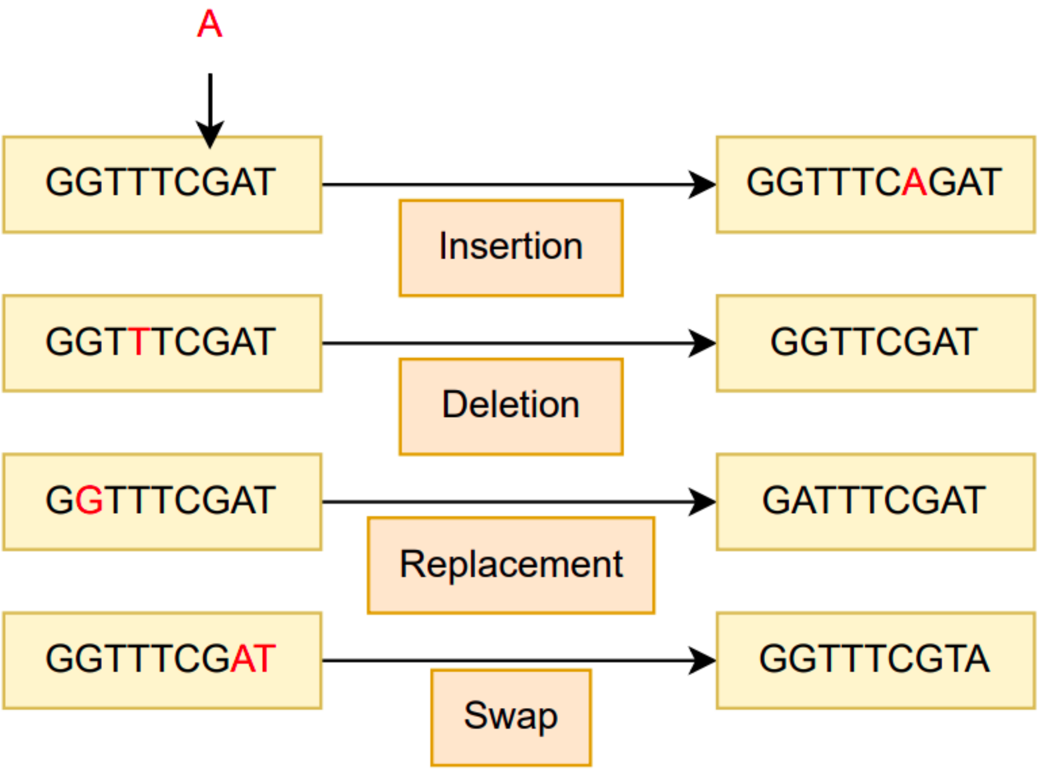
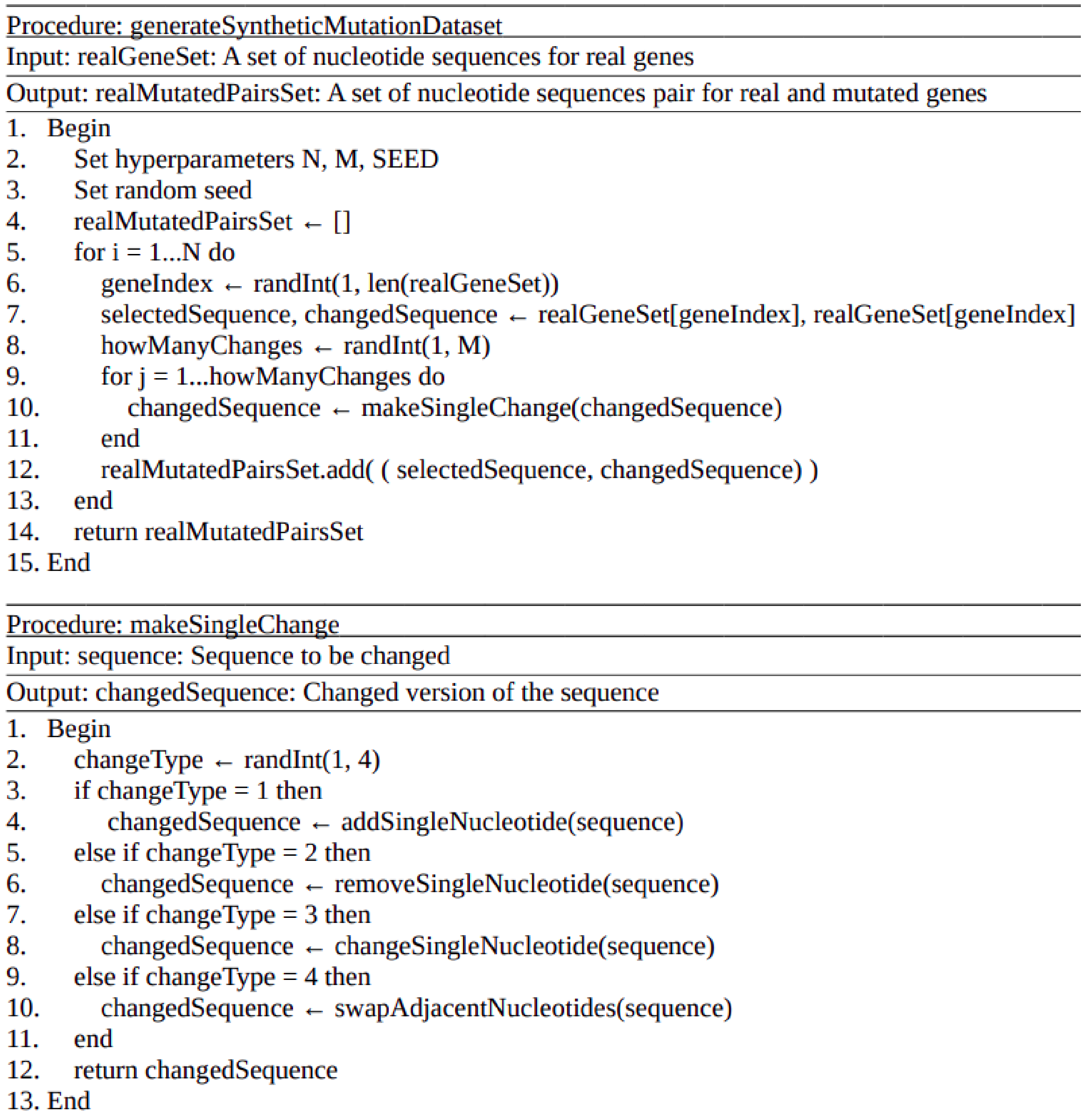


Figure 5.1. Synthetic Mutation Dataset Operation Types.



```
Procedure: generateSyntheticMutationDataset
Input: realGeneSet: A set of nucleotide sequences for real genes
Output: realMutatedPairsSet: A set of nucleotide sequences pair for real and mutated genes
Begin
    Set hyperparameters N, M, SEED
    Set random seed
    realMutatedPairsSet ← []
    for i = 1...N do
        geneIndex ← randInt(1, len(realGeneSet))
        selectedSequence, changedSequence ← realGeneSet[geneIndex], realGeneSet[geneIndex]
        howManyChanges ← randInt(1, M)
        for j = 1...howManyChanges do
            changedSequence ← makeSingleChange(changedSequence)
        end
        realMutatedPairsSet.add( ( selectedSequence, changedSequence) )
    end
    return realMutatedPairsSet
End
```

```
Procedure: makeSingleChange
Input: sequence: Sequence to be changed
Output: changedSequence: Changed version of the sequence
Begin
    changeType ← randInt(1, 4)
    if changeType = 1 then
        changedSequence ← addSingleNucleotide(sequence)
    else if changeType = 2 then
        changedSequence ← removeSingleNucleotide(sequence)
    else if changeType = 3 then
        changedSequence ← changeSingleNucleotide(sequence)
    else if changeType = 4 then
        changedSequence ← swapAdjacentNucleotides(sequence)
    end
    return changedSequence
End
```

Figure 5.2. Procedure for Generating a Synthetic Mutation Dataset.

As stated, this procedure was carried out for all three different values of the maximum number of mutations and three different seeds, leading to nine different synthetic mutation datasets. The corresponding datasets can be found in the related section of the implementation code, which is online [61].

5.2.2. Real Mutation Dataset

Although synthetic datasets are beneficial for measuring the success of different alternatives, checking the performance on the real data is also very important. Consequently, finding mutations for genes in our test set was needed.

Human Gene Mutation Database [62–64] was a nice resource with almost no alternative for our goal. Even though the number of mutations we could find for genes in our test set was only 10, which is very limited, it still helped to show that our models could make nice predictions for real examples as well. Since the database is not publicly available, we will, unfortunately, not be able to share the obtained mutation dataset.

6. EXPERIMENTS AND RESULTS

The code used for implementing the ideas described in the thesis is reproducible and available online [61] for all different techniques used.

6.1. N-gram Language Models

For N-gram language models [65], we experimented with six different values of N, starting from one and going up to six. The perplexity values obtained are given in Table 6.1. Moreover, NLTK library [66] was helpful for both using N-grams and calculating perplexity.

Table 6.1. Results for different n values

N Value	**Perplexity**
1	3.9971
2	3.9298
3	3.9073
4	3.8920
5	3.8787
6	∞

6.1.1. N-gram Language Models with Laplace Smoothing

Getting the perplexity of infinity when N is selected as six was an interesting result since the training set contained thousands of genes having thousands of nucleotides, so encountering an unseen sequence with length six did not look normal. Nevertheless, when the cause of the situation is inspected, it is noticed that the problem was related to understanding the beginning or end of the gene, which is a crucial component for modeling.

Then Laplace smoothing is used to avoid this problem. Different N values are tried starting from six and going up to eight. The corresponding perplexity values obtained are given in Table 6.2

Table 6.2. Laplace smoothing results for different n values

N Value	**Perplexity**
6	3.8652
7	3.8365
8	3.8121

The best model we obtained had about 66,000 parameters.

6.2. Long Short-Term Memory Based Language Model

After experimenting with N-gram language models, we continued with deep learning-based approaches. The first option was using Long Short-Term Memory based models. For getting input, a layer for learning embedding and position information was used at the beginning. Then a single or multiple Long Short-Term Memory layers are used depending on hyperparameter configuration. Lastly, the aim was predicting the next word; because of this, a linear layer with the size of vocabulary many units are used with Softmax to get predictions. The model architecture is visualized in Figure 6.1. Adam was chosen as the optimizer to use.

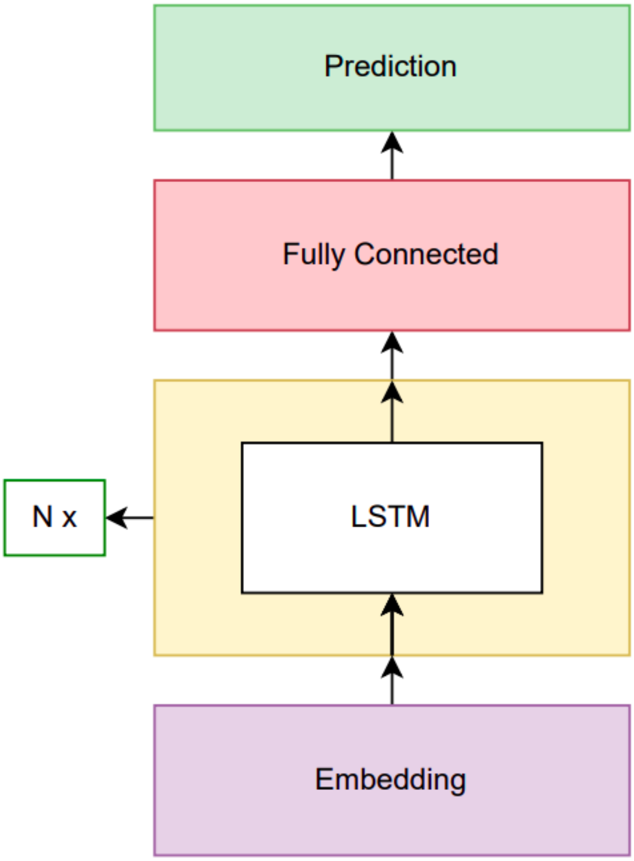


Figure 6.1. Long Short-Term Memory Based Architecture.

Hyperparameter optimization was a crucial component. The basic procedure followed is like the following. We started with a hyperparameter configuration which looked reasonable. After that, if the model was underfitting, relevant hyperparameters are changed so that the model's capacity is increased. When overfitting was seen, the opposite is done. More details about the procedure can be found in the reproducible and online code [61].

There were three main hyperparameters to play with. Embedding dimension, the number of LSTM layers to use and LSTM dimension. For any of these hyperparameter values, the model was run for 40 epochs. Then the result where the absolute difference between training and validation perplexities was less than 0.02 was chosen as the result of that hyperparameter trial.

The initial choice was EMBED_DIM = 256, LSTM_DIM = 256, NUM_LAYERS = 2 (trial 1). Then NUM_LAYERS = 1 (trial 2), NUM_LAYERS = 3 (trial 3) and NUM_LAYERS = 4 (6) were tried to see the effect of NUM_LAYERS. After that, EMBED_DIM = 128 (trial 4), LSTM_DIM = 128 (trial 5), EMBED_DIM = 512 (trial 7) and LSTM_DIM = 512 (trial 8) were tried to see the effect of them.

As a result, the model obtained from trial 3 was chosen since it was the best. It is important to note that even though trial 8 improved the obtained perplexity, results obtained during the training process were very unstable; in other words, the metrics for consecutive epochs fluctuated a lot, so it was not perceived as an improvement. The accepted model can be found in the implementation repository [61]. All the results are given in Table 6.3.

Table 6.3. Long short-term memory hyperparameter experiment results

Trial Number	**Validation Set Perplexity**
1	3.4273
2	3.5816
3	**3.3880**
4	3.4449
5	3.4446
6	3.4131
7	3.4712
8	3.3934 (unstable)

When the whole process was completed, the most promising model was obtained. Lastly, Tensorflow library [67], its high-level API Keras [68] and its NLP tool KerasNLP [69] were very beneficial for implementing stuff related to this section. The corresponding perplexity values obtained are given in Table 6.4

Table 6.4. Long short-term memory result

N Value	Validation Set Perplexity	Test Set Perplexity
1024	3.3880	3.3935

The best model we obtained had 1,841,928 parameters.

6.3. Transformer Based Language Model

After using Long Short-Term Memory based models, transformer-based models were the next candidate. Actually, most of the procedures were very similar to the previous case. For getting input, a layer for learning embedding and position information was used at the beginning. Then a single or multiple transformer blocks are used depending on the hyperparameter configuration. Lastly, the aim was predicting the next word; because of this, a linear layer with the vocabulary size many units are used with Softmax to get predictions. The model architecture is visualized in Figure 6.2. Adam was chosen as the optimizer to use.

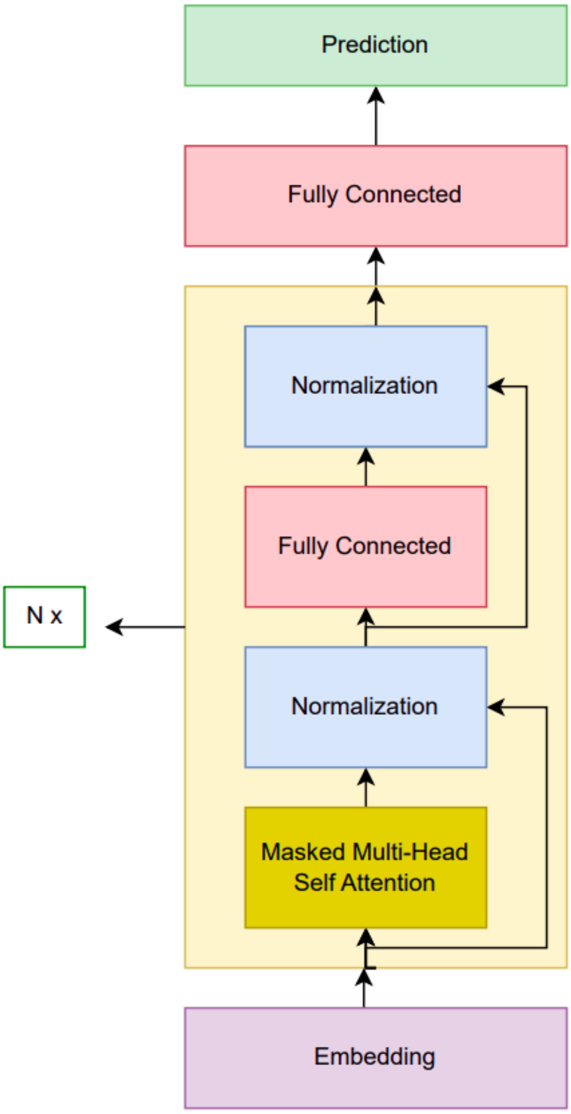


Figure 6.2. Transformer Architecture.

Hyperparameter optimization was a crucial component. The basic procedure followed is like the following. We started with a hyperparameter configuration which looked reasonable. After that, if the model was underfitting, relevant hyperparameters are changed so that the model's capacity is increased. When overfitting was seen, the opposite is done. More details about the procedure can be found in the reproducible and online code [61].

There were four main hyperparameters to play with. Embedding dimension, the number of transformer blocks to use transformer feed forward dimension and number of heads to use for a certain transformer block. The model was run for 40 epochs for any of these hyperparameter values. Then the result where the absolute difference between training and validation perplexities was less than 0.02 was chosen as the result of that hyperparameter trial.

The initial choice was EMBED_DIM = 256, FF_DIM = 256, NUM_LAYERS = 2, NUM_HEADS = 3 (trial 1). The performance was low compared to Long Short-Term Memory, so NUM_LAYERS = 1 or NUM_HEADS = 2 was not tried since they would further decrease the model's capacity. NUM_LAYERS = 3 (trial 2), NUM_HEADS = 4 (trial 3), EMBED_DIM = 512 (trial 4), FF_DIM = 512 (trial 5), NUM_LAYERS = 4 (trial 6), EMBED_DIM = 1024 (trial 7), FF_DIM = 1024 (trial 8) and NUM_LAYERS = 5 (trial 9) were tried to see the effects of these hyperparameters.

As a result, the model obtained from trial 9 was chosen since it was the best. It is important to note the even though combining trials 4, 5 and 9 looks like the best option, this was causing the model to be too big to fit into memories of the machine used; because of this, the best option among them were chosen. The obtained model can be found in the implementation repository [61]. All the results are given in Table 6.5.

Table 6.5. Transformer hyperparameter experiment results

Trial Number	**Validation Set Perplexity**
1	3.6601
2	3.6472
3	3.6614
4	3.6562
5	3.6438
6	3.6377
7	3.6656
8	3.6454
9	**3.6331**

When the whole process was completed, the most promising model was obtained. Lastly, Tensorflow library, its high-level API Keras and its NLP tool KerasNLP were very beneficial for implementing stuff related to this section. The corresponding perplexity values obtained are given in Table 6.6. The best model we obtained had 2,239,737 parameters.

Table 6.6. Transformer result

N Value	**Validation Set Perplexity**	**Test Set Perplexity**
1024	3.6331	3.6361

Lastly, the results obtained for different models based on test set perplexity is visualized in Figure 6.3 using Matplotlib library [70].

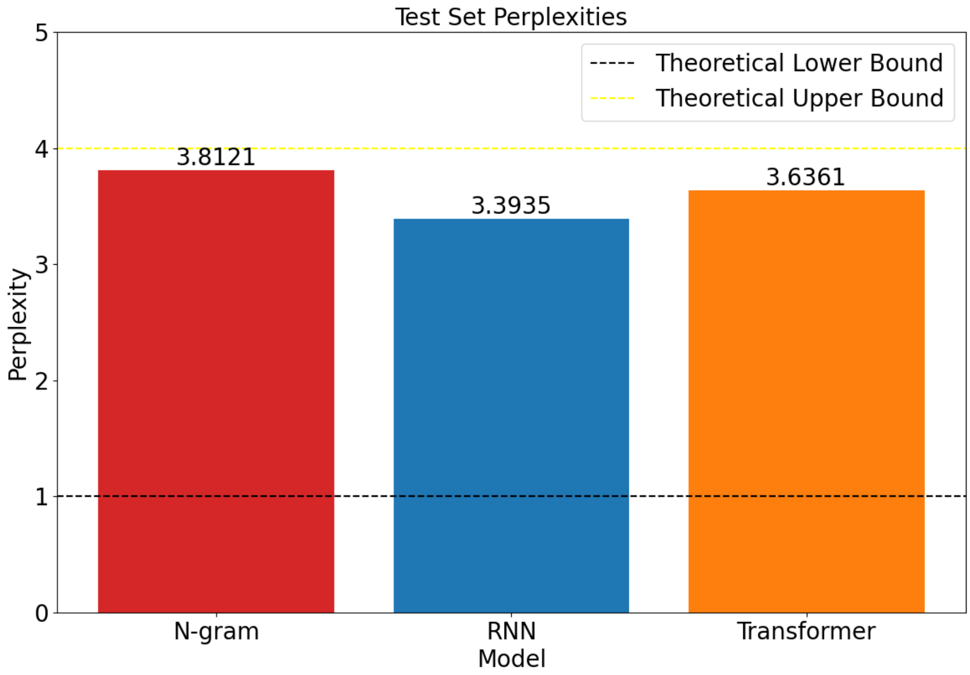


Figure 6.3. Comparison of Different Methods based on Test Set Perplexity.

As a small note, after we completed almost all of our experiments, we noticed that about 5% of the genes in the test set have a nucleotide sequence that actually appears in one of the genes in the training set although they have a different gene id. Nonetheless, because we used test set only for comparing different strategies and not focusing on the number obtained, we thought that it wouldn't make a serious difference due to the fact that they are still competing under the same conditions.

6.4. Performances On Real Life Tasks

6.4.1. Synthetic Mutation Dataset

For synthetically obtained nine different datasets, which were described in Chapter 5, containing real and mutated gene pairs, the best models from N-gram with Laplace smoothing, recurrent neural networks and transformer-based language models were chosen. Their predictions were taken for each couple and the accuracy was calculated for each of them. The results obtained based on perplexity for three different random seeds, which were 623598, 638453 and 419432 in the code [61] are given in Table 6.7 and results obtained according to probability given in Table 6.8.

Table 6.7. Synthetic mutation dataset result based on perplexity predictions

Model	**Max Change: 1**	**Max Change: 5**	**Max Change: 10**
N-gram	(0.685, 0.707, 0.652)	(0.763, 0.736, 0.758)	(0.806, 0.811, 0.813)
RNN	(0.748, 0.724, 0.698)	(0.791, 0.800, 0.803)	(0.850, 0.849, 0.851)
Transformer	(0.668, 0.661, 0.651)	(0.725, 0.714, 0.733)	(0.766, 0.757, 0.776)

Table 6.8. Synthetic mutation dataset result based on probability predictions

Model	**Max Change: 1**	**Max Change: 5**	**Max Change: 10**
N-gram	(0.681, 0.651, 0.624)	(0.728, 0.706, 0.731)	(0.770, 0.764, 0.773)
RNN	(0.720, 0.668, 0.667)	(0.762, 0.778, 0.776)	(0.808, 0.816, 0.790)
Transformer	(0.648, 0.639, 0.637)	(0.704, 0.661, 0.704)	(0.711, 0.734, 0.700)

The results obtained for different models based on synthetic mutation dataset perplexity is visualized in Figure 6.4 using the Matplotlib library.

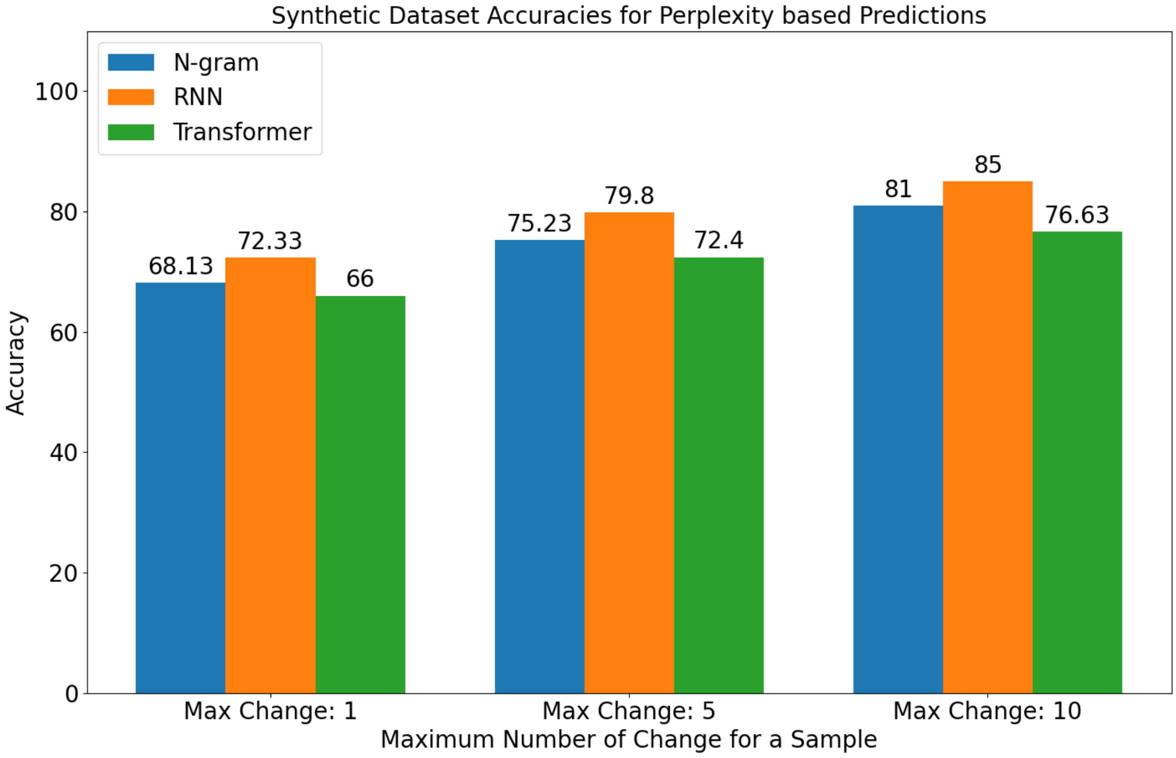


Figure 6.4. Averaged Accuracy Results Using Perplexity for Predictions.

Finally, the results obtained for different models based on synthetic mutation dataset probability is visualized in Figure 6.5 using the Matplotlib library.

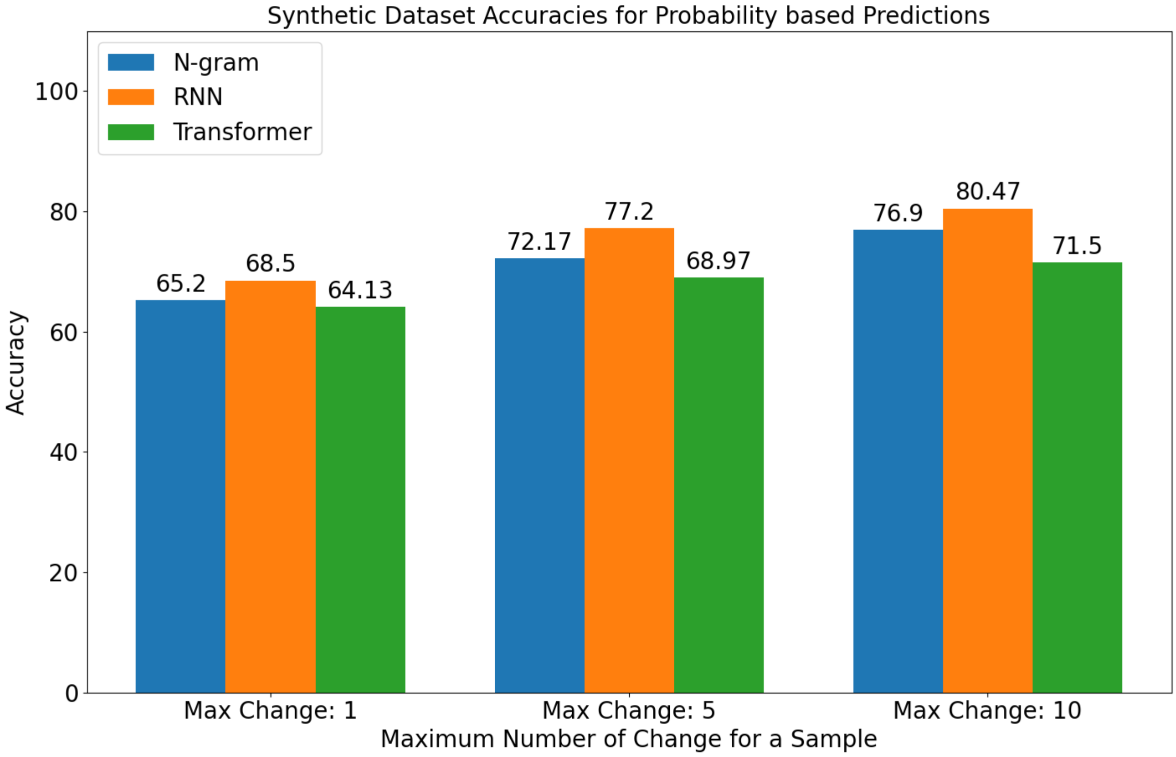


Figure 6.5. Averaged Accuracy Results Using Probability for Predictions.

6.4.2. Real Mutation Dataset

The procedure for real mutation dataset was very similar to the synthetic dataset and results obtained based on perplexity values are given in Table 6.9 and visualized in Figure 6.6 using Matplotlib library. In addition, results obtained based on probability values are given in Table 6.10 and visualized in Figure 6.7 using the Matplotlib library.

Table 6.9. Real mutation dataset results based on perplexity predictions

Model	Accuracy
N-gram	60%
RNN	90%
Transformer	60%

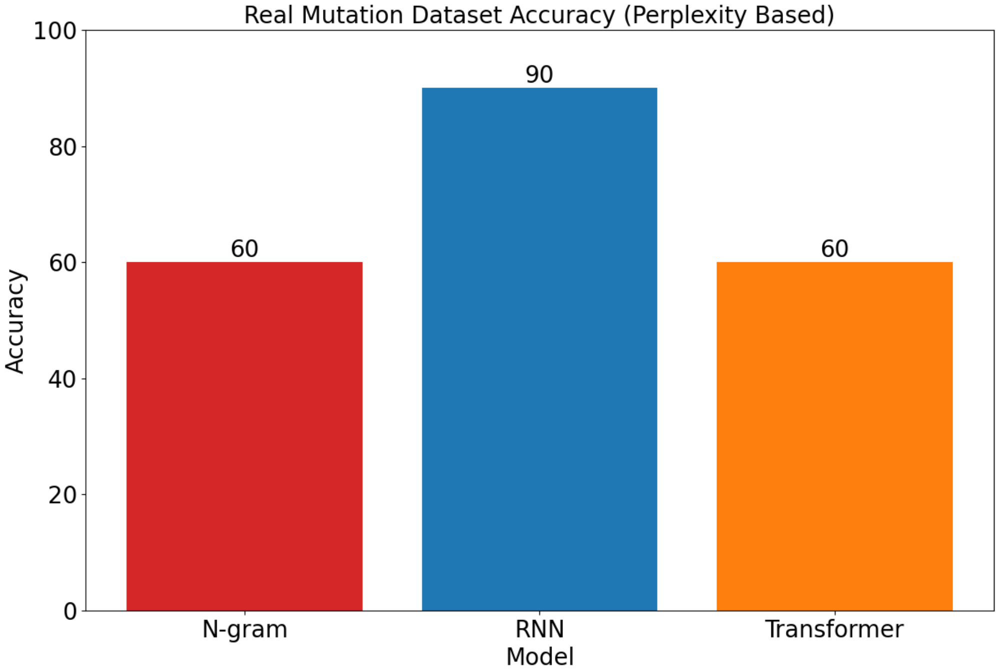


Figure 6.6. Accuracy Results Using Perplexity for Real Mutation Predictions.

Table 6.10. Real mutation dataset results based on probability predictions

Model	**Accuracy**
N-gram	60%
RNN	90%
Transformer	60%

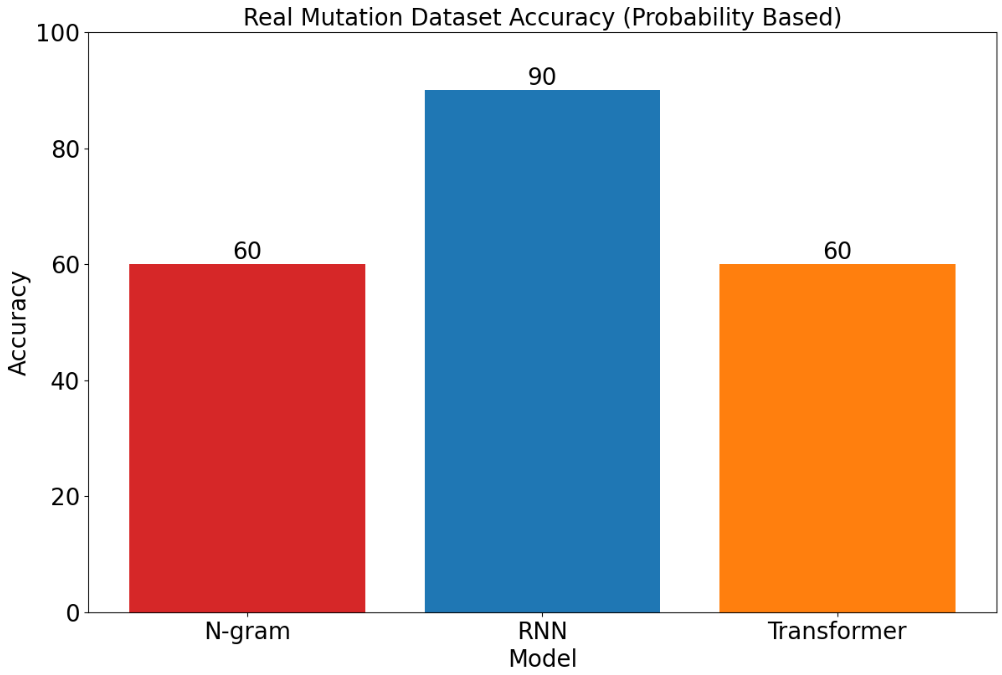


Figure 6.7. Accuracy Results Using Probability for Real Mutation Predictions.

More specific information about the exact mutations for used real genes is given in Table 6.11. Moreover, details about which model made a mistake on which example is shown in Table 6.12. More specific information about this part of the study, such as where the best model made a mistake, can be seen in Table 6.12.

Table 6.11. Mutation information for genes in the real mutation dataset

Gene Number	**Mutation Information**
1	1 Substitution
2	1 Substitution
3	1 Insertion
4	1 Insertion
5	1 Substitution
6	1 Substitution
7	1 Substitution
8	1 Substitution
9	1 Substitution
10	10 Deletions

Table 6.12. Correctness of perplexity based predictions for different models

Gene Number	**Mutation Information**	**N-gram**	**RNN**	**Transformer**
1	1 Substitution	False	True	False
2	1 Substitution	True	True	True
3	1 Insertion	True	True	False
4	1 Insertion	False	False	True
5	1 Substitution	True	True	True
6	1 Substitution	True	True	True
7	1 Substitution	False	True	False
8	1 Substitution	False	True	True
9	1 Substitution	True	True	False
10	10 Deletions	True	True	True

7. CONCLUSION

In this study, we used different natural language processing-based language modeling techniques to model nucleotide sequences of human genes. For big datasets, expecting better performance from deep learning-based methods, primarily transformer-based language models were standard; however, the problem we focused on was inherently the opposite. In other words, the actual genes existing in nature are limited. However, the language in nucleotide sequences looked relatively simple because there were only four words excluding the beginning and end of genes. Consequently, it was interesting to examine whether or not this phenomenon will reduce the gap for deep learning-based models.

When the results obtained are examined, it is clear that simple models such as N-gram language models performed better than their transformer-based competitors and performed really close to recurrent neural network competitors. Also, it was seen that classical metrics used to measure success were not perfect in the sense that N-gram models performed better than transformer-based models in real-world tasks, although they looked worse when the perplexity metric was used for the comparison. Moreover, because deep learning-based models had access to the same or more information about previous elements in the sequence at each prediction, neither recurrent neural networks nor transformer-based language models could achieve the desired level of improvement over classical N-gram models.

In addition, obtained N-gram language models are superior to deep learning approaches regarding space complexity due to the fact that their model sizes are far smaller. When it comes to time complexity, the situation is again similar since deep learning methods require hyperparameter tuning, whereas N-gram language models require only a single training. When the results obtained are comprehensively taken into account, deep learning-based language models do not look more appropriate for the task than simple N-grams, and it seems like the data-hungry nature of deep learning models is still valid even for a case where we work on a language where the vocabulary size is minimal.

8. FUTURE WORK

There is room for improvement in most scientific work. To further improve our work, some factors such as expecting developments from different areas of science, being able to access more powerful computational resources over time, and encouraging other people to work on similar problems by emphasizing possible outcomes of successful works on the studied problem ought to be remembered.

Firstly, genomics is a relatively new discipline compared to other branches of science like mathematics. Because of this, it is very normal to expect that our understanding will get better over time, which means that conducting a similar study in the future could give far more significant insights. Also, a greater theoretical lower bound could be determined for the problem at hand over time. Coming up with such a reliable bound with our current knowledge was not easy for us.

Second of all, more detailed hyperparameter optimization strategies could give more intuition about why deep learning methods are not that successful. Of course, using powerful GPUs is essential for this, yet it could still be helpful for further studies.

Furthermore, expanding the dataset by using nucleotide sequences of genes existing in animals, plants and others could be an interesting work as well. Being able to find them might not be as simple as it was in this study and it requires huge computation resources such as GPUs, but the outcomes of the study could be far more general and conclusive. In addition, further cleaning of the dataset by removing repetitive nucleotide sequences could make drawn conclusions more excellent.

Lastly, finding out more real-life problems strongly related to modeling nucleotide sequences of human genes could increase the scientific community's attention dramatically, which increases the amount of effort and investment put in to develop better strategies to model these incredible structures existing in nature, including our bodies.

REFERENCES

1. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention is all you need”, *Advances in neural information processing systems*, Vol. 30, 2017.

2. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, ArXiv:1810.04805 [cs], 2018.

3. Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners”, *Advances in neural information processing systems*, Vol. 33, pp. 1877–1901, 2020.

4. Ji, Y., Z. Zhou, H. Liu and R. V. Davuluri, “DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome”, *Bioinformatics*, Vol. 37, No. 15, pp. 2112–2120, 2021.

5. OpenAI, “Introducing ChatGPT”, 2022, `https://openai.com/blog/chatgpt`, accessed on March 8, 2023.

6. Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, “Evaluating large language models trained on code”, , 2021, arXiv:2107.03374 [cs], 2021.

7. Ramesh, A., P. Dhariwal, A. Nichol, C. Chu and M. Chen, “Hierarchical text-conditional image generation with clip latentsg”, , 2022, arXiv:2204.06125 [cs], 2022.

8. Campbell, M., A. J. Hoane Jr and F.-h. Hsu, “Deep blue”, *Artificial intelligence*, Vol. 134, No. 1-2, pp. 57–83, 2002.

9. Buchanan, B. G. and R. G. Smith, "Fundamentals of expert systems", *Annual review of computer science*, Vol. 3, No. 1, pp. 23–58, 1988.

10. Kuipers, B., E. A. Feigenbaum, P. E. Hart and N. J. Nilsson, "Shakey: from conception to history", *Ai Magazine*, Vol. 38, No. 1, pp. 88–103, 2017.

11. OpenAI, "GPT-4", 2023, `https://openai.com/research/gpt-4`, accessed on April 5, 2023.

12. Ramesh, A., M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen and I. Sutskever, "Zero-shot text-to-image generation", *International Conference on Machine Learning*, pp. 8821–8831, PMLR, 2021.

13. Mokady, R., A. Hertz and A. H. Bermano, "Clipcap: Clip prefix for image captioning", ArXiv:2111.09734 [cs], 2021.

14. Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision", *International conference on machine learning*, pp. 8748–8763, PMLR, 2021.

15. Hirschberg, J. and C. D. Manning, "Advances in natural language processing", *Science*, Vol. 349, No. 6245, pp. 261–266, 2015.

16. Weizenbaum, J., "ELIZA—a computer program for the study of natural language communication between man and machine", *Communications of the ACM*, Vol. 9, No. 1, pp. 36–45, 1966.

17. Wielemaker, J., T. Schrijvers, M. Triska and T. Lager, "Swi-prolog", *Theory and Practice of Logic Programming*, Vol. 12, No. 1-2, pp. 67–96, 2012.

18. LeCun, Y., Y. Bengio and G. Hinton, "Deep learning", *nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

19. Min, B., H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heinz and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey", ArXiv:2111.01243 [cs], 2021.

20. Li, J., T. Tang, W. X. Zhao and J.-R. Wen, "Pretrained language models for text generation: A survey", ArXiv:2105.10311 [cs], 2021.

21. Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial networks", *Communications of the ACM*, Vol. 63, No. 11, pp. 139–144, 2020.

22. Harshvardhan, G., M. K. Gourisaria, M. Pandey and S. S. Rautaray, "A comprehensive survey and analysis of generative models in machine learning", *Computer Science Review*, Vol. 38, p. 100285, 2020.

23. Arjovsky, M., S. Chintala and L. Bottou, "Wasserstein generative adversarial networks", *International conference on machine learning*, pp. 214–223, PMLR, 2017.

24. Radford, A., L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", ArXiv:1511.06434 [cs], 2015.

25. Zhu, J.-Y., T. Park, P. Isola and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks", *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

26. Karras, T., S. Laine and T. Aila, "A style-based generator architecture for generative adversarial networks", *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

27. Isola, P., J.-Y. Zhu, T. Zhou and A. A. Efros, "Image-to-image translation with conditional adversarial networks", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

28. Iqbal, T. and S. Qureshi, “The survey: Text generation models in deep learning”, *Journal of King Saud University-Computer and Information Sciences*, Vol. 34, No. 6, pp. 2515–2528, 2022.

29. Yang, A., W. Zhang, J. Wang, K. Yang, Y. Han and L. Zhang, “Review on the application of machine learning algorithms in the sequence data mining of DNA”, *Frontiers in Bioengineering and Biotechnology*, Vol. 8, p. 1032, 2020.

30. Li, H.-L., Y.-H. Pang and B. Liu, “BioSeq-BLM: a platform for analyzing DNA, RNA and protein sequences based on biological language models”, *Nucleic acids research*, Vol. 49, No. 22, pp. e129–e129, 2021.

31. McDowall, J. and S. Hunter, “InterPro protein classification”, *Bioinformatics for Comparative Proteomics*, pp. 37–47, 2011.

32. Madani, A., B. McCann, N. Naik, N. S. Keskar, N. Anand, R. R. Eguchi, P.-S. Huang and R. Socher, “Progen: Language modeling for protein generation”, ArXiv:2004.03497 [q-bio.BM], 2020.

33. Nagamine, N. and Y. Sakakibara, “Statistical prediction of protein–chemical interactions based on chemical structure and mass spectrometry data”, *Bioinformatics*, Vol. 23, No. 15, pp. 2004–2012, 2007.

34. Cobanoglu, M. C., C. Liu, F. Hu, Z. N. Oltvai and I. Bahar, “Predicting drug–target interactions using probabilistic matrix factorization”, *Journal of chemical information and modeling*, Vol. 53, No. 12, pp. 3399–3409, 2013.

35. Wang, J. T., S. Rozen, B. A. Shapiro, D. Shasha, Z. Wang and M. Yin, “New techniques for DNA sequence classification”, *Journal of Computational Biology*, Vol. 6, No. 2, pp. 209–218, 1999.

36. Nguyen, N. G., V. A. Tran, D. Phan, F. R. Lumbanraja, M. R. Faisal, B. Abapihi, M. Kubo and K. Satou, “DNA sequence classification by convolutional neural

network", *Journal Biomedical Science and Engineering*, Vol. 9, No. 5, pp. 280–286, 2016.

37. Qi, R., A. Ma, Q. Ma and Q. Zou, "Clustering and classification methods for single-cell RNA-sequencing data", *Briefings in bioinformatics*, Vol. 21, No. 4, pp. 1196–1208, 2020.

38. Altschul, S. F., W. Gish, W. Miller, E. W. Myers and D. J. Lipman, "Basic local alignment search tool", *Journal of molecular biology*, Vol. 215, No. 3, pp. 403–410, 1990.

39. Burge, C. and S. Karlin, "Prediction of complete gene structures in human genomic DNA", *Journal of molecular biology*, Vol. 268, No. 1, pp. 78–94, 1997.

40. Lowe, T. M. and S. R. Eddy, "tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence", *Nucleic acids research*, Vol. 25, No. 5, pp. 955–964, 1997.

41. Trapnell, C., D. G. Hendrickson, M. Sauvageau, L. Goff, J. L. Rinn and L. Pachter, "Differential analysis of gene regulation at transcript resolution with RNA-seq", *Nature biotechnology*, Vol. 31, No. 1, pp. 46–53, 2013.

42. Rannala, B. and Z. Yang, "Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci", *Genetics*, Vol. 164, No. 4, pp. 1645–1656, 2003.

43. Choi, S. H., A. T. Labadorf, R. H. Myers, K. L. Lunetta, J. Dupuis and A. L. DeStefano, "Evaluation of logistic regression models and effect of covariates for case–control study in rna-seq analysis", *BMC bioinformatics*, Vol. 18, No. 1, pp. 1–13, 2017.

44. Lan, K., D.-t. Wang, S. Fong, L.-s. Liu, K. K. Wong and N. Dey, "A survey of data mining and deep learning in bioinformatics", *Journal of medical systems*, Vol. 42,

pp. 1–20, 2018.

45. Zeng, C., Y. Jian, S. Vosoughi, C. Zeng and Y. Zhao, “Evaluating native-like structures of RNA-protein complexes through the deep learning method”, *Nature Communications*, Vol. 14, No. 1, p. 1060, 2023.

46. Luo, R., L. Sun, Y. Xia, T. Qin, S. Zhang, H. Poon and T.-Y. Liu, “BioGPT: generative pre-trained transformer for biomedical text generation and mining”, *Briefings in Bioinformatics*, Vol. 23, No. 6, 2022.

47. Osmanbeyoglu, H. U. and M. K. Ganapathiraju, “N-gram analysis of 970 microbial organisms reveals presence of biological language models”, *BMC bioinformatics*, Vol. 12, pp. 1–12, 2011.

48. Wang, Y., Z.-H. You, S. Yang, X. Li, T.-H. Jiang and X. Zhou, “A high efficient biological language model for predicting protein–protein interactions”, *Cells*, Vol. 8, No. 2, p. 122, 2019.

49. Benegas, G., S. S. Batra and Y. S. Song, “DNA language models are powerful zero-shot predictors of non-coding variant effects”, BioRxiv doi: 10.1101/2022.08.22.504706, 2022.

50. Gankin, D., A. Karollus, M. Grosshauser, K. Klemon, J. Hingerl and J. Gagneur, “Species-aware DNA language modeling”, BioRxiv doi: 10.1101/2023.01.26.525670, 2023.

51. Liang, W., “Segmenting DNA sequence into words based on statistical language model”, *Nature Precedings*, pp. 1–1, 2012.

52. for Biotechnology Information, T. N. C., “National Center for Biotechnology Information”, 1996, `https://www.ncbi.nlm.nih.gov/`, accessed on March 8, 2023.

53. İhtiyar, M. N., “Datasets used including the whole dataset obtained and actual

short gene dataset used including training, test and validation sets", 2023, https://drive.google.com/drive/folders/1bJHrZOv36Om_bY3-nOkuHkKfT_dtJDuN?usp=sharing, accessed on April 14, 2023.

54. Jurafsky, D., *Speech & language processing*, Pearson Education India, 2000.

55. Lidstone, G. J., "Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities", *Transactions of the Faculty of Actuaries*, Vol. 8, No. 182-192, p. 13, 1920.

56. Nair, V. and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

57. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors", *nature*, Vol. 323, No. 6088, pp. 533–536, 1986.

58. Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", ArXiv:1412.6980 [cs], 2014.

59. Hochreiter, S. and J. Schmidhuber, "Long short-term memory", *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

60. Huyen, C., "Evaluation Metrics for Language Modeling", 2019, https://thegradient.pub/understanding-evaluation-metrics-for-language-models/, accessed on March 8, 2023.

61. İhtiyar, M. N., "Implementation of the procedures in the code for all the different methods mentioned in the thesis", 2023, https://github.com/38programmer61/MSThesisCode, accessed on April 02, 2023.

62. Stenson, P. D., E. V. Ball, M. Mort, A. D. Phillips, J. A. Shiel, N. S. Thomas,

S. Abeysinghe, M. Krawczak and D. N. Cooper, "Human gene mutation database (HGMD®): 2003 update", *Human mutation*, Vol. 21, No. 6, pp. 577–581, 2003.

63. Stenson, P. D., M. Mort, E. V. Ball, K. Shaw, A. D. Phillips and D. N. Cooper, "The Human Gene Mutation Database: building a comprehensive mutation repository for clinical and molecular genetics, diagnostic testing and personalized genomic medicine", *Human genetics*, Vol. 133, pp. 1–9, 2014.

64. Stenson, P. D., M. Mort, E. V. Ball, M. Chapman, K. Evans, L. Azevedo, M. Hayden, S. Heywood, D. S. Millar, A. D. Phillips *et al.*, "The Human Gene Mutation Database (HGMD®): optimizing its use in a clinical diagnostic or research setting", *Human genetics*, Vol. 139, pp. 1197–1207, 2020.

65. Shannon, C. E., "A mathematical theory of communication", *The Bell system technical journal*, Vol. 27, No. 3, pp. 379–423, 1948.

66. Toolkit, N. L., "NLTK :: Natural Language Toolkit", 2023, `https://www.nltk.org/`, accessed on March 8, 2023.

67. Google, "TensorFlow", 2023, `https://www.tensorflow.org/`, accessed on April 3, 2023.

68. Google, "Keras: Deep Learning for humans", 2023, `https://keras.io/`, accessed on April 17, 2023.

69. Google, "KerasNLP", 2022, `https://keras.io/keras_nlp/`, accessed on April 17, 2023.

70. Hunter, J. D., "Matplotlib: A 2D graphics environment", *Computing in Science & Engineering*, Vol. 9, No. 3, pp. 90–95, 2007.