NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Applied Mathematics and Informatics"

UDC 004.8

Research Project Report on the Topic:

Identification of biological interrelations in the human genome by methods of language model interpretation

Submitted by the Student:

group #БПАД222, 3rd year of study Bokhyan Roman Beniaminovich

Approved by the Project Supervisor:

Borevsky Andrey Olegovich
Assistant
Faculty of Computer Science, HSE University

Moscow 2025

Contents

Annotation

This study, conducted under the supervision of the International Laboratory of Bioinformatics at HSE University, explores a large language model for DNA analysis, HyenaDNA, which differs from classical transformer architectures and state-based models (SSMs) by using an alternative approach: long convolution instead of an attention mechanism. The model was further fine-tuned on tasks involving DNA secondary structures, Z-DNA and G-quadruplex (G4), and tested on promoter sequences. The model results were partially interpreted using xAI methods, confirming the significance of known sequences associated with the formation of these structures. HyenaDNA outperformed the existing DeepZ and GraphZ models in terms of quality and complexity of solutions, but was inferior to Z-DNABERT. For the first time ever, LLM with long convolutions (as well as closely related SSM architectures) was applied to the task of recognizing secondary DNA structure, which play a critical role in medicine and genomics. This coursework is part of a larger lab study about interpreting LLM in genomics problems that will be published as an article in late 2025/early 2026

Аннотация

В данном исследовании проводимом под руководством Международной Лаборатории Биоинформатики ВШЭ рассматривается большая языковая модель для анализа ДНК — HyenaDNA, отличающаяся от классических архитектур трансформеров и моделей на основе состояний (SSM) использованием альтернативного подхода: длинной свёртки вместо механизма внимания. Модель была дообучена на задачах, связанных с вторичными структурами ДНК — Z-DNA и G-quadruplex (G4), и протестирована на промоторных последовательностях. Результаты модели частично интерпретированы с помощью методов xAI, подтвердивших значимость известных последовательностей, ассоциированных с формированием этих структур. По качеству и сложности решений HyenaDNA превзошла существующие модели DeepZ и GraphZ, но уступила Z-DNABERT. Впервые в истории LLM с длинной свёрткой (а также близкие к ним SSM-архитектуры) была применена к задаче распознавания вторичной структуры ДНК, которая играет критически важную роль в медицине и геномике. Данная курсовая работа часть большого исследования лаборатории про интерпретацию LLM в задачах геномики, которое будет опубликовано в качестве статьи в конце 2025/начале 2026 года.

Keywords

DNA (Deoxyribonucleic Acid) – A molecule that carries genetic instructions for the development, functioning, growth and reproduction of all known living organisms. It consists of two strands that form a double helix.

Z-DNA– A left-handed helical form of DNA (basically secondary structure of it), distinct from the more common right-handed B-DNA, often associated with gene regulation and stress responses.

G-quadruplex (G4) – Four-stranded DNA secondary structure formed by guanine-rich sequences.

Promoter - Sequence located upstream of a gene that serves as a indicator that the gene sequence is here.

Genomics – The study of genomes, including their structure, function, evolution, and mapping, to understand biological processes and disease mechanisms.

Explainable AI (XAI) – A subfield of artificial intelligence focused on making machine learning models interpretable and understandable.

Deep Learning – A subset of machine learning that uses artificial neural networks with multiple layers to learn representations from data.

LLM (Large Language Model) – A deep learning-based model trained on large text datasets to generate and understand human language.

SSM (State Space Models) - Architectures that process sequences by maintaining and updating internal states over time

Attention – A mechanism in deep learning models, particularly in transformers, that dynamically weighs different parts of the input sequence to focus on the most relevant information.

Transformer – A deep learning model architecture that relies on self-attention mechanisms to process sequential data efficiently, widely used in NLP tasks.

Parameter Efficient Fine-Tuning (PEFT) – A method to fine-tune large models by adjusting only a small number of parameters, making it faster and more efficient.

1 Introduction

Today there is a trend towards the widespread use of artificial intelligence methods in various industries. From finance to molecular biology. This research, curated by the International Laboratory of Bioinformatics at HSE, proposes to look at how the currently popular LLM works with a particular data sequence the DNA sequence. In fact, different models (GenaLM [6], HyenaDNA [13], Caduceus [15], DNABert2 [27] and others)already exist to handle popular DNA tasks (next nucleotide generation, DN sequence origin classification, etc.). But what if one has to work with a complex DNA structure that perhaps not all LLMs are ready for?

DNA can be considered the biological code of a human being. It consists of various segments, which may include mutations and provide valuable information about an individual, such as predispositions to diseases or ancestral origins.

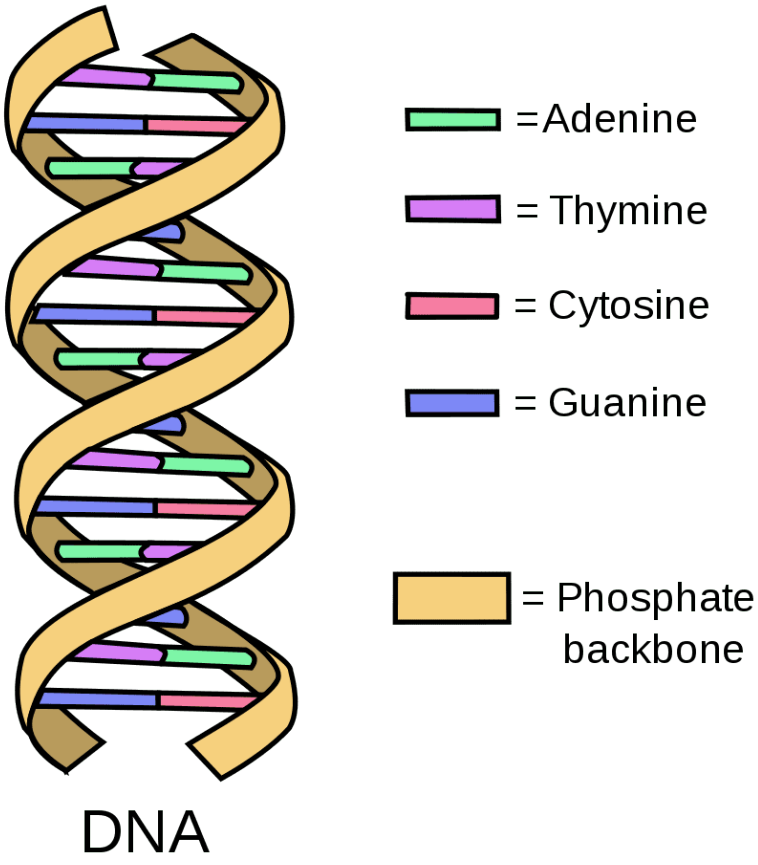

Figure 1.1: Open-source picture

The human genome comprises approximately 3 billion characters and is written using just four nucleotides: adenine (A), thymine (T), guanine (G), and cytosine (C). Despite this simplicity, these four letters combine into sequences that can fold and function in highly complex and diverse

ways.

One well-known motif is the TATA box, a specific sequence (typically TATAAA) that frequently appears near gene promoters. This region serves as a signal for the initiation of transcription and is crucial for gene expression regulation. Promoter regions are of particular interest to biologists, as they mark the beginning of genes, the informative part of DNA, while much of the surrounding sequence is not directly relevant for such studies.

Promoters are a widespread and well-studied topic in genomics, but there is an area that is still open to study, secondary DNA structures. Examples of such secondary structures that I have studied during my research are **Z-DNA** and **G4**.

Only in 1979 did researchers discover ***Z-DNA [23]***, it is a left-handed double helix of DNA. It is formed in regions with alternating purine and pyrimidine bases, especially in sequences (CG)n. This structure appears rarely, but not necessarily a one-time occurrence. A segment of a sequence can flip temporarily into a Z-DNA conformation and then return to its initial state and so on several times.

This structure plays an important role in cellular processes, namely transcription regulation, association with proteins involved in RNA editing, and regulation of the innate immune response [10]. In addition, Z-DNA is associated with Alzheimer's disease [20]. However, to the negative effects, also based on research, it is suggested that the Z-DNA features can be used in methods to fight cancer [26]. The Z-DNA structure itself is complex and rare, but very important, as confirmed by initial research findings. Each year scientists learn more about Z-DNA. Therefore, it is very important to learn how to work with such regions and use modern methods for this task.

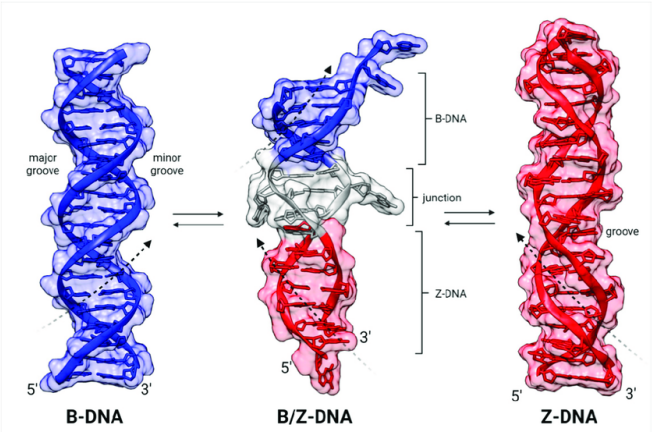

Figure 1.2: Comparative structures of alternative conformations of DNA: left, B-DNA; right, Z-DNA; center, a B-Z composite DNA with an intervening junction. Black dashed arrows indicate handedness (B-DNA, right; Z-DNA, left). Horizontal arrows indicate transitions between depicted DNA conformations. Source [2].

G4 [19] is the second important secondary structure to be considered in this study. Their

special structure makes them important elements in the regulation of genetic information, which can then be used to treat diseases. G4 structures are formed by four guanine bases connected by Hoogstean hydrogen bonds. They are stacked on top of each other to form a four-helix structure as here 1.3. The importance of studying this structure lies in the fact that G4 motifs are often found in promoter parts of genes, including oncogenes [12], where they can repress or activate gene expression by affecting the binding of transcription factors. So, it can also be used as a method to develop new drugs that will create molecules that bind and stabilize with G4.

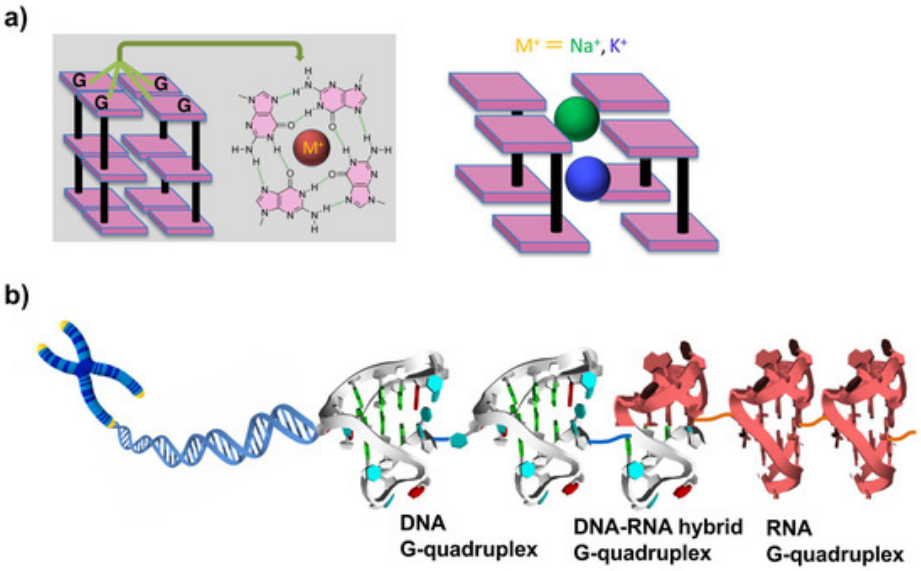

Figure 1.3: (a) Structures of G tetrads and G-quadruplexes. Cations with larger ionic radii, such as K+ (depicted in blue), are positioned between the tetrads. In contrast, smaller cations, such as Na+ (illustrated in green), can coordinate within the plane of the tetrads or assume an intermediate position. (b) Superhelix structure formed at the ends of chromosomes from telomeric DNA G-quadruplexes, DNA:RNA G-quadruplexes, and RNA G-quadruplexes. As the result, this essential portion is sufficiently protected and further functionalized. Source [24].

It becomes particularly intriguing to evaluate how LLMs cope with both canonical promoter regions and more exotic secondary structures—such as Z-DNA and G-quadruplexes—not only in terms of raw predictive quality but also through the lens of interpretability. By probing their ability to surface biologically meaningful relationships between sequence features and functional targets, we can assess how "intelligent" these models truly are at uncovering the hidden rules governing DNA function.

A detailed analysis of the current state of research in this area will be presented in the literature review section. As part of this study, the HyenaDNA model was selected as the primary architecture. This model introduces a unique design choice: it replaces the traditional attention mechanism with a long convolution, which, according to the authors, achieves comparable quality with reduced computational cost. This architecture presents a rare opportunity to evaluate how

such a unique convolutional approach handles complex genomic data. The model was pre-trained on the full human genome.

A key feature of this research is the use of fine-tuning methods (without training from scratch) to evaluate how well the model adapts to narrow domain secondary structures that were not explicitly targeted during pre-training. In summary, HyenaDNA was able to score ***F1 0.82*** on promoter dataset [27], ***0.64*** on Z-DNA **without omics features** and ***0.57*** on G4 **without omics features** using Kouzine et al dataset [9]. The results on the Z-DNA task on the same data are superior to the existing RNN solutions DeepZ (***F1 with omics features 0.4 and without 0.02***) and GNN GraphZ (***F1 with omics features 0.124***), which shows the superiority of the long implicit convolution stucture in LLM HyenaDNA over non-LLM stuctures, but still failed to beat the quality of the classic transformer Z-DNABERT [18] (***F1 without omics features 0.83***), but the resources and running time are far superior to any possible transformer based solution. For task G4, the results were lower than in previous experiments (***F1 0.57***). However, the lab has trained HyenaDNA from scratch [4] (without pre-trained weights and another training approach) and achieved an ***F1 0.75***. Since these are two fundamentally different deep-learning ways, a direct comparison would not be appropriate. Next, all models were tested on training with the LoRa [8] method, which did not gain speed but barely lost any accuracy, making it clear that PEFT methods are not necessary for the HyenaDNA structure. Finally, the weights of the best models were interpreted by IntegratedGradients [17] and Smoothrad [16] methods, and the percentage of TP occurrences of k-mer in the sequence was produced. For G4, the tasks were limited to LoRa and classic fine-tune, due to weak quality metrics.

In future work following this course project, the research will focus on many different experiments such as extending the approach by integrating omics features into language models. One particularly intriguing and non-trivial direction is to extract embeddings from the final layer of the transformer-based model and concatenate them with precomputed multi-omics data available for the hg38 genome across various tissues. This hybrid representation, which combines contextual features at the sequence level with functional genomic signals, can then be used as input to a convolutional model. This framework could open up new opportunities to understand the architecture of DNA through the joint modeling of sequence and functional context.

2 Literature overview

2.1 DeepZ

The first DL approach to the Z-DNA task is the DeepZ model released in 2020 [11]. The paper tried to select a solution from three types of models: CNN, RNN (LSTM) and CNN + RNN. The best approach turned out to be a purely recurrent architecture based on LSTM (***F1 with omics features on small dataset 0.4 and with omics features big dataset 0.02***). Inside a two-layer bi-LSTM (two consecutive blocks of bidirectional layers), this approach gives a two-way look. It is worth noting that the model was trained with omics features (these are sets of genomic features/factors in addition to DNA sequence). The authors of the model took into account the energetic index of the B-Z transition (B-DNA -> Z-DNA) and a set of biological factors. The paper was in its time the first full DL solution, which was a breakthrough in this narrow field of genomics.

Even though this was the first solution in these fields it showed relatively good metrics for a difficult task like Z-DNA, but they did not allow to say that the best solution has been found.

2.2 GraphZ

The idea behind the second approach [22] released in 2022 is the use of graph-based neural network. The authors of DeepZ decided to try replacing the RNN layer of DeepZ with Graph Convolutional Network (GCN), Graph AttentionNetwork (GAT) and inductive representation learning network GraphSAGE. All models were trained with omics data such as DeepZ. The paper considered not only human genomes (HG), but I will not focus on them since the goal of my research is HG. As a result, for the human genome none of the novelties could beat the F1 of DeepZ model and the maximum for GraphZ is ***F1 0.124***. However, GraphSage architecture was able to increase the ***recall from 0.27 (DeepZ) to 0.62.*** The authors note that the model does not whole-genome level HG datasets well.

Despite the potential of the model, the results of this solution show that there is no way to stop and explore other architectures that may show better results.

2.3 Z-DNABERT

In 2023 comes a breakthrough work (from DeepZ and GraphZ authors) the first transformer model based on DNABERT pre-trained *without omics features* called Z-DNABERT [18]. The Kouzine et al (KEx) [9] dataset was used. The result was superior to all existing publicly avail-

able solutions with ***F1 0.83***. Unlike the original DNABERT, which was trained for next-token prediction and sequence-level classification, the Z-DNA task requires per-nucleotide predictions (determining whether each nucleotide is part of the left-handed helix), so they chose a token classification head. The model was trained by the following approach - they spitted the sequences into 6-mers and look at the first nucleotide in the segment, if its label is 1, then the label of the whole k-mer will be 1 and 0 otherwise.

Thus, Z-DNABERT became the first LLM employing the standard attention mechanism to be pretrained on the Z-DNA task without omics features, demonstrating strong performance. However, this raises the question: what if, instead of a conventional transformer-based LLM, we explore an alternative architecture and apply a more rigorous training regimen?

2.4 HyenaDNA benchmark on g4

Moving away from Z-DNA task, the laboratory (they were the authors/coauthors of the models described above) decided to switch to another secondary structure - quadruplexes or G4. An article [4] was published describing experiments with pre-training/training of different LLMs for the quadruplexes task on different datasets, including KEx, which I used in my research as well. When trained from scratch on the G4 prediction task, HyenaDNA achieved an ***F1 0.75***. In comparison, the fine-tuned models performed as follows:

Model	**F1 score**
HyenaDNA	0.75
DNABERT	0.80
DNABERT-2	0.87
GENA-LM	0.84
Caduceus	0.62

Table 2.1: Comparison of F1 scores on the G4 prediction task on KEx dataset.

Based on this, it can be seen that all kinds of LLM architectures for DNA that are currently available are represented here - with the classical attention mechanism [21] (DNABERTs and GENA-LM), SSM [7] (Caduceus) and long convolution [14] (HyenaDNA).

Transformer-based solutions showed their superiority in G4 structure recognition. Notably, HyenaDNA in the experiments at hand was used without any pretrained weights, and no one has yet attempted to fine-tune it on the G4 task leveraging existing DNA "knowledge". So, there is a question: can a long-convolution approach genuinely compete with transformers when fine-tuned for secondary-structure recognition?

3 Dataset and Methods

3.1 Model Architecture

HyenaDNA, a decoder based foundation model for DNA analysis capable of processing sequences up to 1 million in length at the single character level, was chosen as the model. The authors have come up with their own unique variant of self attention replacement called Hyena Operator [14], which allows increase the size of the context window up to a million tokens.

The Hyena operator first projects the input sequence x into three streams u, v, and w using linear layers and short convolutions. Instead of storing a huge convolution kernel of length L, it generates each filter value on-the-fly through a small MLP that takes the position index (and positional encoding) as input, keeping the parameter count constant. The long convolution $h * u$ is then computed in $O(L \log L)$ time via FFT.

Basic FFT Definitions *Below are the key definitions from Fourier transform theory that will help understand the HyenaDNA operator.*

Definition 1: Discrete Fourier Transform (DFT)

Definition. For a finite sequence $x[0], x[1], \ldots, x[N-1]$, the DFT is

$$X[k] \;=\; \sum_{n=0}^{N-1} x[n]\, e^{-i\,(2\pi/N)\,k\,n}, \quad k = 0, 1, \ldots, N-1.$$

Explanation. Converts the sequence from the "time" (or index) domain into its N constituent frequency components, revealing periodic structure.

Definition 2: Fast Fourier Transform (FFT)

Definition. Any divide and conquer algorithm that computes the DFT in $O(N \log N)$ time rather than the naive $O(N^2)$.

Definition 3: Convolution Theorem

Definition. For two sequences $h[n]$ and $u[n]$,

$$\mathcal{F}\{h * u\}(k) \;=\; \mathcal{F}\{h\}(k) \;\cdot\; \mathcal{F}\{u\}(k),$$

Definition 3: Convolution Theorem (cont.)

where $(h * u)[n] = \sum_m h[m]\, u[n-m]$.

Explanation. Shows that convolution in the index domain becomes pointwise multiplication in the frequency domain, enabling efficient filtering via FFTs.

Finally, this convolution output is modulated by v through elementwise multiplication:

$$y = (h * u) \odot v \tag{1}$$

which lets the model capture global, long-range dependencies while adaptively gating the signal at each position.

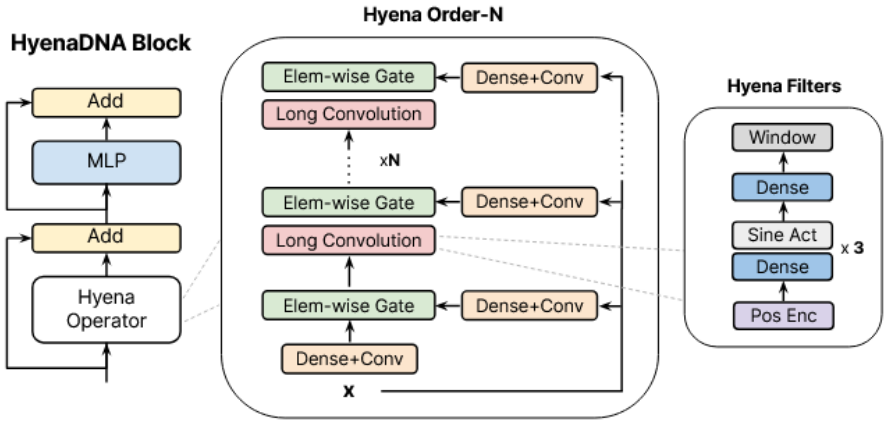

Figure 3.1: HyenaDNA block architecture. Source [13]

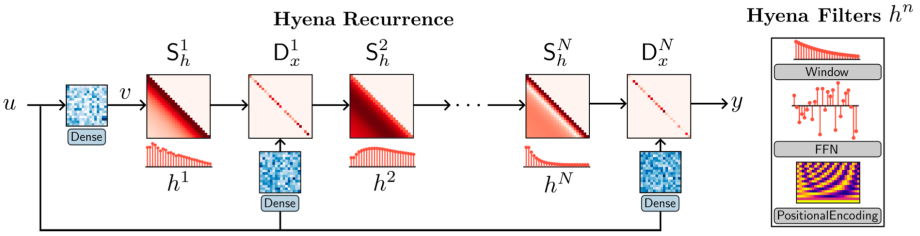

Figure 3.2: The Hyena operator recurrence. Source [14]

Also, their model uses their own variant of tokenization, which breaks the sequence into nucleotides. According to the authors, this approach should solve the issue of loss of accuracy in cases where each nucleotide plays a role. HyenaDNA operates exclusively on nucleotide symbols (A, C, G, T, + special N) without aggregation into k-mers, preserving the maximum resolution.

By combining these two new approaches, the authors have presented a variant of LLM that works for sub-quadratically complexity and having much less parameters. According to their

claims, the model does not fall short in quality compared to classical transformers, and in addition, it can handle such contexts that self attention based models cannot.

3.2 Dataset

For this coursework I decided to use the hg38 Kouzine et al [9] dataset. To confirm that ssDNA-Seq (single-strand DNA sequencing - a method for identifying and mapping segments of the genome where DNA temporarily splices or forms non-standard (non-B) structures) signals in cells corresponded to true G-quadruplexes, the authors overlapped their in-cell data with high-confidence quadruplex sites from the G4-Seq protocol applied to naked human DNA in vitro. They observed substantial concordance between in-cell ssDNA peaks and G4-Seq motifs, reinforcing that many of the in vivo ssDNA signs reflect folded G-quadruplexes in the human genome. As for Z-DNA, they called a region Z-DNA when unpaired thymines identified by ssDNA-seq fell inside sequences a statistical tool predicts can switch to Z form. The modified thymines then mark the two B→Z junctions where the helix flips from right-handed B-DNA into left-handed Z-DNA. Thus, in the KEx lies everything needed to reproduce and analyse exactly where in the genome Z-DNA, G4 structures and other secondary structures are formed in living cells.

As for the promoter dataset, I decided to use the GUE dataset [27]. GUE (Genome Understanding Evaluation) is a comprehensive multi-species genome classification benchmark introduced alongside DNABERT-2. It comprises 36 distinct datasets spanning 9 tasks including promoter detection, core promoter identification, transcription factor binding site prediction, epigenetic mark prediction, and splice site detection across several species, with input sequence lengths ranging from 70 to 10 000 bases.

3.3 Interpretability Methods (XAI)

Integrated Gradients [17] assigns each input feature a contribution score by accumulating the model's gradients along a straight-line path from a baseline input x' (often all zeros) to the actual input x. In practice, this integral is approximated by summing over m steps:

$$\mathrm{IG}_i(x) \;\approx\; (x_i - x_i')\frac{1}{m}\sum_{k=1}^{m}\frac{\partial F\big(x' + \frac{k}{m}(x - x')\big)}{\partial x_i} \tag{2}$$

SmoothGrad [16] reduces noise in gradient-based saliency maps by averaging over many noisy samples of the input. Given n noisy copies $x^{(k)} = x + \mathcal{N}(0, \sigma^2)$, the final attribution is

$$\text{SmoothGrad}(x) \;=\; \frac{1}{n}\sum_{k=1}^{n} \frac{\partial F\big(x^{(k)}\big)}{\partial x} \tag{3}$$

By adding Gaussian noise to embeddings or to the raw input, and then averaging the resulting gradient maps, SmoothGrad highlights consistent attribution patterns and smooths out unexpected peaks.

Informally, the difference between the two methods is that IG will show exactly how each token affected the predictions from the null vector to the vector with real values, while SmoothGrad removes random bursts of tokens by generating n noisy copies of embeddings with real values and counting the gradient over them, which is then averaged.

3.4 LoRA Fine-Tuning

LoRA [8] (Low-Rank Adaptation) makes fine-tuning large pretrained models faster and lighter by keeping the original weight matrix frozen and learning only a small, low-rank update.

Express the update ΔW to the frozen weight matrix W as the product of two much smaller matrices:

$$\Delta W = A\,B \tag{4}$$

Here, A and B capture the task-specific changes without touching W itself.

These two matrices have low rank r, with

$$A \in \mathbb{R}^{d\times r}, \quad B \in \mathbb{R}^{r\times k}, \quad r \ll \min(d,k). \tag{5}$$

Because r is tiny compared to d and k, add only a few new parameters.

Then form the adapted weight matrix W' by adding the update:

$$W' = W + \Delta W. \tag{6}$$

W stays exactly as it was pretrained, and ΔW carries all the fine-tuning.

When the layer sees an input $x \in \mathbb{R}^k$, it computes

$$y = W'\,x = W\,x + A\,(B\,x). \tag{7}$$

This means it still use the original W, plus the small low-rank correction $A(Bx)$.

Since r is much smaller than d and k, training costs drop from $O(dk)$ to $O((d+k)\,r)$. This makes fine-tuning both compute and memory-efficient.

4 Experimental Evaluation

Before delving into the experiments, it was necessary to decide on the quality metrics. At different stages I used different metrics, and below is a list of them with their explanations.

Precision measures the fraction of predicted positives that are actually correct:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8}$$

Recall measures the fraction of actual positives that are correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

F1 score is the harmonic mean of precision and recall, balancing both:

$$F_1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

Matthews Correlation Coefficient (MCC) is a balanced measure of binary classification quality, even on imbalanced data:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{11}$$

ROC AUC (Area Under the Receiver Operating Characteristic curve) summarizes the trade-off between true positive rate and false positive rate over all thresholds:

$$\text{ROC AUC} = \int_0^1 \text{TPR}(t)\,\mathrm{d}\big(\text{FPR}(t)\big) \tag{12}$$

ROC AUC and MCC are generally the most reliable overall metrics; MCC in particular remains robust when classes are highly imbalanced. The F1 score is valuable when a single measure balancing precision and recall is needed, so it is a good metric for imbalanced tasks. Precision and recall themselves are best used to find out the type of error to determine whether the model's errors are mainly false positives or false negatives in model's predictions.

4.1 Promoters

As an initial step of the study, I decided to run HyenaDNA on the GUE [27] dataset promoter task. Despite the fact that the model was pre-trained on the promoters of another dataset, it was interesting to see how it would cope with the data, with data in which motifs other than TATA boxes predominate, since such motifs are difficult to detect.

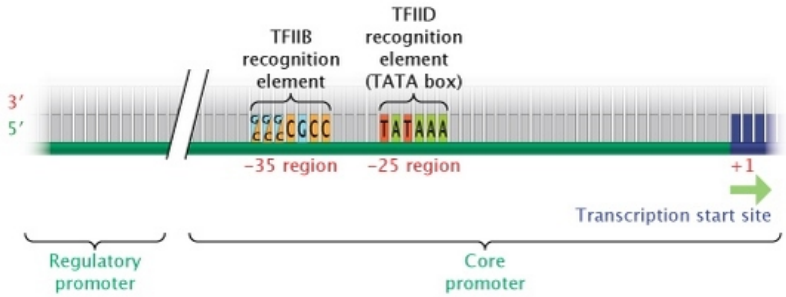

Figure 4.1: Core promoter architecture, highlighting the prevalence of GC-rich motifs and canonical TATA boxes [1].

The data was stored as follows: sequences of length 70, labeled 1 if the sequence is a promoter and 0 otherwise, so it is a sequence classification task. I expected that HyenaDNA would perform well.

4.1.1 Standard fine-tuning

As an experiment, the following hyperparameters were selected

Experiment	Learning Rate	Train / Eval Batch Size	LR Scheduler
1	6×10^{-4}	64 / 64	linear
2	5×10^{-4}	32 / 32	linear
3	2×10^{-5}	64 / 64	cosine
4	5×10^{-4}	8 / 8	linear

Table 4.1: Hyperparameter settings for the four HyenaDNA training runs. All experiments used 10 epochs, warmup ratio 0.1, weight decay 0.01, the `adamw_torch` optimizer, and FP16 precision.

As a result, the best performance ended up being the settings of the **first experiment**. I chose F1 as the main metric.

4.1.2 LoRa Fine-Tuning

The second approach I used was the PEFT method called LoRa [8]. Notably, the training complexity remained almost unchanged compared to the classical variant(about 3 min for both

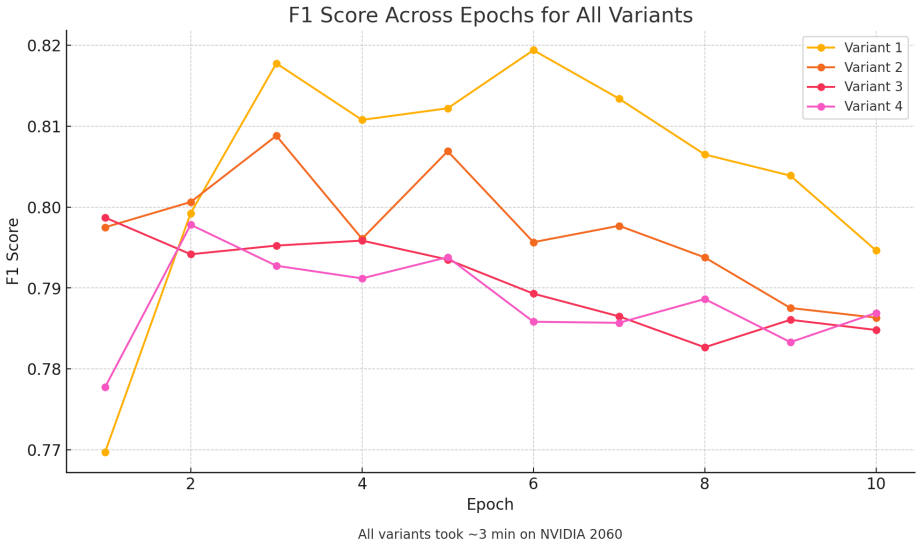

Figure 4.2: F1 on eval through 4 different starts

Metric	**Value**
Validation loss	0.4013
Accuracy	0.8167
Precision	0.8024
Recall	0.8416
F1 score	0.8216
MSS	0.6342
Evaluation runtime (s)	0.8965
Samples per second	6603.3390
Steps per second	103.7350
Epoch	10.0

Table 4.2: Final metrics after 10 epochs on the test dataset. Metrics were calculated based on the weights from the best epoch of the first experiment

on the same GPU, except mixer only approach that took around 20 minutes to complete), which suggests that HyenaDNA does not need LoRa due to the sub-quadratic complexity of the model.

Experiment	r	α	Dropout	% Trainable
Full (Mixer+MLP)	8	32	0.1	2.79%
Mixer Only	8	32	0.1	7.04%
FFN Only	8	32	0.1	5.38%
FFN Only + All Bias	16	16	0.1	9.54%
Embeddings Only	16	32	0.1	0.58%
TokenCls (Mixer)	16	32	0.1	6.33%

Table 4.3: LoRA configuration for each experiment: rank r, α, dropout, and percentage of trainable parameters.

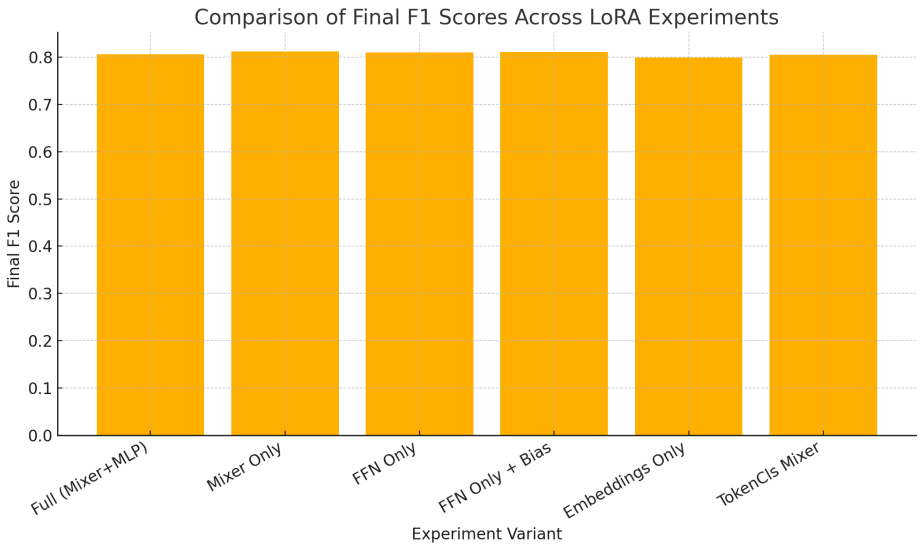

Figure 4.3: F1 from different experiments

Different subsets of the HyenaDNA model were adapted with LoRA (mixer layers, feed-forward layers, their combination, embeddings, etc.) to see where low-rank adaptation yields the best F1. Each variant used the same overall training setup (10 epochs, LR 5e-4, warmup, AdamW, FP16).

In summary, LoRa did not bring any advantage in the task of promoters on the HyenaDNA model, the code execution speed is the same (or slight changes) and the quality is expectedly a bit lower. The method did not prove useful in this case, nevertheless it is interesting to study it on such an unusual architecture as Hyena Operator.

4.1.3 Interpretation

Now we come to the most interesting part - interpretation. The obtained results performance is very good, moreover, they are even comparable with DNABERT2 benchmarks [27] on this dataset. After getting good results the question arises: was the model truly reasoning correctly, or did it latch onto false patterns during training? To answer this question it was necessary to use the interpretation methods I described IG and SmoothGrad. For interpretation I used weights of the model with the best ***F1 (0.82)***.

Although the model evaluates individual characters, tokenization is out of type. To assign a single importance to each k-measure, I slide a window of size k over the cleaned attribution scores, calculated the average attribution in each window, and thus labeled each adjacent k-measure with its average score. Importantly, I limited analysis to true positives (from test dataset) in order to examine how the model reasons when it arrives at correct predictions.

SequenceClassificationExplainer Attribution Aggregation The `SequenceClassificationExplainer` was applied to the trained model and tokenizer to extract token-level attributions for every sequence in the development set that was correctly predicted as positive. After removing special tokens (CLS, SEP, PAD), a sliding window of length k=5 was moved over the remaining tokens, and each 5-mer was assigned the average of its constituent token attributions. By averaging these 5-mer scores across all true positives and sorting them in descending order, the motifs with the strongest positive influence were revealed. ***The algorithm can be found in the appendix part of the paper.***

SmoothGrad-Based K-mer Attribution A custom forward function operating on input embeddings

$$\texttt{forward\_embeddings}(\mathbf{E}, \mathbf{M}) = \texttt{model(inputs\_embeds=}\mathbf{E}\texttt{, attention\_mask=}\mathbf{M}\texttt{).logits}, \tag{13}$$

was wrapped with Captum's `IntegratedGradients` and a `NoiseTunnel` for SmoothGrad. For each true-positive sequence, its token IDs were converted to embeddings $\mathbf{E}$, and a zero tensor of the same shape served as the baseline. SmoothGrad attributions were then computed with 50 noisy samples (stdev = 0.02) to produce a tensor of per-token, per-dimension gradients. Summing over the embedding dimension yielded one score per token; after discarding special tokens, the same sliding-window averaging k=5 produced scores for each contiguous 5-mer, which were then ranked to highlight the most predictive sequence patterns. ***The algorithm can be found in the appendix part of the paper.***

Results of these methods

k-mer	SmoothGrad Attribution
GCTGG	0.09124
CTGGG	0.08750
GGTGG	0.08517
CGCGG	0.08249
TGGGG	0.08248
GCGGG	0.08084
GTTGG	0.08069
TGGGA	0.08046
GCGCG	0.08040
GGAAG	0.08014

Table 4.4: Top 10 5-mers by average SmoothGrad attribution score.

To combine the attributions from Integrated Gradients (IG) and SmoothGrad (SG), all

k-mer	SeqClassExplainer Attribution
GGCTG	0.06518
GCTGG	0.06350
GCTGC	0.06171
TTTCC	0.06139
CTGCT	0.06096
CTTCC	0.05841
TTCCG	0.05808
TTCCT	0.05740
TGCTG	0.05660
GGGAG	0.05548

Table 4.5: Top 10 5-mers by average SequenceClassificationExplainer attribution score.

positive scores for each k-mer were merged into a single table. For each method, the percent deviation of each k-mer's score from the method's mean was computed:

$$\delta_i = \frac{s_i - \bar{s}}{|\bar{s}|} \times 100\%, \tag{14}$$

where s_i is the attribution for the ith k-mer and $\bar{s}$ is the average attribution over all k-mers. The deviations from IG and SG were then averaged to yield a consensus score mean dev).

k-mer	Mean Dev (%)
GCTGG	138.50
GGCTG	125.49
GCTGC	110.13
CTGGG	106.59
GGGAG	104.54
TGCTG	103.48
GGTGG	95.47
TGGGG	88.36
TGGGA	87.88
GGAGG	86.42

Table 4.6: Consensus ranking of top 5-mers by average percent deviation from mean attributions across Integrated Gradients and SmoothGrad.

In summary, the two methods of interpretation and their ranking showed that G and C combinations are in the top, which is consistent with the biological promoter theory [25] 4.1, and most importantly shows that the model was able to identify the correct dependencies in TP.

4.1.4 Promoter Conclusion

HyenaDNA excelled on promoter prediction achieving strong performance and yielding biologically meaningful attributions. In addition, attribution analyzes revealed that HyenaDNA consistently prioritized key promoter elements.

4.2 Z-DNA

4.2.1 Model's Details

As noted above, I used the Kouzine et al. (KEx) dataset for both Z-DNA and G4 prediction. Since positives occur only one-quarter as often as negatives, every train/validation/test split was generated via stratified sampling to preserve the $1:3$ class ratio. The specificity of the task is that it is a single-nucleotide classification and here the HyenaDNA tokenizer gives a big advantage. But there is a nuance that the pre-trained model has no head for Token Classification, so I wrote it myself. In the experiments described below I will mention an attempt to use the head of another model, but in all other cases I used my version.

Algorithm 1 HyenaDNA Token-Level Classification

Init(config, num_labels)
  $L \leftarrow$ HyenaDNAModel(config)
  $C \leftarrow$ Linear(d_model, num_labels)
  $\mathcal{L} \leftarrow$ CrossEntropyLoss(weights, ignore_index)
  `post_init`()
procedure FORWARD(input_ids, inputs_embeds, labels, return_dict)
  $O \leftarrow L($`input_ids`, `inputs_embeds`, `return_dict`$)$
  $H \leftarrow O$.`last_hidden_state`
  `logits` $\leftarrow C(H)$
  if labels provided **then**
    $\ell \leftarrow \mathcal{L}($`logits`$.reshape(-1, \_),$ `labels`$.reshape(-1))$
  else
    $\ell \leftarrow$ `None`
  end if
  if return_dict **then**
    return {`loss`$=\ell$, `logits`, `hidden_states` $= O$.`hidden_states`} **else**
    return `loss`? $(\ell,$ `logits`$)$: $($`logits`$)$ **end if**

The `HyenaDNAForTokenClassification` class extension of `HyenaDNAPreTrainedModel`. In

its constructor, it reads `num_labels` either from the keyword arguments or from the configuration, instantiates the HyenaDNA backbone via `HyenaDNAModel(config)`, and attaches a linear classifier of shape `(d_model, num_labels)` without bias. A cross-entropy loss is set up with class weights (there were experiments to find best for my research) and `ignore_index` $= -100$ to not consider special tokens. The methods `get_input_embeddings` and `set_input_embeddings` expose the word embedding layer for external manipulation. The `forward` method accepts either `input_ids` or precomputed `inputs_embeds`, runs them through the HyenaDNA backbone to obtain last_hidden_state of shape `(batch, seq_len, d_model)`, and applies the token-level linear layer to produce `logits` of shape `(batch, seq_len, num_labels)`. If `labels` are provided, these logits and labels are flattened to compute the weighted cross-entropy loss. Finally, a `TokenClassifierOutput` is returned, containing `loss`, `logits`, and optional hidden states, or a tuple (`loss`, `logits`) when `return_dict=False`.

The `forward` method accepts either `input_ids` or precomputed `inputs_embeds`, runs them through the HyenaDNA backbone to obtain last_hidden_state of shape `(batch, seq_len, d_model)`, and applies the token-level linear layer to produce `logits` of shape `(batch, seq_len, num_labels)`. If `labels` are provided, these logits and labels are flattened to compute the weighted cross-entropy loss. Then, a `TokenClassifierOutput` is returned, containing `loss`, `logits`, and optional hidden states, or a tuple (`loss`, `logits`) when `return_dict=False`.

4.2.2 Standart Fine-Tuning

Through extensive Z-DNA experiments, the most effective setup was freezing both the model's head and body, yielding an ***F1 0.64***, a ***ROC-AUC 0.80***, and an ***MCC 0.62***. Before describing the best solution, I will go into details of my experiments.

The best performance was obtained with a custom two-phase training regimen under the following settings: class-weighted cross-entropy loss with weights $w_- = 0.7$ (negative) and $w_+ = 2.0$; sequence length $L = 100$; batch size 64; no data augmentation; standard HyenaDNA tokenization; stratified 60/20/20 train/val/test split.

Phase 1: Frozen Backbone. Only the classification head was trained for 3 epochs at learning rate $\eta = 10^{-3}$, while all Hyena backbone parameters remained frozen. This allowed the newly-initialized head to specialize rapidly without perturbing pretrained features.

Phase 2: Full Fine-Tuning. The entire model was unfrozen and trained for 6 further epochs with discriminative learning rates: $\eta_{\text{backbone}} = 10^{-5}$ and $\eta_{\text{head}} = 5 \times 10^{-4}$. This produced the final F1 score of 0.64 on the evaluation set.

Configuration	Variants / Settings	Purpose / Hypothesis
Sequence length	100, 256, 512, 1024	Effect of context-window size on accuracy
Batch size	8, 16, 32, 64, 128	Throughput vs. generalization trade-off
Loss function	Weighted CrossEntropy, Focal Loss	Handling class imbalance
Data augmentation	Random cropping (±20), overlapping windows	Robustness to fragment positioning
Reverse-complement augm.	On-the-fly reverse-complement on 50% of batches	Enforce strand-invariance
Freezing strategy	Freeze backbone → train head only, then full fine-tune	Stability of pre-trained features
Model head variant	Custom linear vs. Electra-small [5]	Impact of classification head capacity
Learning rate schedule	Linear warmup, cosine decay, constant + ReduceLROnPlateau	Optimization stability and convergence speed
Tokenization setup	With vs. without special tokens	Influence of token markers on sequence understanding
Data split	Stratified split vs. Stratified K-Fold cross-validation	Variance and bias in validation metrics
Low-batch-size regime	batch_size=1, accum_steps=4, bf16, checkpointing, 10% warmup	Simulate low-memory training with effective batch size=4

Table 4.7: Summary of experimental configurations for Z-DNA prediction and their evaluation focus.

Because HyenaDNA lacks a built-in token-classification head, this two-phase schedule provides a strong initialization for the custom head while safeguarding the pretrained backbone. Class-aware loss weights counteracted label imbalance, and discriminative rates prevented catastrophic forgetting—achieving both stability and targeted adaptation. Fine-tuning all model parameters with "best" hyperparameters yielded nearly identical results to the two-stage scheme **F1 0.60 ± 0.03, ROC-AUC 0.80 ± 0.02, MCC 0.60 ± 0.02** but did not exceed it. The only "novel" trick to match this performance was extensive data augmentation, which achieved the same scores at the expense of roughly three times longer training. By contrast, swapping out cross-entropy for Focal Loss or replacing my linear head with an Electra-small variant consistently underperformed (all metrics <= 0.55). It is s also worth noting that there was a split of 5 folds with best parameters using that averaged a performance of ***F1 0.63***, a ***ROC-AUC 0.80***, and an ***MCC 0.62***.

Experiment	F1	ROC-AUC	MCC	Precision	Recall
Batch=1, accum=4, 5ep, Len=100	0.6283	0.7836	0.6136	0.6846	0.5805
Batch=128, 10ep, Len=100, CE(0.7/2)	0.6378	0.8021	0.6205	0.6562	0.6204
Batch=64, 12ep, Len=100, constant+warmup	0.6319	0.8055	0.6135	0.6347	0.6291
Batch=8, 12ep, Len=100	0.6385	0.8046	0.6209	0.6516	0.6259
5-fold CV, 12ep, Len=100, Batch=64	0.6378	0.8012	0.6207	0.6586	0.6186
Augmented data (80–120bp), 12ep, Batch=64	0.6437	0.7973	0.6282	0.6829	0.6086

Table 4.8: Comparison of evaluation metrics across various training configurations near the optimal solution.

4.2.3 LoRa Fine-Tuning

The LoRa situation on the Z-DNA task is very similar to the outcome of the promoters' task. LoRa once again did not provide any significant speedup, and its performance metrics were almost the same as with standard fine-tuning.

Variant	Trainable %	F1	ROC-AUC	MCC
Full (Mixer+FFN), 3ep	3.86%	0.6363	0.8057	0.6184
Full (Mixer+FFN), 10ep	3.86%	0.6181	0.8014	0.5989
Mixer only (in/out projections), 10ep	1.49%	0.6291	0.8049	0.6106
FFN only (fc1/fc2), 10ep	4.54%	0.6257	0.8022	0.6071
Short-filter only, 10ep	5.38%	0.6279	0.8015	0.6096
FFN + all biases, 10ep	9.54%	0.6301	0.8036	0.6118
Embeddings only, 10ep	0.58%	0.5830	0.7831	0.5620

Table 4.9: Comparison of LoRA experiments with fine-tuning variants on token classification: percentage of trainable parameters, final F1, ROC-AUC and Matthews correlation coefficient.

4.2.4 Interpretation

The idea of interpretation remains the same, but a method that counts the percentage of occurrences of k-mer for auxiliary analysis is added. I will skip in this part the detailed description of IG, SmoothGrad and the ranking method as they are described in **4.1.3** , but I will include the algorithms in ***the appendix part*** and explain the idea of auxiliary analysis. All solutions based on test dataset with only TP sequences. The model with the best metrics was selected for interpretation.

Th Count k-mer True-Positive Importance algorithm is not a gradient-based interpretation method, but rather an idea that can be used to augment complete solutions. To get a quick, model-agnostic importance score, I simply computed, for each token, the fraction

$$\frac{\text{TP occurrences}}{\text{Total occurrences}} \tag{15}$$

over the set (also TP only). Note that this "TP / Total" score only captures tokens inside annotated Z-DNA regions.

Now, an in-depth look at the interpretation results shows that

5-mer	Count	Avg SmoothGrad Score
ATACG	60	534.45
GCGCG	1622	529.61
CGCGC	1597	523.68
TACGT	61	456.88
ACGCG	200	454.00
CGCGT	427	451.16
ATGTA	38	420.26
GCGTG	763	419.04
CGTAT	48	413.98
GTGCG	769	412.44

Table 4.10: Top 10 5-mers ranked by average SmoothGrad attribution scores.

5-mer	Count	Avg IG Score
CGCGC	1597	23.67
GCGCG	1622	21.97
CGTGC	604	21.69
CGCGT	427	21.60
CGTGT	532	20.11
CGCAC	844	19.50
GCGTG	763	19.42
AATGC	1	17.07
GTGCG	769	16.96
AAGAC	1	16.88

Table 4.11: Top 10 5-mers ranked by average Integrated Gradients (IG) attribution scores.

5-mer	TP Count	Occurrences	Importance
CGCAC	844	2895	0.2915
GTGCG	769	2992	0.2570
CGCGT	427	1676	0.2548
GCGCA	756	3006	0.2515
CGTGT	532	2140	0.2486
CGCGG	1069	4390	0.2435
GCGCG	1622	6925	0.2342
CGCGC	1597	6897	0.2315
CGTGC	604	2612	0.2312
GCACG	598	2669	0.2241

Table 4.12: Top 10 5-mers ranked by True-Positive importance: the fraction of times each 5-mer was predicted correctly among its total occurrences.

5-mer	Mean Dev. (%)
CGCGC	542.54
GCGCG	511.67
CGCGT	477.10
CGTGC	461.29
CGTGT	426.80
GCGTG	423.48
CGCAC	417.34
ACGCG	380.64
GTGCG	373.61
GCGCA	351.54

Table 4.13: Top 10 5-mers ranked by their mean deviation (%) between Integrated Gradients and SmoothGrad importance scores.

In summary, interpretation analyses of IG, SmoothGrad and TP k-mer counting have all independently identified GC-rich 5-mers such as `CGCGC`, `GCGCG` and `CGCGT` as the most influential motifs for Z-DNA prediction. This result is consistent with the biological theory of Z-DNA [18], which states that the alternation of such nucleotides occurs in regions of the left-handed helix.

4.2.5 Z-DNA Conclusion

HyenaDNA model with implicit long convolution instead of the classical attention mechanism was first run on the Z-DNA task. The model was fine-tuned on various experiments, resulting in the best model with the following settings:

Parameter	Value
Loss weighting	CE (negative: 0.7, positive: 2)
Stage 1 epochs	3
Stage 1 learning rate	1×10^{-3}
Stage 1 backbone	frozen
Stage 2 epochs	6
Stage 2 learning rate (backbone)	1×10^{-5}
Stage 2 learning rate (head)	5×10^{-4}
Evaluation loss	0.1525
F1 score	0.6401
ROC–AUC	0.8046
MCC	0.6226
Precision	0.6550
Recall	0.6257

Table 4.14: Two-stage fine-tuning setup and resulting test metrics for the best model.

The model clearly showed its superiority over DeepZ [11] and GraphZ [22] by training without omics traits with higher scores. Despite not surpassing the classic Transformer–based Z-DNABERT [18] (***F1 = 0.83***), this experiment reinforces the observation that Transformer architectures still lead in predictive accuracy. HyenaDNA, however, offers dramatically faster training times, and strict token-level labeling protocol for Z-DNA regions is more rigorous than prior benchmarks. Remarkably, even with fewer metrics, the attribution rankings consistently placed the biologically validated 5-mer at the very top, indicating that the model, though less accurate overall, has learned the correct signals. Moving forward, I intend to explore alternative pre-training objectives and data regimes to fully unlock HyenaDNA's potential in this domain.

4.3 G4

And the last experiment in the series of secondary DNA knockdowns was done on G4. This task is very similar to Z-DNA, only the target from the dataset is changed. Therefore, I will skip the description of the dataset and classifier head, as they are identical to what was described above and go straight to the fine-tuning task.

4.3.1 Standart fine-tuning

After running numerous experiments, it became clear that the model, regardless of hyperparameter settings, quickly reaches a performance plateau indicating its limited capacity for further improvement on this task. The best result, however, was achieved with standard training using a batch size of 32, sequence length of 100, 10 percent warmup, class-weighted cross-entropy

Experiment	CE	Batch	Ep.	Scheduler	F1	ROC-AUC
Two-stage	0.7/2	64	3+6	Linear warmup	0.5779	**0.8206**
Single-stage	1/8	64	12	ReduceLROnPlateau	0.5772	0.8178
Single-stage	1/8	**32**	12	Linear warmup	**0.5788**	0.8177
Single-stage	0.7/2	64	12	Linear warmup	0.5761	0.8166

Table 4.15: Comparison of G4-classification setups.

loss (CE = 1 and 8), and 12 epochs, yielding an ***F1 0.5788.*** The last attempt to pre-train the model was LoRa.

4.3.2 LoRa Fine-Tuning

Experiment	**Trainable %**	**Epochs**	**F1**	**ROC-AUC**	**MCC**	**Prec.**	**Rec.**
Full (Mixer+MLP)	7.04	10	**0.5767**	**0.8242**	**0.5438**	**0.4824**	**0.7168**
Mixer only	2.79	10	0.5756	0.8237	0.5426	0.4811	0.7162

Table 4.16: Comparison of LoRA-finetuned HyenaDNA variants on the G4 token-classification task. The "Full" variant adapts both mixer and MLP layers, while "Mixer only" adapts mixer layers alone. Metrics taken at the final epoch (10). Best values in bold.

In the end LoRa showed the same trend as in the previous experiments, no significant code acceleration, but metrics almost like a regular fine-tune. LoRa once again proved unnecessary for the HyenaDNA architecture.

4.3.3 G4 Conclusion

G4 prediction proved to be the weakest link in this study, with metrics too low to warrant deeper interpretation. By comparison, a recent benchmark [4] of HyenaDNA achieved an ***F1 0.75*** roughly 20 points higher when training from scratch. Although differences in pre-training strategies make direct comparison unfair, it is still instructive to note that Transformer-based models adapt far more effectively to G4 classification. In other words, for DNA G4 secondary-structure prediction, HyenaDNA falls short of its peers.

4.4 HyenaDNA's Results

Task	**Accuracy**	**F1**	**MCC**
Promoter	0.8167	0.8216	0.6342
Z-DNA	0.9664	0.6401	0.6226
G4	0.9172	0.5788	0.5443

Table 4.17: Best evaluation metrics achieved on each classification task.

5 Conclusion

To sum it up, my coursework I would like to note once again that this is the first time that an LLM with a special fine-tuned architecture has been used on the task of secondary DNA structures. Detailed study of the model gave hope that it can quickly and efficiently train for DNA tasks. As part of the research, I conducted many different experiments, adapted the model structure to the tasks of the lab, explored biological theory to analyse the performance and was pre-training the LLM using various methods.

The first task was the promoters that the model was pre-trained on, but the interest lay not only in how it would train on a different dataset, but also whether it would be able to detect those segments that are associated with the promoter but are not TATA boxes. As a result, on this task the model showed its skill and with much less time than any transformer it managed all targets.

The second task was Z-DNA. I would say this was my main study, for it is a critical structure in the human genome. I posed the same hypothesis - would the model be able to learn and identify biologically relevant patterns? As it turned out, the model was able to significantly outperform the two existing non-LLM solutions on quality metrics, but was unable to outperform its rival classic transformer. Nevertheless, interpretation of the training showed that the model is excellent at identifying the regions. This performance is a breakthrough of sorts, as it is much faster than any previous solution and understands where it is necessary to look. This leaves an opportunity to further explore the architecture of the model for this task, but moving away from classical fine-tuning, which is what I will do in the framework of the lab research.

The third task was the G4 (G-quadruplex). HyenaDNA's fine-tuned models achieved the lowest performance of all three problems. Despite exploring multiple training strategies, they failed to match the accuracy of transformer-based approaches and even underperformed compared to a HyenaDNA model trained from scratch on the same task. Given these comparatively modest results, it was deemed further interpretability analyses unwarranted.

The experiments with HyenaDNA show that this new model has a lot of potential: it outperforms almost all other approaches I tried, even if it does not quite match transformer models when using standard fine-tuning. This suggests that instead of tweaking hyperparameters for each task, it might be better to train HyenaDNA from scratch or add extra biological data (multi-omics) to its fine-tuned versions. There are many promising directions to explore, and I am excited to continue this work in the lab. In short, HyenaDNA is fast and easy to integrate, but simple fine-tuning on very specialized genomics tasks may not be enough to show its true

strengths.

Over the past year, I have combined my interests in genomics, deep learning, and NLP to study this unconventional LLM for DNA problems. What I have learned is new and largely unexplored, so this project goes well beyond a term paper. It is part of a larger research effort led by the International Laboratory of Bioinformatics. My colleagues and I will keep pushing forward on LLMs for predicting DNA secondary structures, aiming to publish our work in a top-tier journal.

All experiments can be found in my repository [3].

References

[1] Benjamin A. Pierce. *Genetics: A Conceptual Approach*. W. H. Freeman, 2017. ISBN: 978-1-319-72085-3.

[2] Martin Bartas, Kristyna Slychko, Jiri Cerven, Petr Pecinka, Donna J. Arndt-Jovin, Thomas, and M. Jovin. "Extensive Bioinformatics Analyses Reveal a Phylogenetically Conserved Winged Helix (WH) Domain (Z) of Topoisomerase II, Elucidating Its Very High Affinity for Left-Handed Z-DNA and Suggesting Novel Putative Functions". In: *International Journal of Molecular Sciences* (2023).

[3] Roman Bokhyan. *coursework_dna_llm: DNA Large-Language-Model Coursework*. GitHub repository. 2025. URL: https://github.com/perfectteatimer/coursework_dna_llm (visited on May 19, 2025).

[4] O Cherednichenko, A Herbert, and M Poptsova. "Benchmarking DNA large language models on quadruplexes". In: *Computational and Structural Biotechnology Journal* 27 (2025), pp. 992–1000. DOI: 10.1016/j.csbj.2025.03.007.

[5] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *arXiv preprint arXiv:2003.10555* (2020). Proceedings of ICLR 2020. DOI: 10.48550/arXiv.2003.10555. arXiv: 2003.10555 [cs.CL].

[6] Veniamin Fishman, Yuri Kuratov, Aleksei Shmelev, Maxim Petrov, Dmitry Penzar, Denis Shepelin, Nikolay Chekanov, Olga Kardymon, and Mikhail Burtsev. "GENA-LM: a family of open-source foundational DNA language models for long sequences". In: *Nucleic Acids Research* (2025).

[7] Albert Gu, Karan Goel, and Christopher Ré. "Efficiently Modeling Long Sequences with Structured State Spaces". In: *arXiv preprint arXiv:2111.00396* (2021).

[8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "LoRA: Low-Rank Adaptation of Large Language Models". In: *arXiv preprint arXiv:2106.09685* (2021). URL: https://doi.org/10.48550/arXiv.2106.09685.

[9] F. Kouzine, D. Wojtowicz, L. Baranello, A. Yamane, S. Nelson, W. Resch, K.-R. Kieffer-Kwon, C. J. Benham, R. Casellas, T. M. Przytycka, and D. Levens. "Permanganate/S1 nuclease footprinting reveals Non-B DNA structures with regulatory potential across a mammalian genome". In: *Cell Systems* 4.3 (2017).

[10] Beknazarov N, Konovalov D, Herbert A, and Poptsova M. "Z-DNA formation in promoters conserved between human and mouse are associated with increased transcription reinitiation rates". In: *Scientific Reports 14.17786 (2024)* (2024).

[11] Beknazarov N, Jin S, and M. Poptsova. "Deep learning approach for predicting functional Z-DNA regions using omics data". In: *Sci Rep 10, 19134 (2020). https://doi.org/10.1038/s41598-020-76203-1* (2020).

[12] Kosiol N, Juranek S, Brossart P, Heine A, and Paeschke K. "G-quadruplexes: a promising target for cancer therapy." In: *Mol Cancer. 2021 Feb 25;20(1):40. doi: 10.1186/s12943-021-01328-4. PMID: 33632214; PMCID: PMC7905668.* (2021).

[13] Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Callum Birch-Sykes, Michael Wornow, Aman Patel, Clayton Rabideau, Stefano Massaroli, Stefano Ermon Yoshua Bengio, Stephen A. Baccus, and Chris Ré. "HyenaDNA: Long-Range Genomic Sequence Modeling at Single Nucleotide Resolution". In: *arXiv preprint, arXiv:2306.15794 [cs.LG]* (2023).

[14] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. "Hyena Hierarchy: Towards Larger Convolutional Language Models". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 28043–28078. URL: https://proceedings.mlr.press/v202/poli23a.html.

[15] Yair Schiff, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. "Caduceus: Bi-Directional Equivariant Long-Range DNA Sequence Modeling". In: *arXiv preprint, arXiv:2403.03234v2 [q-bio.GN]* (2024).

[16] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. "SmoothGrad: removing noise by adding noise". In: *arXiv preprint arXiv:1706.03825* (2017). URL: https://doi.org/10.48550/arXiv.1706.03825.

[17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic attribution for deep networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3319–3328.

[18] Dmitry Umerenkov, Alan Herbert, Dmitrii Konovalov, Anna Danilova, Nazar Beknazarov, Vladimir Kokh, Aleksandr Fedorov, and Maria Poptsova. "Z-Flipon Variants reveal the many roles of Z-DNA and Z-RNA in health and disease". In: *bioRxiv* (2023). DOI: 10.1101/2023.01.12.523822. eprint: https://www.biorxiv.org/content/early/2023/01/13/2023.

01.12.523822.full.pdf. URL: https://www.biorxiv.org/content/early/2023/01/13/2023.01.12.523822.

[19] Spiegel J Varshney D, Zyner K, Tannahill D, and Balasubramanian S. "The regulation and functions of DNA and RNA G-quadruplexes". In: *Nat Rev Mol Cell Biol 21, 459–474 (2020). https://doi.org/10.1038/s41580-020-0236-x* (2020).

[20] P. Vasudevaraju, Bharathi, R.M. Garruto, K. Sambamurti, and K.S.J. Rao. "Role of DNA dynamics in Alzheimer's disease". In: *Brain Research Reviews* 58.1 (2008), pp. 136–148. DOI: 10.1016/j.brainresrev.2008.01.001.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017, pp. 5998–6008.

[22] Artem Voytetskiy, Alan Herbert, and Maria Poptsova. "Graph Neural Networks for Z-DNA prediction in Genomes". In: *IEEE 10.10.1101/2022.08.23.504929 (2022)* (2022).

[23] AJ Wang, G Quigley, F Kolpak, and et al. "Molecular structure of a left-handed double helical DNA fragment at atomic resolution". In: *Nature 282, 680–686 (1979). https://doi.org/10.1038/* (1979).

[24] Yan Xu and Makoto Komiyama. "G-Quadruplexes in Human Telomere: Structures, Properties, and Applications". In: *Molecules* 29.1 (2024). ISSN: 1420-3049. DOI: 10.3390/molecules29010 URL: https://www.mdpi.com/1420-3049/29/1/174.

[25] Rong Zeng, Zhi Li, Jia Li, and Qing Zhang. "DNA promoter task-oriented dictionary mining and prediction model based on natural language technology". In: *Scientific Reports* 15.1 (Jan. 2025), p. 153. DOI: 10.1038/s41598-024-84105-9.

[26] Ting Zhang, Chaoran Yin, Aleksandr Fedorov, Liangjun Qiao, Hongliang Bao, Nazar Beknazarov, Shiyu Wang, Avishekh Gautam, Riley M Williams, Jeremy Chase Crawford, et al. In: *Nature* 606.7914 (2022), pp. 594–602. DOI: 10.1038/s41586-022-04753-7.

[27] Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. *DNABERT-2: Efficient Foundation Model and Benchmark For Multi-Species Genome*. 2023. arXiv: 2306.15006 [q-bio.GN].

6 Appendix

Algorithm 2 SequenceClassificationExplainer Attribution Aggregation

procedure ExtractTopKmers(model, tokenizer, dataset, k)
    $TP \leftarrow []$ ▷ true positives
    for all example in dataset.dev **do**
        $seq \leftarrow example.sequence$
        $inputs \leftarrow tokenizer(seq)$ on device
        $logits \leftarrow model(inputs)$.logits
        $pred, true \leftarrow \arg\max(logits), example.label$
        if $pred = 1$ **and** $true = 1$ **then**
            TP.append(seq)
        end if
    end for
    $scores \leftarrow$ ▷ map k-mer to list of scores
    for all seq in TP **do**
        $attrs \leftarrow explainer(seq, n_steps = 50)$
        $(tokens, vals) \leftarrow$ filterSpecials($attrs$)
        for $i = 1$ to $|tokens| - k + 1$ **do**
            $kmer \leftarrow tokens[i : i + k]$ as string
            $s \leftarrow \text{mean}(vals[i : i + k])$
            $scores[kmer]$.append(s)
        end for
    end for
    $avgScores \leftarrow (kmer, \text{mean}(scores[kmer]))$
    $sorted \leftarrow \text{sort}_{\text{d}}\text{esc}(avgScores)$
    return sorted
end procedure

Algorithm 3 SmoothGrad-Based K-mer Attribution

procedure RankKmersSmoothGrad(model, tokenizer, dataset, k)
  move model to GPU and set to eval mode
  $TP \leftarrow \emptyset$ ▷ collect true positives
  for all example in dataset.dev **do**
    $seq \leftarrow example$.sequence
    $enc \leftarrow tokenizer(seq)$ on device
    $pred \leftarrow \arg\max(model(enc).logits)$
    if $pred = 1$ and example.label= 1 **then**
      add seq to TP
    end if
  end for
  $scores \leftarrow \{\}$ ▷ map each k-mer to list of attributions
  for all $seq \in TP$ **do**
    $enc \leftarrow tokenizer(seq)$ on device
    $ids \leftarrow enc.input\_ids,\ mask \leftarrow enc.attention\_mask$
    $pred \leftarrow \arg\max(model(enc).logits)$
    $E \leftarrow embedding\_layer(ids)$
    $B \leftarrow \mathbf{0}$ tensor shaped like E
    $A \leftarrow \text{NoiseTunnel(IntegratedGradients)}(E, B, \text{target} = pred, \text{nt\_samples} = 50, \text{stdevs} = 0.02, \text{args} = (mask,))$
    $attr \leftarrow \sum A$ over embedding dim
    $tokens \leftarrow tokenizer.convert\_ids\_to\_tokens(ids)$
    remove special tokens from $tokens$ and $attr$
    for $i = 1$ to $|tokens| - k + 1$ **do**
      $kmer \leftarrow \text{join}(tokens[i : i + k])$
      $s \leftarrow \text{mean}(attr[i : i + k])$
      append s to $scores[kmer]$
    end for
  end for
  $avgScores \leftarrow \{(k, m = \text{mean}(scores[k]))\}$
  return sorted $avgScores$ in descending order
end procedure

Algorithm 4 HyenaDNA Token-Level Z-DNA k-mer Extraction IG

procedure ExtractZDNAKmers(model, dataset, K)
    Initialize empty maps *kmerCounts* and *kmerScoreSums*
    for all sample in dataset **do**
        $seq \leftarrow$ sample.sequence
        $ids,\ mask \leftarrow$ sample.input_ids, sample.attention_mask
        $labels \leftarrow$ sample.labels
        $embeds \leftarrow$ model.get_input_embeddings()(ids)
        $logits \leftarrow$ model$(embeds,\ mask)$.`logits`
        $preds \leftarrow \arg\max(logits,\ \text{axis} = -1)$
        $tpMask \leftarrow (preds = 1)\ \wedge\ (labels = 1)$ ▷ float mask
        if $\sum tpMask = 0$ **then**
            continue
        end if
        Define $forwardTP(e, m) = \sum\big(\text{model}(e, m)\text{.}\texttt{logits}[..., 1] \times tpMask\big)$
        `ig.forward_func` $\leftarrow forwardTP$
        $(attrs, \delta) \leftarrow$ IntegratedGradients.attribute$\big(e = embeds,$ baselines $= 0,$ additional_forward_args $= (mask),\ n\_steps = 50,$ return_convergence_delta $= True\big)$
        $scores \leftarrow \sum(attrs,\ \text{axis} = -1)$
        for $i = 1$ **to** $|seq| - K + 1$ **do**
            if all $labels[i : i + K] = 1$ and $preds[i : i + K] = 1$ **then**
                $kmer \leftarrow seq[i : i + K]$
                $kmerCounts[kmer]$ += 1
                $kmerScoreSums[kmer]$ += $\sum(scores[i : i + K])$
            end if
        end for
    end for
    Build table of $\{kmer,\ Count,\ AvgScore = \frac{kmerScoreSums[kmer]}{kmerCounts[kmer]}\}$
    return table sorted by *AvgScore* descending
end procedure

Algorithm 5 SmoothGrad-Saliency Z-DNA k-mer Extraction

procedure ExtractZDNAKmersSmoothGrad(model, dataset, K)
  Initialize empty maps *kmerCounts* and *kmerScoreSums*
  for all sample in dataset **do**
    $seq \leftarrow$ sample.sequence
    $ids,\ mask \leftarrow$ sample.input_ids, sample.attention_mask
    $labels \leftarrow$ sample.labels
    $embeds \leftarrow$ model.get_input_embeddings()(ids)
    $logits \leftarrow$ model($embeds,\ mask$).`logits`
    $preds \leftarrow \arg\max(logits, \text{axis} = -1)$
    $tpMask \leftarrow (preds = 1) \wedge (labels = 1)$ ▷ float mask
    if $\sum tpMask = 0$ **then**
      continue
    end if
    Define $forwardTP(e, m) = \sum\big(\text{model}(e, m).\texttt{logits}[..., 1] \times tpMask\big)$
    `saliency.forward_func` $\leftarrow forwardTP$
    `nt` $\leftarrow$ NoiseTunnel(`saliency`)
    $attrs \leftarrow$ nt.attribute$\big($inputs $= embeds,$ nt_type $=$ `'smoothgrad'`, nt_samples $= 50,$ nt_samples_batch_size $= 10,$ stdevs $= 0.1,$ additional_forward_args $= (mask)\big)$
    $scores \leftarrow \sum(attrs, \text{axis} = -1)$
    for $i = 1$ **to** $|seq| - K + 1$ **do**
      if all $labels[i : i + K] = 1$ and $preds[i : i + K] = 1$ **then**
        $kmer \leftarrow seq[i : i + K]$
        $kmerCounts[kmer]$ += 1
        $kmerScoreSums[kmer]$ += $\sum(scores[i : i + K])$
      end if
    end for
  end for
  Build table of $\{kmer,\ Count,\ AvgScore = \frac{kmerScoreSums[kmer]}{kmerCounts[kmer]}\}$
  return table sorted by *AvgScore* descending
end procedure

Algorithm 6 Count k-mer True-Positive Importance

procedure ComputeKmerImportance(model, dataset, K)
    Initialize empty maps *tpCounts* and *occCounts*
    for all sample index in $0, \ldots, |\text{dataset}| - 1$ **do**
        seq $\leftarrow$ dataset[index].seq
        labels $\leftarrow$ dataset[index].labels
        Prepare *inputs*, *mask* from sample and move to device
        with no_grad(): *logits* $\leftarrow$ model(*inputs*, *mask*).`logits`
        preds $\leftarrow \arg\max(\textit{logits}, \text{axis} = -1)$
        $L \leftarrow |\textit{seq}|$
        for $i = 0$ **to** $L - K$ **do**
            $\textit{kmer} \leftarrow \textit{seq}[i : i + K]$
            $\textit{occCounts}[\textit{kmer}] \mathrel{+}= 1$
            $\textit{windowLabels} \leftarrow \textit{labels}[i : i + K]$
            $\textit{windowPreds} \leftarrow \textit{preds}[i : i + K]$
            if all entries of *windowLabels* and *windowPreds* equal 1 **then**
                $\textit{tpCounts}[\textit{kmer}] \mathrel{+}= 1$
            end if
        end for
    end for
    Initialize empty list `records`
    for all $(\textit{kmer}, \textit{total})$ in *occCounts* **do**
        $\textit{tp} \leftarrow \textit{tpCounts}.\text{get}(\textit{kmer}, 0)$
        $\textit{importance} \leftarrow \textit{tp}/\textit{total}$
        Append to `records` $\{\textit{kmer}, \textit{tp}, \textit{total}, \textit{importance}\}$
    end for
    return `records` sorted by `importance` descending
end procedure