StackBERT-Enhancer —

A Dual-Layer BERT-Based Framework for Enhancer Identification and Strength Classification in Genomic Data

Phat Tran

A thesis
submitted in partial fulfillment of the
requirements of the degree of

Master of Science

University of Washington
2025

Committee:
Wooyoung Kim
Dong Si
Afra Mashhadi

Program Authorized to Offer Degree:
Computer Science and Software Engineering

University of Washington

Abstract

StackBERT-Enhancer — A Dual-Layer BERT-Based Framework for Enhancer Identification and Strength Classification in Genomic Data

Phat Tran

Chair of the Supervisory Committee:
Wooyoung Kim
Division of Computing and Software Systems

Accurately identifying and classifying crucial regulatory DNA sequences known as enhancers is a significant challenge, as traditional computational methods often struggle with their complex, context-dependent nature and lack interpretability. This thesis introduces StackBERT-Enhancer, a novel deep learning framework to address these limitations, focusing on two primary tasks: distinguishing enhancer sequences from non-enhancer sequences and classifying identified enhancers by their activity levels. The proposed framework employs multiple transformer-based language models, each independently trained on DNA sequences tokenized with different k-mer sizes, allowing for the capture of sequence dependencies across various scales. These individual models are then integrated into a stacking ensemble architecture, which significantly boosts classification accuracy, robustness, and generalization, achieving state-of-the-art results of 83.5% in enhancer identification and 99.0% in enhancer strength classification. The framework utilizes distributed multi-GPU systems for efficient model training and incorporates interpretability techniques such as SHapley Additive exPlanations (SHAP) for feature importance and attention score analysis for sequence motif discovery, bridging predictive power with biological insight. This advanced approach offers a robust and interpretable tool for enhancer analysis, holding strong potential for applications in disease modeling and broader biomedical research.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Wooyoung Kim, for her invaluable guidance, encouragement, and unwavering support throughout my research journey. Her insights and expertise have been instrumental in shaping this study, and I am truly grateful for her mentorship.

I would like to extend my heartfelt appreciation to my friends and peers at the University of Washington. Their friendship, support, insightful conversations, and shared experiences have enriched my learning in ways that words cannot fully capture.

To my family, I am profoundly grateful for your unfailing love, patience, and encouragement, and for always being there for me. Your belief in me has been a constant source of strength and motivation, and I am truly thankful for your presence in my life.

This work is built upon the encouragement, support, and wisdom of many. To everyone who played a part in this chapter of my life, thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Enhancers are crucial regulatory *deoxyribonucleic acid* (DNA) sequences that orchestrate gene expression, often exerting their influence over long genomic distances to modulate transcriptional activity [1]. These elements are fundamental to complex biological processes, including cellular differentiation, organismal development, and responses to environmental cues. Despite their established importance, the accurate identification and functional classification of enhancers remain significant hurdles in genomics [2]. Enhancer activity is highly variable and context-dependent, modulated by factors such as chromatin accessibility, transcription factor binding patterns, and epigenetic modifications [3]. This dynamic nature makes distinguishing enhancers from non-regulatory sequences, and further categorizing them by functional strength (e.g., strong versus weak enhancers), particularly challenging due to their often subtle sequence features. Overcoming these challenges is essential for deepening the understanding of gene regulation, cellular function, and the molecular underpinnings of numerous diseases.

Problem Statement

Historically, computational efforts to identify enhancers have relied on sequence-based heuristics[1] or shallow machine learning models[2]. While providing valuable initial insights, these methods often achieve only moderate success because they struggle to capture the complex, long-range dependencies and multi-scale contextual information inherent in genomic sequences [2]. Furthermore, many traditional and early deep learning approaches, which might use fixed k-mer representations or standard convolutional filters, may not fully model the hierarchical nature of regulatory information. A significant drawback of many existing models is their lack of interpretability; they often function as "black boxes", making it difficult to extract biologically meaningful insights, such as identifying the specific sequence motifs that dictate enhancer function [4]. Consequently, there is a clear need for more sophisticated computational frameworks that can address these limitations in both predictive power and biological insight.

Recent advances in *artificial intelligence* (AI), particularly large language models, such as transformer-based architectures like *bidirectional encoder representations from transformers* (BERT), offer a transformative opportunity for genomic analysis [5]. These models excel at learning contextual relationships and long-range dependencies, making them particularly well-suited for tasks involving genomic sequences.

However, applying large language models to enhancer analysis introduces several challenges [6]. First, DNA sequences do not possess the linguistic structure of natural language, necessitating the development of effective tokenization strategies to capture biologically relevant patterns. Second, enhancers exhibit regulatory activity across multiple scales of sequence context, requiring models that can integrate information from varying k-mer sizes. Third,

[1]Sequence-based heuristics are computational methods that use simple rules or patterns derived from biological sequences to efficiently solve problems like alignment or annotation, often trading some accuracy for speed and practicality.

[2]Shallow machine learning models are simple algorithms, like linear regression, designed for smaller datasets and lower feature complexity, offering efficiency and interpretability.

while large language models are powerful predictive tools, their "black-box" nature poses challenges for biological interpretability, particularly in identifying sequence motifs associated with enhancer activity [4]. Fourth, training these models on genomic data is computationally intensive, necessitating the use of distributed multi-GPU systems to optimize hyperparameters and scale model training efficiently [7]. Finally, combining predictions from multiple base models into an ensemble framework introduces additional complexity but holds promise for improving predictive accuracy and robustness [8].

Proposed Approach

This study introduces StackBERT-Enhancer, a novel framework designed to address these challenges for improved enhancer identification and classification using large language models. The approach involves splitting DNA sequences into k-mers to capture multi-scale contextual information. Each k-mer representation is processed using large language models trained to distinguish enhancers from non-enhancers and further classify enhancers into strong or weak categories. To improve biological interpretability, *SHapley Additive exPlanations* (SHAP) are employed for feature importance analysis from the trained models. Additionally, attention scores derived from the models are leveraged for motif discovery, enabling the identification of key sequence patterns and understanding their contribution to enhancer function. The training process is accelerated through distributed multi-GPU systems to optimize hyperparameters and handle the computational demands of large-scale genomic datasets. Additionally, a stacking ensemble strategy is employed to combine predictions from individual base models into a robust framework that enhances accuracy and generalizability.

The outcomes of this research have the potential to significantly advance both computational biology and genomics by providing a scalable and interpretable framework for enhancer analysis. By integrating cutting-edge machine learning techniques with biological insights, this work aims to not only improve the accuracy of enhancer classification but

also contribute to a deeper understanding of gene regulatory mechanisms. This could lead to new treatments for diseases that involve enhancer-controlled gene regulation.

Specifically, the objectives of this research are as follows:

- Enhancer Identification and Classification: To accurately distinguish enhancers from non-enhancers based on DNA sequence features and further classify identified enhancers into strong or weak categories.
- Optimization of Model Training: To implement distributed multi-GPU training for efficient handling of large-scale genomic datasets and optimization of hyperparameters.
- Development of an Ensemble Framework: To apply stacking techniques for combining predictions from multiple base models into a single robust ensemble model that enhances accuracy and generalizability.
- Evaluation Against State-of-the-Art Methods: To benchmark the proposed framework against existing methods for enhancer identification and classification, using established metrics.
- Feature Importance Analysis with SHAP: To employ SHAP to analyze the importance of features from the base models, identifying key contributors and understanding how the ensemble leverages multi-scale information for enhancer classification.
- Motif Discovery through Attention Mechanisms: To utilize the attention scores generated by the models for identifying sequence motifs that are biologically relevant to enhancer function. This objective seeks to enhance the interpretability of the models by linking predictive performance with underlying biological mechanisms.
- Biological Validation: To validate the discovered motifs and enhancer classifications against experimental datasets or established databases (e.g., STREME), ensuring that the computational predictions align with known biological evidence.

Contributions

The unique contributions of this research lie in the development of StackBERT-Enhancer, a novel computational framework that significantly advances the field of enhancer analysis through its scalability and sophisticated ensemble learning strategy. First, it introduces a highly scalable training pipeline leveraging distributed multi-GPU systems, enabling efficient processing of large-scale genomic data and facilitating extensive hyperparameter optimization, a crucial aspect often limiting the applicability of large language models in genomics. Second, it pioneers the use of a dual-layer stacking ensemble of large language models trained on multi-scale k-mer representations. This architecture allows for a more comprehensive integration of contextual information and diverse model perspectives, potentially leading to superior predictive performance compared to single models or simpler ensemble methods. Finally, by integrating advanced interpretability methods, including attention mechanisms for motif discovery and SHAP for feature importance analysis, within this scalable and high-performing framework, StackBERT-Enhancer offers a pathway towards more comprehensive, interpretable, and biologically meaningful insights into the regulatory code of the genome.

Chapter 2

Literature Review

This chapter surveys the current research landscape on genomic regulatory elements, specifically enhancers, and the evolution of machine learning techniques used for their identification and classification. It traces the progression from traditional machine learning to advanced deep learning architectures, including transformers and large language models, highlighting their contributions and challenges.

2.1 Genomic Regulatory Elements

Genomic regulatory elements are essential components of the genome that control gene expression by influencing transcriptional activity. This activity refers to the process by which *ribonucleic acid* (RNA) polymerase binds to DNA and synthesizes RNA, copying the genetic information from DNA into RNA as the first step of gene expression [9]. These elements are broadly classified into two types: *cis*-regulatory elements and *trans*-regulatory elements, based on their mode of action [10]. Together, they ensure genes are activated or silenced at the right time and place, helping cells grow, change, and maintain balance.

Cis-regulatory elements are DNA sequences located on the same chromosome as the gene they regulate [11]. They directly interact with transcription factors and other proteins to

Table 2.1: Roles of key DNA *cis*-regulatory elements in gene expression.

Element	**Typical Location**	**Main Function**	**Example**	**Note**
Promoter	Upstream of gene	Initiates transcription; recruits RNA polymerase	TATA box, CAAT box	Essential for activation; multiple binding sites
Enhancer	Far from gene, any orientation	Increases transcription rate	SV40 enhancer, LCR	Regulates multiple genes; long-range action
Silencer	Variable, any orientation	Decreases/blocks transcription	NRSE	Can override enhancers; tissue-specific
Insulator	Between enhancer/promoter; variable position	Blocks enhancer-promoter interaction; defines boundaries	CTCF sites	Prevents unwanted activation

Table 2.2: Comparison of *cis*- and *trans*-regulatory elements.

Feature	***Cis*-regulatory**	***Trans*-regulatory**
Location	Same DNA molecule	Different DNA molecule
Mode of Action	Direct binding to DNA	Indirect via diffusible factors
Examples	Promoters, enhancers	Transcription factors, microRNAs
Specificity	Local (allele-specific)	Global (affects multiple genes)
Evolutionary Impact	Phenotypic diversity	Broad regulatory divergence

modulate transcription. Key examples include promoters, which are positioned near the transcription start site and recruit RNA polymerase to initiate transcription; enhancers, which amplify transcriptional activity from a distance regardless of orientation; silencers, which repress gene expression by recruiting repressor proteins or altering chromatin structure; and insulators, which act as boundaries to prevent enhancers from interacting with non-target promoters [12]. These elements often function through DNA looping and bringing in helper proteins to ensure accurate gene control. Table 2.1 provides a summary of key DNA cis-regulatory elements and their roles in gene expression.

In contrast, *trans*-regulatory elements operate indirectly by producing diffusible factors such as proteins or RNAs that regulate gene expression [13]. These include transcription factors, which bind to *cis*-regulatory elements to activate or repress transcription, and non-coding RNA (ncRNAs) like microRNAs (miRNAs) or long non-coding RNAs (lncRNAs), which influence gene expression post-transcriptionally or epigenetically. Trans-regulatory elements enable cross-talk between different parts of the genome and fine-tune gene regulation across various cellular contexts. Table 2.2 illustrates the fundamental differences between *cis*- and *trans*-regulatory elements.

Beyond these classifications, other regulatory mechanisms also contribute to genomic control. For instance, transposable elements (TEs), once considered "junk DNA", have emerged as significant contributors to genetic diversity and regulatory innovation [14]. TEs can influence nearby genes by serving as enhancers or silencers, while host genomes often suppress their activity through epigenetic modifications to maintain stability.

The interplay between *cis*- and *trans*-regulatory elements underscores the complexity of gene regulation. Advances in genomic technologies, such as high-throughput sequencing and chromatin interaction assays, have significantly enhanced the understanding of the roles these elements play in health and disease [13].

Enhancer

Enhancers, a type of *cis*-regulatory element, are short, non-coding DNA sequences[1], typically 50–1500 base pairs, that regulate gene expression by increasing the likelihood of transcription [16]. A base pair refers to two complementary nucleotides on opposite strands of a DNA molecule that are bonded together. The pairing is based on specific rules: adenine (A) pairs with thymine (T) through two hydrogen bonds, and cytosine (C) pairs with guanine (G) through three hydrogen bonds [17]. These base pairs are fundamental units of the double helix structure of DNA.

To understand their role in gene regulation, it is essential to first explore the fundamental process of transcription. Transcription is the fundamental process in molecular biology where the genetic information encoded in DNA is copied into a functional RNA molecule, most commonly messenger RNA (mRNA) [18]. This process is carried out by an enzyme called RNA polymerase, which binds to a specific region of the DNA called the promoter and then unwinds the DNA double helix [19]. As the RNA polymerase moves along the DNA template strand, it synthesizes a complementary RNA molecule by adding nucleotides. The resulting RNA molecule carries the genetic instructions from the DNA in the nucleus to the ribosomes in the cytoplasm, where these instructions are used to synthesize proteins during translation.

In computer science, this is similar to compiling a specific function or class (the gene) within a large program (DNA source code) using a compiler (like RNA polymerase) that recognizes the start of the function definition (like the promoter). This compilation process can be influenced by other parts of the code called enhancers. Enhancers are like separate configuration settings or optimization directives located elsewhere in the program, some-

[1]Non-coding DNA sequences are segments of DNA that do not directly code for proteins. Instead, they play various roles in regulating gene activity, such as controlling when and where genes are turned on or off. Some non-coding DNA also contributes to maintaining chromosome structure and integrity, for example, by forming protective ends called telomeres. While once considered "junk", non-coding DNA is now recognized as essential for many cellular functions [15].

times quite far away from the function. When specific system tools or flags (analogous to transcription factors) interact with these enhancers, they can remotely boost the activity of the compiler at the promoter, increasing the rate at which the function is compiled. The resulting compiled module (RNA) is then transferred from the source code environment (like the nucleus) to the processor (like ribosomes) for execution (translation) to build the final output (protein).

Given that transcription relies on accessibility to DNA, chromatin plays a crucial role in regulating this process. Chromatin refers to the complex of DNA and proteins that constitutes chromosomes within the nucleus of eukaryotic cells. Its primary function is to package the long DNA molecules into a more compact, denser structure, preventing tangling and damage while also regulating gene expression. The dynamic organization of chromatin, ranging from loosely packed euchromatin (generally associated with active transcription) to tightly packed heterochromatin (often associated with gene silencing), plays a crucial role in controlling the accessibility of DNA to the cellular machinery involved in processes like transcription and replication.

Enhancers further influence transcription by facilitating interactions between regulatory elements and target genes. They can activate transcription over long genomic distances, sometimes up to 1 million base pairs away, and function independently of their orientation or position relative to the target gene. Enhancers achieve this by recruiting transcription factors and other proteins that facilitate chromatin looping, bringing the enhancer into physical proximity with the gene's promoter. This interaction is often mediated by proteins such as cohesin and CTCF, which stabilize the loop structure. Their activity is influenced by factors such as chromatin structure (how DNA is packaged with proteins, affecting its accessibility), the binding of transcription factors, and chemical modifications to histones (proteins around which DNA is wrapped, playing a role in gene regulation). For example, histone acetylation (e.g., H3K27ac) and methylation (e.g., H3K4me1) are common markers of active enhancers [20]. Enhancers can be further categorized into subtypes

such as strong, weak, poised, or inactive enhancers based on their chromatin features and activity levels.

Figure 2.1: Diagram illustrating the basic concept of an enhancer's function. This model abstracts away chromatin structure and focuses solely on the interaction between an enhancer, genes, and insulators through DNA looping. Enhancers activate genes over long distances, while insulators ensure specificity by blocking interactions with non-target gene.

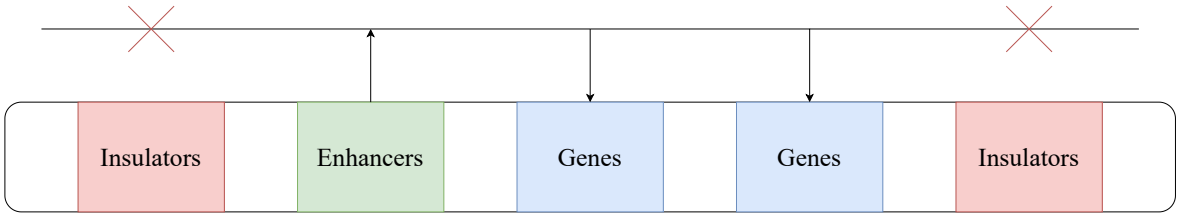
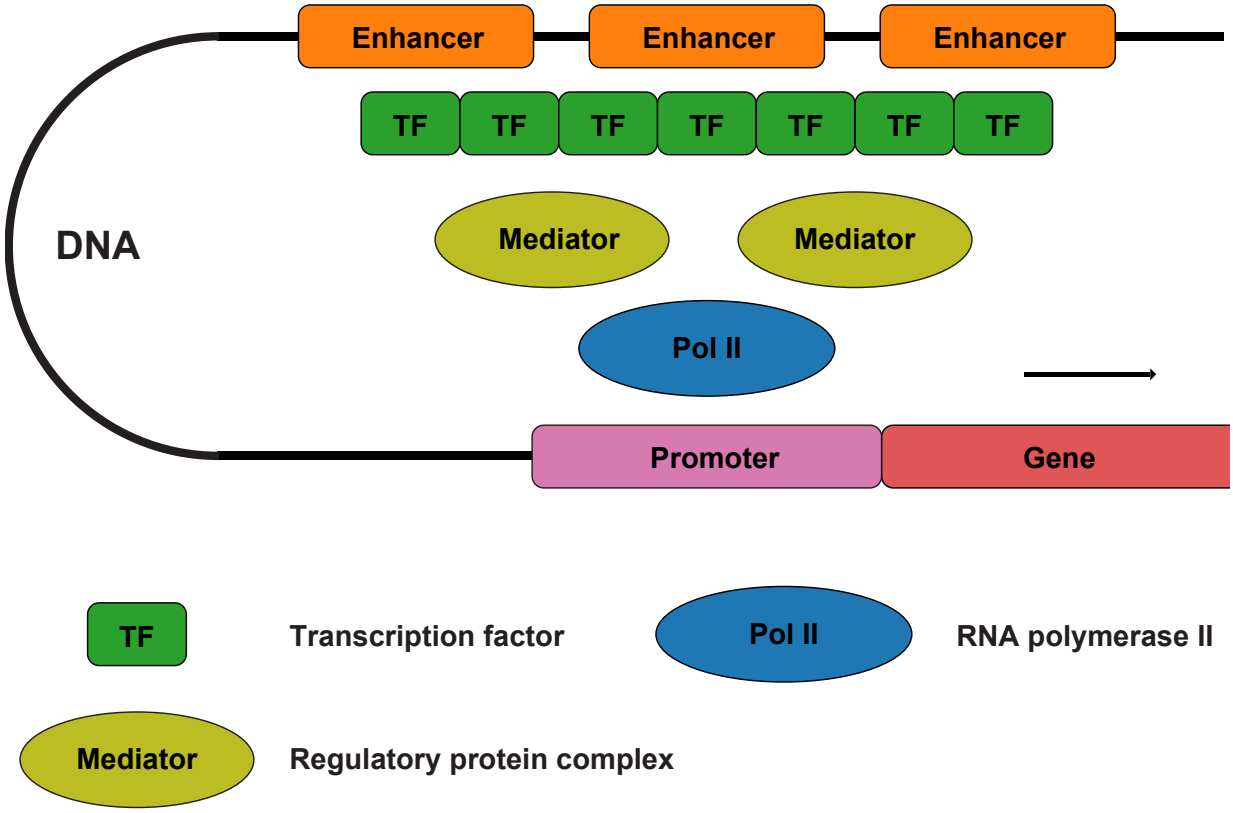


Figure 2.1 represents a highly simplified model of enhancer function, focusing on DNA looping and omitting chromatin's role. Enhancers are DNA sequences that regulate transcription by interacting with gene promoters, often over long distances. In this idealized depiction, DNA loops bring enhancers into proximity with their target promoters to activate transcription. Insulators are included as boundary elements that block enhancers from interacting with non-target genes, ensuring specificity.

Figure 2.2 illustrates how specific regions of DNA called enhancers help control gene activity, based on [21]. Imagine DNA as a long instruction manual. A gene is a specific set of instructions within that manual. Enhancers act like spotlights or amplifiers located some distance away from the gene they regulate. For a gene to be activated (or "turned on"), specialized proteins called transcription factors (TFs) first bind to these enhancer regions. The DNA then loops, bringing the enhancers and their bound TFs physically closer to the target gene. A crucial intermediary, the mediator complex (a collection of regulatory proteins), then bridges the gap, connecting the TFs at the enhancer site to another protein complex called RNA Polymerase II (Pol II), which is positioned at the start of the gene. Pol II is the machine responsible for reading the gene's instructions. This interaction, fa-

Figure 2.2: Mechanism of gene activation. Far away on the DNA are special areas called enhancers. Specific transcription factors (TFs) attach to these enhancers. When they do, they help bring a big protein group called the mediator to the gene. The mediator, in turn, helps to recruit and position RNA polymerase II (Pol II) at the promoter, a regulatory region directly preceding the gene. Once Pol II is in place, it can read the gene's instructions and make a copy in the form of RNA, meaning gene is now active.



cilitated by the mediator, essentially gives Pol II the "go" signal, activating the gene and allowing the cell to carry out the instructions encoded within it.

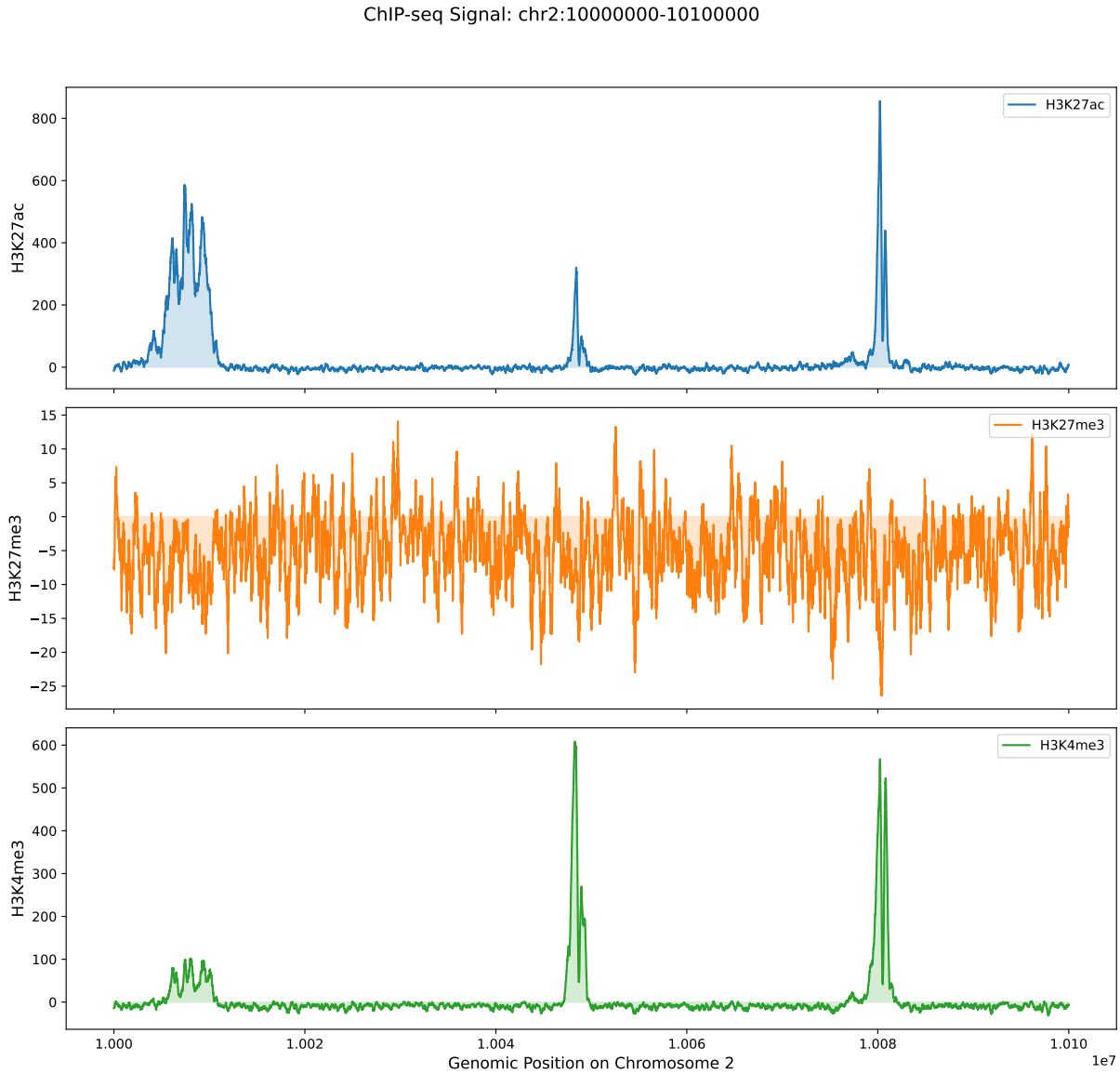
In computer science terms, think of DNA as source code and a gene as a specific function. Transcription is the process where RNA Polymerase (the compiler) reads the function, starting at the promoter (its entry point). Enhancers are like remote optimization directives elsewhere in the code. When specific system tools (transcription factors) interact with an enhancer, a mediator links them to the compiler at the promoter, signaling it to increase the compilation rate of that function. This effectively "activates" or boosts the function's processing.

Experimental Approaches

Identifying enhancers is challenging because their sequences vary greatly and their activity depends on the context, such as the cell type or environmental conditions. Experimental techniques like chromatin immunoprecipitation followed by sequencing (ChIP-seq) and assay for transposase-accessible chromatin using sequencing (ATAC-seq) are commonly used to study enhancers [22], [23]. ChIP-seq helps detect specific protein-DNA interactions and histone modifications, providing detailed insights into enhancer activity. However, it can be expensive and less scalable. On the other hand, ATAC-seq identifies open regions of chromatin with high sensitivity, offering a broader view of accessible DNA but without directly identifying protein-DNA interactions. These methods are complementary and often used together to improve the accuracy of enhancer identification.

Figure 2.3 displays ChIP-seq signal tracks for three distinct histone modifications - H3K27ac (top panel, blue), H3K27me3 (middle panel, orange), and H3K4me3 (bottom panel, green) - across a 100 kilobase region (`chr2:10000000-10100000`) in *Mus musculus*. This data is derived from an experiment investigating the role of Ki-67 in regulating histone modifications in 4T1 mouse mammary carcinoma cells (GSE163479) [24]. The x-axis represents the

Figure 2.3: ChIP-seq analysis reveals distinct patterns of histone modifications within a 100kb region of Chromosome 2 (`chr2:10000000-10100000`). The plots illustrate the signal intensity for three key histone marks associated with different chromatin states: H3K27ac (active transcription), H3K27me3, (gene repression) and H3K4me3 (transcription initiation). The x-axis denotes the genomic coordinates, and the y-axis reflects the relative abundance of each modification, providing insights into the regulatory landscape of this genomic region.



genomic position, while the y-axis indicates the level of enrichment for each histone mark. Notably, H3K27ac and H3K4me3, which are generally associated with active regulatory elements like enhancers and promoters, show regions of strong enrichment, particularly around positions 1.001e7, 1.005e7, and 1.008e7. Conversely, H3K27me3, a mark associated with gene repression, exhibits a relatively low and fluctuating signal across this region, with no prominent peaks. The co-occurrence of high H3K27ac and H3K4me3 signals suggests the presence of active regulatory elements at these genomic locations, potentially indicating the promoters and/or enhancers of actively transcribed genes within this chromosomal segment. The absence of strong H3K27me3 enrichment in these same regions further supports their active regulatory state.

Categories

Shlyueva et al., through their review of genome-wide studies such as histone modification analysis, emphasized that enhancers represent a broad category of functional elements with varied subgroups, including poised and latent enhancers, in addition to the strong and weak types [25]. Despite this diversity, a key classification based on regulatory strength differentiates strong enhancers from weak ones. This classification remains widely recognized and applied in genomics and bioinformatics, where both computational and experimental approaches are used to distinguish these two primary types.

Strong enhancers are DNA elements that robustly increase the transcriptional activity of their target genes. They are characterized by high levels of transcription factor binding, active chromatin marks (e.g., H3K27ac), and pronounced chromatin accessibility. Strong enhancers tend to have a more significant impact on gene expression, often being essential for the activation of key developmental or cell-type-specific genes [26], [27].

Weak enhancers, in contrast, exhibit lower levels of transcriptional activation and are often marked by more modest chromatin features. While they may still bind transcription

factors and possess enhancer-associated histone modifications (e.g., H3K4me1), their overall contribution to gene expression is less pronounced compared to strong enhancers [28]. Weak enhancers may act redundantly or synergistically with other enhancers, and their individual deletion often results in subtle or context-dependent changes in gene expression.

Recent studies have applied machine learning models to classify enhancers into strong and weak categories based on sequence composition, chromatin accessibility, and histone modification data. For example, iEnhancer-EBLSTM and EnhancerPred are computational tools that use two-layer classification frameworks: the first layer distinguishes enhancers from non-enhancers, and the second layer further separates strong enhancers from weak ones [29], [30].

2.2 Machine Learning in Genomics

Machine learning (ML) has become a powerful tool for analyzing genomic sequences and predicting regulatory elements. Early computational approaches to enhancer classification relied heavily on biological experimental techniques, such as DNase I hypersensitivity site sequencing to identify regions of open chromatin[2], ChIP-seq data for histone modifications (e.g., H3K4me1, H3K27ac), and transcription factor binding sites[3]. For instance, models like EnhancerFinder utilized ML techniques to predict enhancers based on sequence features [33], while GKM-SVM employed *support vector machines* (SVMs) using k-mer-based features for classification [34]. Another method, DEEP, utilized deep learning for enhancer prediction but focused solely on identifying enhancers rather than distinguishing subtypes like strong and weak enhancers [35]. These early approaches were limited in their ability to classify enhancers beyond a binary framework.

[2]Open chromatin refers to regions of DNA that are accessible to regulatory proteins, such as transcription factors, due to the absence or displacement of nucleosomes [31].

[3]Transcription factor binding sites are short DNA sequences (5–20 base pairs long) where transcription factors bind to control the activation or repression of nearby genes [32]. These sites are typically located in promoter regions near the transcription start site or in enhancer regions farther away, and their binding helps recruit other proteins, such as RNA polymerase, to initiate transcription.

These early approaches faced several challenges that limited their effectiveness. Kleftogiannis et al. examined over 30 enhancer recognition tools developed between 2000 and 2015, evaluated the use of high-throughput experimental data, discussed obstacles such as data scarcity and overfitting, and provided guidelines for advancing future research [2]. Many models treated enhancer prediction as a binary classification problem (enhancer vs. non-enhancer), which restricted their ability to classify enhancers into functional subtypes. Furthermore, these methods relied heavily on experimental datasets like ChIP-seq and DNase I hypersensitivity data, which are labor-intensive, costly, and often tissue-specific, thereby limiting the generalizability of the models across different cell types and conditions. Additionally, enhancer datasets frequently exhibited class imbalance, with far fewer enhancer samples compared to non-enhancer samples, which affected model performance and generalization. Overfitting was another significant issue, as many models struggled to generalize beyond the specific datasets they were trained on.

To overcome these challenges, Liu et al. developed iEnhancer-2L in 2016 as a two-layer framework that integrates SVMs with feature extraction techniques [36]. This model extracts physicochemical properties and k-mer features from DNA sequences to enhance the precision of enhancer classification. By leveraging these diverse features, iEnhancer-2L improves the ability to identify enhancers with greater accuracy compared to earlier methods. Its two-layer structure ensures a more refined analysis of sequence data, making it a robust tool for enhancer prediction. Additionally, the study proposed a new benchmark dataset, which is also used in this study, to further support the evaluation and improvement of enhancer prediction models.

Since the development of iEnhancer-2L, several advanced models have emerged to further enhance enhancer identification and classification. Among these, EnhancerPred improves on the foundational SVM approach by incorporating optimized sequence-derived features, such as nucleotide composition and dinucleotide properties, to better distinguish enhancers

Table 2.3: Support vector machine-based enhancer classification models.

Model	Key Features	Architecture
iEnhancer-2L	Two-layer SVM; physicochemical properties, k-mer frequency.	SVM
EnhancerPred	Optimized sequence features: nucleotide composition, di-/tri-nucleotide properties, hybrid features.	SVM
iEnhancer-EL	Ensemble SVMs; k-mer composition, physicochemical properties, position-specific features.	Ensemble SVMs
iEnhancer-5Step	Five-step framework; word embedding, k-mer features, feature selection.	SVM

from non-enhancers [30]. On the other hand, iEnhancer-EL can be seen as an enhanced version of iEnhancer-2L, as it employs ensemble learning techniques combined with SVM classifiers [37]. By integrating multiple SVMs trained on diverse feature sets, iEnhancer-EL leverages complementary information from various perspectives. This ensemble-based strategy not only improves prediction accuracy but also ensures greater reliability and consistency in identifying enhancers, setting a new standard in enhancer classification. Subsequently, iEnhancer-5Step utilized Chou's 5-step rule, a standard framework for developing and rigorously validating computational biology prediction models, alongside word embedding techniques to analyze DNA sequences [38]. This model employed SVM-based classifiers in a two-layer structure to predict both enhancer presence and their strength.

In addition to SVM-based models, other ML models have also demonstrated significant advancements. One such model, iEnhancer-MFGBDT, employs a Gradient Boosting Decision Tree (GBDT) framework to classify enhancers and their strength [39]. This model integrates multiple feature sets, including k-mer and reverse complement k-mer nucleotide composition, second-order moving average, normalized Moreau-Broto auto-cross correlation, and Moran auto-cross correlation derived from dinucleotide physical structural properties. Another significant advancement came with the development of iEnhancer-XG,

which utilized XGBoost as its base classifier [40]. This model integrated five feature extraction methods: k-Spectrum Profile, Mismatch k-tuple, Subsequence Profile, Position-Specific Scoring Matrix (PSSM), and Pseudo Dinucleotide Composition (PseDNC). Additionally, iEnhancer-XG incorporates SHAP for interpretability, addressing the "black-box" nature of ML models. Another noteworthy development is iEnhancer-MRBF, which introduces a novel two-layer framework where the first layer identifies enhancers and the second layer classifies them into strong or weak enhancers based on their strength [41]. This model employs a multiple Laplacian-regularized radial basis function network (MLR-RBFN) as its classifier and uses three feature representation methods: k-mer, nucleotide binary profiles (NBP), and accumulated nucleotide frequency (ANF).

Table 2.4: Advanced machine learning-based enhancer classification models.

Model	**Key Features**	**Architecture**
iEnhancer-MFGBDT	Sequence features: k-mer, reverse complement k-mer, moving average, autocorrelations.	GBDT
iEnhancer-XG	Diverse features: k-spectrum, mismatch k-tuple, subsequence profile, PSSM, pseudo dinucleotide composition.	XGBoost
iEnhancer-MRBF	Sequence representations: k-mer, binary profiles (one-hot), accumulated nucleotide frequency.	Multiple LapRBFN network
EnhancerP-2L	Pseudo K-tuple nucleotide composition (sequence/structural), statistical moment features.	RF
iEnhancer-RF	Combines k-mer frequency and pseudo dinucleotide composition.	RF
piEnPred	Optimized hybrid feature vector; cascade multi-level subset feature selection.	RF

The field has also been advanced by various ML-based approaches, such as EnhancerP-2L, iEnhancer-KL, iEnhancer-RF, iEnhancer-PsedeKNC, and piEnPred [42]–[46].

2.3 Deep Learning Approaches

Deep learning (DL) has revolutionized fields such as *natural language processing* (NLP) and sequence modeling by leveraging powerful neural network architectures [47]. While DL is primarily designed to handle high-dimensional data, such as images and videos, its ability to model complex patterns also makes it well-suited for one-dimensional sequential data, including text and time series [48]. Unlike traditional methods, which often rely on handcrafted features and statistical models, DL-based approaches automatically learn hierarchical and sequential representations from raw input [49]. These methods, including *convolutional neural networks* (CNNs), *recurrent neural networks* (RNNs), and hybrid architectures, have achieved remarkable success in various sequence-related tasks by capturing both local and global dependencies within the data.

CNNs have demonstrated exceptional performance in enhancer prediction by identifying local sequence motifs and spatially relevant features within DNA sequences. Models such as iEnhancer-ECNN and iEnhancer-Deep utilize one-hot encoding or k-mer representations to transform DNA sequences into numerical formats suitable for analysis [50], [51]. CNNs excel at learning hierarchical representations, allowing them to detect short-range patterns critical for enhancer classification. By stacking multiple convolutional layers, these models extract increasingly abstract features, enhancing their ability to distinguish enhancers from non-enhancers with high accuracy and robustness.

In addition to CNNs, RNNs-based models, particularly those employing *long short-term memory* (LSTM) networks, have proven highly effective in capturing long-range dependencies within DNA sequences. Models like iEnhancer-EBLSTM or spEnhancer utilize *bidirectional LSTMs* (Bi-LSTMs) to analyze DNA sequences in both forward and reverse directions, enabling them to identify subtle yet important patterns that span longer regions of the genome [29], [52]. This capability is especially valuable for enhancer classification, as enhancers often exhibit complex dependencies that extend beyond local motifs.

Hybrid models that combine CNNs and RNNs have significantly advanced the field of sequence analysis by leveraging the complementary strengths of these architectures. CNNs excel at detecting local motifs, while RNNs, particularly Bi-LSTMs, are adept at capturing long-range dependencies. By integrating these components, hybrid frameworks enhance predictive accuracy and provide a more comprehensive feature representation by capturing both short-range and long-range sequence characteristics.

For example, Enhancer-LSTMAtt and iEnhancer-DCLA are hybrid models that combine CNNs, Bi-LSTMs, and attention mechanisms to identify enhancers and classify their strength [53], [54]. These models use CNNs for local feature extraction and Bi-LSTMs to capture long-range dependencies. Attention mechanisms further enhance these frameworks by highlighting critical features within the sequence data. While Enhancer-LSTMAtt directly processes raw DNA sequences for feature extraction, iEnhancer-DCLA employs word embedding techniques to encode DNA sequences into numerical vectors before analysis, offering an alternative approach to feature representation.

GANs have recently emerged as a promising approach for enhancer prediction by addressing challenges such as class imbalance and data augmentation. GAN-based models consist of two neural networks, a generator and a discriminato, that work in tandem to improve model performance [55]. The generator creates synthetic data resembling real genomic sequences, while the discriminator evaluates whether the input data is real or synthetic. An example is iEnhancer-GAN, which uses GANs to generate realistic synthetic enhancer sequences for training purposes [56]. By augmenting the dataset with high-quality synthetic samples, iEnhancer-GAN effectively mitigates class imbalance issues often encountered in enhancer prediction tasks. Similarly, Rank-GAN focuses on generating synthetic DNA sequences while maintaining their intrinsic characteristics [57]. Unlike iEnhancer-GAN, which primarily addresses class imbalance through data augmentation, Rank-GAN integrates NLP techniques by treating DNA sequences as “sentence” composed of “words”. It

employs FastText for word embedding and transforms sequences into numerical matrices, which are then processed by a LSTM-CNN for enhancer identification.

Similarly to ML-based approaches, numerous other studies have employed DL-based methods to tackle this challenge. Notable examples include iEnhancer-RD, iEnhancer-CNN, ES-ARCNN, DeployEnhancer, Enhancer-DSNet, iEnhancer-DHF, iEnhancer-DCSA, and Enhancer-DRRNN [58]–[65].

Table 2.5: Deep learning-based enhancer classification models.

Model	**Key Features**	**Architecture**
iEnhancer-ECNN	Ensemble CNNs; one-hot encoding, k-mer descriptors for sequence encoding.	Ensemble CNNs
iEnhancer-Deep	One-hot encoding; CNNs for advanced sequence feature extraction.	CNN
spEnhancer	Two-layer Bi-LSTM with attention; integrates position-specific encoding.	Bi-LSTM + Attention
iEnhancer-EBLSTM	Ensemble Bi-LSTM; k-mer-based subsequence encoding.	Ensemble Bi-LSTM
Enhancer-LSTMAtt	Bi-LSTM, deep residual networks, feed-forward attention for feature extraction.	Bi-LSTM + Attention
iEnhancer-DCLA	Word embedding; CNN, Bi-LSTM, and attention for feature extraction.	CNN + Bi-LSTM with Attention
iEnhancer-GAN	Word embedding, Seq-GAN for artificial sequence generation; skip-gram encoding, CNN for classification.	Seq-GAN + CNN
RankGAN	Word embedding, RankGAN for dataset amplification; LSTM-CNN for feature extraction.	RankGAN + LSTM-CNN

2.4 Transformer Models for Sequence Analysis

Building on the advancements of deep learning architectures, the introduction of transformer models marked a paradigm shift in sequence analysis by addressing the limitations of these earlier approaches. The transformer architecture, introduced by Vaswani et al. in 2017, is built on an encoder-decoder structure where the encoder processes input sequences to generate contextualized representations, and the decoder uses these representations to produce output sequences [66]. Unlike earlier sequence-to-sequence models that relied on RNNs or CNNs, transformers eliminate the need for recurrence or convolution by using self-attention mechanisms and positional encodings to capture relationships across all tokens in a sequence simultaneously. This innovation has significantly advanced sequence analysis, particularly in fields like NLP, by addressing the limitations of traditional deep learning architectures such as CNNs and RNNs, which often struggle with capturing long-range dependencies due to their sequential or localized processing nature [67]. By leveraging self-attention mechanisms to efficiently model global relationships within sequences, transformers have enabled breakthroughs in tasks requiring an understanding of complex dependencies.

In genomics, transformer models have been adapted to analyze DNA and protein sequences, leveraging the analogy between genomic data and natural language [68], [69]. Genomic sequences, much like text, consist of ordered elements (e.g., nucleotides or amino acids) where context and interactions across distant positions are essential for understanding biological function. Transformers' ability to process sequences in parallel and capture long-range dependencies has allowed them to outperform traditional methods in a variety of genomic tasks, variant effect prediction, and gene expression modeling.

Several transformer-based models have been specifically developed for genomic applications. DNABERT, inspired by the BERT model in NLP, adapts the transformer architecture for DNA sequences [70]. By pretraining on genomic data, DNABERT learns

meaningful nucleotide representations that can be fine-tuned for downstream tasks such as transcription factor binding site detection and variant effect prediction. This study uses DNABERT to leverage its pre-trained models for accurate genomic analysis. Another notable model is Enformer, which extends the receptive field of transformers to capture long-range interactions up to 100 kilobases away [71]. Enformer has demonstrated superior performance in predicting gene expression and enhancer-promoter interactions directly from DNA sequences, making it particularly effective at modeling distal regulatory elements. Similarly, the Nucleotide Transformer represents a family of transformer-based models trained on diverse genomic datasets [72]. These models excel at generalizing across tasks such as promoter prediction and splicing analysis while maintaining high accuracy even with reduced parameter sizes.

The advantages of transformers are numerous. Their self-attention mechanism enables them to consider all positions in a sequence simultaneously, overcoming the limitations of CNNs and RNNs in capturing distal interactions. Unlike RNNs, which process data sequentially, transformers operate on entire sequences in parallel, significantly improving computational efficiency. Additionally, transformers incorporate positional encodings to preserve sequence structure, ensuring that the order of nucleotides or amino acids is appropriately accounted for during analysis.

Despite their success, transformer models face challenges when applied to genomics [73]–[75]. One major limitation is their computational cost; training large transformer models requires significant resources due to their quadratic scaling with sequence length. Furthermore, the heterogeneity of genomic datasets can introduce biases during training, necessitating careful curation and preprocessing to ensure robust performance across diverse tasks [76]. Another challenge lies in model interpretability, while attention weights provide insights into which regions of a sequence are most influential for predictions, understanding their biological significance remains an open area of research.

2.5 Large Language Models in Genomics

The progression from traditional DL architectures to transformers has revolutionized sequence analysis by enabling the modeling of complex dependencies through self-attention mechanisms. Building on this foundation, *large language models* (LLMs) represent the next leap forward, combining transformer architectures with pretraining on vast datasets to capture even richer contextual representations. In genomics, LLMs have emerged as transformative tools, demonstrating significant promise in applications such as enhancer classification, motif discovery, and variant effect prediction [77].

Models such as Enhancer-BERT and iEnhancer-BERT leverage transformer-based architectures to achieve high accuracy in enhancer sequence prediction. Enhancer-BERT is built on Google's original BERT model and treats enhancer sequences as a "biological language", using n-gram tokenization to divide sequences into overlapping n-grams for downstream analysis [78]. In contrast, iEnhancer-BERT is based on DNABERT, a transformer model pre-trained specifically on genomic data, which uses k-mer tokenization to break DNA sequences into overlapping segments of fixed length k [79]. Both models extract contextual information through self-attention mechanisms and further integrate additional layers, such as CNNs, to process these features for classification tasks. These approaches highlight the effectiveness of transformer architectures in advancing enhancer prediction and improving the understanding of regulatory DNA elements.

Similarly, LLMs have proven effective in motif discovery, a critical area in understanding gene regulation [80]. Attention-based models can detect biologically relevant motifs and analyze their interactions with transcription factors, providing insights into complex regulatory mechanisms [81]. These advancements are further enhanced by interpretability techniques such as DeepLIFT, which help researchers understand how models make predictions [82].

Another area where LLMs excel is Variant Effect Prediction (VEP). These models can predict the functional impact of genetic variants, including Variants of Uncertain Significance (VUS), which are particularly important in clinical genomics. Models like ESM1b and AlphaMissense leverage both DNA and protein sequence data to classify variants with high accuracy, outperforming traditional methods based on sequence homology [83], [84]. This capability has far-reaching implications for precision medicine, as understanding the effects of genetic variants is crucial for diagnosing genetic disorders and tailoring treatments. Additionally, LLMs have been recently applied to automate functional genomics tasks such as gene set enrichment analysis [85]. By analyzing curated gene sets with minimal errors or hallucinations, these models streamline labor-intensive processes and improve the interpretation of gene functions. Furthermore, genomic language models trained on DNA sequences enable transfer learning across various tasks, including functional constraint prediction and sequence design [6], [74]. These models demonstrate remarkable generalizability across species, offering scalable solutions for studying large genomes.

Despite their potential, adapting LLMs to genomic data presents unique challenges that require innovative solutions. One major challenge is tokenization [86], [87]. Genomic sequences differ from natural language text in their structure and complexity, necessitating specialized tokenization strategies. Approaches such as k-mer-based tokenization and byte-pair encoding have been employed to balance computational efficiency with biological relevance. Another challenge lies in handling variable-length genomic sequences. Unlike natural language sentences with relatively consistent lengths, genomic sequences can vary widely in size. Attention mechanisms within transformer architectures help address this issue by focusing on biologically significant regions while managing computational constraints [75].

Biological interpretability is another critical hurdle. While LLMs excel at making predictions based on patterns in data, ensuring that these predictions align with biological insights remains a challenge [88]. Post-hoc interpretability methods and attention-based

motif analysis have been used to shed light on model decisions; however, there is still a gap between model outputs and their biological relevance [89]. Data availability also poses a significant limitation for genomic LLMs. Unlike general-purpose LLMs trained on massive text corpora, genomic datasets are relatively small and often domain-specific [74]. This constraint necessitates the development of high-quality datasets through unsupervised learning or expert curation to ensure robust model training.

The "black boxes" nature of LLMs presents additional challenges for their adoption in clinical settings [90]. Trust is crucial when deploying these models for tasks such as variant annotation or disease diagnosis. To address this issue, researchers are exploring augmented language models with enhanced reasoning capabilities and rigorous validation processes to build confidence among clinicians and researchers alike [91], [92].

Looking ahead, LLMs hold immense potential to revolutionize genomics research by uncovering complex gene regulatory mechanisms and advancing precision medicine [93]. Enhancing model interpretability will also be critical to ensuring that predictions align with biological insights and can be trusted by researchers and clinicians alike. Additionally, integrating multi-modal data, such as DNA, RNA, and protein sequences, will enable holistic analyses of biological systems [94]–[96].

Table 2.6: BERT-based enhancer classification models.

Model	**Key Features**	**Architecture**
Enhancer-BERT	Tokenizes with k-mers, n-grams; multi-head self-attention.	Pre-trained BERT model from Google
iEnhancer-BERT	DNA-language model with transfer learning; k-mer tokenization and embedding for feature representation.	Transfer learning-based model from DNABERT

Chapter 3

Methodology

This chapter details the experimental design and procedures employed in this study. It encompasses a comprehensive overview of the dataset, including its characteristics and preprocessing steps. The chapter introduces DNABERT, a crucial component of the methodology, and elaborates on the k-mer tokenization technique used for sequence representation. Furthermore, it presents the proposed dual-layer architecture, outlining the data preprocessing, token embedding, layer structure, and loss function. The training framework, including the distributed GPU training setup and hyperparameter optimization, is thoroughly described. The chapter also specifies the evaluation metrics used to assess the model's performance. Finally, it explains the stacking ensemble technique, including the base models and meta-model, and the attention-based motif mechanism approach.

3.1 Dataset Overview

The dataset used in this study, which has been widely used in enhancer prediction research, originates from the work of Liu et al [36]. This dataset is specifically designed for enhancer identification and classification tasks and is structured to support both training and independent testing phases. The benchmark dataset was constructed based on chromatin state information derived from nine human cell lines: H1ES, K562, GM12878, HepG2, HUVEC,

HSMM, NHLF, NHEK, and HMEC. Table 3.1 provides details about the human cell lines included in the benchmark dataset. The chromatin state annotations were generated using ChromHMM, a computational tool that employs a multivariate Hidden Markov model to analyze combinations of chromatin marks such as H3K4me1, H3K4me3, and H3K27ac [97]. ChromHMM partitions the genome into 200-base pair intervals, approximately the size of a nucleosome, and determines the presence or absence of these marks to infer chromatin states. Each sequence is standardized to 200 base pairs (bp) to ensure uniformity across samples.

Table 3.1: Description of the human cell lines used in the benchmark dataset.

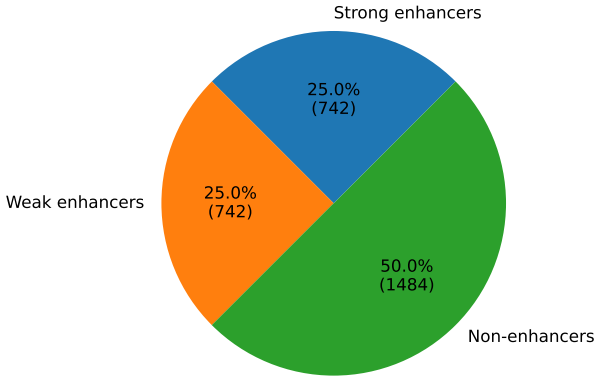
Cell Line	**Description**	**Origin**
H1ES	Human Embryonic Stem Cells (hESC)	Embryonic tissue
K562	Chronic Myelogenous Leukemia (CML)	Bone marrow (Leukemia patient)
GM12878	Lymphoblastoid Cell Line (LCL)	B-lymphocytes (EBV transformed)
HepG2	Hepatocellular Carcinoma	Liver (Tumor)
HUVEC	Human Umbilical Vein Endothelial Cells	Umbilical cord vein endothelium
HSMM	Human Skeletal Muscle Myoblasts	Skeletal muscle tissue
NHLF	Normal Human Lung Fibroblasts	Lung tissue
NHEK	Normal Human Epidermal Keratinocytes	Skin (Epidermis)
HMEC	Human Mammary Epithelial Cells	Mammary gland tissue

The dataset was carefully balanced for both classification layers:

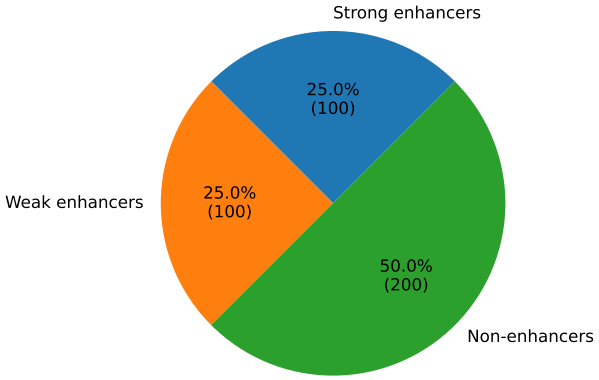
- Layer I: This layer contains 1,484 enhancer samples and 1,484 non-enhancer samples. These are used to train models that distinguish enhancers from non-enhancers.
- Layer II: The enhancer samples from Layer I are further divided into 742 strong enhancers and 742 weak enhancers. This layer supports classification based on enhancer strength.

Figure 3.1: Class distribution in dataset partitions. The development set 3.1a and the independent test set 3.1b both maintain a $25\% - 25\% - 50\%$ split between strong enhancers, weak enhancers, and non-enhancers, respectively.

(a) Development set partition.



(b) Independent test set partition.



An independent test set is also included to evaluate model performance on unseen data. This test set consists of 200 enhancer sequences (100 strong and 100 weak) and 200 non-enhancer sequences. Additionally, CD-HIT software was used to filter out pairwise sequences with more than 20% similarity to avoid redundancy [98]. CD-HIT is a bioinformatics tool designed to cluster biological sequences based on user-defined similarity thresholds. By grouping similar sequences into clusters, it minimizes redundancy, enhances computational efficiency, and improves the accuracy of downstream analyses through its fast, greedy algorithm.

The dataset used in this study can be represented as:

$$\begin{cases} S = S^+ \cup S^- \\ S^+ = S^+_{\text{strong}} \cup S^+_{\text{weak}} \end{cases} \tag{3.1}$$

Here, S^+ represents the positive dataset of enhancers, and S^- is the negative dataset of non-enhancers. The positive set is further composed of S^+_{strong}, which are strong enhancers, and S^+_{weak}, which are weak enhancers.

Table 3.2: Dataset composition.

Layer	Category	Notation	Training Samples	Test Samples
I	Strong enhancer	S^+_{strong}	742	100
	Weak enhancer	S^+_{weak}	742	100
	Non-enhancer	S^-	1,484	200
II	Strong enhancer	S^+_{strong}	742	100
	Weak enhancer	S^+_{weak}	742	100

3.1.1 Nucleotide Composition

The nucleotide composition analysis of enhancers and non-enhancers reveals intriguing patterns that shed light on their structural differences. As shown in Table 3.3, enhancers display a more balanced distribution of nucleotides compared to non-enhancers. The frequencies of A, C, G, and T in enhancers are relatively similar, ranging from 0.2447 to 0.2571. In contrast, non-enhancers exhibit a clear preference for A and T nucleotides, with frequencies of 0.3115 and 0.3139 respectively, while C and G are less represented (0.1880 and 0.1865).

Table 3.3: Nucleotide frequencies in enhancers and non-enhancers.

Category	A	C	G	T
Enhancers	0.2571	0.2460	0.2447	0.2523
Non-enhancers	0.3115	0.1880	0.1865	0.3139

This compositional difference is further emphasized by the GC content (sum of C and G frequencies), which is notably higher in enhancers (0.4907) compared to non-enhancers

(0.3745). This suggests that enhancers tend to be more GC-rich, a characteristic that may contribute to their functional properties.

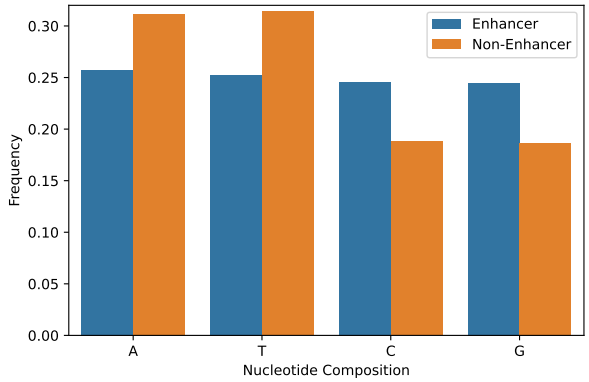
When examining the nucleotide composition of strong and weak enhancers, as presented in Table 3.4, subtle differences are observed. Strong enhancers have a slightly higher proportion of C and G nucleotides (0.2373 and 0.2347) compared to weak enhancers (0.2181 and 0.2104). Conversely, weak enhancers show a marginally higher frequency of A and T nucleotides (0.2418 and 0.2397) compared to strong enhancers (0.2276 and 0.2173). This results in a higher GC content in strong enhancers (0.4720) than in weak enhancers (0.4285), which may contribute to their increased enhancer activity.

Table 3.4: Nucleotide frequencies in strong and weak enhancers.

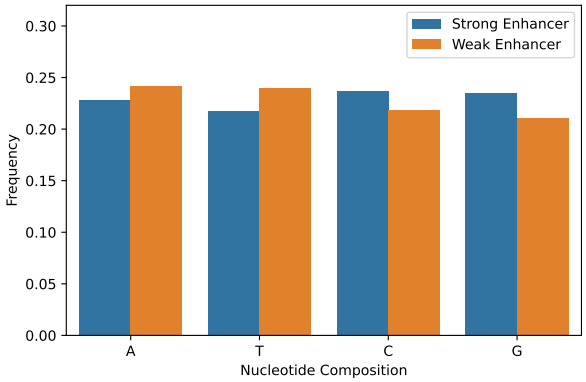
Category	**A**	**C**	**G**	**T**
Strong enhancers	0.2276	0.2373	0.2347	0.2173
Weak enhancers	0.2418	0.2181	0.2104	0.2397

Figure 3.2: Nucleotide compositions across different categories.

(a) Enhancers and non-enhancers.



(b) Strong and weak enhancers.



3.1.2 Dimensionality Reduction

To better understand the underlying structure of DNA enhancer sequence data beyond just the basic nucleotide makeup, dimensionality reduction techniques were used for visualization. The main goals were to find possible patterns or clusters in the data, represent the high-dimensional information in a simpler, lower-dimensional space, and reveal any hidden relationships. This initial analysis helps to qualitatively explore how the data is organized and whether it can be separated into meaningful groups. These insights can guide the development of encoding methods for classification tasks.

PCA

Principal component analysis (PCA), while a powerful linear dimensionality reduction technique focused on variance preservation, is not utilized in this study. The primary objective is to visualize the intricate structural relationships within DNA enhancer sequence data, particularly for the purpose of discerning inherent data patterns that may correspond to class labels. To achieve this, t-SNE and UMAP are employed. These methods excel at preserving local data point relationships, which is essential for revealing meaningful clusters and structures within high-dimensional biological datasets.

While the dataset contains labels facilitating classification tasks, the initial visualization step using t-SNE and UMAP aims to explore the data's inherent organization and potential separability, providing a qualitative assessment of the data's structure prior to classification modeling. This visualization aids in understanding the data's complexity and informs subsequent classification strategies.

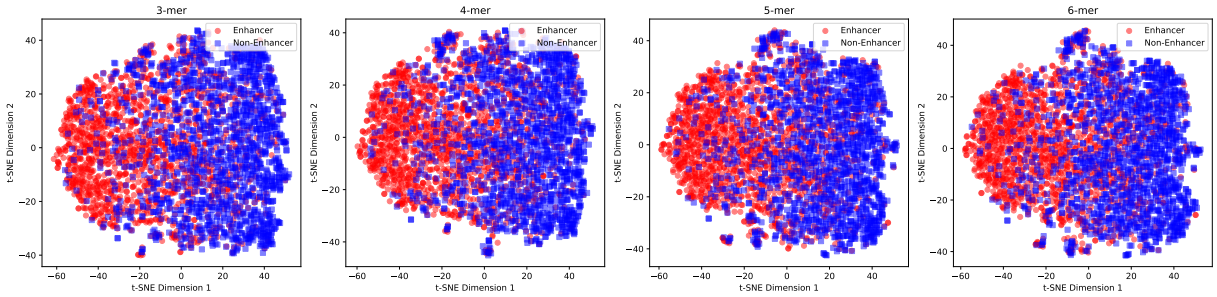
t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is a dimensionality reduction technique for visualizing high-dimensional data [99]. It projects data into a low-dimensional space (typically 2D or 3D) by preserving local relationships, keeping similar points close and

dissimilar points apart. This helps identify clusters and patterns.

Figure 3.3 shows t-SNE plots of enhancer (red dots) and non-enhancer (blue squares) sequences for k-mer sizes 3, 4, 5, and 6. Each panel displays how sequence feature representation changes with k-mer length. The visualizations reveal that k-mer frequency encoding does not effectively separate enhancer and non-enhancer sequences for any tested k-mer size. Significant overlap between the classes persists despite increasing k-mer length.

Figure 3.3: t-SNE visualization using k-mers frequency encoding. Overlap indicates poor separation, suggesting k-mer frequency alone is insufficient for effective classification.



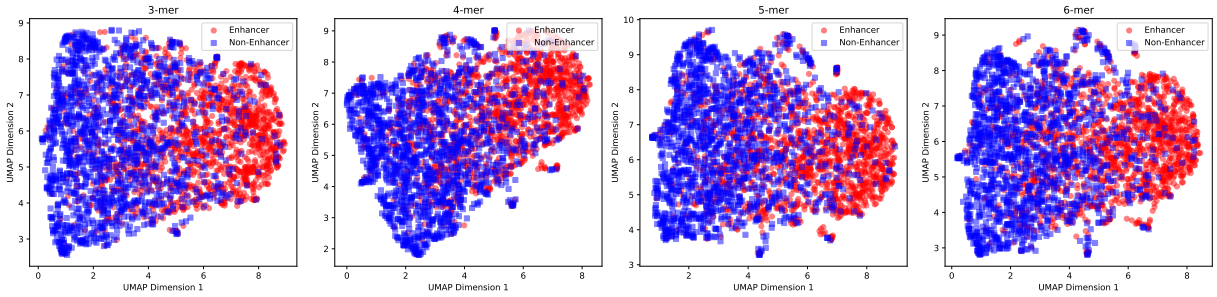
However, a subtle trend is visible: red dots tend towards the left, while blue squares are more concentrated on the right. This pattern is consistent across all k-mer sizes, hinting at underlying compositional differences. Still, these are insufficient for clear clustering or effective classification.

The results indicate that k-mer frequency encoding fails to distinguish between sequence classes. The lack of clear separation suggests this feature space is not optimal for linear classification. Consequently, a linear classifier would likely perform poorly. Advanced feature extraction or nonlinear classification methods are needed to improve performance.

UMAP

Uniform manifold approximation and projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE, but it also serves as a general non-linear dimension reduction method [100]. UMAP is based on three key assumptions about the data: it is uniformly distributed on a Riemannian manifold, the Riemannian metric is locally constant, and the manifold is locally connected. These foundational principles guide UMAP in effectively capturing the intrinsic geometry of high-dimensional datasets.

Figure 3.4: UMAP visualization using k-mers frequency encoding. While UMAP shows slightly tighter enhancer clusters, overall separation remains poor, similar to t-SNE.



UMAP employs a cross-entropy loss function that incorporates both attractive and repulsive forces, contributing to its effectiveness in maintaining global structure compared to t-SNE's reliance on KL divergence. Additionally, UMAP utilizes stochastic gradient descent for optimization, enabling it to handle large datasets efficiently without compromising performance. While UMAP is generally less sensitive to hyperparameters than t-SNE, the choice of the number of neighbors (k) remains crucial, as it influences the balance between local and global structure preservation.

Figure 3.4 illustrates the clustering of enhancers (red dots) and non-enhancers (blue squares) based on k-mer frequency encoding for different k-mer sizes: 3-mer, 4-mer, 5-mer, and 6-mer. Across all panels, there is a clear attempt to separate the two classes, with enhancers

generally forming tighter clusters compared to the more dispersed non-enhancers. The general structure of the UMAP projections remains similar across different k-mer values, but higher k-mers result in tighter and more well-separated clusters. Enhancers appear more localized in specific regions, while non-enhancers are more dispersed. But in general they are similar to t-SNE in that they fail to achieve clear clustering.

3.2 DNABERT Overview

DNABERT is a BERT-based deep learning model specifically adapted for analyzing genomic DNA sequences. It treats DNA like a "language", using its transformer architecture to understand nucleotide relationships. A key component of this architecture is the self-attention mechanism, detailed further in Appendix A.1, which enables the model to weigh the significance of different nucleotides when analyzing the entire sequence. This allows DNABERT to capture both local and global context within DNA, making it effective for various genomic applications.

Preprocessing and Pre-training

A key aspect of DNABERT is its pre-training process, which utilizes the human reference genome (GRCh38/Hg38.p13). Before being fed into the model, the raw DNA sequence is processed using k-mer tokenization. This technique involves breaking the sequence into overlapping segments of a fixed length k. For instance, tokenizing the sequence "AGCTAG" with $k = 3$ results in the tokens `[AGC, GCT, CTA, TAG]`. DNABERT was pre-trained using different values of $k \in \{3, 4, 5, 6\}$, leading to four distinct models: DNABERT-3, DNABERT-4, DNABERT-5, and DNABERT-6.

Applications

DNABERT excels in a wide range of sequence-level task. Its transformer-based architecture enables it to capture both short- and long-range dependencies in DNA sequences,

outperforming traditional methods such as CNNs and RNNs, which often struggle with these challenges. Additionally, DNABERT provides nucleotide-level interpretability, allowing researchers to identify conserved sequence motifs and functional variants that are biologically significant.

One of DNABERT's key advantages is its versatility and ability to generalize across different organisms and datasets after fine-tuning. It has been successfully applied to diverse genomic problems, including species classification and clustering through its advanced embedding capabilities. The model outputs embeddings, which are dense vector representations, $e \in \mathbb{R}^d$, for sequences or sequence tokens, where d is the embedding dimension. Furthermore, the model performs exceptionally well in data-scarce scenarios, reducing the reliance on large labeled datasets that are often expensive and time-consuming to generate.

Limitations

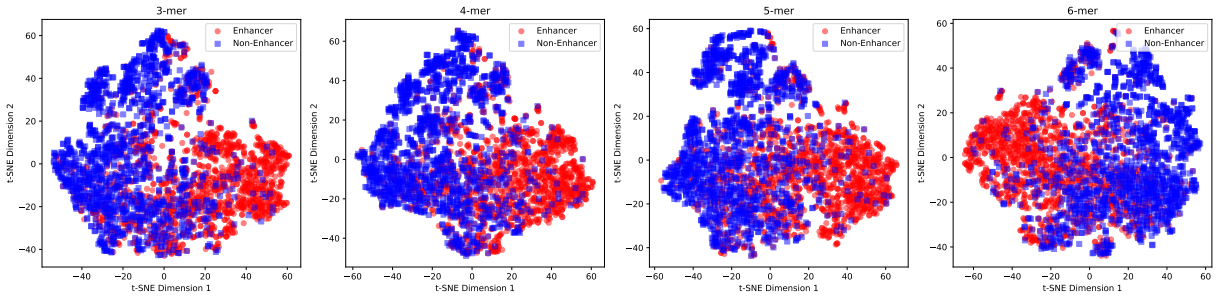
Despite its strengths, DNABERT does have some limitations. The transformer-based architecture is computationally intensive, requiring significant hardware resources for both training and inference. This can make it less accessible compared to lightweight methods such as Tetranucleotide Frequencies (TNF) [101]. TNF is a computational technique that analyzes DNA sequences by calculating the frequency of all possible combinations of four nucleotides (e.g., AGTC or TTGG). These frequencies are used to generate statistical profiles that can help identify genomic relationships or classify fragments in metagenomic studies. While TNF is fast and resource-efficient, it provides weaker phylogenetic signals compared to alignment-based methods or advanced models like DNABERT. However, DNABERT's ability to generate high-quality embeddings and achieve state-of-the-art performance on genomic tasks makes it an invaluable tool for researchers seeking to analyze complex DNA sequences.

Utilization

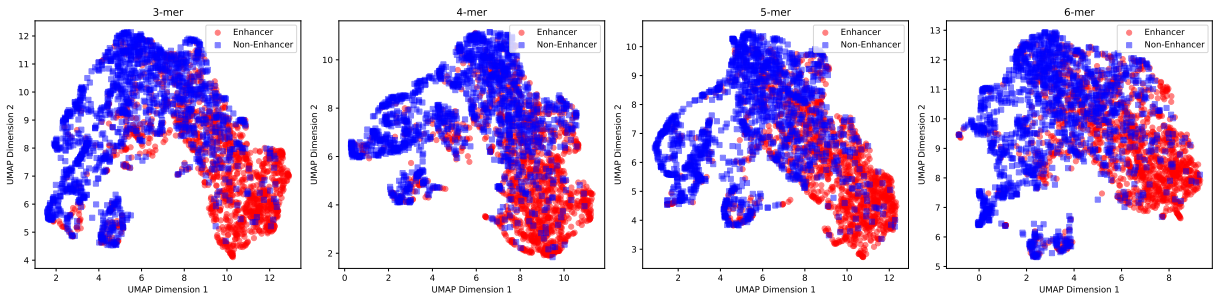
This research utilizes DNABERT to generate dense embedding vectors ($e \in \mathbb{R}^7 68$) that capture the contextual features of input sequences. These embeddings are then leveraged for dimensionality reduction and visualization using t-SNE and UMAP techniques. Figure 3.5a and Figure 3.5b illustrate the effectiveness of DNABERT embeddings in generating distinct clusters that separate enhancers from non-enhancers, showcasing the model's ability to capture meaningful sequence features. DNABERT embeddings demonstrate significantly improved clustering and separation between enhancers and non-enhancers across all k-mers. This indicates that DNABERT's transformer-based model effectively learns higher-order dependencies and contextual relationships within DNA sequences, leading to better differentiation between the two classes.

Figure 3.5: t-SNE and UMAP visualizations using DNABERT.

(a) t-SNE.



(b) UMAP.



3.3 *k*-mer Tokenization

Computational analysis of DNA sequences often requires preprocessing these potentially long strings into standardized units suitable for modeling. This study utilizes k-mer tokenization, a fundamental technique in bioinformatics, to segment DNA sequences into fixed-length substrings known as k-mers. These k-mers serve as the basic tokens for subsequent computational analysis, effectively capturing local sequence patterns critical for tasks such as enhancer prediction and genomic sequence classification.

A k-mer is formally defined as a contiguous substring of length k nucleotides derived from a longer DNA sequence. Given a DNA sequence S of length L, denoted as $S = s_1, s_2, \ldots s_L$ where s_j is the nucleotide at position j, the process of generating k-mers involves sliding a window of size k across the sequence. Each k-mer starts at a specific position i and includes k consecutive nucleotides. The i-th k-mer, T_i, generated from sequence S is defined as:

$$T_i = s_i s_{i+1} \ldots s_{i+k-1}, \quad \text{for } i = 1, 2, \ldots, L - k + 1 \tag{3.2}$$

Here, k is a user-defined hyperparameter representing the length of the substring (e.g., $k = 3$ or $k = 6$). This method produces overlapping k-mers, ensuring that each nucleotide (except for those near the ends of the sequence) contributes to multiple k-mers, thereby preserving local contextual information.

The total number of k-mers, denoted by n, that can be extracted from a sequence of length L using a k-mer size of k is calculated by the formula:

$$n = L - k + 1 \tag{3.3}$$

For instance, consider the DNA sequence $S = \text{GATCCTACTGATGC}$, which has a length $L = 14$. If a k-mer size of $k = 8$ is chosen, the total number of k-mers generated is $n = 14 - 8 + 1 = 7$. Applying the sliding window approach yields the following set of

8-mers:

- T_1: GATCCTAC
- T_2: ATCCTACT
- T_3: TCCTACTG
- T_4: CCTACTGA
- T_5: CTACTGAT
- T_6: TACTGATG
- T_7: ACTGATGC

Note that the actual nucleotides are represented as text, not variables, within the list. The complete set of k-mers generated from a sequence S, denoted as $K(S)$, can be formally represented as:

$$K(S) = \{s_i s_{i+1} \dots s_{i+k-1} \mid 1 \leq i \leq L - k + 1\} \tag{3.4}$$

The choice of the hyperparameter k significantly influences the nature of the features extracted [102]. Smaller values of k (e.g., 3 or 4) capture finer-grained local sequence motifs but might miss broader contextual patterns. Conversely, larger values of k (e.g., 6 or 8) incorporate more extensive local context but can lead to a much larger vocabulary of possible k-mers, potentially increasing computational demands and introducing redundancy if k becomes too large relative to the information content.

In this research, the strategy employed follows the approach used in DNABERT [70]. Specifically, distinct sets of tokens were generated using $k \in \{3, 4, 5, 6\}$. Four separate models were subsequently trained, each utilizing k-mers from one of these sizes. This allows for the capture and analysis of sequence patterns at varying resolutions.

3.4 Proposed Dual-Layer Architecture

The proposed framework builds upon DNABERT, a transformer-based model pre-trained specifically for genomic sequences. It consists of two primary components: Layer I (the Identifier Layer), which distinguishes enhancer sequences from non-enhancers, and Layer II (the Classifier Layer), which further categorizes enhancers into strong and weak types.

Figure 3.6: Workflow of StackBERT-Enhancer. The model uses Layer I for enhancer identification and Layer II for classifying enhancers into strong or weak categories.

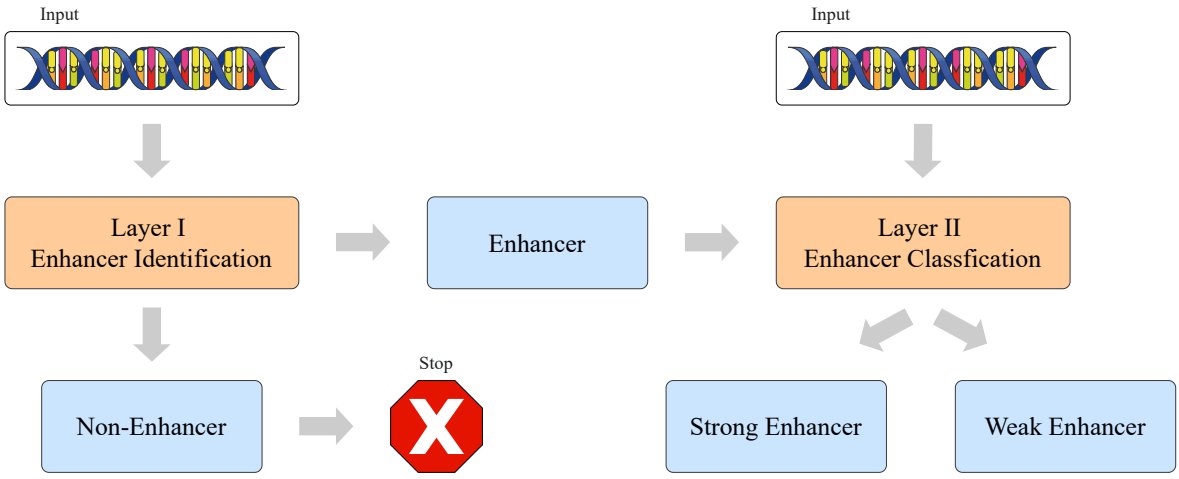


Figure 3.6 illustrates the workflow of StackBERT-Enhancer, which operates in two distinct layers to process DNA sequences and classify enhancers. The first layer, enhancer identification, takes the input DNA sequence and determines whether it is an enhancer or a non-enhancer. If classified as a non-enhancer, the workflow terminates at this stage. However, if identified as an enhancer, the sequence progresses to the second layer, enhancer classification, where it is further categorized as either a strong enhancer or a weak enhancer. Figure 3.7 shows the architectural details of the proposed model.

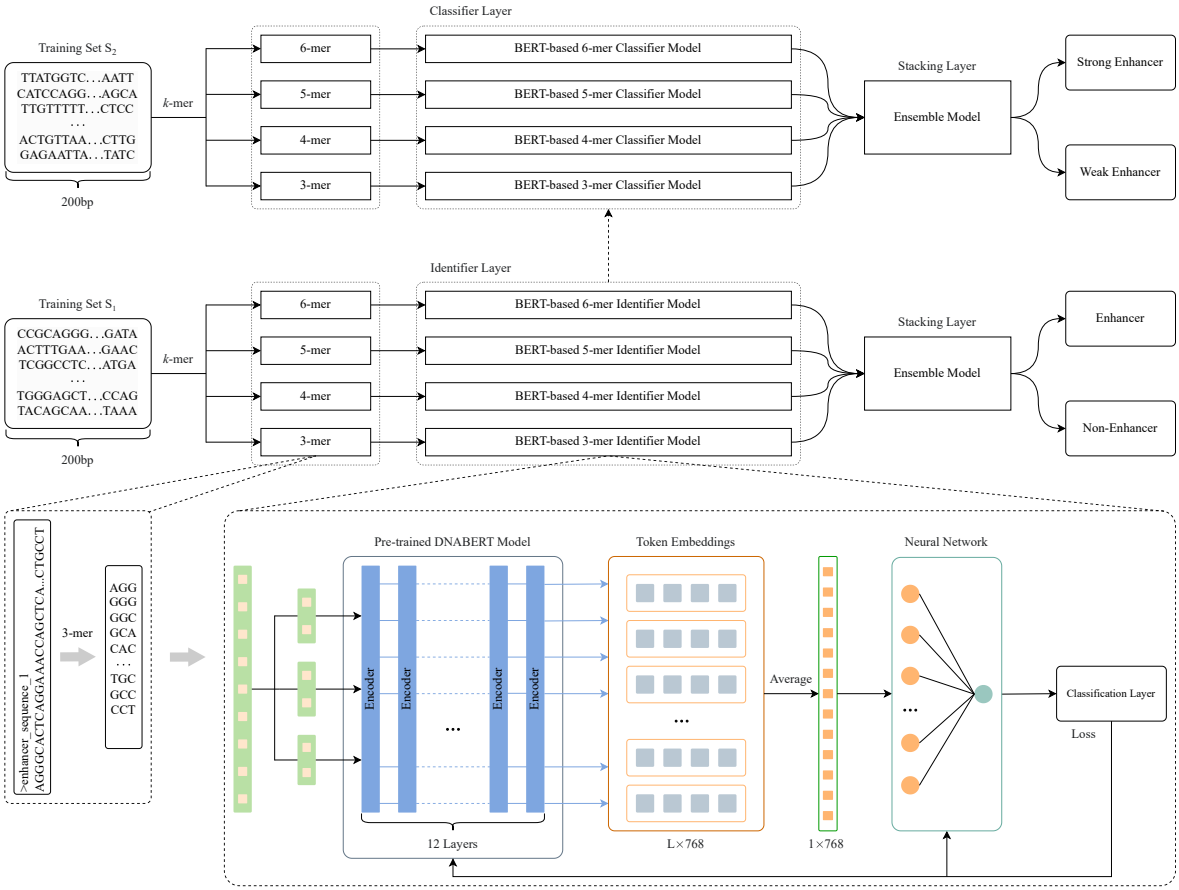
3.4.1 Data Preprocessing

The model begins by preprocessing DNA sequences from two training sets:

- S_1: Used for training the Identifier Layer.
- S_2: Used for training the Classifier Layer, $S_2 \subseteq S_1$.

The framework processes DNA sequences using a k-mer tokenization strategy, which captures sequence patterns and dependencies. Given a DNA sequence of 200 bp, overlapping

Figure 3.7: Two-layer model for enhancer identification and strength classification, where Layer II is fine-tuned based on the learned representations from Layer I.



k-mers are extracted to form tokens, where each token represents a contiguous subsequence of length k. For example, with $k = 3$, a $200-$character sequence is split into $200-k+1 = 198$ overlapping tokens. These k-mers serve as input tokens for DNABERT, enabling the model to process genomic sequences as structured "words".

3.4.2 Token Embeddings

Figure 3.8: Core structure of a classification model.

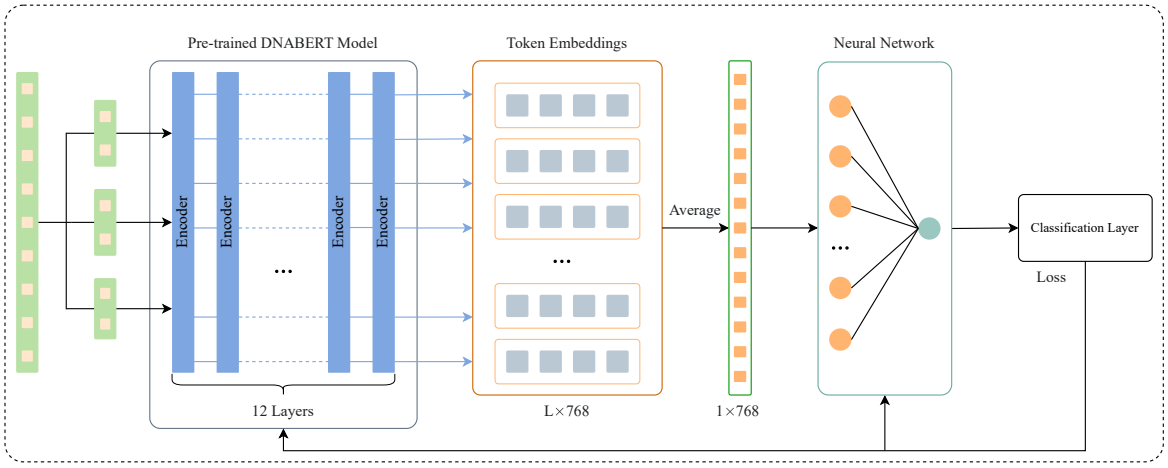


Figure 3.8 illustrates the core structure of a classification model based on a pre-trained DNABERT model, focusing on the stages following the tokenization phase. While the tokenization process varies across models, employing different k-mer sizes (3-mer to 6-mer) to segment DNA sequences into tokens, the subsequent architecture remains identical for all models. After tokenization, the pre-trained DNABERT model processes the input tokens through its 12-layer transformer encoder, generating token embeddings. These embeddings are averaged and passed through a neural network, culminating in a classification layer optimized with a loss function.

Pre-trained DNABERT model processes genomic sequences by leveraging its architecture of 12 transformer layers to encode the contextual meaning of DNA sequences. The in-

put DNA sequence is first tokenized into L tokens using a k-mer representation. Each token is then embedded into a 768-dimensional vector, resulting in an embedding matrix $\mathbf{E} \in \mathbb{R}^{L \times 768}$, where L represents the number of tokens in the sequence. These embeddings capture the contextual and semantic relationships within the genomic data.

This specific dimensionality of 768 for the embedding vectors is a design choice inherent to the base architecture of the BERT model upon which DNABERT is built. In the original BERT paper, Devlin et al. explored different model sizes and found that a hidden layer size of 768 with 12 attention heads offered a strong balance between model capacity and computational efficiency for a wide range of natural language processing tasks [5]. DNABERT adopted this same base architecture, including the 768-dimensional embedding size, to effectively capture the contextual and semantic relationships within genomic data, drawing upon the representational power demonstrated by the original BERT model. Additionally, 768 is divisible by the number of attention heads (12), allowing each head to operate on a subspace of size 64, which ensures efficient computation during multi-head self-attention.

The architecture includes 12 transformer layers, which sequentially refine the token embeddings by applying self-attention mechanisms and feed-forward neural networks. This depth strikes a balance between capturing hierarchical representations of input sequences and maintaining computational efficiency. While deeper models like BERT Large (24 layers) can capture more complex patterns, they come with significantly higher resource demands. The 12-layer configuration in BERT Base, adopted by DNABERT, has been empirically validated to perform well across diverse tasks.

To derive a single feature vector representing the entire sequence, the embeddings corresponding to the special tokens `[CLS]` and `[SEP]` are excluded. The `[CLS]` token, which is placed at the beginning of every input sequence, serves as a global representation of the sequence during processing, often used for classification tasks. Similarly, the `[SEP]` token

is used to mark boundaries between segments or indicate the end of a sequence, helping the model to distinguish between different parts of the input. While these tokens play crucial roles in structuring and processing input data, their embeddings are omitted in this approach to focus solely on the contextual information from the remaining tokens. The remaining embeddings are averaged to produce a unified representation:

$$\mathbf{v} = \frac{1}{L'} \sum_{i=1}^{L'} \mathbf{e}_i \tag{3.5}$$

Here, $L' = L - 2$ (accounting for the exclusion of special tokens), and $\mathbf{e}_i$ represents the i^{th} row of the embedding matrix $\mathbf{E}$, corresponding to the embedding of the i^{th} token in the sequence.. This step condenses token-level contextual information into a single feature vector $\mathbf{v} \in \mathbb{R}^{768}$, which serves as a compact representation of the entire sequence.

To mitigate overfitting and enhance generalization, dropout regularization is applied to $\mathbf{v}$. During training, each element of $\mathbf{v}$ is randomly set to zero with a probability p_{dropout}, while scaling the remaining elements by $\frac{1}{1-p_{\text{dropout}}}$. This can be expressed as:

$$v_i' = \begin{cases} 0 & \text{with probability } p_{\text{dropout}}, \\ v_i/(1 - p_{\text{dropout}}) & \text{otherwise.} \end{cases} \tag{3.6}$$

Consider a neural network model that includes a hidden layer represented by a vector $v = [v_1, v_2, v_3, v_4]$. During training, dropout is applied to this layer with a dropout probability of $p_{\text{dropout}} = 0.5$. This means that each element of v has a 50% chance of being set to zero during each forward pass of training.

For instance, if $v = [0.8, 0.4, 0.6, 0.9]$ before dropout, after applying dropout, it might become $v' = [0.8, 0, 0.6, 0]$, where v_2 and v_4 are deactivated (set to zero). This deactivation is random and varies for each iteration, ensuring that no single feature or neuron dominates the learning process.

Through this mechanism, the model is encouraged to learn more generalized representations of the data [103]. When dropout is not applied during inference (dropout is turned off), all elements of v are used, typically scaled by $\frac{1}{1-p_{\text{dropout}}}$ to account for the missing activations during training.

This step ensures that the model does not rely too heavily on specific features during training, thereby improving its robustness.

Following dropout, the dimensionality reduction from 768 to 32 is achieved through a fully connected linear layer with a subsequent non-linear activation. This transformation is defined as:

$$\mathbf{z} = f(\mathbf{W}\mathbf{v}' + \mathbf{b}) \tag{3.7}$$

Here, $f(x) = \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$, $\mathbf{W} \in \mathbb{R}^{32\times768}$ is a weight matrix of size 32×768, and $\mathbf{b} \in \mathbb{R}^{32}$ is a bias vector of size 32. The hyperbolic tangent (tanh) function serves as the non-linear activation in this dimensionality reduction step for several critical reasons. First and foremost, tanh introduces essential non-linearity, enabling the model to learn and represent the complex, non-linear relationships inherent in the high-dimensional 768-dimensional input space. Without such a non-linearity, the transformation performed by the weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ would remain a purely linear operation, fundamentally limiting the model's capacity to capture intricate data patterns. Second, the output range of tanh, spanning $(-1, 1)$, provides a form of inherent normalization to the 32-dimensional output. This bounded output can contribute to more stable gradient flow during training and potentially lead to faster convergence compared to unbounded activation functions. The zero-centered nature of the tanh function's output can also be advantageous for subsequent layers, facilitating learning by preventing persistent biases towards positive or negative values [104].

Finally, the reduced feature vector $\mathbf{z} \in \mathbb{R}^{32}$ is fed into an output layer with a sigmoid activation function. The output is computed as:

$$y = g(\mathbf{w}^\top \mathbf{z} + b) \tag{3.8}$$

Here, $g(x) = \sigma(x) = \frac{1}{(1+e^{-x})}$. The sigmoid function maps the output to a value between 0 and 1, representing the probability of the sequence belonging to a specific class (e.g., enhancer or non-enhancer).

The combined equation representing the computation from the averaged token embeddings to the final classification probability is:

$$y = \sigma\left(\mathbf{w}^\top \tanh\left(\mathbf{W}\left(\text{dropout}\left(\frac{1}{L'}\sum_{i=1}^{L'}\mathbf{e}_i\right)\right) + \mathbf{b}_1\right) + b_2\right) \tag{3.9}$$

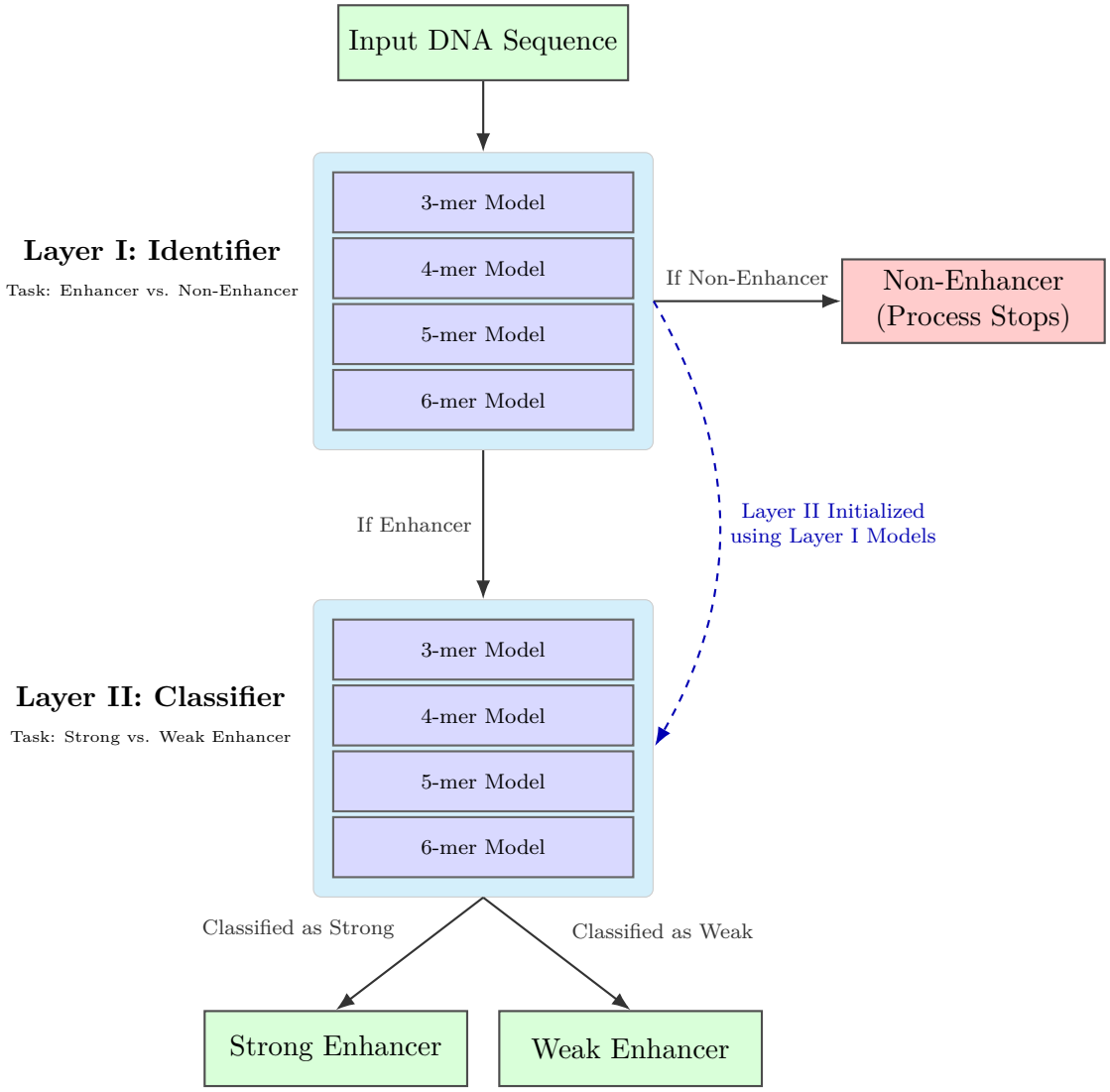
Here:

- y is the final predicted probability.
- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function.
- $\mathbf{w}^\top$ and b_2 are the weight vector and bias scalar of the final output layer.
- $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ is the hyperbolic tangent activation function.
- $\mathbf{W} \in \mathbb{R}^{32\times 768}$ and $\mathbf{b}_1 \in \mathbb{R}^{32}$ are the weight matrix and bias vector of the fully connected hidden layer.
- dropout$(\cdot)$ represents the application of dropout regularization during training.
- $\mathbf{e}_i \in \mathbb{R}^{768}$ is the embedding of the i^{th} token from the DNABERT model.
- $L' = L - 2$ is the number of sequence tokens after excluding special tokens (`[CLS]`, `[SEP]`), where L is the total number of tokens.
- $\sum_{i=1}^{L'} \mathbf{e}_i$ is the sum of the relevant token embeddings.

3.4.3 Hierarchical Two-Layer Architecture

This work introduces a hierarchical, two-layer architecture designed for accurate enhancer identification and subsequent classification into strong or weak categories. The architecture processes DNA sequences through two distinct layers, as depicted in Figure 3.9.

Figure 3.9: Hierarchical two-layer architecture for enhancer identification and classification.



Identifier Layer

The Identifier Layer constitutes the initial stage of the architecture. Its primary function is to process an input DNA sequence and determine whether it possesses the characteristics of an enhancer element. This layer is trained and evaluated using a carefully balanced dataset designed for binary classification, containing 1,484 known enhancer sequences (positive samples) and 1,484 non-enhancer sequences (negative samples). The models within this layer are trained specifically to distinguish sequences exhibiting enhancer properties from those that do not, based on this dataset.

Layer I employs four distinct BERT-based models. Each model is specialized for analyzing sequences based on a specific k-mer length, covering $k \in 3, 4, 5, 6$. These models independently process the input sequence and predict the likelihood of it being an enhancer. The final classification output for Layer I is generated by combining the predictions from these four individual models using an ensemble approach.

Employing models trained on different k-mer lengths enables Layer I to capture relevant sequence patterns and motifs at multiple scales or resolutions. The subsequent ensemble method integrates the insights from these diverse models, enhancing the overall accuracy and robustness of the enhancer identification process compared to relying on a single k-mer perspective.

Classifier Layer

The Classifier Layer is responsible for distinguishing between strong and weak enhancers. This layer is trained and evaluated using a specific, balanced subset derived from the full dataset used for Layer I. This subset consists solely of the 1,484 sequences known to be enhancers from the original dataset, which are further categorized equally into 742 strong enhancer samples and 742 weak enhancer samples. Crucially, Layer II models are trained directly on these ground-truth strong and weak enhancer sequences. It does not take as

input the sequences predicted as enhancers by the Layer I models in a sequential pipeline; rather, it operates on this predefined subset of known enhancers.

Similar to Layer I, Layer II utilizes four BERT-based models trained on k-mers, where $k \in [3, 6]$. A key feature of the training process is the use of transfer learning from Layer I. The Layer II models are initialized and subsequently fine-tuned using representations (e.g., embeddings) and parameters learned by the Layer I models during the enhancer identification task.

This initialization strategy allows Layer II to leverage the foundational sequence features relevant to identifying enhancers (learned by Layer I). By building upon this knowledge, the Layer II can more effectively focus on learning the subtle differences within enhancer sequences that distinguish strong activity from weak activity, thereby improving its classification performance.

3.4.4 Loss Function

The model employs task-specific loss functions to optimize the two layers, both of which use binary cross entropy with logits loss. This loss function is well-suited for binary classification tasks and combines the benefits of sigmoid activation and binary cross-entropy in a reliable and efficient way:

- Layer I minimizes the binary cross-entropy loss to distinguish enhancers from non-enhancers.
- Layer II minimizes binary cross-entropy loss to distinguish strong enhancers from weak enhancers.

To mitigate overfitting and improve the model's ability to generalize to unseen data, L2 regularization (also known as weight decay or Ridge Regression) is applied [105]. This technique adds a penalty to the loss function proportional to the sum of the squared values

of the model's learnable weights ($\mathbf{w}$). By penalizing large weight values, L2 regularization encourages the model to learn smaller, more distributed weights, reducing reliance on any single input feature.

The L2 regularization term is defined as:

$$L_{reg} = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_j w_j^2 \tag{3.10}$$

Here, $\|\mathbf{w}\|_2^2$ is the squared Euclidean norm (L2 norm) of the model's weight vector $\mathbf{w}$ (summing over all individual weights w_j), and $\lambda > 0$ is the regularization hyperparameter that controls the strength of the penalty. A higher λ value imposes a stronger penalty on large weights.

The total loss for the model is augmented with an L2 regularization term to prevent overfitting and encourage generalization. The combined total loss is defined as:

$$\text{Total Loss} = \text{Loss}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \|\mathbf{w}\|_2^2 \tag{3.11}$$

The binary cross-entropy loss, denoted as $\text{Loss}(\mathbf{y}, \hat{\mathbf{y}})$, measures the difference between the true labels $\mathbf{y}$ and the predicted logits $\hat{\mathbf{y}}$. The expression $\lambda \|\mathbf{w}\|_2^2$ serves as the L2 regularization term, where λ is a hyperparameter that controls the degree of regularization applied to the model.

3.5 Trainning Framework

Modern deep learning models necessitate considerable computational resources, particularly when dealing with large-scale genomic sequences, as seen in this study of enhancer and non-enhancer classification using 200bp sequences. To efficiently train the model across multiple GPUs, a distributed training setup is utilized, leveraging PyTorch's Distributed

Data Parallel (DDP) framework [106]. This section details the setup, initialization, synchronization mechanisms, and hyperparameter optimization employed in the distributed training pipeline. Figure 3.10 illustrates the workflow of DDP employed in this study, which utilizes four GPUs.

Figure 3.10: Workflow of Distributed Data Parallel. A mini-batch from the dataloader is split across n GPUs. Each GPU computes gradients for its data shard using a local model replica. Gradients are then synchronized (e.g., averaged) across all GPUs before each replica performs an identical weight update.

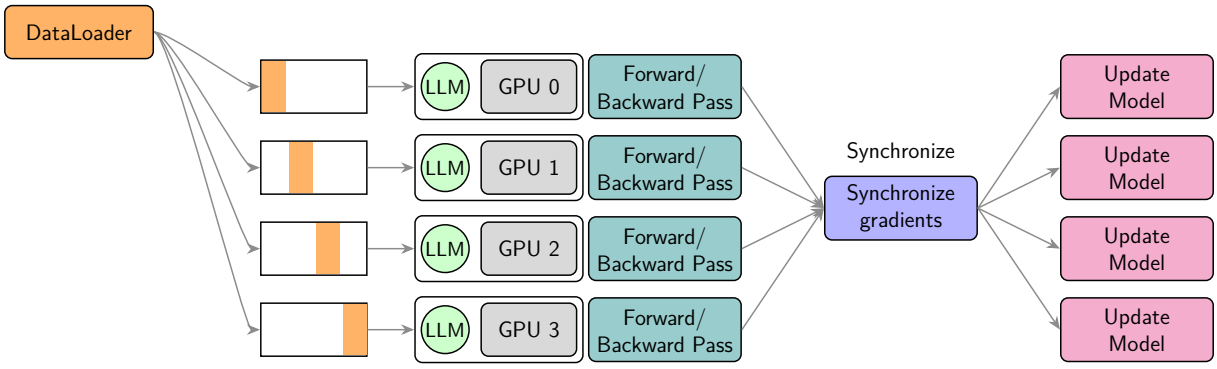


Figure 3.10 illustrates DDP, a technique to accelerate LLM training with multiple GPUs. The process starts with a `DataLoader` supplying a mini-batch of training data, which is then divided into shards and distributed among the GPUs. Each GPU hosts an identical replica of the LLM. Concurrently, every GPU executes a forward pass with its data shard to generate predictions, followed by a backward pass to compute local gradients. Crucially, these gradients are synchronized across all GPUs. Subsequently, each GPU applies these identical, aggregated gradients to update its local model replica. This synchronized update ensures all model replicas stay identical, effectively parallelizing computation while maintaining training consistency.

3.5.1 Distributed GPU Training Setup

To efficiently train deep learning models on large-scale genomic data, a distributed GPU training setup is implemented using PyTorch's `torch.distributed` package. This setup

enables parallelization across multiple GPUs and nodes, reducing the training time while ensuring effective utilization of computational resources.

Initialization of Distributed Training

The distributed training process, using `torch.multiprocessing`, follows a multi-processing approach. The script initializes multiple processes, where each process is associated with a specific GPU device. The initialization of the distributed process group is achieved through the `init_process_group` function, which sets up the communication backend and establishes ranks among processes. The following parameters are configured:

- Backend: The `nccl` backend is used, which is optimized for NVIDIA GPUs and provides efficient communication primitives.
- Initialization Method: The `init_method` is set to a TCP address that synchronizes processes across nodes.
- World Size: The total number of processes participating in training, computed as the product of nodes and GPUs per node.
- Rank: Each process is assigned a unique rank to coordinate training steps.

To ensure reproducibility in the distributed training process, a fixed random seed is set at the beginning of each process. The random seed, defined as `seed = 1337`, is applied across multiple libraries to maintain consistent results. Specifically, `torch.manual_seed(seed)` and `torch.cuda.manual_seed_all(seed)` are used to initialize the random number generators for CPU and all GPUs, respectively. Additionally, `np.random.seed(seed)` is employed to set the seed for NumPy's random number generator. Finally, by enabling `torch.backends.cudnn.deterministic = True`, the behavior of the cuDNN backend is made deterministic, further ensuring reproducibility during training. These measures collectively help mitigate variability in results caused by non-deterministic operations or hardware differences.

Dataset Partitioning and Data Loaders

Efficiently processing large datasets in a distributed environment demands careful partitioning and synchronization. In this study, for instance, the 2968 enhancer sequences are accelerated using four GPUs. This is achieved through data parallelism, where the dataset is divided among the GPUs to maximize computational efficiency and speed.

The main tool used here is PyTorch's `DistributedSampler`. It divides the data indices so that each GPU receives a unique portion. With 2968 sequences and four GPUs, each GPU gets indices corresponding to about $\frac{2968}{4} = 742$ sequences. This stops different GPUs from working on the same data, which saves time.

Next, the `DataLoader` loads the sequence data for each GPU according to the indices it received from the sampler. The data is loaded in mini-batches. Using a `batch_size` of 64 means each batch contains 64 sequences from the GPU's assigned data subset. The final batch on each GPU might be smaller if the number of assigned sequences is not perfectly divisible by the batch size.

To help the model generalize better, the training data order is shuffled every epoch[1]. The `DistributedSampler` takes care of this. By setting the current epoch number, by using `sampler.set_epoch(epoch)`, at the start of each epoch, the sampler ensures the data is shuffled differently each time.

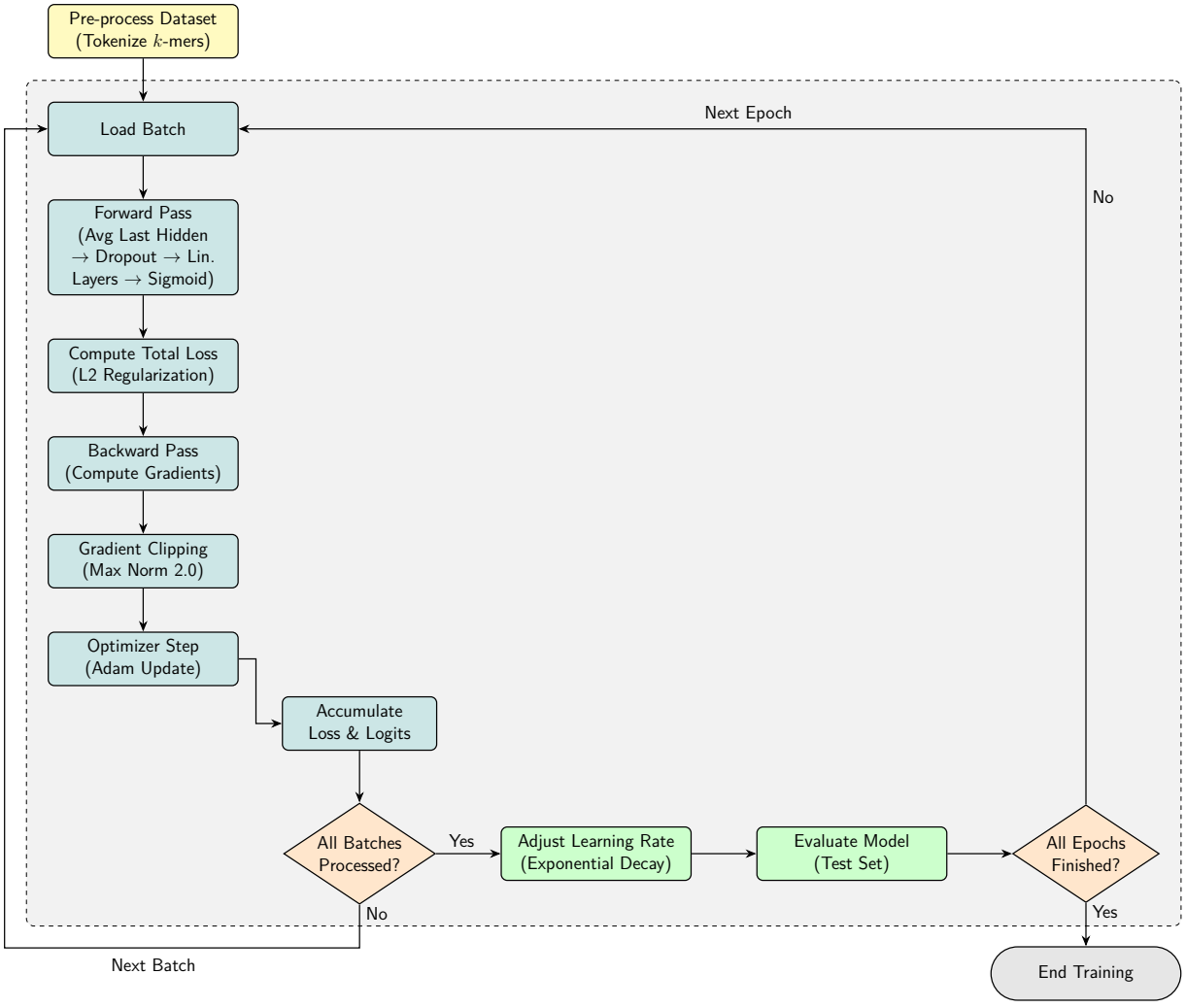
In short, using `DistributedSampler` to split the data and `DataLoader` to load it in batches allows large datasets to be processed efficiently across multiple GPUs, leading to faster training.

[1]Shuffling the training data at the start of each epoch prevents the model from learning spurious patterns related to the data order.

3.5.2 Training Loop and Regularization

Prior to the main training loop, the dataset undergoes a pre-processing phase. During this step, input sequences, already represented as k-mers, are loaded and tokenized using the specified BERT tokenizer. This entire dataset, comprising the tokenized sequences and their corresponding labels, is prepared upfront. Figure 3.11 represents a flowchart that visualizes the process of training the model.

Figure 3.11: Flowchart of the model training loop. The diagram illustrates the iterative process, starting from data pre-processing and proceeding through multiple epochs. Each epoch involves batch-wise forward/backward passes, optimization updates, and concludes with learning rate scheduling and evaluation.



The training process then iterates over a predefined number of epochs. Within each epoch, the core training loop processes the data in batches, facilitated by a `DataLoader`. For distributed training scenarios, the distributed sampler's epoch is updated at the beginning of each training epoch to ensure proper data shuffling across processes. For every batch processed within the loop, the following sequence of operations is performed.

First, the forward pass is executed. Input IDs and attention masks are fed into the base BERT model to generate the last hidden states. The token embeddings corresponding to the input sequence (excluding special tokens like `[CLS]` and `[SEP]`, and adjusted based on the k-mer value) are then averaged across the sequence length dimension. This averaged embedding is subjected to dropout, passed through a hidden linear layer utilizing a tanh activation function, and finally processed by an output linear layer with a Sigmoid activation to produce the logits. The primary loss, binary cross-entropy, is then computed between these predicted logits and the true labels.

Next, to mitigate overfitting, an L2 regularization term is calculated. This involves summing the squared L2 norms of all the model's parameters. This sum is scaled by a regularization strength hyperparameter λ and added to the previously computed primary loss to form the total loss.

Following the loss computation, the backward pass calculates the gradients of this total loss with respect to each model parameter using backpropagation. This determines how much each parameter contributed to the overall loss.

To prevent issues associated with exploding gradients[2] during training, gradient clipping is applied. The computed gradient norms are clipped, ensuring they do not exceed a max-

[2]Exploding gradients refer to a problem during neural network training where gradients become excessively large, causing unstable updates to model parameters. This can lead to numerical instability and prevent the model from converging.

imum value of 2.0.

Consistency among multiple processing units in the distributed setting is achieved through synchronization. Gradient updates are synchronized using `all_reduce` operations after the backward pass and before the optimizer step. Similarly, losses and predictions are gathered from all processes to compute global evaluation metrics accurately.

Subsequently, the optimizer step is performed using Adam (Adaptive Moment Estimation) [107]. This step is where the model learns by adjusting its parameters to minimize the loss. Adam is an adaptive optimizer, meaning it computes individual learning rates for different parameters based on estimates of the first moment (mean, similar to momentum) and the second moment (uncentered variance) of the gradients, using exponentially decaying averages of past gradients and squared gradients. It includes bias correction terms to improve estimates early in training. In this specific process, Adam utilizes the previously computed and clipped gradients to calculate the parameter updates, ensuring stability, and operates using a global learning rate that is adjusted by the learning rate scheduler between epochs.

Concurrently, learning rate scheduling refines the training process. This adjustment occurs once after each full epoch completes, not within the individual batch processing loop. Specifically, an exponential decay scheduler modifies the optimizer's learning rate, multiplying it by a decay factor of 0.98 at the end of every epoch. This gradual reduction in the learning rate helps the model to converge more effectively, allowing for finer adjustments to the parameters as training progresses and the model approaches an optimal solution.

Throughout the epoch loop, the loss is accumulated, and the true labels and predicted logits from each batch are concatenated. This collected information is used for performance evaluation once the epoch is complete.

Periodically, typically at the end of each epoch, the model's performance is assessed through an evaluation step on a separate validation dataset. Standard classification metrics are computed to monitor training progress and model generalization.

Synchronization

In distributed training environments, ensuring consistency across multiple processing units (workers) after each epoch is crucial for accurate evaluation. Once the training computations for an epoch complete on each worker, the resulting tensors containing information like the ground truth labels and the model's predictions need to be aggregated from all workers.

This post-epoch aggregation is performed using a specific process involving the `all_gather` operation. This operation collects the specified outputs (e.g., the true labels and the model's predictions) from all workers involved in the training. It pulls these individual pieces of information together from each worker and combines them into one complete set available on every worker. Combining all results this way is necessary to correctly calculate overall performance measures (e.g., training accuracy) based on the entire dataset's outcome, instead of just looking at the results from one worker, which might not show the whole story.

Model Saving and Checkpointing

To facilitate resumption and reproducibility, model checkpoints are saved periodically. This operation is restricted to the primary process (rank 0) to prevent conflicts. The model weights, optimizer state, and training progress are stored for future retraining or inference tasks.

Execution and Process Management

The training script utilizes `torch.multiprocessing.spawn` to launch multiple worker processes. The main function assigns appropriate arguments to each process and orchestrates execution across GPUs.

Through this distributed training setup, large-scale genomic classification tasks benefit from increased computational throughput, reduced training time, and improved model convergence, thereby enabling the effective identification of enhancer sequences from genomic data.

3.5.3 Hyperparameter Optimization

Hyperparameter optimization is performed for different configurations of the k-mer identifier and classifier models. The key hyperparameters tuned include:

- Learning Rate: The step size for updating model weights during training (e.g., 1×10^{-4} for 3-mer), balancing convergence speed and precision.
- Lambda (λ): A coefficient controlling the regularization penalty added to the loss function to prevent overfitting by penalizing large weights.
- Epochs: The number of full passes through the entire training dataset, allowing iterative refinement of model parameters.
- Batch Size: The number of training samples processed per iteration before weight updates, distributed across GPUs.
- Dropout Rate: The fraction of neurons randomly ignored during training (e.g., 3×10^{-1} for 30% dropout) to prevent overfitting and promote robust feature learning.

The optimization procedure systematically explores various k-mer configurations, such as 3-mers, by iteratively training a model for each. To ensure reproducibility, random seeds

Table 3.5: Ranges of different hyperparameters explored in the search space.

Hyperparameter	**Search Space**
Learning Rate	$1 \times 10^{-4}, 2 \times 10^{-4}, 3 \times 10^{-4}, 4 \times 10^{-4}, 5 \times 10^{-4},$ $1 \times 10^{-5}, 2 \times 10^{-5}, 3 \times 10^{-5}, 4 \times 10^{-5}, 5 \times 10^{-5}$
Lambda	$1 \times 10^{-3}, 2 \times 10^{-3}, \ldots, 8 \times 10^{-3}, 9 \times 10^{-3},$ $1 \times 10^{-4}, 2 \times 10^{-4}, \ldots, 8 \times 10^{-4}, 9 \times 10^{-4},$ $1 \times 10^{-5}, 2 \times 10^{-5}, \ldots, 8 \times 10^{-5}, 9 \times 10^{-5}$
Epochs	$10, 20, 30, 40, 50, 60, 70, 80, 90, 100$
Batch Size	$8, 16, 32, 64, 128$
Dropout Rate	$1 \times 10^{-1}, 2 \times 10^{-1}, 3 \times 10^{-1}, 4 \times 10^{-1}, 5 \times 10^{-1}$

Table 3.6: Optimal hyperparameter settings for dual-layer.

Hyperparameter	**Model**	**Enhancer Identification**	**Strength Classification**
Learning Rate	3-mer	5×10^{-5}	1×10^{-4}
	4-mer	1×10^{-4}	1×10^{-4}
	5-mer	1×10^{-4}	1×10^{-4}
	6-mer	4×10^{-5}	5×10^{-5}
Lambda	3-mer	7×10^{-4}	8×10^{-3}
	4-mer	3×10^{-5}	2×10^{-5}
	5-mer	6×10^{-4}	8×10^{-4}
	6-mer	1×10^{-5}	5×10^{-4}
Epochs		50	100
Batch Size		64	64
Dropout Rate		3×10^{-1}	3×10^{-1}

are initialized at the beginning of each iteration. Subsequently, training and evaluation datasets corresponding to the current k-mer target are loaded. The model is then trained using the Adam optimizer, incorporating gradient clipping and an exponential learning rate scheduler to enhance stability and convergence. Finally, the performance of the trained model is evaluated by computing key metrics on both the training and test datasets. Results are logged periodically during training and saved to `.log` files for further analysis, alongside being stored in comma-separated values (CSV) files.

Table 3.5 provides an overview of the range of hyperparameters used in this study, while Table 3.6 highlights the optimal values determined for each hyperparameter.

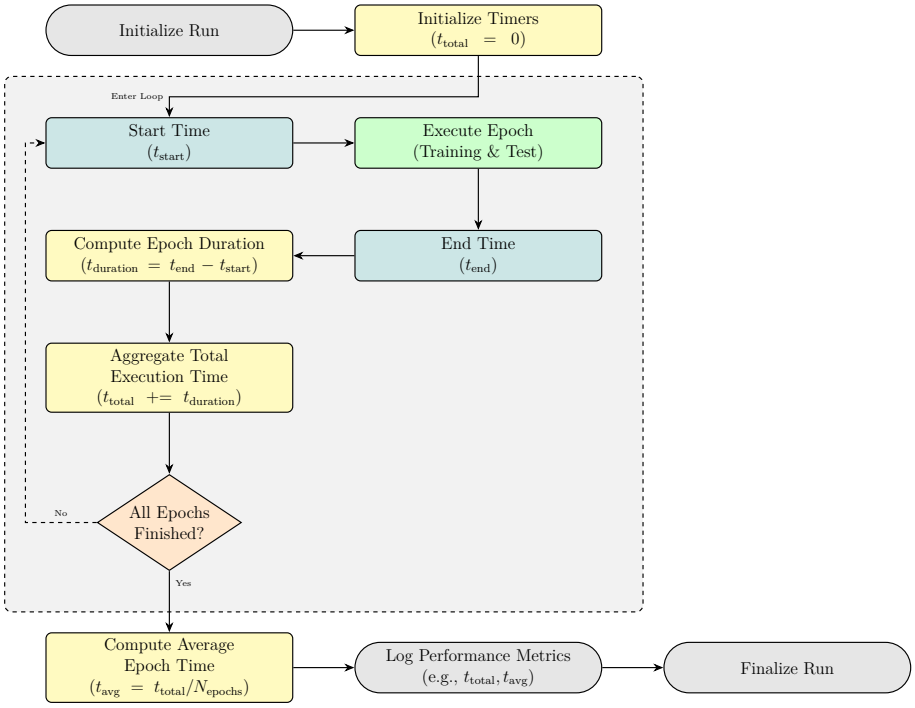
3.5.4 Running Time

Multi-GPU distributed training is utilized to achieve faster optimization and improve training efficiency. The runtime is measured by recording the time at the beginning and end of each epoch's processing. The duration for each individual epoch is calculated, and these durations are accumulated over the entire training process for a given hyperparameter configuration. Finally, the total accumulated time is divided by the total number of epochs to compute the average time per epoch for that specific training run. This process is illustrated in Figure 3.12.

The average runtime per iteration using a single GPU without DDP is approximately $t_1' = 35.2195$ seconds, while with DDP it is reduced to $t_1 = 30.2502$ seconds. Using multiple GPUs, the runtime further decreases: $t_2 = 15.3022$ seconds for two GPUs, $t_3 = 10.3917$ seconds for three GPUs, and $t_4 = 7.8085$ seconds for four GPUs. The speedup S can be calculated using Equation 3.12:

$$S = \frac{t_1'}{t_n} \tag{3.12}$$

Figure 3.12: Flowchart illustrating the process of timing and logging execution metrics across multiple epochs.



Here, t_n represents the runtime with n GPUs. Substituting the given values yields:

$$S_1 = \frac{35.2195}{30.2502} \approx 1.1643 \qquad S_2 = \frac{35.2195}{15.3022} \approx 2.3018$$

$$S_3 = \frac{35.2195}{10.3917} \approx 3.3906 \qquad S_4 = \frac{35.2195}{7.8085} \approx 4.5104$$

The maximum observed speedup of approximately 4.51x with four GPUs demonstrates the scalability and efficiency of distributed training for large-scale computations. These results are further illustrated in Figure 3.13 and Table 3.7, showcasing the runtime reductions and corresponding speedups across different configurations.

Figure 3.13: Comparison of runtime across different configurations. This graph compares the runtime per iteration (bars) and the resulting speedup (line) as the number of GPUs used for training increases. The baseline performance is established using a single GPU without Distributed Data Parallel, with the most significant speedup of 4.51x achieved when employing four GPUs.

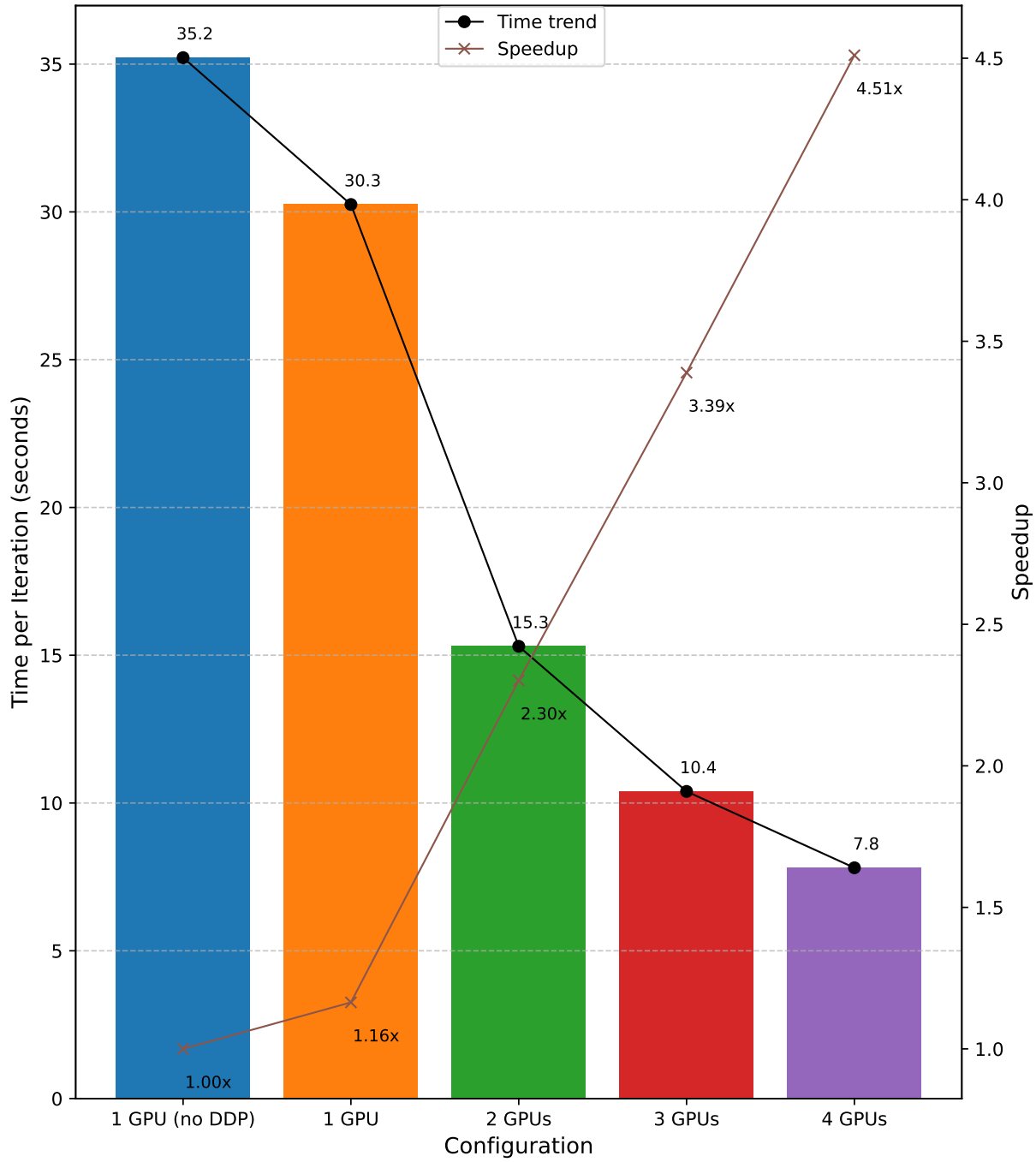


Table 3.7: Performance comparison of multi-GPU distributed training. This table presents the runtime, speedup, and percentage improvement achieved when training a model using 1 to four GPUs.

Configuration	Runtime	Speedup	Improvement
1x GPU (no DDP)	35.2195 s	1.0000x	-
1x GPU	30.2502 s	1.1643x	+14.11%
2x GPUs	15.3022 s	2.3018x	+49.41%
3x GPUs	10.3917 s	3.3906x	+32.09%
4x GPUs	7.8085 s	4.5104x	+24.86%

3.6 Evaluation Metrics

The performance of the proposed framework is assessed using a comprehensive set of standard evaluation metrics. These include accuracy (ACC), sensitivity (SN), specificity (SP), Matthews correlation coefficient (MCC), and area under the receiver operating characteristic curve (AUC) [108]. Appendix B presents these metrics in a more intuitive and understandable way.

Together, these metrics provide a robust evaluation of the model's predictive capabilities, encompassing both overall performance and class-specific behavior. These metrics are derived from the confusion matrix. This matrix serves as a structured summary of the classification performance, categorizing predictions against the true outcomes. It provides the essential breakdown needed for detailed evaluation. The components of the confusion matrix are defined in Table 3.8.

Accuracy

Accuracy measures the proportion of correctly classified instances across all classes relative to the total number of predictions. It provides an overall measure of model performance

Table 3.8: Definition and examples of confusion matrix components.

Component	Definition	Example
True Positives (TP)	Instances where the model correctly identifies a positive case.	Correctly predicting an actual enhancer.
True Negatives (TN)	Instances where the model correctly identifies a negative case.	Correctly rejecting a sequence that is not an enhancer.
False Positives (FP)	Instances where the model incorrectly classifies a negative case as positive. Also known as a Type I error.	Falsely predicting a non-enhancer sequence as an enhancer.
False Negatives (FN)	Instances where the model fails to identify a positive case, misclassifying it as negative. Also known as a Type II error.	Failing to predict (missing) an actual enhancer sequence.

and is calculated as:

$$\text{ACC} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.13}$$

While accuracy is intuitive and widely used, it may not be sufficient in scenarios with class imbalance, where one class significantly outweighs the other.

Sensitivity

Sensitivity, also known as recall or the true positive rate, evaluates the model's ability to correctly identify positive instances (e.g., enhancers). It is particularly important when false negatives are costly and is defined as:

$$\text{SN} = \frac{TP}{TP + FN} \tag{3.14}$$

High sensitivity is paramount in applications where missing positive instances (false negatives) incurs a significant cost.

Specificity

Specificity, or the true negative rate, measures the model's ability to correctly classify negative instances (e.g., non-enhancers). It complements sensitivity by focusing on how well negative cases are identified and is computed as:

$$\text{SP} = \frac{TN}{TN + FP} \tag{3.15}$$

High specificity is crucial when incorrectly classifying a negative instance as positive (a false positive) has detrimental consequences.

Matthews Correlation Coefficient

The Matthews correlation coefficient, or phi coefficient in statistics, provides a balanced evaluation by considering all elements of the confusion matrix. Unlike accuracy, MCC accounts for both correct and incorrect classifications across all classes and is particularly useful for imbalanced datasets. It is calculated as:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.16}$$

The MCC value ranges from -1 to 1, where:

- 1 indicates perfect classification,
- 0 represents random guessing,
- -1 reflects complete disagreement between predictions and ground truth.

This metric is highly informative in scenarios involving imbalanced data distributions.

Area Under the Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across various thresholds. It plots the true positive rate (TPR),

also known as sensitivity, against the false positive rate (FPR), also known as $1-$specificity, where:

$$\text{TPR} = \text{SN} = \frac{TP}{TP + FN} \tag{3.17}$$

$$\text{FPR} = 1 - \text{SP} = \frac{FP}{FP + TN} \tag{3.18}$$

The ROC curve provides insights into the trade-off between sensitivity and specificity at different threshold levels. An ideal classifier achieves a TPR of 1 and an FPR of 0, represented by a point in the upper left corner of the graph. An AUC value ranges from 0 to 1, where:

- 1.0: Perfect discrimination.
- 0.5: No better than random guessing.
- < 0.5: Worse than random guessing.

A higher AUC generally signifies a better model in terms of its ability to distinguish between the positive and negative classes, independent of a specific classification threshold.

3.7 Stacking Ensemble Technique

This study utilizes an ensemble learning approach by integrating four distinct models per layer, each specialized for a specific k-mer. The core idea is to combine these specialized models, leveraging their diverse strengths to potentially achieve superior overall predictive performance than any single model could alone. This integrated method allows for evaluating if the ensemble truly outperforms its individual components, aiming for a more robust and accurate system that mitigates the limitations of single models and enhances classification and identification capabilities.

Stacking is a powerful ensemble learning technique that combines predictions from multiple base models to improve overall predictive performance [109], [110]. The approach

Figure 3.14: Detailed architecture of Layer I for enhancer identification.

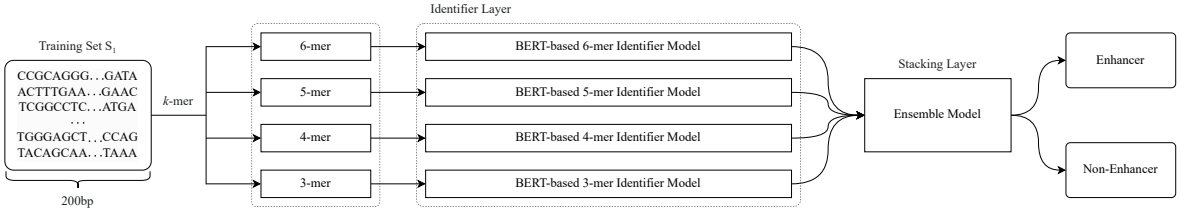
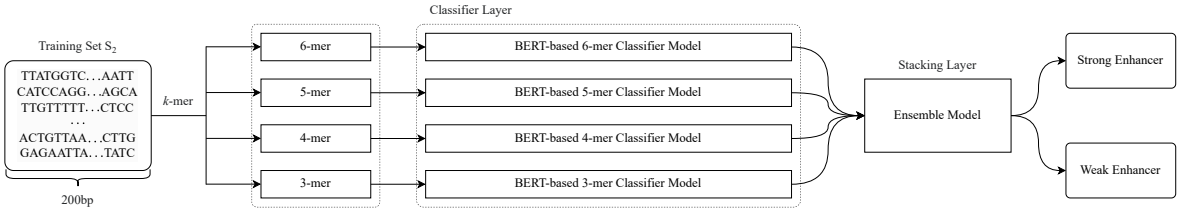


Figure 3.15: Detailed architecture of Layer II for enhancer strength classification.



leverages the strengths of various algorithms by integrating their outputs through a meta-model, which learns to optimally combine these predictions. It consists of two primary components: base models and a meta-model.

Let $h_1, h_2, \ldots, h_n$ represent the base models, and x be the input features. Each base model h_i produces a prediction $h_i(x)$. These predictions are then used as input features for the meta-model g, which outputs the final prediction y:

$$y = g(h_1(x), h_2(x), \ldots, h_n(x)) \tag{3.19}$$

Here:

- $h_i(x)$: Prediction from the i^{th} base model.
- g: Meta-model that learns to combine the predictions of the base models.
- y: Final prediction.

Equation 3.19 demonstrates how stacking integrates multiple models to leverage their individual strengths and produce a more accurate and robust solution.

Base Models

Base models, also referred to as level-0 models, are individual machine learning algorithms trained on the original dataset. These models can be diverse and include various types, such as decision trees, support vector machines, or neural networks. The key idea is to leverage the strengths of different algorithms, ensuring that the base models provide varied perspectives on the data. Their predictions serve as inputs for the next stage in the stacking process.

In this study, multiple BERT-based models were utilized as base models to generate predictions, including the BERT-based 3-mer Model, BERT-based 4-mer Model, BERT-based 5-mer Model, and BERT-based 6-mer Model. Figure 3.14 and Figure 3.15 illustrate that each layer's ensemble model is constructed with these four separate BERT-based models as its base models.

Meta-Model

The stacking ensemble architecture employed in this research incorporates a meta-model, also designated as the level-1 model [111]. The primary function of this higher-level model is to synthesize the predictions generated by the individual base (level-0) models [112]. It achieves this by learning an optimal combination strategy, utilizing the outputs of the base models as its input feature set. The objective is to minimize the overall prediction error, potentially surpassing the performance of any single base model by leveraging the diverse perspectives captured in their collective predictions.

In this study, the outputs from the BERT-based base models, specifically their predicted logits or probabilities, serve as the input features for the meta-model. Given the nature

of these inputs and the objective of optimizing the final prediction, a comprehensive evaluation was conducted to select the most effective meta-model. These included (but are not limited to) linear models (logistic regression and SGD classifier); tree-based models (decision tree, random forest, gradient boosting methods including XGBoost, LightGBM, and CatBoost, as well as AdaBoost); support vector machines; nearest neighbors algorithms (k-nearest neighbors and fixed-radius near neighbors); Bayesian models (Gaussian Naïve Bayes); discriminant analysis techniques (linear discriminant analysis and quadratic discriminant analysis); neural networks (multilayer perceptron); semi-supervised learning methods (label propagation); and other exploratory techniques.

3.8 Attention-Based Motif Discovery

Recent advancements in deep learning, particularly attention mechanisms, are revolutionizing the way researchers approach the challenging task of motif discovery in bioinformatics.

Motif Finding in Bioinformatics

Motif finding is a central challenge in bioinformatics, focused on identifying short, recurring patterns, known as motifs, within collections of DNA, RNA, or protein sequences [113]. These motifs are typically conserved subsequences that play critical biological roles, such as serving as transcription factor binding sites, splice sites, or other regulatory elements essential for gene expression and cellular function. The discovery of such motifs provides valuable insights into gene regulation, the mechanisms underlying biological processes, and evolutionary relationships among organisms [114].

The process of motif discovery is often likened to searching for a "needle in a haystack", as motifs are usually short (often 6–12 base pairs for DNA) and can be highly variable, making them difficult to distinguish from background sequence noise [115]. Motifs may be represented as consensus sequences, which capture the most common nucleotides or amino

acids at each position, or as position-specific weight matrices (PWMs), which describe the probability of each residue occurring at each position within the motif [116].

Traditional motif finding algorithms have relied on a variety of computational strategies. Probabilistic approaches, such as the expectation–maximization (EM) algorithm and Gibbs sampling, iteratively refine motif models to maximize the likelihood of observing the given sequences, often using PWMs to represent motifs [117]. Enumerative and combinatorial methods systematically search for overrepresented patterns, while tree-based and graph-theoretic algorithms exploit structural properties of the sequence data to enhance motif detection [118]. Despite their effectiveness, these methods can be computationally demanding and may struggle with the large search space and variability of biological sequences.

With the advent of high-throughput sequencing technologies and the resulting explosion of genomic data, deep learning approaches have emerged as powerful alternatives for motif discovery [119]. These methods, particularly those incorporating attention mechanisms, are capable of capturing complex and subtle sequence patterns that may be missed by traditional algorithms. Deep learning models can learn hierarchical representations of sequence data, enabling the identification of motifs even in the presence of significant noise and variability.

Median String Problem

In the field of computer science, the motif finding problem is closely related to the concept of the median string problem, where the goal is to find a string that minimizes the total distance, typically measured by the Hamming distance, to a set of input strings [120].

The Hamming distance is a metric used to compare two strings of equal length by counting the number of positions at which the corresponding symbols are different [121]. In other words, it measures how many substitutions are required to change one string into the other.

For example, the Hamming distance between the strings “phattt” and “phattr” is 1, since they differ at one position. This metric is widely used in information theory, coding theory, and computer science for tasks such as error detection and correction, as well as in motif finding algorithms where sequence similarity is important. An example of this problem is presented at Appendix A.2.

Attention Mechanism Overview in DNABERT

The attention mechanism in DNABERT enables the model to capture both local and global dependencies within DNA sequences. Each sequence is tokenized into overlapping k-mers, which are then embedded into high-dimensional vectors. These embeddings are processed through multiple layers of self-attention, where each attention head computes the importance of one k-mer relative to others in the sequence.

The attention mechanism in DNABERT is represented as shown in Equation 3.8:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here, Q, K, and V are the query, key, and value matrices derived from the input embeddings. The term d_k represents the dimensionality of the key vectors. The softmax function ensures that the attention weights sum to 1, highlighting important regions of the sequence. This mechanism enables DNABERT to uncover biologically meaningful patterns, including motifs, by assigning higher attention scores to regions of interest.

Extraction of Attention Scores for Motif Identification

The process of extracting attention scores for motif identification in DNABERT involves aggregating the attention weights from the model’s self-attention mechanism and mapping them back to the original DNA sequence.

The first step in this process is extracting attention scores from the model. DNA sequences are tokenized into overlapping k-mers and fed into the DNABERT model using a `dataloader`. During evaluation, the model is set to evaluation mode (`model.eval()`), and gradients are disabled (`torch.no_grad()`). For each batch of input sequences, attention scores are retrieved from the last layer of the Transformer model. These scores represent relationships between all token pairs in the sequence, with a shape of:

(`batch_size`, `num_heads`, `sequence_length`, `sequence_length`)

Once the attention scores are extracted, they are aggregated across k-mers to map them back to the original DNA sequence length. The attention scores matrix is first preprocessed by slicing it to exclude special tokens like `[CLS]` and `[SEP]`. Scores across all attention heads are summed for each position, resulting in a single attention score vector per sequence. The tokenized sequence length (which includes overlapping k-mers) is then mapped back to the original DNA sequence length using the formula $L = \text{tokens} - 2 + k - 1$, where k is the size of the k-mer. For each nucleotide position in the original sequence, attention scores are distributed across overlapping k-mers using a running sum and count. The aggregated scores are averaged by dividing by their respective counts, ensuring accurate representation of attention across k-mer overlaps.

Normalization is applied to ensure consistency across sequences. The aggregated attention vectors are normalized to have an L2 Euclidean unit norm, making them comparable across different sequences. The final output consists of k-mer aggregated attention vectors for all sequences, stored in an array with a shape of:

(`num_sequences`, `original_sequence_length`)

These vectors highlight regions of high importance within DNA sequences, enabling downstream tasks such as motif discovery. By leveraging DNABERT's self-attention mechanism, this process efficiently extracts biologically relevant patterns and translates them into interpretable insights about DNA motifs. Algorithm 1 summarizes this process, and Appendix A.2 provides an example.

Algorithm 1 k-mer attention vector extraction.

Require: Model $\mathcal{M}$, DataLoader $\mathcal{D}$, k-mer size k, input length T, original sequence length L, batch size B

Ensure: k-mer aggregated attention vectors $\mathcal{A} \in \mathbb{R}^{N \times L}$, where N is the number of sequences

Initialize attention score tensor $\mathbf{S} \in \mathbb{R}^{N \times H \times T \times T}$, where H is the number of attention heads
for each batch $\mathbf{x}$ in $\mathcal{D}$ **do**
    Pass inputs through model: $\mathbf{S}_{\text{batch}} \leftarrow \mathcal{M}(\mathbf{x})$
    Extract and store last layer attention scores into $\mathbf{S}$
end for
for each sequence $i \in \{1, \ldots, N\}$ **do**
    Extract attention scores $\mathbf{S}_i \in \mathbb{R}^{H \times T \times T}$ for sequence i
    Slice attention scores to remove `[CLS]` and `[SEP]`: $\mathbf{s} \leftarrow \mathbf{S}_i[:, 0, 1 : T-1]$
    Sum over attention heads: $\mathbf{v} \leftarrow \sum_{h=1}^{H} \mathbf{s}_h$
    Initialize $\mathbf{a}_i \in \mathbb{R}^L$ and count vector $\mathbf{c}_i \in \mathbb{R}^L$ to zeros
    for each position j in $\mathbf{v}$ **do**
        for offset $o = 0$ to $k-1$ **do**
            $\mathbf{a}_i[j+o] \mathrel{+}= \mathbf{v}[j]$
            $\mathbf{c}_i[j+o] \mathrel{+}= 1$
        end for
    end for
    Average: $\mathbf{a}_i \leftarrow \dfrac{\mathbf{a}_i}{\mathbf{c}_i}$
    Normalize: $\mathbf{a}_i \leftarrow \dfrac{\mathbf{a}_i}{\|\mathbf{a}_i\|_2}$
    Store in $\mathcal{A}[i, :] \leftarrow \mathbf{a}_i$
end for
return $\mathcal{A}$

Chapter 4

Results

This chapter delves into the core findings of the research, presenting a detailed analysis of the results obtained. The chapter begins by examining the ensemble model's performance, providing insights into its predictive capabilities and overall effectiveness. Key patterns and trends identified in the data are illustrated through a series of visualizations, including Venn diagrams and UpSet plots for set intersections, ROC curves, and PR curves. Following this, SHAP analysis is employed to identify the most influential features derived from the BERT-based model outputs. Finally, the biological relevance of the discovered motifs is explored, discussing their potential implications and significance within the biological context. Additional details about model evaluation methodologies and performance metrics are presented in Appendix B for further reference.

Researchers have developed a substantial number of computational methods for enhancer classification, numbering at least 30 distinct approaches. Validation of these approaches varies, commonly employing techniques such as jackknife tests, k-fold cross-validation (typically 5 or 10 folds), and independent tests. The use of such diverse validation approaches underscores the difficulty of the prediction problem. These computational strategies also pursue different objectives, for instance, distinguishing enhancers from background sequences or classifying enhancers based on their regulatory strength.

Given the computational intensity of jackknife tests, particularly for deep learning methods, the evaluation of these state-of-the-art approaches primarily relied on 5-fold cross-validation, 10-fold cross-validation, and independent tests [122]. The evaluation process employed multiple performance indices, each offering insights into different aspects of predictive capability. These metrics, however, may not always align, making it challenging to determine the overall effectiveness of a method based on these indicators alone.

Therefore, this study aims to comprehensively evaluate all the discussed factors. This thorough assessment is essential for effectively addressing the inherent complexity of enhancer classification and for establishing a solid foundation for future research in this area.

4.1 Ensemble Model Results

This section presents the results of two distinct ensemble modeling approaches. The first approach, weighted soft voting, involved averaging the predicted probabilities from a selection of base models to arrive at a final prediction. The second approach utilized a stacking methodology, where the predicted probabilities from the same base models were fed as input features into a meta-learning model.

4.1.1 Weighted Soft Voting

Initially, a soft voting ensemble approach was implemented and tested. Here, the final prediction is determined by averaging the predicted probabilities (logits) generated by each base model. To enhance fairness, individual base models were assigned weights proportional to their accuracy on the training set. These weights were then normalized to ensure they summed to 1 before being applied to the ensemble predictions.

Let the predicted probabilities from n base models for a given class be denoted as $p_1, p_2, \ldots, p_n$,

and their corresponding weights as $w_1, w_2, \ldots, w_n$. In the standard soft voting approach without weighting, the final predicted probability P_{soft}, given n is the total number of base models, is calculated as:

$$P_{\text{soft}} = \frac{1}{n} \sum_{i=1}^{n} p_i \tag{4.1}$$

The weights are proportional to the accuracy of each model on the training set and are normalized such that:

$$\sum_{i=1}^{n} w_i = 1 \tag{4.2}$$

The final predicted probability P_{final} for a class is given by the weighted average of the logits:

$$P_{\text{final}} = \sum_{i=1}^{n} w_i \cdot p_i \tag{4.3}$$

To compute w_i, let the accuracy of the i-th model be a_i. The weights are calculated as:

$$w_i = \frac{a_i}{\sum_{j=1}^{n} a_j} \tag{4.4}$$

For example, if there are base three models with accuracies $a_1 = 0.85$, $a_2 = 0.90$, and $a_3 = 0.80$, the weights would be:

$$w_1 = \frac{0.85}{0.85 + 0.90 + 0.80}, \quad w_2 = \frac{0.90}{0.85 + 0.90 + 0.80}, \quad w_3 = \frac{0.80}{0.85 + 0.90 + 0.80}$$

The normalized weights are applied in the weighted average formula to calculate P_{final}. Since each individual predicted probability p_i lies within the range $[0, 1]$ and the weights w_i are non-negative and sum to 1, the final predicted probability P_{final} will also be within the range of $[0, 1]$, i.e., $0 \leq P_{\text{final}} \leq 1$. This method ensures that models with higher accuracy contribute more to the final prediction, while maintaining a balanced weight distribution. By doing so, this approach effectively leverages the strengths of individual models and considers their relative performance on the training data.

Table 4.1 demonstrates clear advantages over individual models in both Layer I and Layer

II. By aggregating predictions, the ensemble method effectively mitigates the weaknesses of individual models, leading to improved performance metrics across the board.

Table 4.1: Result of weighted soft voting ensemble model.

Layer	Dataset	ACC	SN	SP	MCC	AUC
I	Training	0.8999	0.8315	0.9683	0.8075	0.9317
	Test	0.8325	0.7600	0.9050	0.6721	0.8504
II	Training	0.9272	0.8747	0.9798	0.8592	0.9729
	Test	0.8850	0.8600	0.9100	0.7710	0.9494

In Layer I, the soft voting ensemble achieves higher metrics compared to individual k-mer models. For instance, on the test set, the best-performing individual model is the 4-mer with an ACC of 0.8000 and an AUC of 0.8412. However, when these models are combined in an ensemble, ACC improves to 0.8325, and AUC rises to 0.8504. This improvement highlights the ensemble's ability to balance the trade-offs between SN and SP, ensuring a more robust performance.

Similarly, in Layer II, the ensemble outperforms individual models significantly. On the test set, the best individual model (4-mer) achieves an accuracy of 0.8850 and an AUC of 0.8874. In contrast, the soft voting ensemble in this layer achieves the same accuracy of 0.8850 but a higher AUC of 0.9489. These results underscore how combining predictions from diverse classifiers enhances predictive reliability by reducing bias and variance.

4.1.2 Extra Trees

Across the meta-models detailed in Section 3.7, and based on the top ten training and test results presented in Table 4.2 and Table 4.3 respectively, the Extra Trees classifier stood out with its strong standalone performance.

The Extra Trees classifier excelled, achieving perfect training metrics in Layer I and strong test generalization (ACC: 0.8350, AUC: 0.8652), attributed to its proficiency with high-dimensional data and robust randomized splitting. As a meta-model in Layer II, it continued to dominate with near-perfect test results (ACC: 0.9900, SN: 1.0000, SP: 0.9800, MCC: 0.9802, AUC: 0.9900), highlighting its superior generalization. The Extra Trees models' performance is summarized in Table 4.4. For further comparison, Figure 4.1 and Table 4.5 detail the soft voting ensemble alongside the Extra Trees classifier.

The Extra Trees classifier builds an ensemble of decision trees using random feature subsets and, crucially, random thresholds for splits, enhancing randomization compared to methods like Random Forests [123]. This study's superior Extra Trees performance is likely due to this high randomness fostering diverse trees, which reduces inter-tree correlation, mitigates overfitting, and improves generalization. Furthermore, by not being confined to locally optimal splits, Extra Trees may better capture complex non-linear relationships, while its randomized splitting also offers computational benefits, likely contributing to its consistently high performance across both Layer I and Layer II.

Figure 4.2 uses confusion matrices to depict the performance of the Extra Trees classifier across two stages. Layer I discriminates enhancers from non-enhancers with reasonable success but shows a tendency towards false negatives (50 instances). Layer II further refines this by classifying enhancer strength, demonstrating excellent performance in distinguishing strong from weak enhancers with minimal errors (only 2 misclassifications out of 200).

Figure 4.3 shows density histograms of predicted probabilities for two classification tasks, separated by training and test sets. Training histograms exhibit strong class separation, while test histograms indicate good generalization with a slightly wider spread and minor prediction uncertainties.

Table 4.2: Results for top 10 meta-models on the training set. The Extra Trees model in both layers exhibited perfect performance (1.0000) across all metrics, suggesting excellent fit to the training data.

Meta-Model	ACC	SN	SP	MCC	AUC
		Layer I			
Extra Trees	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
SGD Classifier	0.9104	0.8632	0.9575	0.8244	0.9271
Gaussian Process Classifier	0.9070	0.8504	0.9636	0.8193	0.9274
Multilayer Perceptron	0.9070	0.8491	0.9650	0.8195	0.9273
Label Propagation	0.9161	0.8699	0.9623	0.8358	0.9316
Label Spreading	0.9161	0.8699	0.9623	0.8358	0.9311
NuSVC	0.9077	0.8538	0.9616	0.8201	0.9294
Support Vector Machine	0.9181	0.8827	0.9535	0.8384	0.9309
Logistic Regression	0.9003	0.8322	0.9683	0.8081	0.9270
Linear Discriminant Analysis	0.9003	0.8329	0.9677	0.8079	0.9273
		Layer II			
Extra Trees	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
XGBoost	0.9993	0.9987	1.0000	0.9987	1.0000
LightGBM	1.0000	1.0000	1.0000	1.0000	1.0000
Random Forest	0.9737	0.9474	1.0000	0.9488	1.0000
Hist Gradient Boosting	1.0000	1.0000	1.0000	1.0000	1.0000
Gradient Boosting	0.9690	0.9394	0.9987	0.9397	0.9922
Radius Neighbors	0.9252	0.8679	0.9825	0.8560	0.9518
Linear Discriminant Analysis	0.9259	0.8747	0.9771	0.8563	0.9700
Logistic Regression	0.9272	0.8774	0.9771	0.8587	0.9701
SGD Classifier	0.9218	0.8558	0.9879	0.8511	0.9700

Table 4.3: Results for the top 10 meta-models on the independent test set. The Extra Trees model achieved the best performance in both Layer I and Layer II.

Meta-Model	ACC	SN	SP	MCC	AUC
		Layer I			
Extra Trees	**0.8350**	**0.7750**	**0.8950**	**0.6749**	**0.8652**
SGD Classifier	0.8275	0.7650	0.8900	0.6602	0.8528
Gaussian Process Classifier	0.8250	0.7500	0.9000	0.6574	0.8525
Multilayer Perceptron	0.8225	0.7450	0.9000	0.6529	0.8531
Label Propagation	0.8225	0.7550	0.8900	0.6510	0.8590
Label Spreading	0.8225	0.7550	0.8900	0.6510	0.8584
NuSVC	0.8200	0.7500	0.8900	0.6464	0.8530
Support Vector Machine	0.8200	0.7750	0.8650	0.6426	0.8039
Logistic Regression	0.8150	0.7150	0.9150	0.6430	0.8526
Linear Discriminant Analysis	0.8150	0.7150	0.9150	0.6430	0.8513
		Layer II			
Extra Trees	**0.9900**	**1.0000**	**0.9800**	**0.9802**	**0.9900**
XGBoost	0.9150	1.0000	0.8300	0.8423	0.9591
LightGBM	0.9100	1.0000	0.8200	0.8336	0.9621
Random Forest	0.9100	0.9400	0.8800	0.8215	0.9667
Hist Gradient Boosting	0.9050	1.0000	0.8100	0.8250	0.9409
Gradient Boosting	0.9000	0.9400	0.8600	0.8026	0.9528
Radius Neighbors	0.8850	0.8400	0.9300	0.7731	0.9248
Linear Discriminant Analysis	0.8850	0.8600	0.9100	0.7710	0.9498
Logistic Regression	0.8850	0.8600	0.9100	0.7710	0.9495
SGD Classifier	0.8800	0.8300	0.9300	0.7638	0.9485

Table 4.4: Result of Extra Trees ensemble model.

Layer	Dataset	ACC	SN	SP	MCC	AUC
I	Training	1.0000	1.0000	1.0000	1.0000	1.0000
	Test	0.8350	0.7750	0.8950	0.6749	0.8652
II	Training	1.0000	1.0000	1.0000	1.0000	1.0000
	Test	0.9900	1.0000	0.9800	0.9802	0.9900

Figure 4.1: Performance comparison of ensemble models on identification and classification tasks using multiple evaluation metrics for both training and test sets.

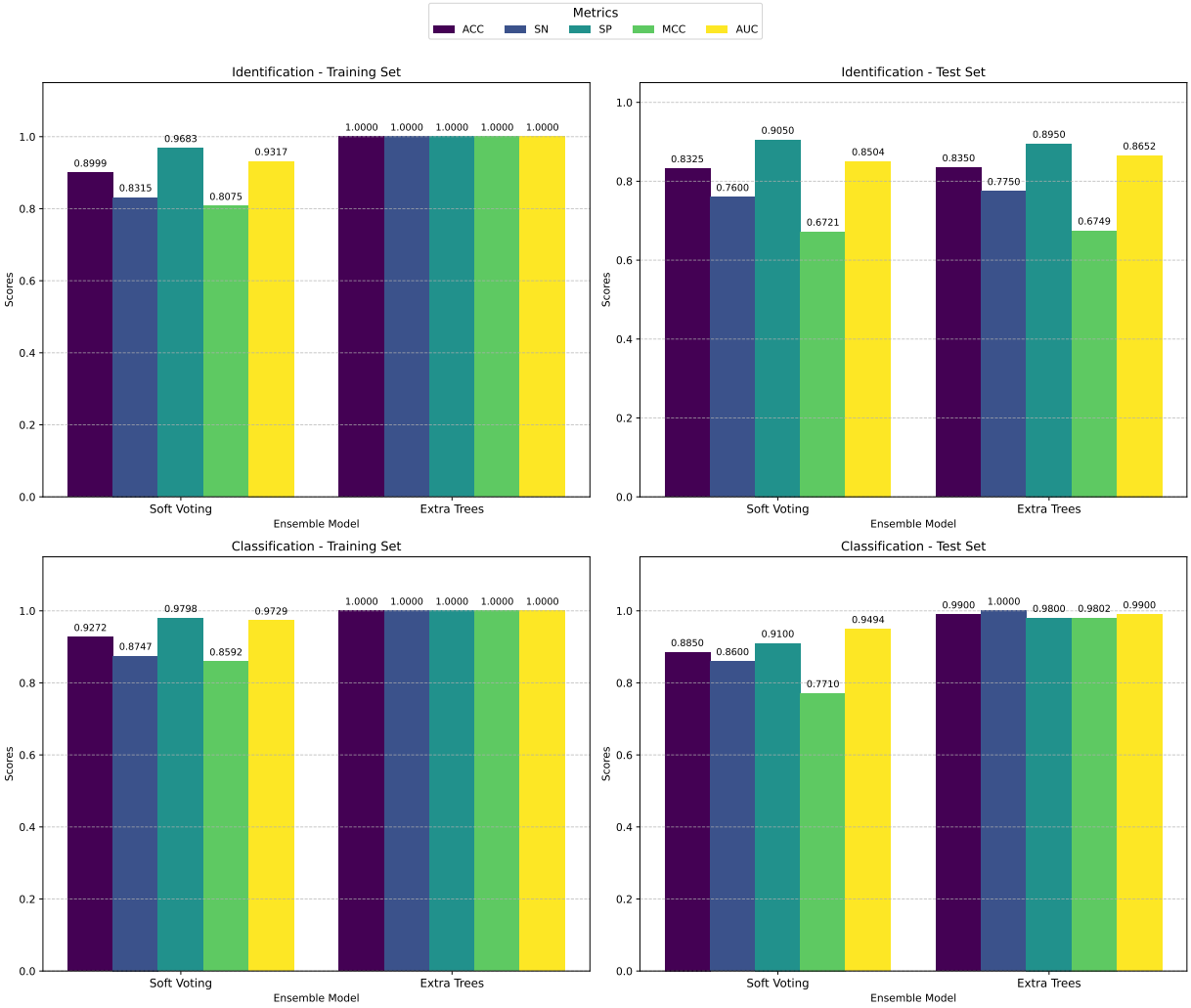
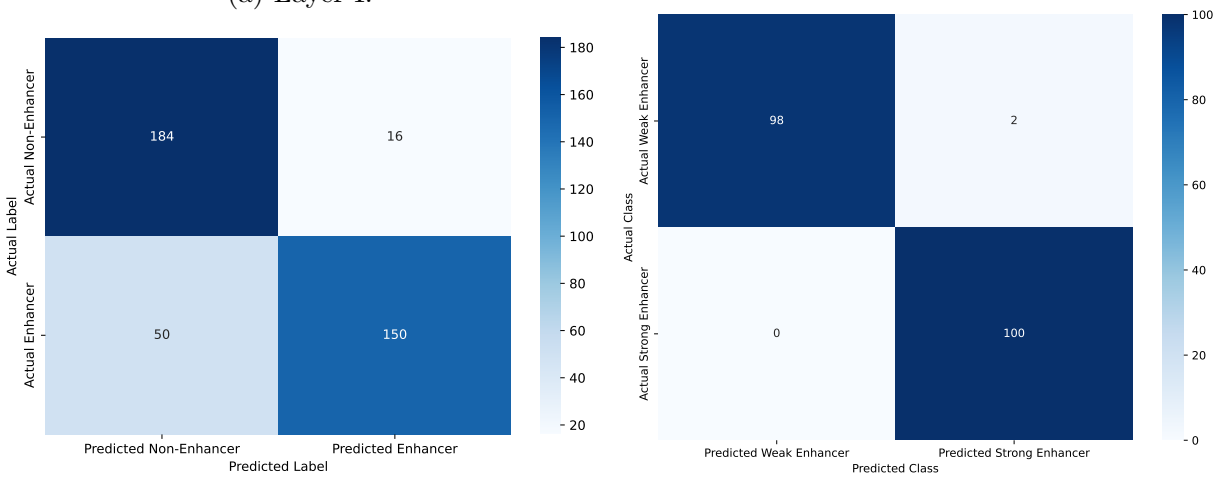


Figure 4.2: Performance of Extra Trees classifier illustrated by confusion matrices.

(a) Layer I.

(b) Layer II.



Given that the Extra Trees classifier consistently outperformed other models across both layers and datasets, it was selected as the ensemble meta-model for this study. Its exceptional performance metrics and robust generalization capabilities make it an ideal choice for integrating predictions from multiple base models into a cohesive and highly accurate ensemble framework.

Figure 4.3: Density histograms showing the distribution of predicted probabilities for enhancer classification on the training and test sets using Extra Trees classifier.

(a) Layer I.

(b) Layer II.

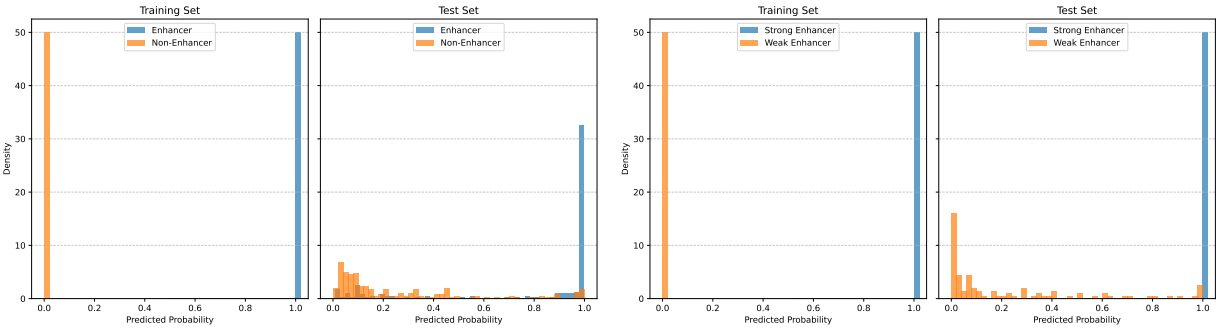


Table 4.5: Performance comparison of ensemble models across two layers.

Layer	Model	Dataset	ACC	SN	SP	MCC	AUC
I	Soft Voting	Training	0.8999	0.8315	0.9683	0.8075	0.9317
		Test	0.8325	0.7600	0.9050	0.6721	0.8504
	Extra Trees	Training	1.0000	1.0000	1.0000	1.0000	1.0000
		Test	0.8350	0.7750	0.8950	0.6749	0.8652
II	Soft Voting	Training	0.9272	0.8747	0.9798	0.8592	0.9729
		Test	0.8850	0.8600	0.9100	0.7710	0.9494
	Extra Trees	Training	1.0000	1.0000	1.0000	1.0000	1.0000
		Test	0.9900	1.0000	0.9800	0.9802	0.9900

4.2 Plots

Plots are visual tools used to understand patterns, trends, and relationships in data, crucial for data analysis, machine learning, and research. They simplify complex information through various forms.

Venn Diagrams

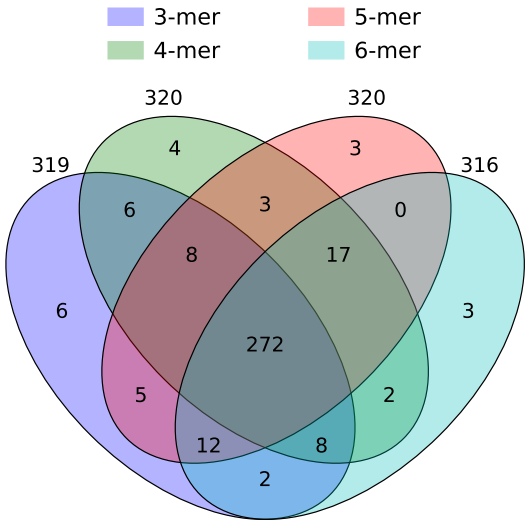
Venn diagrams visually represent set relationships using overlapping circles, effectively illustrating the distribution of elements like compounds, genes, or, in this context, k-mers [124]. Each circle corresponds to a set, with overlaps indicating shared elements. This method simplifies the visualization of complex data relationships.

Figures 4.4a and 4.4b depict the relationships between sets of k-mers for Layer I and Layer II, respectively, showing unique and shared k-mers across different models.

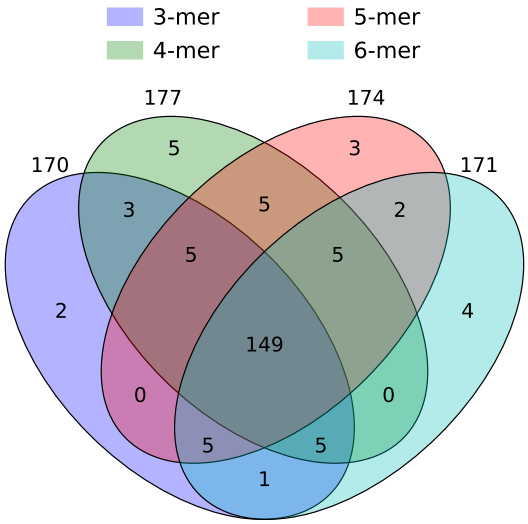
In Layer I, as shown in Figure 4.4a, the models for 3-mer, 4-mer, 5-mer, and 6-mer show

Figure 4.4: Venn diagrams illustrating the overlap in correctly classified samples among the four k-mer models on the independent test set for Layer I and Layer II.

(a) Layer I.



(b) Layer II.



substantial unique sets of correctly classified samples, with approximately 316 to 320 each. A significant central overlap of 272 is common to all four models. Smaller overlaps between pairs of models are also observed (e.g., six between 3-mer and 4-mer, eight between 4-mer and 5-mer, and seventeen between 5-mer and 6-mer).

For Layer II, as shown in Figure 4.4b, the number of unique elements in each set is smaller, ranging from approximately 170 to 177 samples. This reduction is expected, as Layer II was trained on a subset of the Layer I data. Further details on the overlap of incorrectly predicted samples for both layers are available in Appendix B.3.

UpSet Plots

UpSet plots offer a scalable alternative to Venn diagrams for visualizing intersections of multiple sets, particularly useful for more than a few sets [125]. They use a matrix to show set intersections, with vertical bars indicating the size of each intersection and connected

filled circles below denoting the contributing sets. Horizontal bars on the left display the size of each individual set.

Figure 4.5: UpSet plot for Layer I. The top bar chart shows the size of each intersection, while the matrix below indicates which sets are included in each intersection. The bar chart on the left displays the size of each individual set.

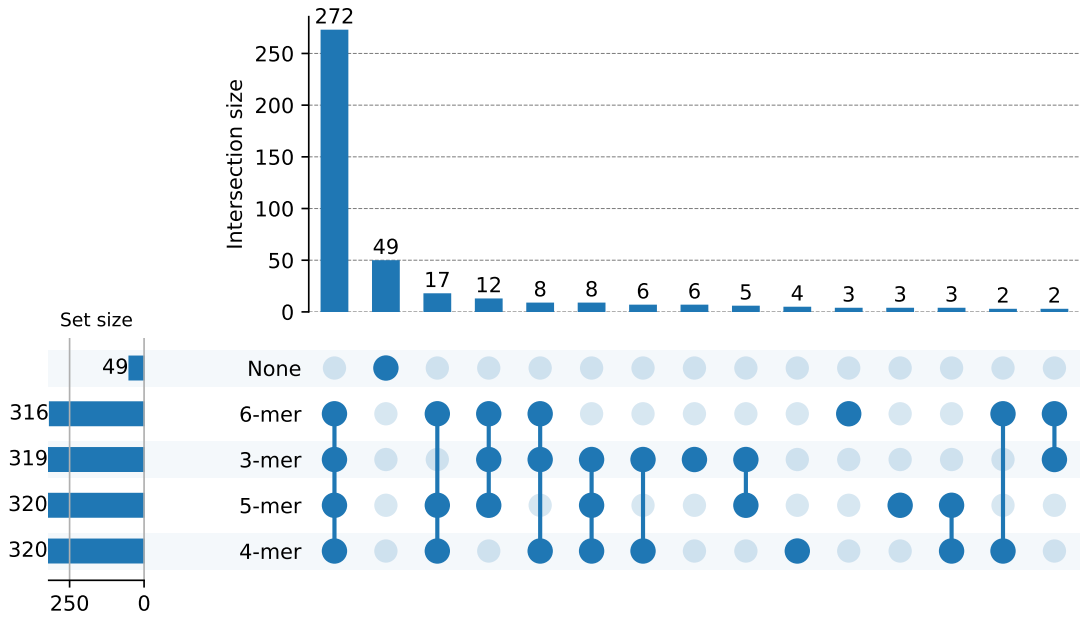
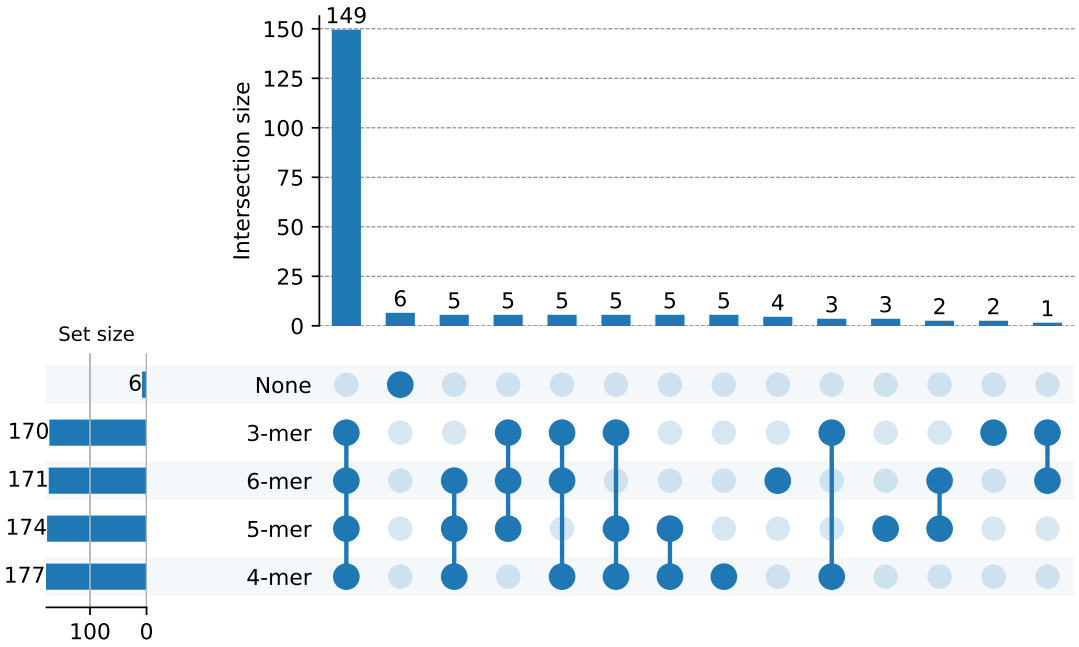


Figure 4.5 employs an UpSet plot for Layer I, illustrating intersections among k-mer sets (3-mer, 4-mer, 5-mer, 6-mer) and a "None" category. The top bars quantify intersection sizes; for example, one major intersection identifies 271 sequences not found in any of the 3-mer to 6-mer sets (this intersection is associated with the "None" concept). Similarly, Figure 4.6 presents an UpSet plot for Layer II. A key intersection here shows 149 sequences absent from the 3-mer to 6-mer sets.

ROC Curves

The ROC curve is a graphical representation of a binary classifier's performance across various thresholds. It plots the TPR against the FPR at different classification thresholds. The ROC curve provides a visual way to assess the trade-off between SN and SP,

Figure 4.6: UpSet plot for Layer II.



with the AUC serving as a single metric to summarize overall model performance. A perfect classifier would have an AUC of 1, while a random classifier would have an AUC of 0.5.

The ROC curves in Figure 4.7 illustrate the performance of the ensemble dual-layer model for enhancer analysis.

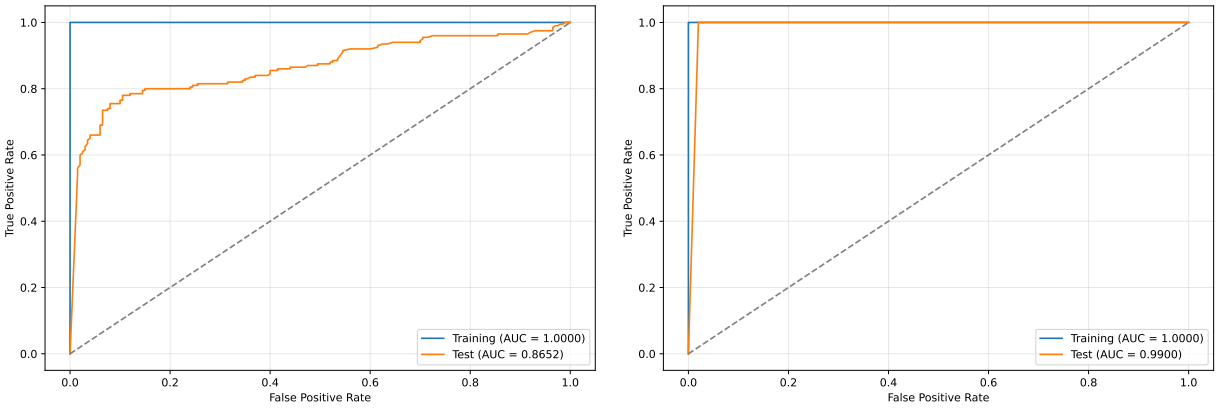
Layer I shows perfect performance on the training set (AUC = 1.0000) but a lower AUC of 0.8652 on the test set. This indicates good predictive power but suggests some overfitting, as the model's generalization to unseen data is not perfect.

Layer II demonstrates exceptional performance on both training (AUC = 1.0000) and test sets (AUC = 0.9900). This near-perfect test performance indicates excellent generalization and suggests that the features used for strength classification are highly informative.

The contrast between Layer I and II test performances is notable. Layer II maintains

Figure 4.7: ROC curves of ensemble model for dual-layer.

(a) Layer I. (b) Layer II.



almost perfect performance on unseen data, while Layer I shows a more substantial drop.

PR Curves

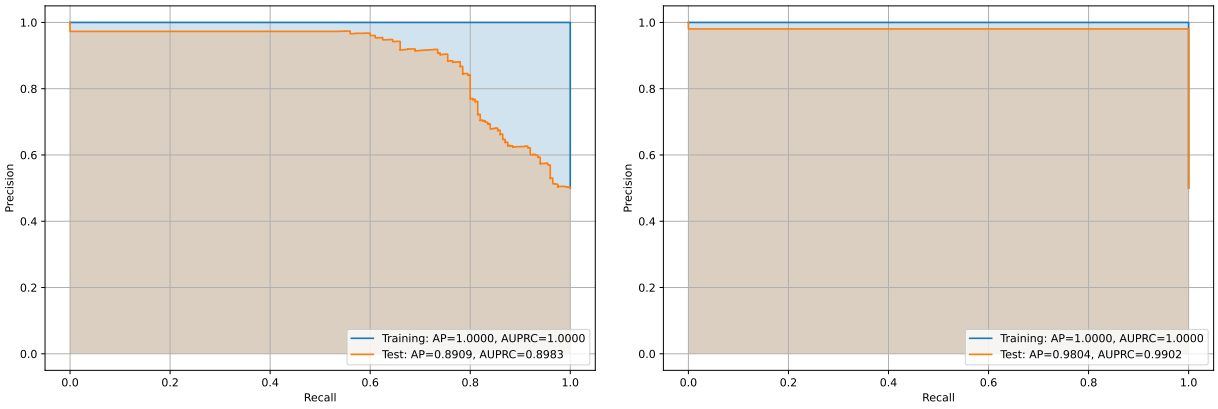
The Precision-Recall (PR) curve plots precision (y-axis) against recall (x-axis) at various classification thresholds, offering a trade-off visualization for binary classification. It is especially useful for imbalanced datasets where the positive class is rare. The area under the PR curve (AUPRC) and the closely related average precision (AP) score summarize model performance. AP, an approximation of AUPRC, is a weighted mean of precisions at each threshold, ranging from 0 to 1, with 1 indicating a perfect model.

Figure 4.8 illustrates the PR curves for the ensemble model's dual-layer architecture. In Layer I, the training PR curve demonstrates perfect performance, achieving an AP and AUPRC of 1.0000. However, on the test dataset, the AP and AUPRC slightly decrease to 0.8909 and 0.8983, respectively, indicating strong but slightly diminished generalization. The PR curve shows a gradual decline in precision as recall increases, which is typical in imbalanced datasets where achieving high recall often reduces precision.

In contrast, Layer II exhibits superior performance in both training and test datasets.

Figure 4.8: PR curves of ensemble model for dual-layer.

(a) Layer I. (b) Layer II.



The training PR curve remains perfect with AP and AUPRC values of 1.0000, while the test dataset achieves AP and AUPRC values of 0.9804 and 0.9902, respectively. The PR curve for Layer II is nearly flat at high precision levels across all recall values, highlighting robust generalization and classification capability for enhancer strength prediction. Compared to Layer I, Layer II demonstrates better generalization on unseen data, suggesting that enhancer strength classification may be a less complex task for the model than enhancer identification. Overall, the dual-layer ensemble model performs exceptionally well in both tasks, with Layer II showing slightly better results on the test dataset.

4.3 SHAP Analysis

Machine learning models, particularly complex non-linear models like deep neural networks or ensemble methods, often function as “black boxes”, making it challenging to understand the reasoning behind their predictions. While these models may achieve high predictive accuracy, a lack of interpretability can be a significant barrier to their adoption in critical applications where trust, fairness, and accountability are paramount. Understanding how individual features influence a model’s output is crucial for debugging, identifying biases, gaining scientific insights, and building user trust.

To address this challenge, this research employed SHapley Additive exPlanations (SHAP), a state-of-the-art model-agnostic interpretability method [126]. SHAP connects optimal credit allocation with local explanations using the classic Shapley values from cooperative game theory. It provides a unified framework for interpreting predictions, assigning each feature an importance value for a particular prediction, indicating how much that feature's presence changes the prediction compared to the baseline prediction.

At its core, SHAP is based on the concept of Shapley values, developed by Lloyd Shapley. In game theory, Shapley values provide a way to distribute the total gain of a collaborative game among the players, based on their marginal contribution to all possible coalitions. In the context of machine learning interpretability, the "game" is the prediction task for a single instance, the "players" are the feature values of that instance, and the "gain" is the difference between the actual prediction and the average prediction for the dataset.

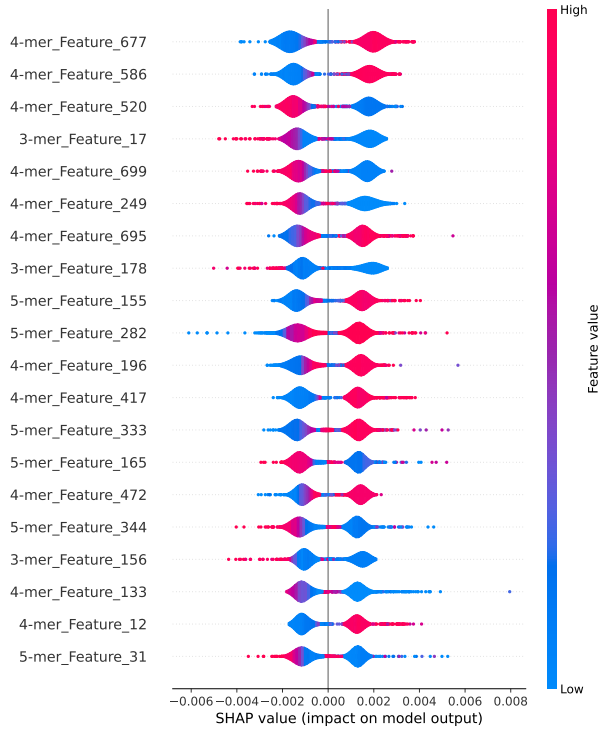
Due to the significant computational resources and time required for SHAP analysis on complex models and large datasets, this research focused the SHAP interpretability efforts exclusively on the enhancer identification task performed by Layer I of the framework. The analysis of Layer II, which handles enhancer strength classification, was omitted from the SHAP analysis to manage computational feasibility.

Lundberg et al. introduced TreeExplainer as a method to compute exact Shapley values efficiently for tree-based models [127]. Therefore, `shap.TreeExplainer` was employed to efficiently calculate SHAP values for the tree-based models utilized in this study (XGBoost and Extra Trees). For XGBoost, TreeExplainer can usually compute exact SHAP values without needing a background dataset, which helps keep the process efficient. In contrast, when using Extra Trees, TreeExplainer typically requires a background dataset and more computational resources to estimate SHAP values, making it slower and more resource-intensive compared to XGBoost.

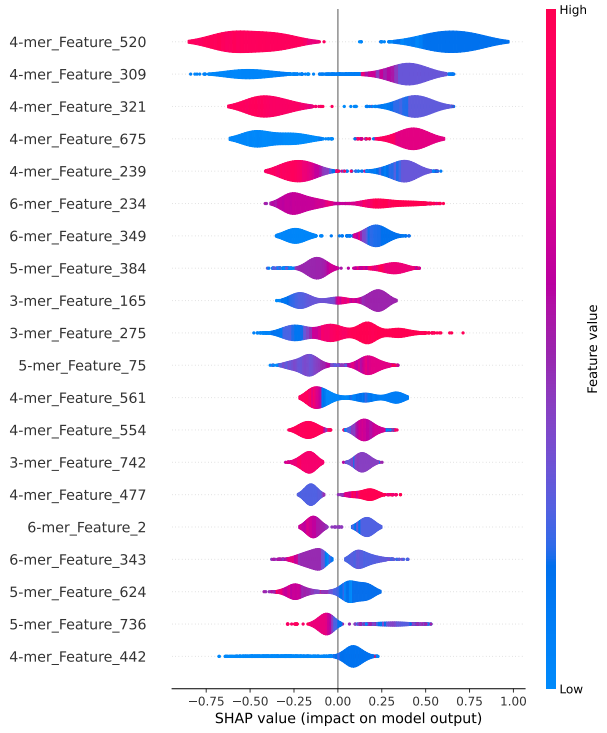
For each of the 2968 sequence samples in the training set, the data was processed through each of the four BERT models. From each BERT model, a 768-dimensional feature vector representing the sequence was extracted (after handling special tokens like `[SEP]` and `[CLS]`). These four 768-dimensional vectors, one from each BERT model, were then concatenated together, forming a single, combined feature vector of $4 \times 768 = 3072$ dimensions for every sample. The models were trained using these 3072 features per sample.

Figure 4.9: SHAP summary plots for the Extra Trees and XGBoost models, showing the impact of the top 20 most important features on enhancer predictions using the training set.

(a) Extra Trees model.



(b) XGBoost model.



To assess the global importance of each feature across the dataset, the mean absolute SHAP value was calculated for each feature across all analyzed samples. Features were then ranked based on these mean absolute SHAP values in descending order. The top 20

features with the highest mean absolute SHAP values were identified as the most influential features in the model's predictions.

Figure 4.9a presents the SHAP summary plot for the Extra Trees model, visualizing the impact of the top 20 features on enhancer predictions. These features are primarily 4-mers, interspersed with several 3-mer and 5-mer features, ranked on the y-axis by their mean absolute SHAP value. The x-axis quantifies the SHAP value, representing each feature's contribution to the model's output for individual samples. Positive SHAP values on the right signify that a feature's value pushed the prediction towards the enhancer class, while negative SHAP values on the left indicate a push towards the non-enhancer class. The color of each point directly corresponds to the feature value for that specific instance, transitioning from low (blue) to high (red or pink). Notably, the prominence of 4-mer features among the top 20 SHAP values aligns with the observation that the 4-mer base model demonstrated the highest performance metrics among the individual k-mer models.

SHAP values are tightly distributed around zero, ranging approximately from -0.006 to 0.008. This indicates that, while these features are important, their individual effects on the model's predictions are modest. This visualization clearly shows how variations in a feature's value influence its predictive effect. For example, for `4-mer_Feature_677`, instances with high values (red) predominantly show positive SHAP values, strongly contributing to enhancer predictions. The violin-like shapes further reveal the distribution and consistency of these impacts across the dataset.

Figure 4.9b shows the SHAP summary plot for the XGBoost model, again focusing on the top 20 features. While there is some overlap in important features, such as `4-mer_Feature_520`, which appears highly ranked in both models, the XGBoost plot includes a greater diversity of feature lengths, with several 6-mer features among the most influential. The range of SHAP values is much wider, spanning from about -0.75 to 1.0, which indicates that individual features can have a substantially larger impact on the model's output compared to

the Extra Trees model. This broader distribution suggests that XGBoost leverages certain features more aggressively, allowing for stronger positive or negative shifts in prediction based on specific k-mer patterns. The shapes of the distributions for each feature are often more asymmetric and spread out compared to the Extra Trees model, reflecting the greater sensitivity and nonlinearity inherent in gradient boosting algorithms.

Taken together, these plots highlight both similarities and differences in how the two models interpret sequence features for enhancer identification. Both models identify short k-mer motifs as crucial, but XGBoost attributes much greater individual influence to certain features and captures more complex relationships, as evidenced by the presence of longer k-mers and the broader range of SHAP values. The Extra Trees model, by contrast, distributes importance more evenly across features, with generally smaller individual impacts. Additional SHAP plots are available in Appendix B.3.

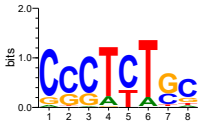
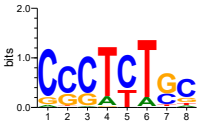
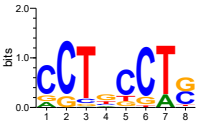
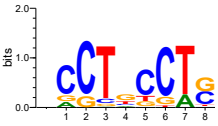
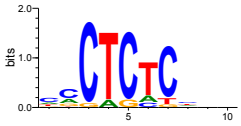
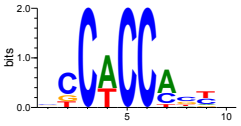
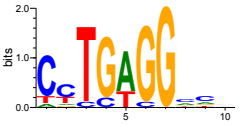
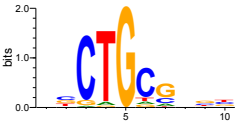
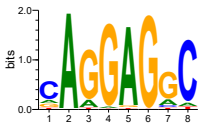
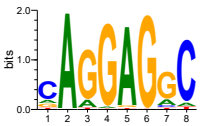
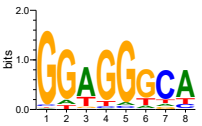
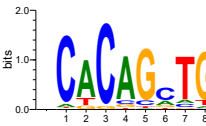
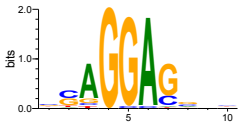
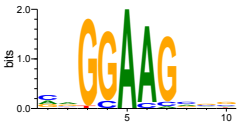
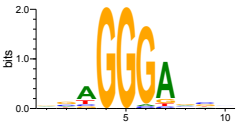
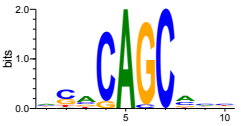
4.4 Biological Relevance of Identified Motifs

This study investigates the identification of biologically significant DNA motifs by leveraging the self-attention mechanism within the DNABERT model, facilitated by the DNABERT-viz tool. The core hypothesis is that regions within DNA sequences assigned high attention scores by the model during predictive tasks correspond to functionally relevant motifs.

The methodology extracts attention scores generated by DNABERT's self-attention layers. To relate these scores back to the original sequence context, attention values associated with overlapping k-mers (the input units for DNABERT) are aggregated and mapped onto their corresponding nucleotide positions in the full DNA sequence. This process generates an attention profile across the sequence, where peaks indicate regions the model deemed important.

To isolate potentially significant motifs from background attention noise, contiguous high-

Table 4.6: Comparative analysis of sequence motifs identified by STREME and attention mechanism. STREME detected 11 distinct motifs, while the attention-based model found 18. The table displays 8 motif pairs where the sequences were determined to be significantly similar between the two methods.

STREME				
Attention				
STREME				
Attention				

attention regions are identified and filtered based on statistical criteria. Candidate motifs are selected based on the statistical significance of their aggregated attention scores. In this study, regions were considered statistically significant if their associated p-value or false discovery rate (FDR) fell below a cutoff of 0.05. This filtering step aims to ensure the identified motifs are both statistically robust and likely biologically relevant, moving beyond simpler thresholding methods like merely exceeding the mean attention value.

The effectiveness of this attention-based approach was evaluated by comparing its results against those obtained using STREME (Statistical Regulatory Elements Miner) [128], a widely adopted tool for traditional motif discovery. STREME operates as a discriminative algorithm, excelling at finding short, ungapped motifs significantly enriched in a primary sequence set compared to a control set. It employs a robust statistical framework for evaluating motif significance and utilizes an efficient exhaustive search strategy, making it a strong benchmark, particularly for speed and sensitivity in detecting well-defined motifs.

Table 4.6 presents a comparative analysis of the motifs identified by both methods. In this analysis, the attention-based approach identified 18 distinct motifs, while STREME detected 11 distinct motifs from the same sequence data. The table specifically showcases 8 motif pairs for which the sequence patterns discovered by the attention mechanism and STREME were determined to be significantly similar, illustrated visually through sequence logos. This direct comparison demonstrates that the attention-based method can successfully recover motifs consistent with those found by established techniques like STREME. Although the attention mechanism identified a larger number of distinct motifs overall, the significant similarity observed for these 8 pairs strongly supports its efficacy in capturing biologically relevant sequence patterns comparable to those identified by STREME.

Chapter 5

Discussion

This chapter contextualizes the findings of the research, primarily focusing on the performance of the proposed StackBERT-Enhancer model. It begins by positioning StackBERT-Enhancer within the current landscape through a rigorous benchmark comparison against over 30 state-of-the-art methods, drawing upon the methods detailed in Chapter 2. Furthermore, this chapter includes an ablation study to evaluate the impact of different architectural choices, specifically comparing CNNs with an averaging approach for processing token embeddings from a pre-trained BERT model, on the performance of genomic large language models. This analysis investigates the effects of these designs on predictive accuracy, computational efficiency, and ensemble performance.

5.1 Benchmarking Against State-of-the-Art Studies

The performance of StackBERT-Enhancer model is rigorously benchmarked against more than 30 existing methods, with the results for Layer I and Layer II tasks presented in the heatmaps shown in Figure 5.1. For the initial task of enhancer identification, the heatmap in Figure 5.1a demonstrates that StackBERT-Enhancer achieves state-of-the-art performance. It secures the highest scores across several key metrics, including an ACC of 0.8350, SP of 0.8950, and an AUC of 0.8652. This performance surpasses other leading models

such as Enhancer-DHF and iEnhancer-DCSA. The performance gap between the proposed model and others becomes even more significant in the context of the more complex task of enhancer strength classification, as detailed in Figure 5.1b. Here, StackBERT-Enhancer exhibits exceptional classification power, attaining near-perfect scores with an ACC of 0.9900, a SN of 1.0000, an SP of 0.9800, a MCC of 0.9802, and an AUC of 0.9900. This outcome represents a significant leap in performance compared to the next-best models such ás iEnhancer-DCSA and Enhancer-LSTMAtt, firmly establishing StackBERT-Enhancer as the top-performing method for both identification and strength classification of enhancers. Appendix B.1 presents more details on performance comparisons between different studies.

5.2 Ablation Study

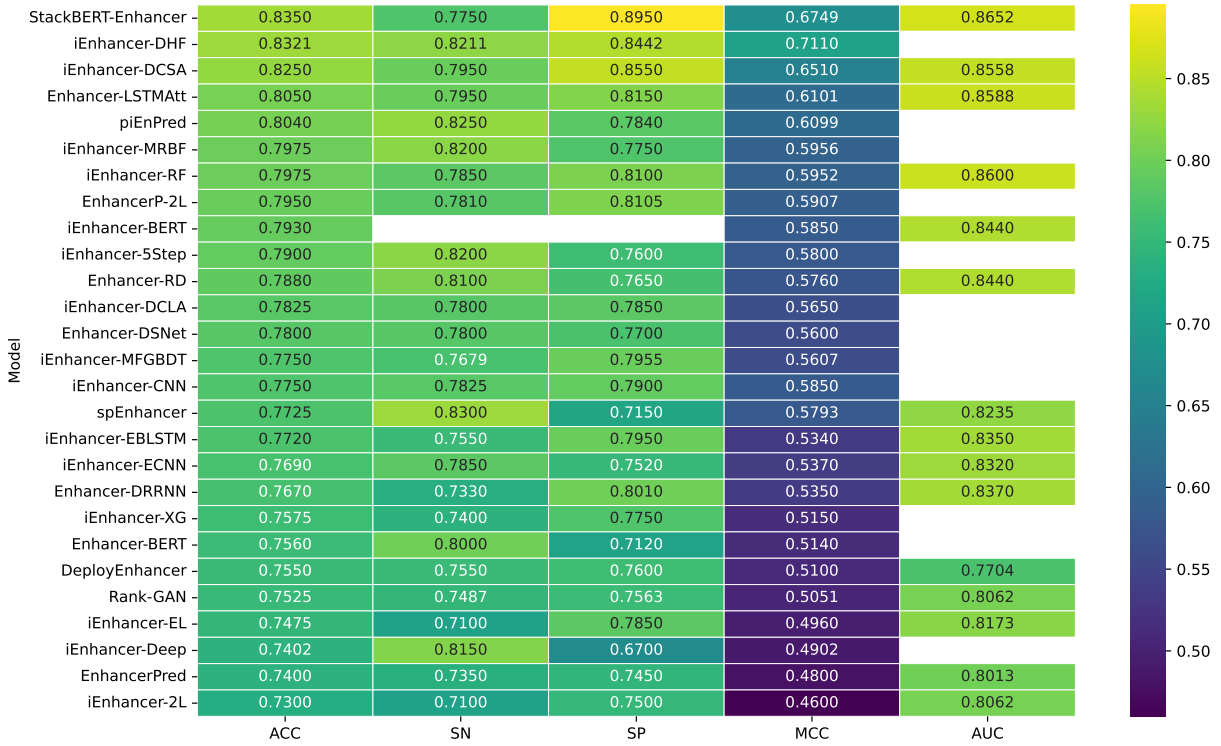
This ablation study investigates the impact of downstream architectural choices when analyzing genomic sequences using features from the pre-trained DNABERT model. Specifically, it compares two approaches for processing token embeddings derived from a pre-trained DNABERT model: one utilizing CNNs and another employing a simple averaging method. The analysis focuses on how these designs influence predictive accuracy, computational efficiency, and ensemble performance.

Architectures Comparison

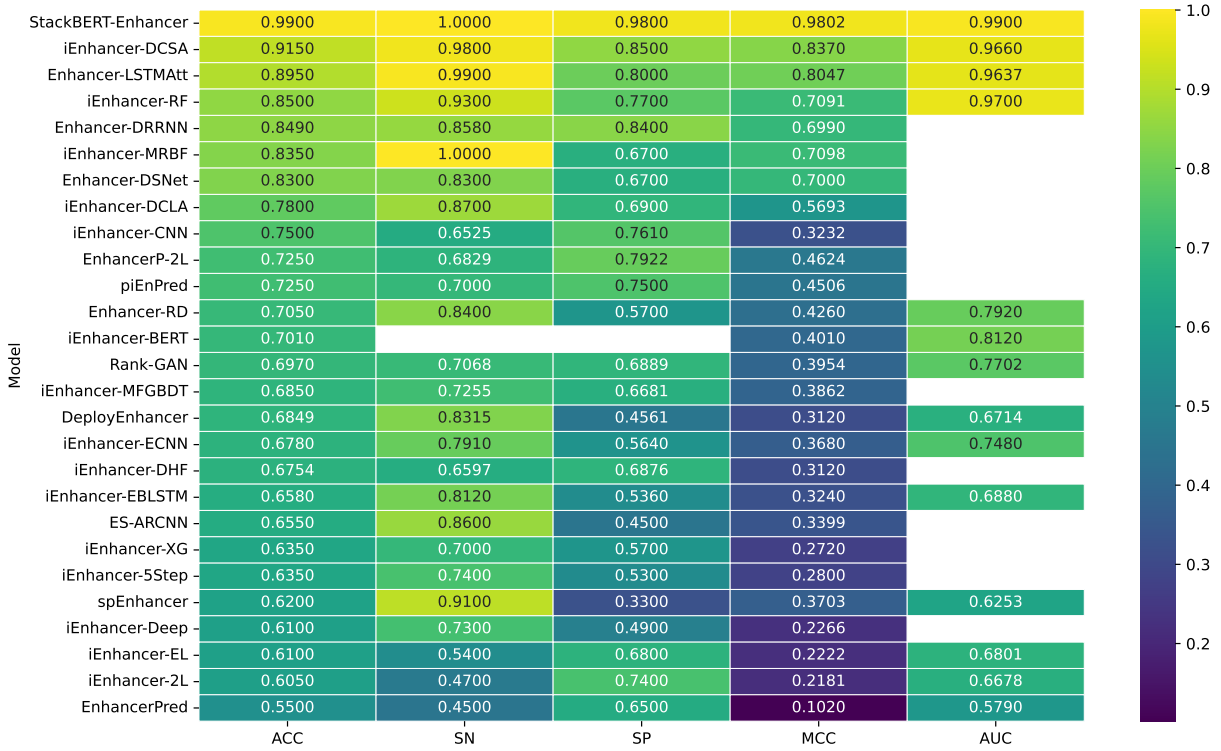
The first architecture employs a CNN layer to further process the token embeddings generated by DNABERT, as illustrated in Figure 5.2. Specifically, the $L \times 768$ dimensional embeddings undergo a one-dimensional convolution with $N_f = 64$ filters and a kernel size $k = 3$. This is followed by a ReLU activation function and max pooling along the sequence length dimension (L). This CNN block is designed to extract higher-level, local features from the token embeddings. The resulting features are then flattened and passed through two fully connected layers with ReLU activation after the first layer and a sigmoid (σ) ac-

Figure 5.1: Model performance heatmaps for Layer I and Layer II.

(a) Layer I.

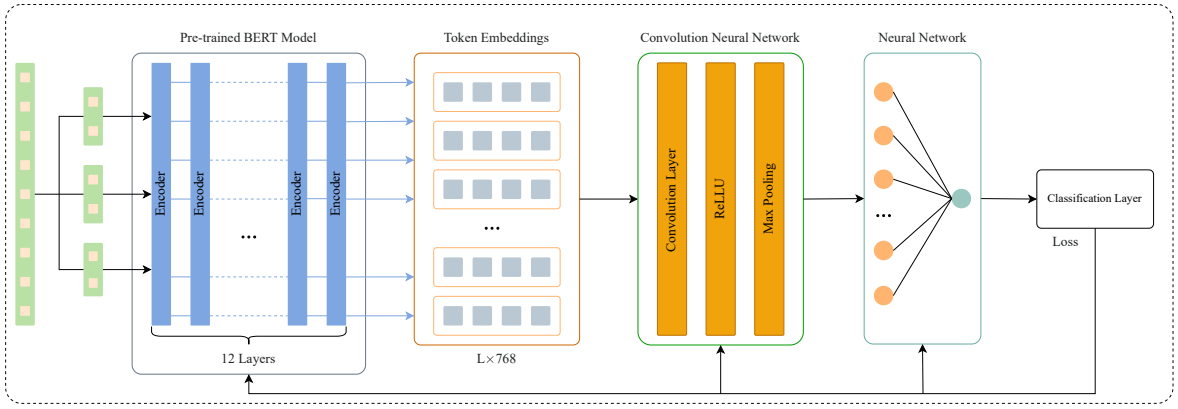


(b) Layer II.



tivation after the second layer for binary classification. While this CNN-based architecture aims to capture intricate sequence patterns, it has a tendency to produce overconfident predictions, which can negatively impact the diversity of predictions in an ensemble setting and thus hinder overall ensemble performance.

Figure 5.2: Model architecture employing a CNN to process token embeddings from a pre-trained BERT model. This more complex approach was initially explored to learn from the enriched output.



In contrast, the current architecture employs an averaging approach, recall Figure 3.8. Instead of using a CNN, the token embeddings are averaged into a single vector of dimensions 1×768. This averaged representation is then passed directly to a neural network for classification. The averaging approach simplifies the model and ensures that predictions are more balanced, avoiding the overconfidence observed in CNN-based outputs.

Results Analysis

The results presented in Table 5.1 and Table 5.2 illustrate a contrast in the range of prediction scores and their implications for model behavior.

Table 5.1 reveals that the individual CNN-based k-mer models often produce prediction scores close to the extremes (0.0 or 1.0). This characteristic suggests a high degree of

certainty in their individual classifications, implying that these models have effectively identified and strongly linked specific k-mer sequences to their corresponding class labels during the training phase. While this strong association contributes to high accuracy on the training data for these individual k-mers, it also raises a potential concern regarding overfitting. The models might have become overly specialized in recognizing the exact k-mer patterns encountered during training, which could limit their capacity to generalize accurately to new, unseen data that may contain slightly different or novel k-mer combinations.

Table 5.1: Results of models with CNN for Layer I, using 10 random samples from the independent test set.

ID	**3-mer**	**4-mer**	**5-mer**	**6-mer**	**Ensemble**	**Label**	**Prediction**	**Correct**
27	0.9994	0.9988	0.9557	0.9995	0.9884	0	1	False
44	0.0002	0.0010	0.6054	0.0023	0.1522	0	0	True
65	0.9516	0.9988	0.9557	0.9996	0.9764	0	1	False
185	0.0001	0.0010	0.0052	0.0001	0.0016	0	0	True
188	0.0001	0.3485	0.9554	0.3412	0.4113	0	0	True
199	0.0001	0.0010	0.0054	0.0001	0.0016	0	0	True
200	0.9994	0.0009	0.9487	0.0001	0.4873	0	0	True
201	0.9994	0.9987	0.9556	0.9994	0.9883	1	1	True
202	0.9994	0.9988	0.9548	0.9995	0.9882	1	1	True
261	0.9993	0.9988	0.9555	0.9995	0.9883	1	1	True
301	0.0001	0.0010	0.0048	0.0001	0.0015	1	0	False
359	0.0002	0.0010	0.0059	0.0001	0.0018	1	0	False
376	0.9988	0.9987	0.9557	0.9996	0.9882	1	1	True
380	0.9994	0.9988	0.9557	0.9995	0.9884	1	1	True

Conversely, the prediction scores observed in Table 5.2 (models without CNN) exhibit a

more moderate range. The individual k-mer models in this table tend to produce prediction scores that are less extreme, with values distributed more towards the center of the probability range (i.e., away from 0.0 and 1.0). This suggests that these models are less certain in their individual predictions compared to their CNN-based counterparts. This lower confidence might indicate that these models have learned more nuanced relationships between k-mers and class labels, potentially making them less susceptible to overfitting the specific patterns in the training data. The more distributed prediction scores could imply a greater ability to handle variations in k-mer sequences and thus potentially lead to better generalization performance on unseen data, even if their accuracy on the training data might be slightly lower than that of the highly confident CNN-based models.

Ensemble Performance and Model Selection

The ensemble predictions in both tables reflect the aggregation of these individual k-mer scores, and their performance will be influenced by the underlying characteristics of the individual models. Specifically, the ensemble scores in Table 5.1 show greater separation between true positives and false predictions compared to the ensemble outputs in Table 5.2. For example, ensemble scores near 0.9884 (in CNN-based models) consistently correspond to correct predictions, whereas similar ensemble confidence levels in the MLP-based models tend to fall in a more ambiguous (lower) range, sometimes misclassifying samples despite moderate confidence. This highlights the advantage of CNNs in providing richer feature representations that enhance ensemble decision-making. However, it also raises concerns regarding potential overconfidence, especially when all individual k-mer models align strongly despite possibly learning redundant or biased representations.

Overall, these findings indicate a trade-off between model confidence and generalization. CNN-based models offer stronger feature learning and higher confidence, potentially boosting accuracy but risking overfitting. The averaging-based models, while perhaps less precise on edge cases (as reflected in lower comparative ensemble performance here), appear

more robust and demonstrate characteristics associated with better generalization potential based on the full set of performance metrics evaluated in this study. Furthermore, the significantly reduced complexity of the averaging architecture translates to substantial savings in training time and computational resources. Given these factors, and prioritizing generalization and efficiency for this specific study, the averaging-based model (without CNN) was deemed the more suitable approach.

Table 5.2: Results of models without CNN for Layer I, using 10 random samples from the independent test set.

ID	**3-mer**	**4-mer**	**5-mer**	**6-mer**	**Ensemble**	**Label**	**Prediction**	**Correct**
27	0.7274	0.8544	0.8206	0.7123	0.7787	0	1	False
44	0.7273	0.1380	0.1712	0.3456	0.3455	0	0	True
65	0.7258	0.1421	0.8206	0.7124	0.6002	0	1	False
185	0.2676	0.1379	0.1712	0.2834	0.2150	0	0	True
188	0.7273	0.1380	0.8206	0.7122	0.5995	0	1	False
199	0.2676	0.1379	0.1712	0.2834	0.2150	0	0	True
200	0.7273	0.1379	0.1734	0.6161	0.4137	0	0	True
201	0.7274	0.8543	0.8206	0.7123	0.7786	1	1	True
202	0.7273	0.8544	0.8206	0.7124	0.7787	1	1	True
261	0.7272	0.8544	0.8206	0.7124	0.7786	1	1	True
301	0.2677	0.1379	0.1712	0.2834	0.2150	1	0	False
359	0.2676	0.1379	0.1712	0.2834	0.2150	1	0	False
376	0.7274	0.8335	0.8206	0.7124	0.7735	1	1	True
380	0.7274	0.8544	0.8206	0.7124	0.7787	1	1	True

Chapter 6

Conclusion

Concluding this study, this chapter synthesizes the research undertaken to develop a novel computational framework for enhancer identification and strength classification using large language models. It provides a summary of key contributions, an assessment of the approach's limitations, a discussion of potential applications within functional genomics, and an outlook on future research directions stemming from this work, reflecting on the framework's development and potential evolution.

Summary of Contributions

Overall, this research has introduced StackBERT-Enhancer, a novel framework, leveraging the power of LLMs like transformer-based architectures. A key innovation is its multi-scale k-mer tokenization strategy, enabling the model to capture intricate sequence patterns and long-range dependencies across various genomic resolutions. These capabilities are often lacking in traditional approaches reliant on fixed k-mer representations or standard convolutional filters. This advanced contextual learning proves effective in distinguishing strong and weak enhancers with potentially higher accuracy.

Addressing the critical need for biological insight, the framework integrates dual inter-

pretability mechanisms: attention scores are analyzed for identifying potential sequence motifs associated with enhancer function, providing valuable insights into regulatory elements, while SHAP analysis offers transparency into the model's decision-making process by interpreting feature importance derived from the BERT embedding's last hidden layer. Furthermore, this work implements a sophisticated dual-layer stacking ensemble approach, which enhances prediction robustness and generalizability by optimally combining predictions from base models trained on the different k-mer scales, mitigating potential biases. Recognizing the computational intensity, distributed multi-GPU training was employed to optimize efficiency, making large-scale genomic analysis and hyperparameter tuning feasible.

Collectively, these contributions establish StackBERT-Enhancer not merely as a high-performing predictive tool, but as a comprehensive and interpretable framework. It successfully bridges advanced AI techniques, multi-scale sequence analysis, sophisticated ensemble learning, and robust interpretability methods, paving the way for more accurate enhancer characterization and a deeper understanding of the complex regulatory code embedded within the genome.

Limitations of the Proposed Approach

Despite its advancements, the proposed framework has certain limitations. One primary challenge is the inherent "black-box" nature of transformer models, which, although mitigated by attention mechanisms, still poses difficulties in fully interpreting model predictions. While attention scores offer a degree of biological insight, they do not entirely replace traditional motif discovery techniques, necessitating further validation through experimental datasets.

In addition, finding optimal hyperparameter configurations proved to be a significant challenge during development. Hyperparameter tuning is critical for improving model perfor-

mance and resource efficiency but often requires extensive experimentation and expertise.

Another limitation is the computational complexity of training large-scale transformer models on genomic sequences. Although distributed multi-GPU systems improve efficiency, the resource-intensive nature of training remains a bottleneck, limiting accessibility for research groups with constrained computational resources. This issue becomes even more pronounced when scaling models to billions of parameters, as seen in recent genomic studies. Additionally, while the stacking ensemble enhances classification performance, it introduces increased model complexity, requiring careful optimization to balance performance and computational overhead.

The framework also focuses primarily on sequence-based features without incorporating additional regulatory elements such as chromatin accessibility or histone modification data. Enhancer function is influenced by multiple epigenetic factors, and the current model does not explicitly integrate these aspects, potentially limiting its predictive power in certain biological contexts.

Potential Applications

Notwithstanding these limitations, the findings of this research hold significant promise for functional genomics. The ability to accurately classify enhancers and distinguish strong from weak regulatory elements can facilitate a deeper understanding of gene expression dynamics. This framework can be employed in large-scale enhancer annotation projects, aiding in the identification of functional regulatory regions across different cell types and conditions.

Moreover, the motif discovery capabilities of the model provide valuable insights for identifying transcription factor binding sites and other regulatory elements critical for gene regulation. These findings can support biomedical research by helping to identify poten-

tial targets for gene therapy and personalized medicine, particularly in diseases driven by enhancer dysregulation, such as cancer and genetic disorders.

Furthermore, the scalable nature of the proposed approach makes it adaptable for analyzing other genomic regulatory elements beyond enhancers. The principles established in this study can be extended to promoters, silencers, and other non-coding regulatory regions, contributing to a broader understanding of genome function.

Directions for Future Research

Building upon the current work, several promising avenues for future research emerge. Future research can expand upon this work by integrating multi-modal genomic data, incorporating chromatin accessibility, histone modifications, and transcription factor binding information to enhance enhancer classification accuracy. By combining sequence-based models with epigenetic features, predictive performance can be further improved, leading to more biologically informed models. Although this framework is currently applied to enhancer DNA sequence data, it can also be readily adapted for promoter data, as promoters share similar sequence characteristics and regulatory features with enhancers.

Another promising direction involves improving model interpretability. While attention mechanisms provide some insight, developing explainable AI techniques tailored for genomic data can enhance the understanding of enhancer function. Techniques such as layer-wise relevance propagation or integrated gradients may offer deeper insights into the contribution of specific sequence elements to enhancer activity.

Further optimization of computational efficiency remains a key area for improvement. Exploring more efficient model architectures, such as sparse transformers or hybrid models that integrate convolutional neural networks with transformers, could reduce computational demands while maintaining high predictive performance. Additionally, developing

methods for training on limited data, such as self-supervised learning or transfer learning, could enable better generalization across different genomic datasets. Incorporating reinforcement learning (RL) for hyperparameter optimization presents another exciting avenue, allowing for automated and potentially more effective tuning of model parameters to maximize performance.

Finally, experimental validation of predicted enhancer motifs and classifications is crucial for bridging the gap between computational predictions and biological reality. Collaborations with experimental biologists to test model predictions in laboratory settings would provide a direct assessment of the framework's biological relevance and its potential for real-world applications in functional genomics.

In conclusion, this work demonstrates a substantial step forward in applying LLMs to enhancer analysis. StackBERT-Enhancer framework provides a solid foundation with clear applications and numerous exciting paths for future research aimed at decoding gene regulation.

References

[1] L. A. Pennacchio, W. Bickmore, A. Dean, M. A. Nobrega, and G. Bejerano, "Enhancers: five essential questions," *Nature Reviews Genetics*, vol. 14, no. 4, pp. 288–295, 2013, ISSN: 1471-0064. DOI: 10.1038/nrg3458. [Online]. Available: https://doi.org/10.1038/nrg3458.

[2] D. Kleftogiannis, P. Kalnis, and V. B. Bajic, "Progress and challenges in bioinformatics approaches for enhancer identification," *Briefings in Bioinformatics*, vol. 17, no. 6, pp. 967–979, 2015, ISSN: 1467-5463. DOI: 10.1093/bib/bbv101. [Online]. Available: https://doi.org/10.1093/bib/bbv101.

[3] E. Calo and J. Wysocka, "Modification of Enhancer Chromatin: What, How, and Why?" *Molecular Cell*, vol. 49, no. 5, pp. 825–837, 2013, ISSN: 1097-2765. DOI: 10.1016/j.molcel.2013.01.038. [Online]. Available: https://doi.org/10.1016/j.molcel.2013.01.038.

[4] V. Chen, M. Yang, W. Cui, J. S. Kim, A. Talwalkar, and J. Ma, "Applying interpretable machine learning in computational biology—pitfalls, recommendations and opportunities for new developments," *Nature Methods*, vol. 21, no. 8, pp. 1454–1461, 2024, ISSN: 1548-7105. DOI: 10.1038/s41592-024-02359-7. [Online]. Available: https://doi.org/10.1038/s41592-024-02359-7.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*,

Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[6] J. Liu, M. Yang, Y. Yu, *et al.*, *Advancing bioinformatics with large language models: components, applications and perspectives*, 2025. arXiv: 2401.04155 [q-bio.QM]. [Online]. Available: https://arxiv.org/abs/2401.04155.

[7] M. Zvyagin, A. Brace, K. Hippe, *et al.*, "GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics," *bioRxiv*, 2022. DOI: 10.1101/2022.10.10.511571. [Online]. Available: https://www.biorxiv.org/content/early/2022/11/23/2022.10.10.511571.

[8] M. Liang, T. Chang, B. An, *et al.*, "A Stacking Ensemble Learning Framework for Genomic Prediction," *Frontiers in Genetics*, vol. 12, 2021, ISSN: 1664-8021. DOI: 10.3389/fgene.2021.600040. [Online]. Available: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2021.600040.

[9] N. J. Fuda, M. B. Ardehali, and J. T. Lis, "Defining mechanisms that regulate RNA polymerase II transcription *in vivo*," *Nature*, vol. 461, no. 7261, pp. 186–192, 2009, ISSN: 1476-4687. DOI: 10.1038/nature08449. [Online]. Available: https://doi.org/10.1038/nature08449.

[10] B. P. H. Metzger, F. Duveau, D. C. Yuan, S. Tryban, B. Yang, and P. J. Wittkopp, "Contrasting Frequencies and Effects of *cis*- and *trans*-Regulatory Mutations Affecting Gene Expression," *Molecular Biology and Evolution*, vol. 33, no. 5, pp. 1131–1146, 2016, ISSN: 0737-4038. DOI: 10.1093/molbev/msw011. [Online]. Available: https://doi.org/10.1093/molbev/msw011.

[11] P. J. Wittkopp and G. Kalay, "*Cis*-regulatory elements: molecular mechanisms and evolutionary processes underlying divergence," *Nature Reviews Genetics*, vol. 13, no. 1, pp. 59–69, 2012, ISSN: 1471-0064. DOI: 10.1038/nrg3095. [Online]. Available: https://doi.org/10.1038/nrg3095.

[12] S. Preissl, K. J. Gaulton, and B. Ren, "Characterizing *cis*-regulatory elements using single-cell epigenomics," *Nature Reviews Genetics*, vol. 24, no. 1, pp. 21–43, 2023, ISSN: 1471-0064. DOI: 10.1038/s41576-022-00509-1. [Online]. Available: https://doi.org/10.1038/s41576-022-00509-1.

[13] S. A. Signor and S. V. Nuzhdin, "The evolution of gene expression in *cis* and *trans*," *Trends in Genetics*, vol. 34, no. 7, pp. 532–544, 2018, ISSN: 0168-9525. DOI: 10.1016/j.tig.2018.03.007. [Online]. Available: https://doi.org/10.1016/j.tig.2018.03.007.

[14] V. Sundaram and J. Wysocka, "Transposable elements as a potent source of diverse *cis*-regulatory sequences in mammalian genomes," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 375, no. 1795, p. 20 190 347, 2020. DOI: 10.1098/rstb.2019.0347. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2019.0347.

[15] Genomics Education Programme, *Non-coding DNA — Knowledge Hub*, 2024. [Online]. Available: https://www.genomicseducation.hee.nhs.uk/genotes/knowledge-hub/non-coding-dna/.

[16] M. Erokhin, Y. Vassetzky, P. Georgiev, and D. Chetverina, "Eukaryotic enhancers: common features, regulation, and participation in diseases," *Cellular and Molecular Life Sciences*, vol. 72, no. 12, pp. 2361–2375, 2015, ISSN: 1420-9071. DOI: 10.1007/s00018-015-1871-9. [Online]. Available: https://doi.org/10.1007/s00018-015-1871-9.

[17] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*, 4th. Garland Science, 2002, ISBN: 0-8153-3218-1.

[18] J. Guo, "Transcription: the epicenter of gene expression," *Journal of Zhejiang University SCIENCE B*, vol. 15, no. 5, pp. 409-4–11, 2014, ISSN: 1862-1783. DOI: 10.1631/jzus.B1400113. [Online]. Available: https://doi.org/10.1631/jzus.B1400113.

[19] M. Bansal, A. Kumar, and V. R. Yella, "Role of DNA sequence based structural features of promoters in transcription initiation and gene expression," *Current Opinion in Structural Biology*, vol. 25, pp. 77–85, 2014, Theory and simulation / Macromolecular machines, ISSN: 0959-440X. DOI: 10.1016/j.sbi.2014.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959440X14000086.

[20] Y. Kang, Y. W. Kim, J. Kang, and A. Kim, "Histone H3K4me1 and H3K27ac play roles in nucleosome eviction and eRNA transcription, respectively, at enhancers," *The FASEB Journal*, vol. 35, no. 8, e21781, 2021. DOI: 10.1096/fj.202100488R. [Online]. Available: https://faseb.onlinelibrary.wiley.com/doi/abs/10.1096/fj.202100488R.

[21] W. Hu, Y. Li, Y. Wu, L. Guan, and M. Li, "A deep learning model for DNA enhancer prediction based on nucleotide position aware feature encoding," *iScience*, vol. 27, no. 6, 2024, ISSN: 2589-0042. DOI: 10.1016/j.isci.2024.110030. [Online]. Available: https://doi.org/10.1016/j.isci.2024.110030.

[22] P. J. Park, "ChIP–seq: advantages and challenges of a maturing technology," *Nature Reviews Genetics*, vol. 10, no. 10, pp. 669–680, 2009, ISSN: 1471-0064. DOI: 10.1038/nrg2641. [Online]. Available: https://doi.org/10.1038/nrg2641.

[23] F. C. Grandi, H. Modi, L. Kampman, and M. R. Corces, "Chromatin accessibility profiling by ATAC-seq," *Nature Protocols*, vol. 17, no. 6, pp. 1518–1552, 2022, ISSN: 1750-2799. DOI: 10.1038/s41596-022-00692-9. [Online]. Available: https://doi.org/10.1038/s41596-022-00692-9.

[24] D. Chahar, K. Mrouj, G. Dubra, L. Krasinska, and D. Fisher, *Next Generation Sequencing Quantitative Analysis of Wild Type and Mki67-/- 4T1 mouse mammary carcinoma cell histone modification*, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE163479.

[25] D. Shlyueva, G. Stampfel, and A. Stark, "Transcriptional enhancers: from properties to genome-wide predictions," *Nature Reviews Genetics*, vol. 15, no. 4, pp. 272–286, 2014, ISSN: 1471-0064. DOI: 10.1038/nrg3682. [Online]. Available: https://doi.org/10.1038/nrg3682.

[26] A. Tafessu and L. A. Banaszynski, "Establishment and function of chromatin modification at enhancers," *Open Biology*, vol. 10, no. 10, p. 200 255, 2020. DOI: 10.1098/rsob.200255. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsob.200255.

[27] A. Barral and J. D. and, "The chromatin signatures of enhancers and their dynamic regulation," *Nucleus*, vol. 14, no. 1, p. 2 160 551, 2023, PMID: 36602897. DOI: 10.1080/19491034.2022.2160551. [Online]. Available: https://doi.org/10.1080/19491034.2022.2160551.

[28] F. Grosveld, J. van Staalduinen, and R. Stadhouders, "Transcriptional Regulation by (Super)Enhancers: From Discovery to Mechanisms," *Annual Review of Genomics and Human Genetics*, vol. 22, no. Volume 22, 2021, pp. 127–146, 2021, ISSN: 1545-293X. DOI: 10.1146/annurev-genom-122220-093818. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-genom-122220-093818.

[29] K. Niu, X. Luo, S. Zhang, Z. Teng, T. Zhang, and Y. Zhao, "iEnhancer-EBLSTM: Identifying Enhancers and Strengths by Ensembles of Bidirectional Long Short-Term Memory," *Frontiers in Genetics*, vol. 12, 2021, ISSN: 1664-8021. DOI: 10.3389/fgene.2021.665498. [Online]. Available: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2021.665498.

[30] C. Jia and W. He, "EnhancerPred: a predictor for discovering enhancers based on the combination and selection of multiple features," *Scientific Reports*, vol. 6, no. 1, p. 38 741, 2016, ISSN: 2045-2322. DOI: 10.1038/srep38741. [Online]. Available: https://doi.org/10.1038/srep38741.

[31] S. L. Klemm, Z. Shipony, and W. J. Greenleaf, "Chromatin accessibility and the regulatory epigenome," *Nature Reviews Genetics*, vol. 20, no. 4, pp. 207–220, 2019, ISSN: 1471-0064. DOI: 10.1038/s41576-018-0089-8. [Online]. Available: https://doi.org/10.1038/s41576-018-0089-8.

[32] G. Wang, F. Wang, Q. Huang, Y. Li, Y. Liu, and Y. Wang, "Understanding Transcription Factor Regulation by Integrating Gene Expression and DNase I Hypersensitive Sites," *BioMed Research International*, vol. 2015, no. 1, p. 757 530, 2015. DOI: 10.1155/2015/757530. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2015/757530.

[33] G. D. Erwin, N. Oksenberg, R. M. Truty, *et al.*, "Integrating Diverse Datasets Improves Developmental Enhancer Prediction," *PLOS Computational Biology*, vol. 10, no. 6, pp. 1–20, 2014. DOI: 10.1371/journal.pcbi.1003677. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003677.

[34] M. Ghandi, D. Lee, M. Mohammad-Noori, and M. A. Beer, "Enhanced Regulatory Sequence Prediction Using Gapped k-mer Features," *PLOS Computational Biology*, vol. 10, no. 7, pp. 1–15, 2014. DOI: 10.1371/journal.pcbi.1003711. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003711.

[35] S. G. Kim, M. Harwani, A. Grama, and S. Chaterji, "EP-DNN: A Deep Neural Network-Based Global Enhancer Prediction Algorithm," *Scientific Reports*, vol. 6, no. 1, p. 38 433, 2016, ISSN: 2045-2322. DOI: 10.1038/srep38433. [Online]. Available: https://doi.org/10.1038/srep38433.

[36] B. Liu, L. Fang, R. Long, X. Lan, and K.-C. Chou, "iEnhancer-2L: a two-layer predictor for identifying enhancers and their strength by pseudo k-tuple nucleotide composition," *Bioinformatics*, vol. 32, no. 3, pp. 362–369, 2015, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv604. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv604.

[37] B. Liu, K. Li, D.-S. Huang, and K.-C. Chou, "iEnhancer-EL: identifying enhancers and their strength with ensemble learning approach," *Bioinformatics*, vol. 34, no. 22, pp. 3835–3842, 2018, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty458. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty458.

[38] N. Q. K. Le, E. K. Y. Yapp, Q.-T. Ho, N. Nagasundaram, Y.-Y. Ou, and H.-Y. Yeh, "iEnhancer-5Step: Identifying enhancers using hidden information of DNA sequences via Chou's 5-step rule and word embedding," *Analytical Biochemistry*, vol. 571, pp. 53–61, 2019, ISSN: 0003-2697. DOI: 10.1016/j.ab.2019.02.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003269719300788.

[39] Y. Liang, S. Zhang, H. Qiao, and Y. Cheng, "iEnhancer-MFGBDT: Identifying enhancers and their strength by fusing multiple features and gradient boosting decision tree," *Mathematical Biosciences and Engineering*, vol. 18, no. 6, pp. 8797–8814, 2021, ISSN: 1551-0018. DOI: 10.3934/mbe.2021434. [Online]. Available: https://www.aimspress.com/article/doi/10.3934/mbe.2021434.

[40] L. Cai, X. Ren, X. Fu, L. Peng, M. Gao, and X. Zeng, "iEnhancer-XG: interpretable sequence-based enhancers and their strength predictor," *Bioinformatics*, vol. 37, no. 8, pp. 1060–1067, 2020, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa914. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa914.

[41] Z. Xiao, L. Wang, Y. Ding, and L. Yu, "iEnhancer-MRBF: Identifying enhancers and their strength with a multiple Laplacian-regularized radial basis function network," *Methods*, vol. 208, pp. 1–8, 2022, ISSN: 1046-2023. DOI: 10.1016/j.ymeth.2022.10.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1046202322002183.

[42] A. H. Butt, S. Alkhalaf, S. Iqbal, and Y. D. Khan, "EnhancerP-2L: A Gene regulatory site identification tool for DNA enhancer region using CREs motifs," *bioRxiv*,

2020. DOI: 10.1101/2020.01.20.912451. [Online]. Available: https://www.biorxiv.org/content/early/2020/01/20/2020.01.20.912451.

[43] Y. Lyu, Z. Zhang, J. Li, W. He, Y. Ding, and F. Guo, "iEnhancer-KL: A Novel Two-Layer Predictor for Identifying Enhancers by Position Specific of Nucleotide Composition," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 6, pp. 2809–2815, 2021. DOI: 10.1109/TCBB.2021.3053608.

[44] D. Y. Lim, J. Khanal, H. Tayara, and K. T. Chong, "iEnhancer-RF: Identifying enhancers and their strength by enhanced feature representation using random forest," *Chemometrics and Intelligent Laboratory Systems*, vol. 212, p. 104 284, 2021, ISSN: 0169-7439. DOI: 10.1016/j.chemolab.2021.104284. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169743921000526.

[45] B. Liu, "iEnhancer-PsedeKNC: Identification of enhancers and their subgroups based on Pseudo degenerate kmer nucleotide composition," *Neurocomputing*, vol. 217, pp. 46–52, 2016, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.12.138.

[46] Z. U. Khan, D. Pi, S. Yao, A. Nawaz, F. Ali, and S. Ali, "piEnPred: a bi-layered discriminative model for enhancers and their subtypes via novel cascade multi-level subset feature selection algorithm," *Frontiers of Computer Science*, vol. 15, no. 6, p. 156 904, 2021, ISSN: 2095-2236. DOI: 10.1007/s11704-020-9504-3. [Online]. Available: https://doi.org/10.1007/s11704-020-9504-3.

[47] D. W. Otter, J. R. Medina, and J. K. Kalita, "A Survey of the Usages of Deep Learning for Natural Language Processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021. DOI: 10.1109/TNNLS.2020.2979670.

[48] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019, ISSN: 1573-756X. DOI: 10.1007/s10618-019-00619-1. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1.

[49] N. Bento, J. Rebelo, M. Barandas, *et al.*, "Comparing Handcrafted Features and Deep Neural Representations for Domain Generalization in Human Activity Recognition," *Sensors*, vol. 22, no. 19, 2022, ISSN: 1424-8220. DOI: 10.3390/s22197324. [Online]. Available: https://www.mdpi.com/1424-8220/22/19/7324.

[50] Q. H. Nguyen, T.-H. Nguyen-Vo, N. Q. K. Le, T. T. Do, S. Rahardja, and B. P. Nguyen, "iEnhancer-ECNN: identifying enhancers and their strength using ensembles of convolutional neural networks," *BMC Genomics*, vol. 20, no. 9, p. 951, 2019, ISSN: 1471-2164. DOI: 10.1186/s12864-019-6336-3. [Online]. Available: https://doi.org/10.1186/s12864-019-6336-3.

[51] H. Kamran, M. Tahir, H. Tayara, and K. T. Chong, "iEnhancer-Deep: A Computational Predictor for Enhancer Sites and Their Strength Using Deep Learning," *Applied Sciences*, vol. 12, no. 4, 2022, ISSN: 2076-3417. DOI: 10.3390/app12042120. [Online]. Available: https://www.mdpi.com/2076-3417/12/4/2120.

[52] X. Mu, Y. Wang, M. Duan, *et al.*, "A Novel Position-Specific Encoding Algorithm (SeqPose) of Nucleotide Sequences and Its Application for Detecting Enhancers," *International Journal of Molecular Sciences*, vol. 22, no. 6, 2021, ISSN: 1422-0067. DOI: 10.3390/ijms22063079. [Online]. Available: https://www.mdpi.com/1422-0067/22/6/3079.

[53] G. Huang, W. Luo, G. Zhang, *et al.*, "Enhancer-LSTMAtt: A Bi-LSTM and Attention-Based Deep Learning Method for Enhancer Recognition," *Biomolecules*, vol. 12, no. 7, 2022, ISSN: 2218-273X. DOI: 10.3390/biom12070995. [Online]. Available: https://www.mdpi.com/2218-273X/12/7/995.

[54] M. Liao, J.-p. Zhao, J. Tian, and C.-H. Zheng, "iEnhancer-DCLA: using the original sequence to identify enhancers and their strength based on a deep learning framework," *BMC Bioinformatics*, vol. 23, no. 1, p. 480, 2022, ISSN: 1471-2105. DOI: 10.1186/s12859-022-05033-x. [Online]. Available: https://doi.org/10.1186/s12859-022-05033-x.

[55] L. Gonog and Y. Zhou, "A Review: Generative Adversarial Networks," in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2019, pp. 505–510. DOI: 10.1109/ICIEA.2019.8833686.

[56] R. Yang, F. Wu, C. Zhang, and L. Zhang, "iEnhancer-GAN: A Deep Learning Framework in Combination with Word Embedding and Sequence Generative Adversarial Net to Identify Enhancers and Their Strength," *International Journal of Molecular Sciences*, vol. 22, no. 7, 2021, ISSN: 1422-0067. DOI: 10.3390/ijms22073589. [Online]. Available: https://www.mdpi.com/1422-0067/22/7/3589.

[57] Q. Geng, R. Yang, and L. Zhang, "A deep learning framework for enhancer prediction using word embedding and sequence generation," *Biophysical Chemistry*, vol. 286, p. 106 822, 2022, ISSN: 0301-4622. DOI: 10.1016/j.bpc.2022.106822. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301462222000643.

[58] H. Yang, S. Wang, and X. Xia, "iEnhancer-RD: Identification of enhancers and their strength using RKPK features and deep neural networks," *Analytical Biochemistry*, vol. 630, p. 114 318, 2021, ISSN: 0003-2697. DOI: 10.1016/j.ab.2021.114318. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003269721002190.

[59] J. Khanal, H. Tayara, and K. T. Chong, "Identifying enhancers and their strength by the integration of word embedding and convolution neural network," *IEEE Access*, vol. 8, pp. 58 369–58 376, 2020. DOI: 10.1109/ACCESS.2020.2982666.

[60] T.-H. Zhang, M. Flores, and Y. Huang, "ES-ARCNN: Predicting enhancer strength by using data augmentation and residual convolutional neural network," *Analytical Biochemistry*, vol. 618, p. 114 120, 2021, ISSN: 0003-2697. DOI: 10.1016/j.ab.2021.114120. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000326972100021X.

[61] K. K. Tan, N. Q. K. Le, H.-Y. Yeh, and M. C. H. Chua, "Ensemble of Deep Recurrent Neural Networks for Identifying Enhancers via Dinucleotide Physicochemical Properties," *Cells*, vol. 8, no. 7, 2019, ISSN: 2073-4409. DOI: 10.3390/cells8070767. [Online]. Available: https://www.mdpi.com/2073-4409/8/7/767.

[62] M. N. Asim, M. A. Ibrahim, M. I. Malik, A. Dengel, and S. Ahmed, "Enhancer-DSNet: A Supervisedly Prepared Enriched Sequence Representation for the Identification of Enhancers and Their Strength," in *Neural Information Processing*, Cham: Springer International Publishing, 2020, pp. 38–48, ISBN: 978-3-030-63836-8.

[63] N. Inayat, M. Khan, N. Iqbal, *et al.*, "iEnhancer-DHF: Identification of Enhancers and Their Strengths Using Optimize Deep Neural Network With Multiple Features Extraction Methods," *IEEE Access*, vol. 9, pp. 40 783–40 796, 2021. DOI: 10.1109/ACCESS.2021.3062291.

[64] W. Wang, Q. Wu, and C. Li, "iEnhancer-DCSA: identifying enhancers via dual-scale convolution and spatial attention," *BMC Genomics*, vol. 24, no. 1, p. 393, 2023, ISSN: 1471-2164. DOI: 10.1186/s12864-023-09468-1. [Online]. Available: https://doi.org/10.1186/s12864-023-09468-1.

[65] Q. Li, L. Xu, Q. Li, and L. Zhang, "Identification and Classification of Enhancers Using Dimension Reduction Technique and Recurrent Neural Network," *Computational and Mathematical Methods in Medicine*, vol. 2020, no. 1, p. 8 852 258, 2020. DOI: 10.1155/2020/8852258. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2020/8852258.

[66] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[67] S. Islam, H. Elmekki, A. Elsebai, *et al.*, "A comprehensive survey on applications of transformers for deep learning tasks," *Expert Systems with Applications*, vol. 241,

p. 122 666, 2024, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2023.122666. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423031688.

[68] S. Zhang, R. Fan, Y. Liu, S. Chen, Q. Liu, and W. Zeng, "Applications of transformer-based language models in bioinformatics: a survey," *Bioinformatics Advances*, vol. 3, no. 1, vbad001, Jan. 2023, ISSN: 2635-0041. DOI: 10.1093/bioadv/vbad001. [Online]. Available: https://doi.org/10.1093/bioadv/vbad001.

[69] A. Chandra, L. Tünnermann, T. Löfstedt, and R. Gratz, "Transformer-based deep learning for predicting protein properties in the life sciences," *eLife*, vol. 12, e82819, 2023, ISSN: 2050-084X. DOI: 10.7554/eLife.82819. [Online]. Available: https://doi.org/10.7554/eLife.82819.

[70] Y. Ji, 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, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab083. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab083.

[71] Ž. Avsec, V. Agarwal, D. Visentin, *et al.*, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature Methods*, vol. 18, no. 10, pp. 1196–1203, 2021, ISSN: 1548-7105. DOI: 10.1038/s41592-021-01252-x. [Online]. Available: https://doi.org/10.1038/s41592-021-01252-x.

[72] H. Dalla-Torre, L. Gonzalez, J. Mendoza-Revilla, *et al.*, "Nucleotide Transformer: building and evaluating robust foundation models for human genomics," *Nature Methods*, vol. 22, no. 2, pp. 287–297, 2025, ISSN: 1548-7105. DOI: 10.1038/s41592-024-02523-z. [Online]. Available: https://doi.org/10.1038/s41592-024-02523-z.

[73] F. D. Keles, P. M. Wijewardena, and C. Hegde, "On the computational complexity of self-attention," in *International Conference on Algorithmic Learning Theory*, PMLR, 2023, pp. 597–619.

[74] G. Benegas, C. Ye, C. Albors, J. C. Li, and Y. S. Song, "Genomic Language Models: Opportunities and Challenges," 2024.

[75] S. R. Choi and M. Lee, "Transformer Architecture and Attention Mechanisms in Genome Data Analysis: A Comprehensive Review," *Biology*, vol. 12, no. 7, 2023, ISSN: 2079-7737. DOI: 10.3390/biology12071033. [Online]. Available: https://www.mdpi.com/2079-7737/12/7/1033.

[76] F. Vallania, A. Tam, S. Lofgren, *et al.*, "Leveraging heterogeneity across multiple datasets increases cell-mixture deconvolution accuracy and reduces biological and technical biases," *Nature Communications*, vol. 9, no. 1, p. 4735, 2018, ISSN: 2041-1723. DOI: 10.1038/s41467-018-07242-6. [Online]. Available: https://doi.org/10.1038/s41467-018-07242-6.

[77] T. Yue, Y. Wang, L. Zhang, *et al.*, "Deep Learning for Genomics: From Early Neural Nets to Modern Large Language Models," *International Journal of Molecular Sciences*, vol. 24, no. 21, 2023, ISSN: 1422-0067. DOI: 10.3390/ijms242115858. [Online]. Available: https://www.mdpi.com/1422-0067/24/21/15858.

[78] N. Q. K. Le, Q.-T. Ho, T.-T.-D. Nguyen, and Y.-Y. Ou, "A transformer architecture based on BERT and 2D convolutional neural network to identify DNA enhancers from sequence information," *Briefings in Bioinformatics*, vol. 22, no. 5, bbab005, 2021, ISSN: 1477-4054. DOI: 10.1093/bib/bbab005. [Online]. Available: https://doi.org/10.1093/bib/bbab005.

[79] H. Luo, C. Chen, W. Shan, P. Ding, and L. Luo, "iEnhancer-BERT: A Novel Transfer Learning Architecture Based on DNA-Language Model for Identifying Enhancers and Their Strength," in *Intelligent Computing Theories and Application*, Cham: Springer International Publishing, 2022, pp. 153–165, ISBN: 978-3-031-13829-4.

[80] Y. Hwang, A. L. Cornman, E. H. Kellogg, S. Ovchinnikov, and P. R. Girguis, "Genomic language model predicts protein co-regulation and function," *Nature Communications*, vol. 15, no. 1, p. 2880, 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-46947-9. [Online]. Available: https://doi.org/10.1038/s41467-024-46947-9.

[81] F. Ullah and A. Ben-Hur, "A self-attention model for inferring cooperativity between regulatory features," *Nucleic Acids Research*, vol. 49, no. 13, e77, May 2021, ISSN: 0305-1048. DOI: 10.1093/nar/gkab349. [Online]. Available: https://doi.org/10.1093/nar/gkab349.

[82] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences," in *International Conference on Machine Learning*, PMlR, 2017, pp. 3145–3153.

[83] N. Brandes, G. Goldman, C. H. Wang, C. J. Ye, and V. Ntranos, "Genome-wide prediction of disease variant effects with a deep protein language model," *Nature Genetics*, vol. 55, no. 9, pp. 1512–1522, 2023, ISSN: 1546-1718. DOI: 10.1038/s41588-023-01465-0. [Online]. Available: https://doi.org/10.1038/s41588-023-01465-0.

[84] J. Cheng, G. Novati, J. Pan, *et al.*, "Accurate proteome-wide missense variant effect prediction with AlphaMissense," *Science*, vol. 381, no. 6664, eadg7492, 2023. DOI: 10.1126/science.adg7492. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.adg7492.

[85] M. Hu, S. Alkhairy, I. Lee, *et al.*, "Evaluation of large language models for discovery of gene set function," *Nature Methods*, vol. 22, no. 1, pp. 82–91, 2025, ISSN: 1548-7105. DOI: 10.1038/s41592-024-02525-x. [Online]. Available: https://doi.org/10.1038/s41592-024-02525-x.

[86] S. Cheng, Y. Wei, Y. Zhou, *et al.*, *Deciphering genomic codes using advanced NLP techniques: a scoping review*, 2024. [Online]. Available: https://arxiv.org/abs/2411.16084.

[87] L. M. Lindsey, N. L. Pershing, A. Habib, W. Z. Stephens, A. J. Blaschke, and H. Sundar, "A Comparison of Tokenization Impact in Attention Based and State Space Genomic Language Models," *bioRxiv*, 2024. DOI: 10.1101/2024.09.09.612081. [Online]. Available: https://www.biorxiv.org/content/early/2024/09/14/2024.09.09.612081.

[88] W. Esser-Skala and N. Fortelny, "Reliable interpretability of biology-inspired deep neural networks," *npj Systems Biology and Applications*, vol. 9, no. 1, p. 50, 2023, ISSN: 2056-7189. DOI: 10.1038/s41540-023-00310-8. [Online]. Available: https://doi.org/10.1038/s41540-023-00310-8.

[89] R. S. Ghotra, N. K. Lee, and P. K. Koo, "Uncovering motif interactions from convolutional-attention networks for genomics," in *NeurIPS 2021 AI for Science Workshop*, 2021.

[90] I. S. Schwartz, K. E. Link, R. Daneshjou, and N. Cortés-Penfield, "Black Box Warning: Large Language Models and the Future of Infectious Diseases Consultation," *Clinical Infectious Diseases*, vol. 78, no. 4, pp. 860–866, 2023, ISSN: 1058-4838. DOI: 10.1093/cid/ciad633. [Online]. Available: https://doi.org/10.1093/cid/ciad633.

[91] T. Savage, A. Nayak, R. Gallo, E. Rangan, and J. H. Chen, "Diagnostic reasoning prompts reveal the potential for large language model interpretability in medicine," *npj Digital Medicine*, vol. 7, no. 1, p. 20, 2024, ISSN: 2398-6352. DOI: 10.1038/s41746-024-01010-1. [Online]. Available: https://doi.org/10.1038/s41746-024-01010-1.

[92] D. Wang and S. Zhang, "Large language models in medical and healthcare fields: applications, advances, and challenges," *Artificial Intelligence Review*, vol. 57, no. 11, p. 299, 2024, ISSN: 1573-7462. DOI: 10.1007/s10462-024-10921-0. [Online]. Available: https://doi.org/10.1007/s10462-024-10921-0.

[93] M. E. Consens, C. Dufault, M. Wainberg, *et al.*, *To Transformers and Beyond: Large Language Models for the Genome*, 2023. [Online]. Available: https://arxiv.org/abs/2311.07621.

[94] M. S. Jain, K. Polanski, C. D. Conde, *et al.*, "MultiMAP: dimensionality reduction and integration of multimodal data," *Genome Biology*, vol. 22, no. 1, p. 346, 2021, ISSN: 1474-760X. DOI: 10.1186/s13059-021-02565-y. [Online]. Available: https://doi.org/10.1186/s13059-021-02565-y.

[95] T. Ashuach, M. I. Gabitto, R. V. Koodli, G.-A. Saldi, M. I. Jordan, and N. Yosef, "MultiVI: deep generative model for the integration of multimodal data," *Nature Methods*, vol. 20, no. 8, pp. 1222–1231, 2023, ISSN: 1548-7105. DOI: 10.1038/s41592-023-01909-9. [Online]. Available: https://doi.org/10.1038/s41592-023-01909-9.

[96] X. Tang, J. Zhang, Y. He, *et al.*, "Explainable multi-task learning for multi-modality biological data analysis," *Nature Communications*, vol. 14, no. 1, p. 2546, 2023, ISSN: 2041-1723. DOI: 10.1038/s41467-023-37477-x. [Online]. Available: https://doi.org/10.1038/s41467-023-37477-x.

[97] J. Ernst and M. Kellis, "ChromHMM: automating chromatin-state discovery and characterization," *Nature Methods*, vol. 9, no. 3, pp. 215–216, 2012, ISSN: 1548-7105. DOI: 10.1038/nmeth.1906. [Online]. Available: https://doi.org/10.1038/nmeth.1906.

[98] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl158. [Online]. Available: https://doi.org/10.1093/bioinformatics/btl158.

[99] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[100] L. McInnes, J. Healy, N. Saul, and L. Großberger, "UMAP: Uniform Manifold Approximation and Projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018. DOI: 10.21105/joss.00861. [Online]. Available: https://doi.org/10.21105/joss.00861.

[101] Z. Zhou, W. Wu, H. Ho, *et al.*, *Dnabert-s: Pioneering species differentiation with species-aware dna embeddings*, 2024. arXiv: 2402.08777 [q-bio.GN].

[102] C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. Chikhi, "Data structures based on k-mers for querying large collections of sequencing datasets," *bioRxiv*, 2020. DOI: 10.1101/866756. [Online]. Available: https://www.biorxiv.org/content/early/2020/07/18/866756.

[103] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[104] L. Datta, *A Survey on Activation Functions and their relation with Xavier and He Normal Initialization*, 2020. arXiv: 2004.06632 [cs.NE]. [Online]. Available: https://arxiv.org/abs/2004.06632.

[105] E. Cule and M. De Iorio, "Ridge Regression in Prediction Problems: Automatic Choice of the Ridge Parameter," *Genetic Epidemiology*, vol. 37, no. 7, pp. 704–714, 2013. DOI: 10.1002/gepi.21750. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/gepi.21750.

[106] S. Li, Y. Zhao, R. Varma, *et al.*, "PyTorch distributed: experiences on accelerating data parallel training," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3005–3018, 2020, ISSN: 2150-8097. DOI: 10.14778/3415478.3415530. [Online]. Available: https://doi.org/10.14778/3415478.3415530.

[107] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980.

[108] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer Series in Statistics), 2nd ed. Springer, 2009, ISBN: 9780387848846.

[109] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Ensemble Learning," in *Data Mining: Practical Machine Learning Tools and Techniques*, ser. Morgan Kaufmann Series in Data Management Systems, 4th, Morgan Kaufmann, 2016, ch. 8, pp. 351–374, ISBN: 9780128042915.

[110] A. Mohammed and R. Kora, "A comprehensive review on ensemble deep learning: Opportunities and challenges," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 757–774, 2023, ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2023.01.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157823000228.

[111] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992, ISSN: 0893-6080. DOI: 10.1016/S0893-6080(05)80023-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[112] M. P. Sesmero, A. I. Ledezma, and A. Sanchis, "Generating ensembles of heterogeneous classifiers using Stacked Generalization," *WIREs Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 21–34, 2015. DOI: 10.1002/widm.1143. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1143.

[113] F. Zambelli, G. Pesole, and G. Pavesi, "Motif discovery and transcription factor binding sites before and after the next-generation sequencing era," *Briefings in Bioinformatics*, vol. 14, no. 2, pp. 225–237, 2012, ISSN: 1467-5463. DOI: 10.1093/bib/bbs016. [Online]. Available: https://doi.org/10.1093/bib/bbs016.

[114] F. A. Hashim, M. S. Mabrouk, and W. Al-Atabany, "Review of Different Sequence Motif Finding Algorithms," *Avicenna J Med Biotechnol*, vol. 11, no. 2, pp. 130–148, 2019.

[115] N. E. Davey, R. J. Edwards, and D. C. Shields, "The SLiMDisc server: short, linear motif discovery in proteins," *Nucleic Acids Research*, vol. 35, no. suppl_2, W455–W459, 2007, ISSN: 0305-1048. DOI: 10.1093/nar/gkm400. [Online]. Available: https://doi.org/10.1093/nar/gkm400.

[116] K. D. MacIsaac and E. Fraenkel, "Practical Strategies for Discovering Regulatory DNA Sequence Motifs," *PLOS Computational Biology*, vol. 2, no. 4, pp. 1–10, 2006. DOI: 10.1371/journal.pcbi.0020036. [Online]. Available: https://doi.org/10.1371/journal.pcbi.0020036.

[117] M. K. Das and H.-K. Dai, "A survey of DNA motif finding algorithms," *BMC Bioinformatics*, vol. 8, no. 7, S21, 2007, ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-S7-S21. [Online]. Available: https://doi.org/10.1186/1471-2105-8-S7-S21.

[118] E. Zaslavsky and M. Singh, "A combinatorial optimization approach for diverse motif finding applications," *Algorithms for Molecular Biology*, vol. 1, no. 1, p. 13, 2006, ISSN: 1748-7188. DOI: 10.1186/1748-7188-1-13. [Online]. Available: https://doi.org/10.1186/1748-7188-1-13.

[119] E. Routhier and J. Mozziconacci, "Genomics enters the deep learning era," *PeerJ*, vol. 10, e13613, 2022, ISSN: 2167-8359. DOI: 10.7717/peerj.13613. [Online]. Available: https://doi.org/10.7717/peerj.13613.

[120] T. Kohonen, "Median strings," *Pattern Recognition Letters*, vol. 3, no. 5, pp. 309–313, 1985, ISSN: 0167-8655. DOI: 10.1016/0167-8655(85)90061-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167865585900613.

[121] S. Ling and C. Xing, *Coding Theory: A First Course*. Cambridge University Press, 2004.

[122] R. F. Barber, E. J. Candes, A. Ramdas, and R. J. Tibshirani, *Predictive inference with the jackknife+*, 2020. arXiv: 1905.02928 [stat.ME]. [Online]. Available: https://arxiv.org/abs/1905.02928.

[123] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006, ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1.

[124] C.-H. Gao, G. Yu, and P. Cai, "ggVennDiagram: An Intuitive, Easy-to-Use, and Highly Customizable R Package to Generate Venn Diagram," *Frontiers in Genetics*, vol. 12, 2021, ISSN: 1664-8021. DOI: 10.3389/fgene.2021.706907. [Online]. Available: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2021.706907.

[125] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister, "UpSet: Visualization of Intersecting Sets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1983–1992, 2014. DOI: 10.1109/TVCG.2014.2346248.

[126] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Curran Associates Inc., 2017, pp. 4768–4777, ISBN: 9781510860964.

[127] S. M. Lundberg, G. Erion, H. Chen, *et al.*, "From local explanations to global understanding with explainable AI for trees," *Nature Machine Intelligence*, vol. 2, no. 1, pp. 56–67, 2020, ISSN: 2522-5839. DOI: 10.1038/s42256-019-0138-9. [Online]. Available: https://doi.org/10.1038/s42256-019-0138-9.

[128] T. L. Bailey, "STREME: accurate and versatile sequence motif discovery," *Bioinformatics*, vol. 37, no. 18, pp. 2834–2840, 2021, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab203. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab203.

Appendices

Appendix A

Technical Details

This appendix details the technical foundations of the research, covering the concept of the self-attention mechanism, a practical example of the algorithm, and the hardware and software environment used for implementation.

A.1 Self-Attention Mechanism

One of the main reasons transformers work so well is their self-attention mechanism [66]. This powerful feature allows the model, when processing any given part of an input sequence, to determine the relative significance of all other parts. Specifically, for every element within the input sequence, the self-attention process calculates how much focus or attention should be allocated to every other element (and the element itself).

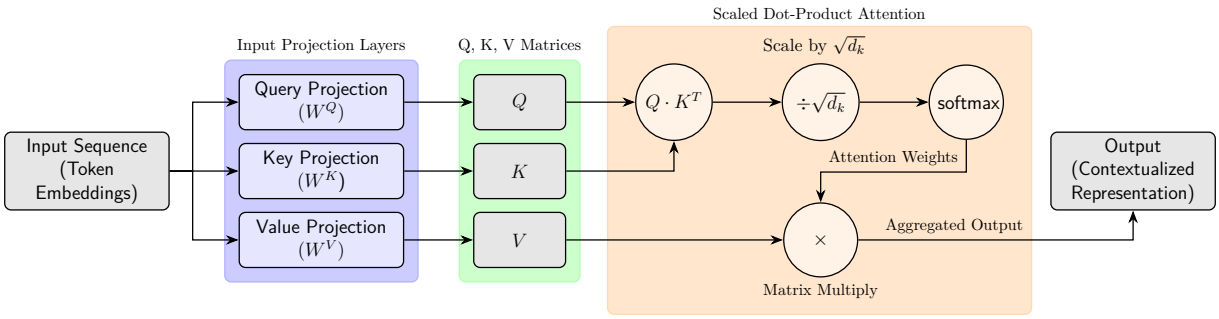
The most common implementation of this is scaled dot-product attention. Figure A.1 depicts the scaled dot-product attention mechanism. Recall Equation 3.8 the calculation is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A.1}$$

Within this formula, Q (Query), K (Key), and V (Value) are matrices derived from linear transformations of input sequence embeddings. Conceptually, Q represents the current po-

sition asking for information, K signifies information other positions possess, and V denotes the actual content of those positions. The calculation starts by computing the dot product QK^T, which measures similarity between queries and keys. These raw similarity scores are then scaled by $\sqrt{d_k}$, where d_k is the dimension of K. This scaling is vital as it prevents dot products from becoming excessively large, thereby stabilizing the softmax function and ensuring more stable gradients during training. Following this, the softmax function normalizes these scaled scores, converting them into probabilities (attention weights) that sum to 1, indicating how much importance should be given to each Value vector. Finally, these attention weights are multiplied by the Value matrix V to compute a weighted sum. This process yields an output representation for each position, now enriched by relevant context aggregated from the entire sequence based on learned attention patterns.

Figure A.1: Scaled dot-product attention: Inputs are projected to Q, K, V. Attention weights, derived from scaled ($\frac{1}{\sqrt{d_k}}$) dot-products and softmax, are applied to V.



While powerful, a single attention calculation might struggle to capture the diverse types of relationships present in complex sequences. To address this, transformers employ multi-head attention. The core idea is to perform the scaled dot-product attention multiple times (h times, where h is the number of "heads") in parallel.

Crucially, each attention head (i) does not use the original Q, K, and V directly. Instead, it first learns its own unique linear projections for them using distinct weight matrices: $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$. Here, d_{model} is the main

embedding dimension of the model, while d_k and d_v are the dimensions for keys and values specific to each head (typically set as $d_k = d_v = \frac{d_{model}}{h}$). This projection step allows each head to focus on different aspects or "representation subspaces" of the input sequence.

The attention calculation for a single head i is then:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{A.2}$$

Note that the Attention function here is the same scaled dot-product mechanism defined in Equation 3.8, but applied to the uniquely projected Q, K, and V for head i, using the head-specific dimension d_k for scaling.

After computing the attention output for all h heads in parallel, their results need to be combined. This is done by concatenating the output vectors from each head:

$$\text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h) \tag{A.3}$$

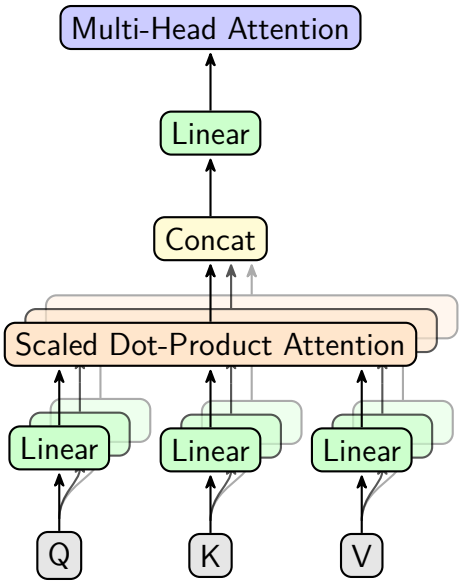
This concatenated result, which now has dimension $h \times d_v$, is then passed through a final linear transformation, parameterized by another learned weight matrix $W^O \in \mathbb{R}^{hd_v \times d_{model}}$, to produce the final output of the multi-head attention layer, projecting it back to the model's primary dimension d_{model}:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O \tag{A.4}$$

Figure A.2 visualizes the multi-head attention mechanism, describing the parallel application of scaled dot-product attention to linear projections of queries, keys, and values, followed by concatenation and a final linear transformation. This multi-head strategy enables the model to jointly attend to information from different learned perspectives (representation subspaces) simultaneously, significantly enhancing its ability to capture complex and varied dependencies within the input DNA sequence compared to using just a single

attention calculation.

Figure A.2: Multi-head attention: Inputs Q, K, and V are first linearly projected h times. Each of these projected versions is then fed into a separate scaled dot-product attention function. The outputs of these h attention heads are concatenated and passed through a final linear layer.



To illustrate this with a real-world analogy, consider how a chef adjusts the seasoning of a complex soup. When deciding how much salt (representing the current element or Query) to add, the chef does not evaluate the salt in isolation. Instead, they consider the entire flavor profile of the soup. They mentally assess the other ingredients present: Each existing ingredient possesses characteristics that act as Keys (searchable aspects relevant to the Query) and contributes its specific flavor profile as its Value. For instance, consider the sweetness of the carrots (representing k_1 offering sweetness as v_1), the acidity of the tomatoes (k_2 offering acidity as v_2), the richness of the broth (k_3 offering richness as v_3), and so forth. The chef intuitively calculates how strongly the need for saltiness relates to each of these existing flavor components, akin to computing the QK^T scores and applying scaling. Some components, like sweetness, might strongly signal the need for salt to create balance, thus having a high compatibility score, while others might be less relevant. Based on this assessment, the chef assigns relative importance, analogous to the attention weights

derived from the softmax function, to each existing flavor component. Finally, the decision on the precise amount of salt to add (the resulting output representation for the "salt" element) is a weighted consideration of all these other flavors (v's), guided by their calculated relevance (attention weights) to the overall goal of achieving a balanced taste profile. Extending this analogy, multi-head attention would be akin to having several assistant chefs simultaneously tasting and focusing on different specific flavor interactions (e.g., one focuses on the sweet-savory balance, another on the sour-savory balance). Their individual assessments are then combined to inform a final, more nuanced and comprehensive seasoning adjustment. Similarly, the self-attention mechanism enables the transformer model to weigh the influence of different parts of an input sequence (such as DNA bases or segments) based on their learned relationships and relevance, thereby creating a contextually enriched understanding of each part within the sequence.

A.2 Algorithm Examples

This section provides supplementary, step-by-step examples to illustrate the core methodologies discussed in the main text.

Median String Problem

Given t DNA sequences $D = \{D_1, D_2, \ldots, D_t\}$, each of length n, and a motif length l, the median string problem is defined as follows:

> Find a string v of length l that minimizes the sum of the minimum Hamming distances between v and any substring of length l in each D_i.

This can be written as:

$$\text{Score}(v) = \sum_{i=1}^{t} \min_{1 \le j \le n-l+1} d_H\Big(v, D_i[j : j+l-1]\Big) \tag{A.5}$$

Here $d_H(x, y)$ is the Hamming distance between strings x and y, and $D_i[j : j + l - 1]$ denotes the substring of D_i starting at position j and of length l.

Consider $t = 2$ DNA sequences $D = \{D_1, D_2\}$, each of length $n = 5$, where:

$$D_1 = \text{ACGTG} \qquad\qquad D_2 = \text{GTCAG}$$

Let the motif length $l = 2$. The goal is to find a string v of length l that minimizes Equation A.5. To illustrate this process, consider one possible candidate string $v = \text{AC}$ and compute its score $\text{Score}(v)$ using Equation A.5. Note that this computation does not guarantee that AC is the optimal median string; it simply evaluates $\text{Score}(v)$ for this choice of v.

For $D_1 = \text{ACGTG}$, the substrings of length $l = 2$ are:

$$D_1[1:2] = \text{AC} \qquad\qquad D_1[2:3] = \text{CG}$$
$$D_1[3:4] = \text{GT} \qquad\qquad D_1[4:5] = \text{TG}$$

The Hamming distances are:

$$d_H(\text{AC}, \text{AC}) = 0 \qquad\qquad d_H(\text{AC}, \text{CG}) = 2$$
$$d_H(\text{AC}, \text{GT}) = 2 \qquad\qquad d_H(\text{AC}, \text{TG}) = 2$$

The minimum Hamming distance is:

$$\min_{1 \le j \le n-l+1} d_H(\text{AC}, D_1[j : j + l - 1]) = 0 \tag{A.6}$$

For $D_2 = \text{GTCAG}$, the substrings of length $l = 2$ are:

$$D_2[1:2] = \text{GT} \qquad D_2[2:3] = \text{TC}$$
$$D_2[3:4] = \text{CA} \qquad D_2[4:5] = \text{AG}$$

$$d_H(\text{AC}, \text{GT}) = 2 \qquad d_H(\text{AC}, \text{TC}) = 1$$
$$d_H(\text{AC}, \text{CA}) = 2 \qquad d_H(\text{AC}, \text{AG}) = 1$$

The minimum Hamming distance is:

$$\min_{1 \le j \le n-l+1} d_H(\text{AC}, D_2[j : j+l-1]) = 1 \tag{A.7}$$

From (A.6) and (A.7), the total score for $v = \text{AC}$ is:

$$\text{Score}(\text{AC}) = 0 + 1 = 1$$

This calculation shows that the candidate string AC achieves a score of 1, but it does not guarantee that AC is the optimal median string. To find the true median string, all possible strings of length $l = 2$ would need to be evaluated. If any candidate string achieves a score of $\text{Score}(v) = 0$, then that string would definitively be the optimal median string.

Attention Score

Consider a DNA sequence $S = \texttt{ACGTC}$ (length $L = 5$). The objective is to derive a nucleotide-level attention score vector for this sequence using a pre-trained DNABERT model configured with $k = 3$ for 3-mer tokenization.

First, the sequence is tokenized into overlapping 3-mers, including special tokens:

$$[\texttt{[CLS]}, \texttt{[ACG]}, \texttt{[CGT]}, \texttt{[GTC]}, \texttt{[SEP]}]$$

This results in a token sequence length of 5. For this example, to simplify, assume that the DNABERT model utilizes 2 attention heads ($\texttt{num\_heads} = 2$). During the evaluation phase, after feeding the tokenized sequence into the model (batch size = 1), attention scores are extracted from the final layer. These scores initially possess the dimensions $(1, 2, 5, 5)$, representing ($\texttt{batch\_size}, \texttt{num\_heads}, \texttt{sequence\_length}, \texttt{sequence\_length}$).

Hypothetical raw attention scores summed across all source tokens "received" by each 3-mer token from the two heads are assumed as follows:

- For Head 1, the summed attention scores received are: Score(`ACG`) = 0.8, Score(`CGT`) = 0.9, Score(`GTC`) = 1.2.
- For Head 2, the summed attention scores received are: Score(`ACG`) = 0.6, Score(`CGT`) = 1.0, Score(`GTC`) = 0.7.

Next, preprocessing involves summing the attention scores across the heads for each 3-mer token position. Interactions involving only special tokens `[CLS]` and `[SEP]` are excluded from the direct aggregation mapping, although their influence is implicitly captured within the calculated token scores. The summed attention for each token is calculated:

$$\text{Summed Attention}(\texttt{ACG}) = 0.8 + 0.6 = 1.4$$
$$\text{Summed Attention}(\texttt{CGT}) = 0.9 + 1.0 = 1.9$$
$$\text{Summed Attention}(\texttt{GTC}) = 1.2 + 0.7 = 1.9$$

This yields a token-level attention vector: $[1.4, 1.9, 1.9]$. The subsequent step aggregates these token-level scores back to the original nucleotide sequence length. The original sequence length L is calculated using the formula $L = \text{tokens} - 2 + k - 1$. With 5 tokens and

$k = 3$, $L = 5 - 2 + 3 - 1 = 5$. Recall the sequence is $\texttt{A}_1\texttt{C}_2\texttt{G}_3\texttt{T}_4\texttt{C}_5$.

The aggregation proceeds using a running sum and count for each nucleotide position based on the overlapping 3-mers. The tokens cover the nucleotides as follows:

- Token 1 (`ACG`, score 1.4) covers nucleotides 1, 2, 3.
- Token 2 (`CGT`, score 1.9) covers nucleotides 2, 3, 4.
- Token 3 (`GTC`, score 1.9) covers nucleotides 3, 4, 5.

An aggregation vector `AggScores` of length 5 and a count vector `Counts` of length 5 are initialized to zeros. The scores are distributed step-by-step:

1. Initialize:

$$\texttt{AggScores} = [0.0, 0.0, 0.0, 0.0, 0.0], \quad \texttt{Counts} = [0, 0, 0, 0, 0]$$

2. After distributing Token 1 score (1.4) covering nucleotides 1, 2, 3:

$$\texttt{AggScores}_{\text{step 1}} = [1.4, 1.4, 1.4, 0.0, 0.0]$$
$$\texttt{Counts}_{\text{step 1}} = [1, 1, 1, 0, 0]$$

3. After distributing Token 2 score (1.9) covering nucleotides 2, 3, 4:

$$\texttt{AggScores}_{\text{step 2}} = [1.4, 1.4 + 1.9, 1.4 + 1.9, 1.9, 0.0] = [1.4, 3.3, 3.3, 1.9, 0.0]$$
$$\texttt{Counts}_{\text{step 2}} = [1, 1 + 1, 1 + 1, 1, 0] = [1, 2, 2, 1, 0]$$

4. After distributing Token 3 score (1.9) covering nucleotides 3, 4, 5:

$$\texttt{AggScores}_{\text{step 3}} = [1.4, 3.3, 3.3 + 1.9, 1.9 + 1.9, 1.9] = [1.4, 3.3, 5.2, 3.8, 1.9]$$
$$\texttt{Counts}_{\text{step 3}} = [1, 2, 2 + 1, 1 + 1, 1] = [1, 2, 3, 2, 1]$$

5. The aggregated scores are then averaged by dividing `AggScores` by `Counts`:

$$\begin{aligned}\texttt{AvgScores} &= \left[\frac{1.4}{1}, \frac{3.3}{2}, \frac{5.2}{3}, \frac{3.8}{2}, \frac{1.9}{1}\right] \\ &\approx [1.40, 1.65, 1.73, 1.90, 1.90]\end{aligned}$$

6. Finally, normalization is applied to ensure the vector has an L2 Euclidean unit norm. The L2 norm is calculated as:

$$\begin{aligned}\text{L2} &= \sqrt{\sum_{i=1}^{L}(\texttt{AvgScores}_i)^2} \\ &= \sqrt{1.40^2 + 1.65^2 + 1.73^2 + 1.90^2 + 1.90^2} \\ &= \sqrt{1.96 + 2.7225 + 2.9929 + 3.61 + 3.61} = \sqrt{14.8954} \approx 3.86\end{aligned}$$

7. The averaged scores are divided by the L2 norm:

$$\begin{aligned}\texttt{NormScores} &= \frac{\texttt{AvgScores}}{\text{L2}} = \left[\frac{1.40}{3.86}, \frac{1.65}{3.86}, \frac{1.73}{3.86}, \frac{1.90}{3.86}, \frac{1.90}{3.86}\right] \\ &\approx [0.36, 0.43, 0.45, 0.49, 0.49]\end{aligned}$$

The final output for the sequence `ACGTC` is the normalized attention vector:

$$v = [0.36, 0.43, 0.45, 0.49, 0.49]$$

This vector v, shaped $(1, 5)$ to match $(\texttt{num\_sequences}, \texttt{original\_sequence\_length})$, contains the normalized attention scores for every nucleotide position. These scores highlight potentially crucial regions (demonstrated by positions 4 and 5), making the vector useful for subsequent analyses like motif discovery.

A.3 Hardware and Software Specifications

This section details the hardware and software specifications used for the experiments conducted in this research. These specifications are provided to enable the reproduction of the study's results.

The hardware specifications of the servers are summarized in Table A.1.

Table A.1: Hardware specifications.

Component	**Specifications**
GPU	4x NVIDIA RTX A5000, each with 24GB GDDR6 memory
	CUDA version 12.3, driver version 545.23.08
CPU	2x Intel® Xeon® Gold 6354 @ 3.00GHz
Memory	1055926060 kB ≈ 1055 GB ≈ 1.04 TB
Storage	7.3 TB of high-speed storage
OS	Rocky Linux 8.9 (Green Obsidian)

The servers are equipped with powerful GPUs and CPUs, along with ample memory and storage, to support the computational demands of deep learning models on genomic data.

The software environment used for this work was set up to train and evaluate deep learning models on genomic data. Table A.2 details the primary software components and their versions.

The software stack includes various libraries for data manipulation, machine learning, and deep learning. Specifically, `torch` and `transformers` were crucial for implementing and training the deep learning models. `accelerate` was used for distributed training across multiple GPUs. Libraries such as `pandas`, `numpy`, and `scikit-learn` were used for data

Table A.2: Software dependencies and their versions for distributed deep learning on genomic data.

Name	Version	Name	Version	Name	Version
Python	3.9.13	scikit-learn	1.3.2	torch	2.3.0
accelerate	0.30.0	scipy	1.10.1	transformers	4.29.2
biopython	1.85	seaborn	0.13.2	xgboost	2.1.4
catboost	1.2.7	statsmodels	0.14.4	lightgbm	4.6.0
datasets	2.19.1	numpy	1.26.4	peft	0.10.0
joblib	1.4.2	pandas	2.0.3	umap-learn	0.5.7
safetensors	0.4.3	triton	2.3.0	venny4py	1.0.3

preprocessing and analysis. Genomic data processing was facilitated by `biopython`. Visualization was aided by `seaborn` and `umap-learn`. `catboost`, `xgboost`, and `lightgbm` were used for gradient boosting models.

Appendix B

Performance and Results

This appendix presents the quantitative and qualitative results of the study, detailing the model evaluation metrics and providing supplementary plots for visual analysis.

Simplified Representation of Evaluation Metrics

The traditional definitions and mathematical formulations of these metrics, often found in textbooks, can be challenging for many biological scientists to grasp. Therefore, Table B.1 provides an overview of these classification metrics, tailored to help biological scientists understand their application, such as in identifying enhancers. Complementing this overview, the four metrics used to evaluate classification performance can also be presented

mathematically in a more intuitive and understandable manner, as follows:

$$\begin{cases} \mathrm{SN} = 1 - \dfrac{N_-^+}{N^+}, \quad 0 \le \mathrm{SN} \le 1 \\ \mathrm{SP} = 1 - \dfrac{N_+^-}{N^-}, \quad 0 \le \mathrm{SP} \le 1 \\ \mathrm{ACC} = 1 - \dfrac{N_-^+ + N_+^-}{N^+ + N^-}, \quad 0 \le \mathrm{ACC} \le 1 \\ \mathrm{MCC} = \dfrac{1 - \left(\dfrac{N_-^+}{N^+} + \dfrac{N_+^-}{N^-}\right)}{\sqrt{\left(1 + \dfrac{N_+^- - N_-^+}{N^+}\right)\left(1 + \dfrac{N_-^+ - N_+^-}{N^-}\right)}}, \quad -1 \le \mathrm{MCC} \le 1 \end{cases}$$

N^+ denotes the total count of positive samples analyzed, whereas N_-^+ refers to the count of positive samples that were mistakenly predicted as negative. Similarly, N^- indicates the total count of negative samples analyzed, while N_+^- refers to the count of negative samples that were incorrectly predicted as positive.

B.1 Benchmarking Study Performance

Table B.2 provides a comprehensive comparison of the performance metrics of various models on the independent test set for Layer I. The results clearly demonstrate the superior performance of StackBERT-Enhancer across multiple key metrics. Notably, it achieves the highest ACC of 0.8350 and SP of 0.8950, indicating its exceptional ability to correctly classify both enhancer and non-enhancer sequences. Furthermore, the model exhibits a high MCC of 0.6749 and an AUC of 0.8652, signifying robust overall performance and excellent discriminatory power between the two classes.

While the SN of 0.7750 for the proposed model is slightly lower than some other models, this minor trade-off is effectively compensated by its outstanding SP of 0.8950 and overall ACC. This balance suggests that StackBERT-Enhancer excels at minimizing false positives, a cru-

Table B.1: Summary of classification metrics for enhancer identification.

Metric	**Definition**	**Example**
ACC	The overall proportion of genomic regions (both enhancers and non-enhancers) that were correctly classified by the model.	If a model tested on 1000 genomic regions correctly identifies 950 of them (regardless of whether they are enhancers or non-enhancers), the accuracy is 95%.
SN	The proportion of actual enhancer regions that were correctly identified as enhancers by the model. It measures how well the model finds the positive class.	If there are 200 true enhancer regions in the test set and the model correctly predicts 160 of them as enhancers, the sensitivity is $\frac{160}{200} = 80\%$.
SP	The proportion of actual non-enhancer regions that were correctly identified as non-enhancers by the model. It measures how well the model avoids misclassifying negatives.	If there are 800 true non-enhancer regions in the test set and the model correctly identifies 720 of them as non-enhancers, the specificity is $\frac{720}{800} = 90\%$.
MCC	A correlation coefficient between the observed and predicted classifications, providing a balanced measure even when the classes (enhancers vs. non-enhancers) are of very different sizes. Ranges from -1 (total disagreement) to $+1$ (perfect prediction), with 0 indicating random performance.	An MCC of $+0.7$ indicates a strong agreement between the predicted enhancer/non-enhancer labels and the true labels, accounting for both correct enhancer predictions and correct non-enhancer predictions, useful when non-enhancer regions vastly outnumber enhancer regions.
AUC	Measures the ability of the model to distinguish between enhancer and non-enhancer regions across all possible classification thresholds. It represents the probability that the model ranks a randomly chosen true enhancer higher than a randomly chosen true non-enhancer.	An AUC of 0.85 means there is an 85% chance that the model will assign a higher prediction score (indicating higher likelihood of being an enhancer) to a randomly selected true enhancer region compared to a randomly selected non-enhancer region.

Table B.2: Performance comparison of identifier models on the independent test set.

Model	ACC	SN	SP	MCC	AUC
iEnhancer-2L	0.7300	0.7100	0.7500	0.4600	0.8062
EnhancerPred	0.7400	0.7350	0.7450	0.4800	0.8013
iEnhancer-EL	0.7475	0.7100	0.7850	0.4960	0.8173
iEnhancer-5Step	0.7900	0.8200	0.7600	0.5800	-
iEnhancer-MRBF	0.7975	0.8200	0.7750	0.5956	-
iEnhancer-MFGBDT	0.7750	0.7679	0.7955	0.5607	-
iEnhancer-XG	0.7575	0.7400	0.7750	0.5150	-
EnhancerP-2L	0.7950	0.7810	0.8105	0.5907	-
iEnhancer-RF	0.7975	0.7850	0.8100	0.5952	0.8600
piEnPred	0.8040	0.8250	0.7840	0.6099	-
iEnhancer-ECNN	0.7690	0.7850	0.7520	0.5370	0.8320
iEnhancer-Deep	0.7402	0.8150	0.6700	0.4902	-
iEnhancer-EBLSTM	0.7720	0.7550	0.7950	0.5340	0.8350
spEnhancer	0.7725	0.8300	0.7150	0.5793	0.8235
iEnhancer-DCLA	0.7825	0.7800	0.7850	0.5650	-
iEnhancer-DCSA	0.8250	0.7950	0.8550	0.6510	0.8558
Enhancer-LSTMAtt	0.8050	0.7950	0.8150	0.6101	0.8588
Rank-GAN	0.7525	0.7487	0.7563	0.5051	0.8062
Enhancer-RD	0.7880	0.8100	0.7650	0.5760	0.8440
iEnhancer-CNN	0.7750	0.7825	0.7900	0.5850	-
DeployEnhancer	0.7550	0.7550	0.7600	0.5100	0.7704
Enhancer-DSNet	0.7800	0.7800	0.7700	0.5600	-
iEnhancer-DHF	0.8321	0.8211	0.8442	0.7110	-
Enhancer-DRRNN	0.7670	0.7330	0.8010	0.5350	0.8370
Enhancer-BERT	0.7560	0.8000	0.7120	0.5140	-
iEnhancer-BERT	0.7930	-	-	0.5850	0.8440
StackBERT-Enhancer	**0.8350**	**0.7750**	**0.8950**	**0.6749**	**0.8652**

Table B.3: Performance comparison of classifier models on the independent test set.

Model	ACC	SN	SP	MCC	AUC
iEnhancer-2L	0.6050	0.4700	0.7400	0.2181	0.6678
EnhancerPred	0.5500	0.4500	0.6500	0.1020	0.5790
iEnhancer-EL	0.6100	0.5400	0.6800	0.2222	0.6801
iEnhancer-5Step	0.6350	0.7400	0.5300	0.2800	-
iEnhancer-MRBF	0.8350	1.0000	0.6700	0.7098	-
iEnhancer-MFGBDT	0.6850	0.7255	0.6681	0.3862	-
iEnhancer-XG	0.6350	0.7000	0.5700	0.2720	-
EnhancerP-2L	0.7250	0.6829	0.7922	0.4624	-
iEnhancer-RF	0.8500	0.9300	0.7700	0.7091	0.9700
piEnPred	0.7250	0.7000	0.7500	0.4506	-
iEnhancer-ECNN	0.6780	0.7910	0.5640	0.3680	0.7480
iEnhancer-Deep	0.6100	0.7300	0.4900	0.2266	-
iEnhancer-EBLSTM	0.6580	0.8120	0.5360	0.3240	0.6880
spEnhancer	0.6200	0.9100	0.3300	0.3703	0.6253
iEnhancer-DCLA	0.7800	0.8700	0.6900	0.5693	-
iEnhancer-DCSA	0.9150	0.9800	0.8500	0.8370	0.9660
Enhancer-LSTMAtt	0.8950	0.9900	0.8000	0.8047	0.9637
Rank-GAN	0.6970	0.7068	0.6889	0.3954	0.7702
Enhancer-RD	0.7050	0.8400	0.5700	0.4260	0.7920
iEnhancer-CNN	0.7500	0.6525	0.7610	0.3232	-
DeployEnhancer	0.6849	0.8315	0.4561	0.3120	0.6714
Enhancer-DSNet	0.8300	0.8300	0.6700	0.7000	-
iEnhancer-DHF	0.6754	0.6597	0.6876	0.3120	-
Enhancer-DRRNN	0.8490	0.8580	0.8400	0.6990	-
ES-ARCNN	0.6550	0.8600	0.4500	0.3399	-
iEnhancer-BERT	0.7010	-	-	0.4010	0.8120
StackBERT-Enhancer	**0.9900**	**1.0000**	**0.9800**	**0.9802**	**0.9900**

Table B.4: Performance comparison of models using 5-fold cross-validation.

Model	ACC	SN	SP	MCC	AUC
			Layer I		
iEnhancer-2L	0.7689	0.7809	0.7588	0.5400	-
EnhancerP-2L	0.9168	0.9077	0.9259	0.8340	0.9400
iEnhancer-PsedeKNC	0.7678	0.7731	0.7630	0.5400	0.8500
iEnhancer-5Step	0.8230	0.8110	0.8350	0.6500	-
iEnhancer-KL	0.8423	0.8322	0.8524	0.6800	-
iEnhancer-RF	0.7618	0.7364	0.7871	0.5264	0.8400
iEnhancer-MRBF	0.8123	0.7952	0.8295	0.6254	0.8809
piEnPred	0.8788	0.9228	0.8047	0.7660	0.9603
iEnhancer-RD	0.7888	0.8100	0.7656	0.5761	0.8444
DeployEnhancer	0.7483	0.7325	0.7642	0.4980	0.7694
iEnhancer-CNN	0.8063	0.7588	0.8888	0.6929	0.8957
Enhancer-LSTMAtt	0.7655	0.7304	0.8006	0.5339	0.8259
Enhancer-BERT	0.7620	0.7950	0.7300	0.5250	-
StackBERT-Enhancer	**0.9175**	**0.8848**	**0.9504**	**0.8368**	**0.9383**
			Layer II		
iEnhancer-2L	0.6193	0.6221	0.6182	0.2400	-
EnhancerP-2L	0.6193	0.6221	0.6182	0.2400	0.9000
iEnhancer-PsedeKNC	0.6341	0.6262	0.6441	0.2700	0.6900
iEnhancer-5Step	0.6810	0.7530	0.6080	0.3700	-
iEnhancer-KL	0.9313	0.9340	0.9287	0.8600	-
iEnhancer-RF	0.6253	0.6846	0.5661	0.2529	0.6700
iEnhancer-MRBF	0.7695	0.7723	0.7969	0.5419	0.8409
piEnPred	0.6824	0.6554	0.7094	0.3654	0.7568
iEnhancer-RD	0.7050	0.8400	0.5700	0.4260	0.7920
DeployEnhancer	0.5896	0.7965	0.3828	0.1970	0.6068
iEnhancer-CNN	0.7643	0.7364	0.7680	0.4505	0.8109
Enhancer-LSTMAtt	0.6395	0.6765	0.6024	0.2804	0.6439
StackBERT-Enhancer	**0.9373**	**0.9143**	**0.9599**	**0.8758**	**0.9734**

cial aspect in biological applications where incorrect enhancer predictions can lead to costly downstream experiments. Although models like iEnhancer-DHF and iEnhancer-DCSA also demonstrate strong performance in enhancer identification, StackBERT-Enhancer consistently matches or exceeds their performance in many evaluated metrics.

Similarly, Table B.3 presents a detailed analysis of the performance metrics for different models on the independent test set for Layer II. The data unequivocally highlights the exceptional capabilities of StackBERT-Enhancer, which achieves remarkable results across all evaluated metrics. With an ACC of 0.9900, a perfect SN of 1.0000, a high SP of 0.9800, an MCC of 0.9802, and an AUC of 0.9900, StackBERT-Enhancer significantly outperforms all other listed models in accurately classifying enhancer strength. The perfect SN indicates that the model correctly identifies all instances of a particular strength category, while the high SP ensures minimal misclassification into that category.

Overall, based on the compelling evidence presented in Tables B.2 and B.3, it is evident that the proposed StackBERT-Enhancer model represents the best-performing solution for both Layer I and Layer II. Its consistently superior performance across critical metrics underscores its robustness and reliability in these tasks. While certain competing models may exhibit strengths in specific aspects, none achieve the same level of comprehensive excellence demonstrated by the proposed model across both layers of the prediction task. These findings firmly establish StackBERT-Enhancer as a state-of-the-art solution for enhancer identification and strength classification.

To further validate the robustness and generalizability of the proposed model, Tables B.4 and B.5 provide a comparative analysis of various classifier models for enhancer prediction across both layers using rigorous 5-fold and 10-fold cross-validation techniques. The consistent trend observed across both validation strategies reinforces the superiority of StackBERT-Enhancer. Overall, StackBERT-Enhancer consistently outperforms other models across both Layer I and Layer II, irrespective of the cross-validation method em-

ployed. This consistent outperformance across different validation schemes strengthens the confidence in the model's ability to generalize to unseen data and its potential for real-world applications in regulatory genomics. Jackknife testing was not employed due to its considerable computational intensity, especially for deep learning methods.

Table B.5: Performance comparison of models using 10-fold cross-validation.

Model	ACC	SN	SP	MCC	AUC
			Layer I		
EnhancerP-2L	0.9172	0.8653	0.9690	0.8398	0.9700
iEnhancer-XG	0.8110	0.7570	0.8650	0.6265	-
iEnhancer-GAN	0.9510	0.9510	0.9510	0.9020	-
iEnhancer-MFGBDT	0.7867	0.7754	0.7978	0.5735	-
Enhancer-LSTMAtt	0.7658	0.7414	0.7873	0.5298	0.8256
StackBERT-Enhancer	**0.9178**	**0.8846**	**0.9521**	**0.8378**	**0.9392**
			Layer II		
ES-ARCNN	0.6617	0.7278	0.5957	0.3263	0.6604
EnhancerP-2L	0.8723	0.8049	0.9397	0.7519	0.9300
iEnhancer-XG	0.6674	0.7494	0.5855	0.3395	-
iEnhancer-GAN	0.8720	0.8730	0.8710	0.7440	-
iEnhancer-MFGBDT	0.6604	0.7056	0.6163	0.3232	-
Enhancer-LSTMAtt	0.6429	0.6463	0.6380	0.2851	0.6550
StackBERT-Enhancer	**0.9387**	**0.9171**	**0.9594**	**0.8787**	**0.9719**

B.2 Base Models Performance

Cross Validation

Cross-validation is a crucial statistical method used in machine learning to assess the performance and generalizability of predictive models. It helps in evaluating how well a model will perform on unseen data, thereby providing a more robust estimate of its predictive accuracy. The primary goal of cross-validation is to prevent overfitting and to ensure that the model's performance is consistent across different subsets of the data.

In k-fold cross-validation, the dataset is divided into k equally sized folds. For each fold i, the model is trained on $k-1$ folds and tested on the remaining fold. The overall performance metric M can be expressed as:

$$M = \frac{1}{k} \sum_{i=1}^{k} M_i \tag{B.1}$$

Here, M_i represents the performance metric, such as accuracy, precision, or recall, for fold i, and k denotes the total number of folds.

5-fold

5-fold cross-validation is a specific type of k-fold cross-validation where $k = 5$. In this method, the dataset is divided into five equal (or nearly equal) subsets or "folds". The process involves training the model on four of the five folds (80% of the data) and then validating it on the remaining fold (20% of the data). This process is repeated five times, with each fold serving as the validation set exactly once. The performance metrics are then averaged across all five iterations to provide a more reliable estimate of the model's performance.

Table B.6 and Table B.7 show the results of 5-fold cross-validation for different k-mer models on both the training and validation sets. Based on these results, it can be observed

Table B.6: Results of average 5-fold on the training set.

Layer	Model	ACC	SN	SP	MCC	AUC
I	3-mer	0.8999	0.8477	0.9522	0.8043	0.9015
	4-mer	0.9067	0.8504	0.9629	0.8185	0.9174
	5-mer	0.8848	0.8228	0.9468	0.7755	0.9083
	6-mer	0.8757	0.7978	0.9535	0.7606	0.8963
II	3-mer	0.8888	0.8302	0.9474	0.7830	0.9163
	4-mer	0.9111	0.8639	0.9582	0.8258	0.9218
	5-mer	0.8982	0.8330	0.9636	0.8034	0.8944
	6-mer	0.9043	0.8424	0.9663	0.8149	0.8987

Table B.7: Results of average 5-fold on the validation set.

Layer	Model	ACC	SN	SP	MCC	AUC
I	3-mer	0.8999	0.8475	0.9524	0.8045	0.9018
	4-mer	0.9067	0.8505	0.9631	0.8186	0.9173
	5-mer	0.8848	0.8229	0.9470	0.7756	0.9088
	6-mer	0.8757	0.7976	0.9537	0.7607	0.8965
II	3-mer	0.8888	0.8310	0.9466	0.7828	0.9174
	4-mer	0.9110	0.8648	0.9572	0.8252	0.9217
	5-mer	0.8982	0.8347	0.9627	0.8036	0.8952
	6-mer	0.9043	0.8440	0.9655	0.8148	0.8980

that the 4-mer model consistently performs the best across all metrics for both training and validation sets. There is minimal overfitting, as the performance on the training and validation sets is very similar for all models. Additionally, the 3-mer and 4-mer models show higher performance compared to the 5-mer and 6-mer models, suggesting that increasing the k-mer size beyond 4 may not necessarily improve the model's predictive power in this particular case.

10-fold

10-fold cross-validation is a variation of k-fold cross-validation where $k = 10$. In this method, the dataset is divided into ten equal (or nearly equal) subsets. The model is trained on nine folds (90% of the data) and validated on the remaining fold (10% of the data). This process is repeated ten times, with each fold serving as the validation set exactly once. The performance metrics are then averaged across all ten iterations, providing a more comprehensive assessment of the model's performance compared to 5-fold cross-validation.

Table B.8: Results of average 10-fold on the training set.

Layer	Model	ACC	SN	SP	MCC	AUC
I	3-mer	0.8999	0.8477	0.9522	0.8043	0.9015
	4-mer	0.9067	0.8504	0.9629	0.8185	0.9174
	5-mer	0.8848	0.8228	0.9468	0.7755	0.9083
	6-mer	0.8757	0.7978	0.9535	0.7606	0.8963
II	3-mer	0.8888	0.8302	0.9474	0.7830	0.9162
	4-mer	0.9111	0.8639	0.9582	0.8258	0.9218
	5-mer	0.8982	0.8329	0.9636	0.8034	0.8944
	6-mer	0.9043	0.8423	0.9663	0.8149	0.8987

The results of the 10-fold cross-validation for different k-mer models on both the training

Table B.9: Results of average 10-fold on the validation set.

Layer	Model	ACC	SN	SP	MCC	AUC
I	3-mer	0.8999	0.8479	0.9523	0.8045	0.9024
	4-mer	0.9067	0.8510	0.9633	0.8188	0.9174
	5-mer	0.8848	0.8237	0.9472	0.7762	0.9094
	6-mer	0.8757	0.7982	0.9538	0.7610	0.8967
II	3-mer	0.8889	0.8311	0.9462	0.7824	0.9168
	4-mer	0.9110	0.8643	0.9570	0.8248	0.9218
	5-mer	0.8983	0.8338	0.9621	0.8037	0.8938
	6-mer	0.9043	0.8439	0.9659	0.8148	0.8976

and validation sets are summarized in Table B.8 and Table B.9. The results obtained are somewhat similar to those obtained with 5-fold.

Identifier Layer Performance

The performance of Layer I was evaluated using various models with different k-mer sizes (3-mer, 4-mer, 5-mer, and 6-mer). The evaluation was conducted on both training and test datasets, with metrics including ACC, SN, SP, MCC, and AUC. The results are presented in Table B.10 and Table B.11.

Layer I Training Set Performance

Table B.10 provides the results of the models on the training set. The 4-mer model demonstrated the highest performance, achieving an ACC of 0.9067, an SN of 0.8504, an SP of 0.9629, an MCC of 0.8185, and an AUC of 0.9174. These results indicate that the 4-mer model is particularly effective at distinguishing between classes in the training set.

The 3-mer model also performed well, with an ACC of 0.8999 and an MCC of 0.8043,

Table B.10: Results of identifier enhancer models on the training set.

Model	ACC	SN	SP	MCC	AUC
3-mer	0.8999	0.8477	0.9522	0.8043	0.9015
4-mer	0.9067	0.8504	0.9629	0.8185	0.9174
5-mer	0.8848	0.8228	0.9468	0.7755	0.9083
6-mer	0.8757	0.7978	0.9535	0.7606	0.8962

although its AUC (0.9015) was slightly lower than that of the 4-mer model. The 5-mer and 6-mer models exhibited slightly lower performance compared to the top-performing models, with ACC values of 0.8848 and 0.8757, respectively. Both models maintained high SP, but their SN and MCC values were reduced in comparison to the 4-mer model.

Layer I Test Set Performance

Table B.11: Results of identifier enhancer models on the independent test set.

Model	ACC	SN	SP	MCC	AUC
3-mer	0.7975	0.7600	0.8350	0.5967	0.8324
4-mer	0.8000	0.7100	0.8900	0.6100	0.8412
5-mer	0.7950	0.7550	0.8350	0.5919	0.8111
6-mer	0.7900	0.7200	0.8600	0.5858	0.8300

The generalization capability of the models was assessed on the test set, as shown in Table B.11. Consistent with the training set results, the 4-mer model achieved strong performance on the test set, with an ACC of 0.8000, an SN of 0.7100, an SP of 0.8900, an MCC of 0.6100, and an AUC of 0.8412. This indicates that the 4-mer model maintains a good balance between SN and SP while demonstrating robust predictive ability.

The other models showed comparable but slightly lower performance on the test set. For

example, the 3-mer model achieved an ACC of 0.7975 and an AUC of 0.8324, while the ACC values for the 5-mer and 6-mer models were recorded at 0.7950 and 0.7900, respectively. A slight decrease in performance was observed for all models when transitioning from the training set to the test set.

Classifier Layer Performance

The performance of Layer II was also assessed using models with varying k-mer sizes. The evaluation was conducted on both training and test datasets. The results are summarized in Tables B.12 and B.13.

Layer II Training Set Performance

Table B.12: Results of classifier enhancer models on the training set.

Model	ACC	SN	SP	MCC	AUC
3-mer	0.8888	0.8302	0.9474	0.7830	0.9162
4-mer	0.9111	0.8639	0.9582	0.8258	0.9218
5-mer	0.8982	0.8329	0.9636	0.8034	0.8944
6-mer	0.9043	0.8423	0.9663	0.8149	0.8987

Table B.12 presents the results for the training set. The 4-mer model achieved the highest performance, with an ACC of 0.9111, an SN of 0.8639, an SP of 0.9582, an MCC of 0.8258, and an AUC of 0.9218. This indicates that the 4-mer model is particularly effective in capturing the features necessary for classification tasks on the training data.

The 3-mer model also performed well, with an ACC of 0.8888, an SN of 0.8302, and an SP of 0.9474, alongside an AUC of 0.9162. However, its MCC value of 0.7830 was lower than that of the 4-mer model, suggesting slightly reduced overall predictive power. The 5-mer and 6-mer models showed comparable performance to each other, with ACC values

of 0.8982 and 0.9043, respectively. Both models maintained high SP values (0.9636 and 0.9663), but their SN and MCC values (5-mer: SN = 0.8329, MCC = 0.8034; 6-mer: SN = 0.8423, MCC = 0.8149) were lower than those of the top-performing 4-mer model.

Layer II Test Set Performance

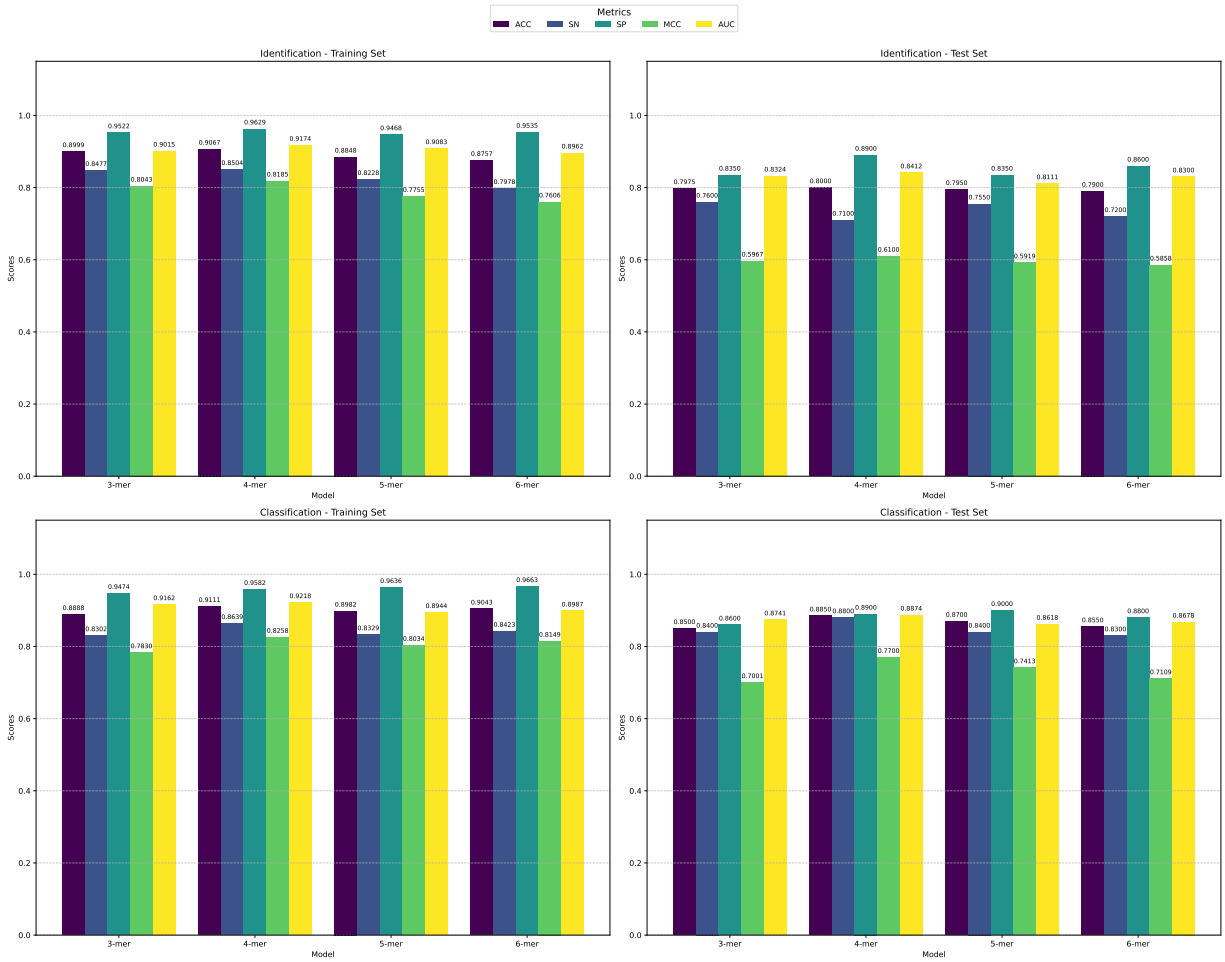
Table B.13: Results of classifier enhancer models on the independent test set.

Model	ACC	SN	SP	MCC	AUC
3-mer	0.8500	0.8400	0.8600	0.7001	0.8741
4-mer	0.8850	0.8800	0.8900	0.7700	0.8874
5-mer	0.8700	0.8400	0.9000	0.7413	0.8618
6-mer	0.8550	0.8300	0.8800	0.7109	0.8678

The generalization capability of the models was evaluated on the test set, as shown in Table B.13. The 4-mer model demonstrated the strongest performance, achieving an ACC of 0.8850, an SN of 0.8800, an SP of 0.8900, an MCC of 0.7700, and an AUC of 0.8874. These results indicate that the 4-mer model effectively balances SN and SP while maintaining robust predictive ability on unseen data.

The other models exhibited comparable but slightly lower performance on the test set. The 3-mer model achieved an ACC of 0.8500, with an SN of 0.8400, an SP of 0.8600, and an AUC of 0.8741, though its MCC of 0.7001 was lower than that of the 4-mer model. Similarly, the 5-mer model achieved an ACC of 0.8700 and an AUC of 0.8618, with a strong SP of 0.9000 but a slightly lower SN of 0.8400. The 6-mer model showed comparable performance to the others, with an ACC of 0.8550, an SN of 0.8300, an SP of 0.8800, and an AUC of 0.8678. All models exhibited a slight decrease in performance when transitioning from the training set to the test set, which is consistent with observations made in the Layer I results.

Figure B.1: Performance comparison of k-mer models on identification and classification tasks using multiple evaluation metrics for both training and test sets.



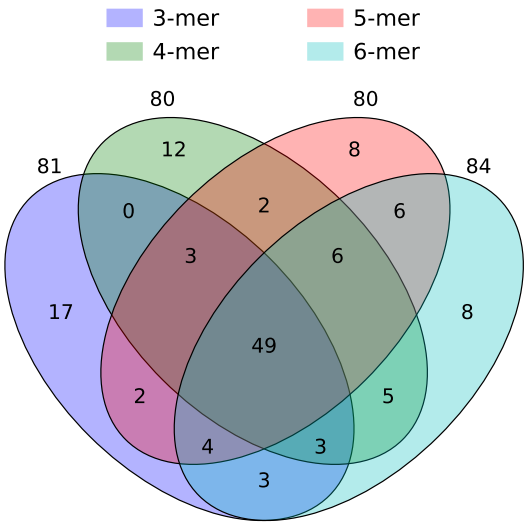
B.3 Extra Plots

This section presents a collection of extra plots that provide supplementary visualizations and further insights into the data and model analysis.

Venn Diagrams

Figure B.2: Venn diagrams illustrating the overlap of incorrectly predicted samples among different models in both layers.

(a) Layer I.



(b) Layer II.

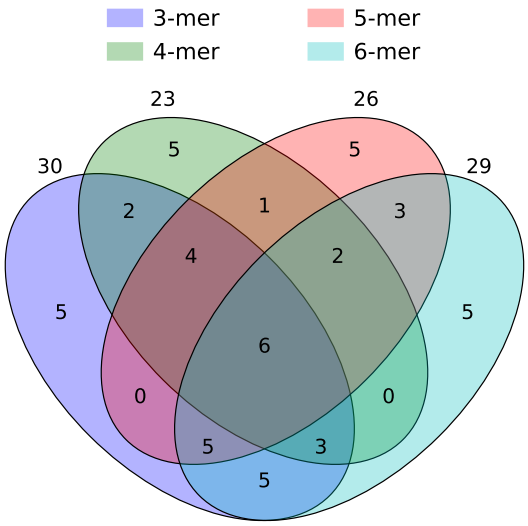


Figure B.2a and Figure B.2b present Venn diagrams illustrating the overlap of incorrectly predicted samples among four k-mer models across two different layers.

In Layer I, each model exhibits a substantial number of unique incorrect predictions (3-mer: 81, 4-mer: 80, 5-mer: 80, 6-mer: 84), with a significant central overlap of 49 samples misclassified by all four models, suggesting persistent classification challenges regardless of k-mer length.

In contrast, Layer II demonstrates a dramatic reduction in both unique misclassifications (3-mer: 30, 4-mer: 23, 5-mer: 26, 6-mer: 29) and shared errors, with only 6 samples misclassified by all models, indicating either improved model performance or less complex classification tasks in this layer.

UpSet Plots

The UpSet plots in Figure B.3 and Figure B.4 illustrate the distribution of incorrectly predicted sequences in the two tasks.

In Layer I, the errors are highly concentrated, with the largest intersection involving 49 sequences that share specific k-mer features, and several other intersections of moderate size. The set sizes for each k-mer group are also relatively large, ranging from 80 to 84. This pattern indicates that mispredictions in enhancer identification tend to cluster around certain sequence motifs, suggesting that these motifs pose particular challenges for the model.

In contrast, Layer II, which uses a subset of the Layer I data to classify enhancer strength, shows a more dispersed error pattern. The largest intersection size drops to 6, and the set sizes for k-mer groups are much smaller (23 to 30). Errors in Layer II are more evenly distributed across different k-mer combinations, reflecting the increased difficulty and subtlety of distinguishing strong from weak enhancers.

Overall, these plots highlight that while Layer I errors are dominated by a few problematic motifs, Layer II errors are more scattered, likely due to the finer distinctions required in enhancer strength classification.

Figure B.3: UpSet plot visualizing the intersections of incorrectly predicted peptide sequences in Layer I, categorized by the presence of specific k-mers. The bar chart shows the number of incorrect predictions sharing each combination of k-mers.

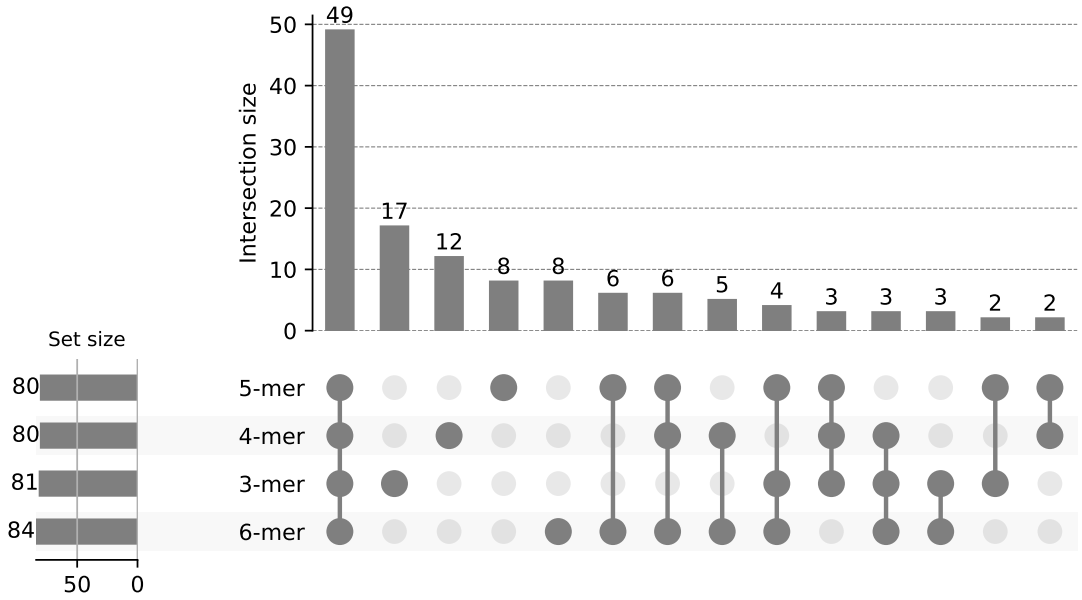
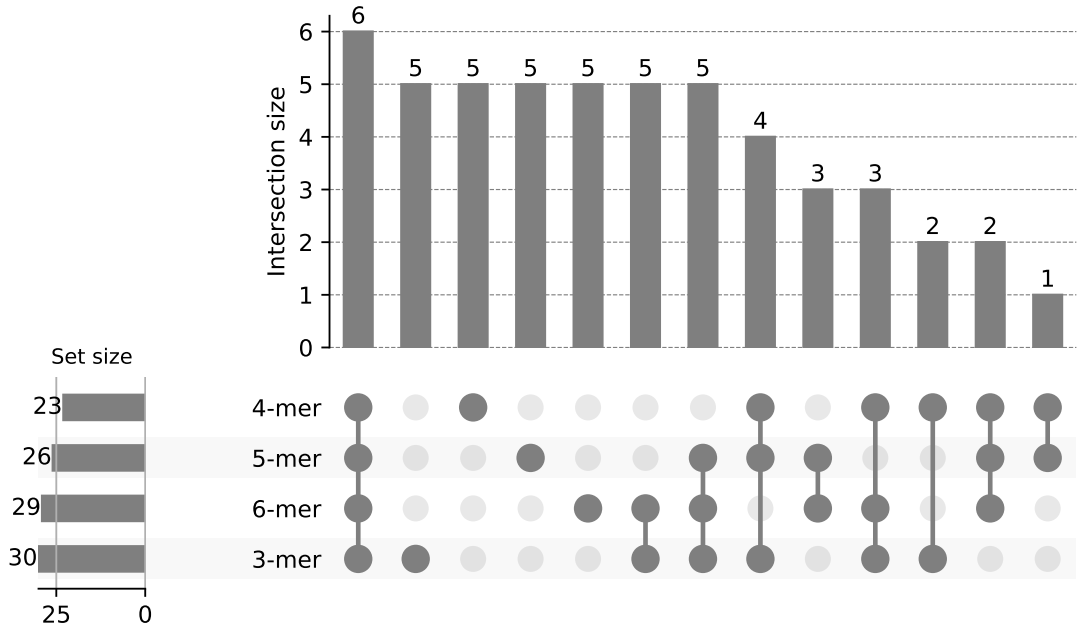


Figure B.4: UpSet plot visualizing the intersections of incorrectly predicted peptide sequences in Layer II.

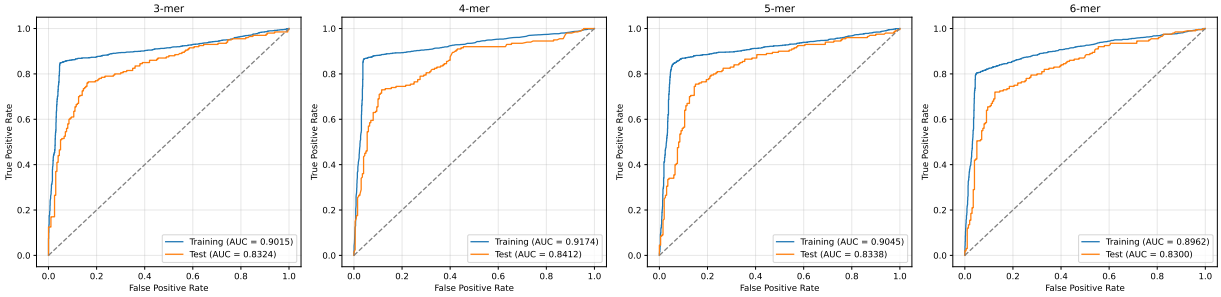


ROC and PR Curves

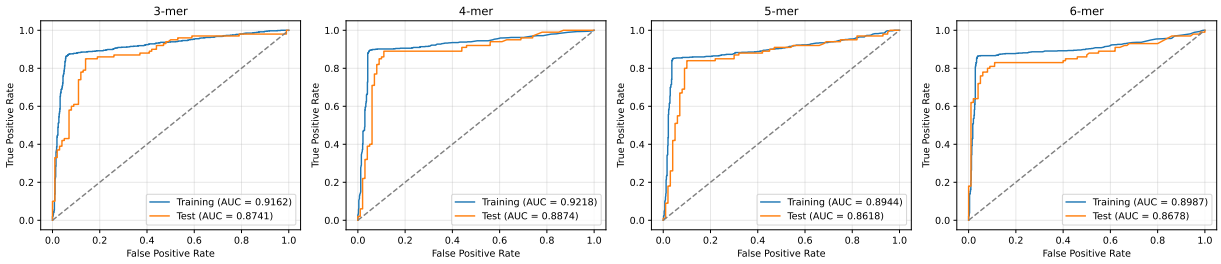
Figure B.5 and Figure B.6 illustrates the ROC and PR curves for each k-mer base model. It can be seen that the ensemble model gives better results than the individual models.

Figure B.5: ROC curves of base models for dual-layer.

(a) Layer I.



(b) Layer II.



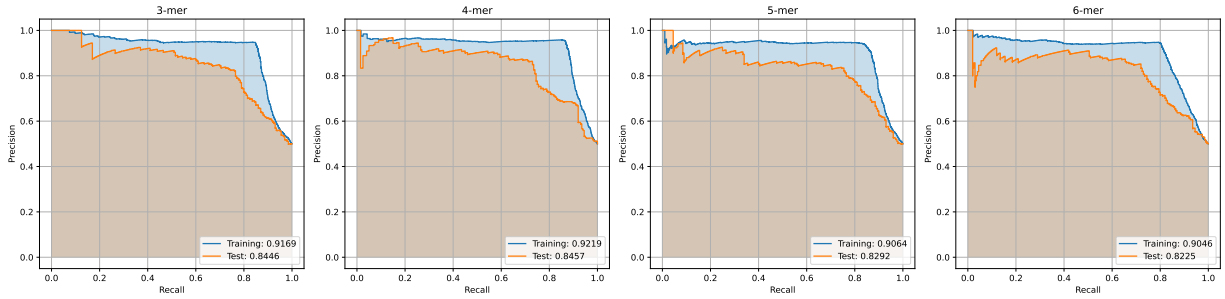
SHAP Plots

Evaluating on the independent test set, Figure B.7a and Figure B.7b show similar trends to the training data.

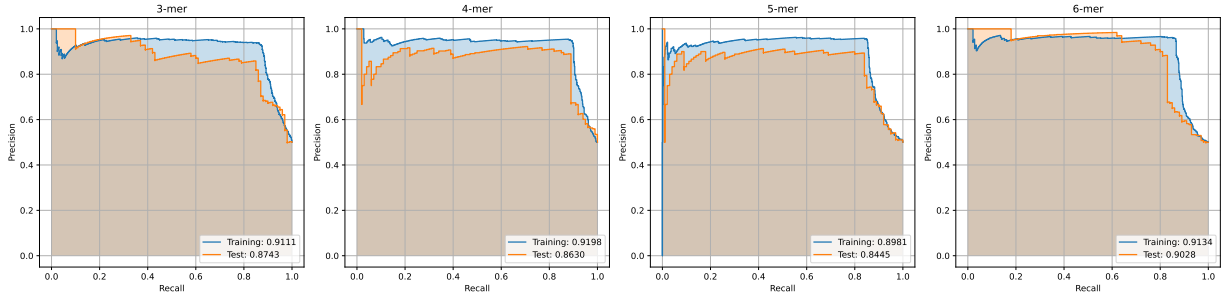
In Figure B.9, both models show that 4-mers yield the highest total mean absolute SHAP values, indicating that features derived from 4-mers are most influential in both models. However, the magnitude of SHAP values is much higher for XGBoost (ranging from 7.32 to 9.34) compared to Extra Trees (ranging from 0.09 to 0.21), suggesting that XGBoost attributes greater overall importance to its features, or that its predictions are more sen-

Figure B.6: PR curves of base models for dual-layer.

(a) Layer I.



(b) Layer II.



sitive to individual feature contributions.

Figure B.8 further details the top 20 most important features for each model. For both models, 4-mer features dominate the top ranks, reinforcing the earlier finding that 4-mer derived features are most predictive. While Extra Trees displays a more even distribution of feature importance among the top features, XGBoost exhibits a steeper drop-off, with a few 4-mer features standing out with particularly high SHAP values. Notably, XGBoost also includes some 6-mer features among its top contributors, indicating a broader sensitivity to feature length. Overall, these figures highlight that both models prioritize 4-mer features, but XGBoost does so with greater intensity and a slightly broader feature scope.

Figure B.7: SHAP summary plots for the Extra Trees and XGBoost models, showing the impact of the top 20 most important features on enhancer predictions using the independent test set.

(a) Extra Trees model.

(b) XGBoost model.

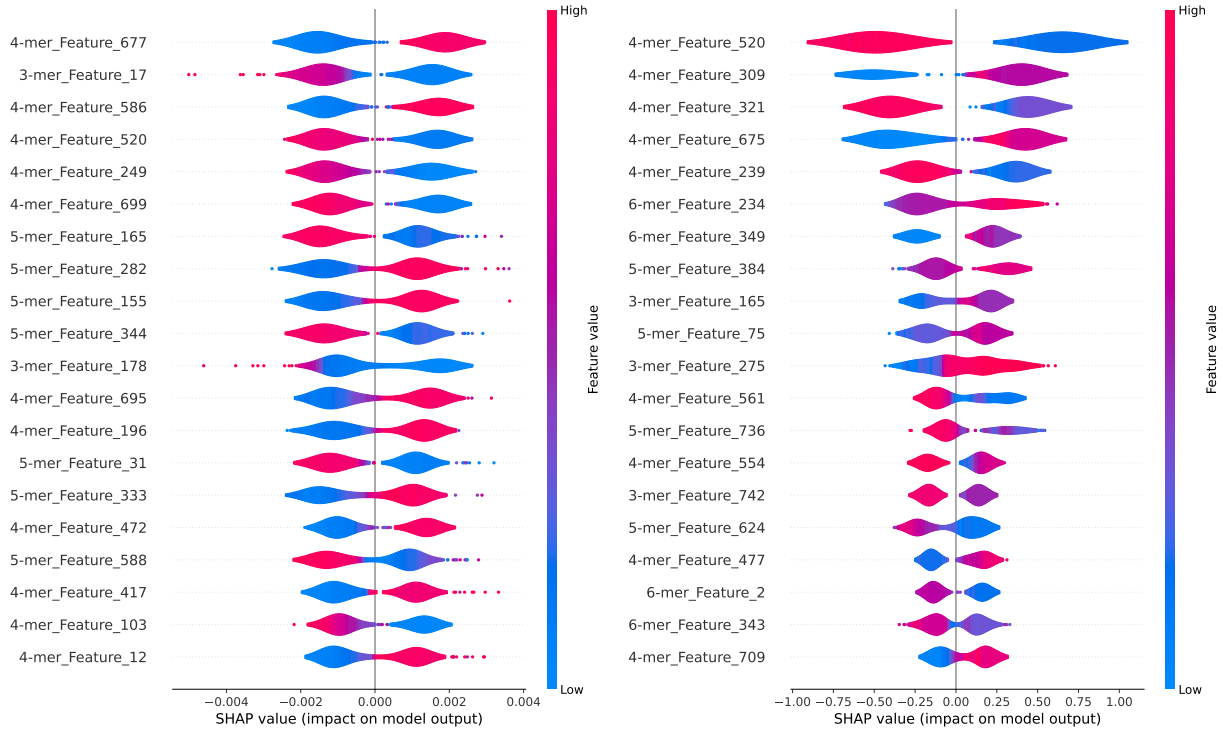


Figure B.8: Top 20 most important features using the training set.

(a) Extra Trees model.

(b) XGBoost model.

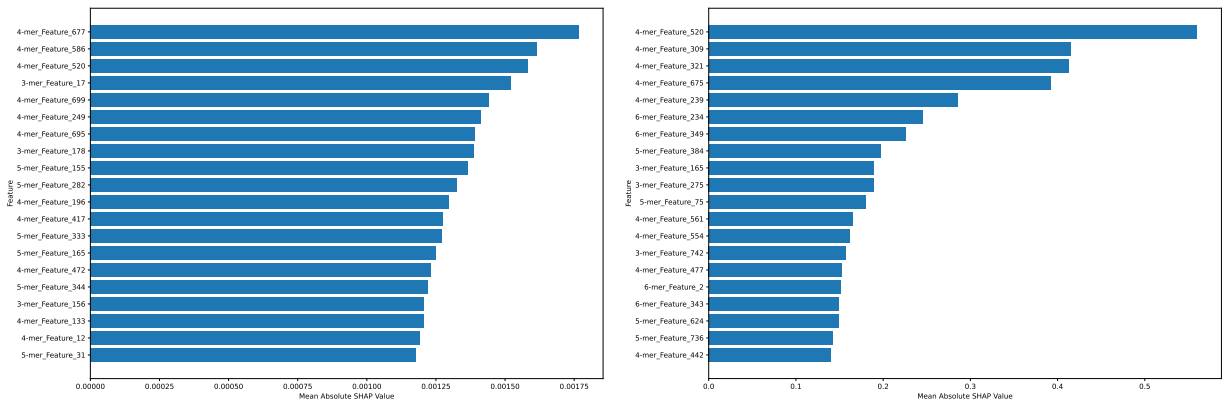
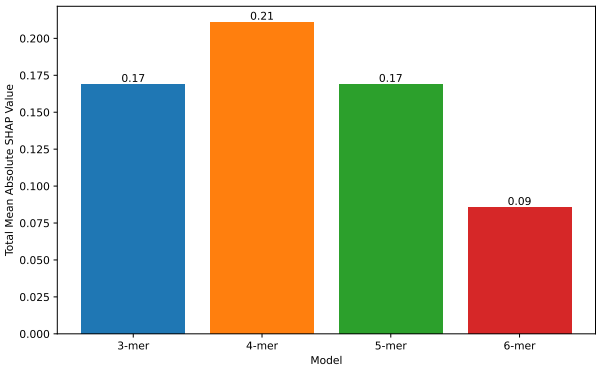


Figure B.9: Total mean absolute SHAP values analyzed by k-mer size using the training set.

(a) Extra Trees model.



(b) XGBoost model.

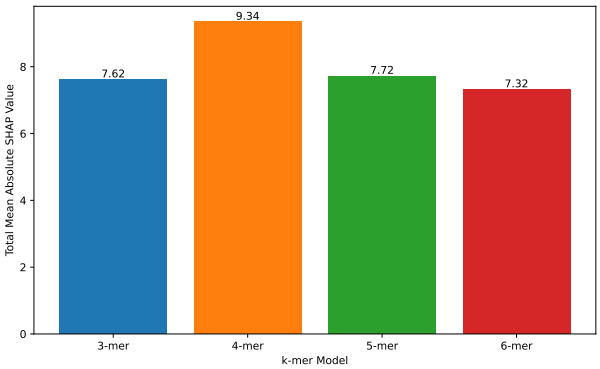
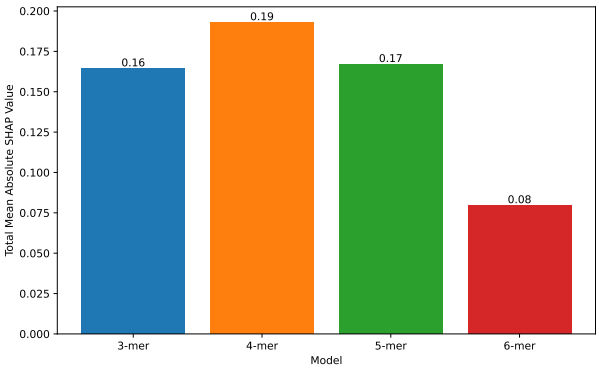
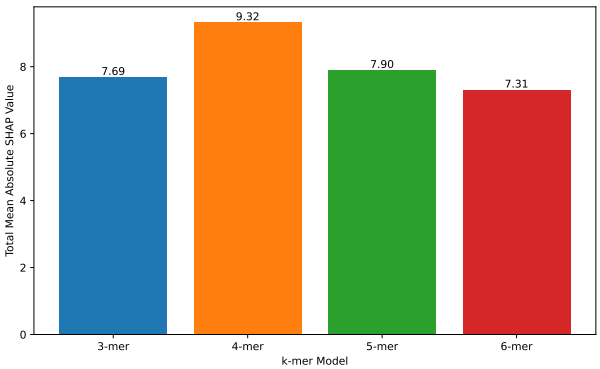


Figure B.10: Total mean absolute SHAP values analyzed by k-mer size using the independent test set.

(a) Extra Trees model.



(b) XGBoost model.



B.4 Trending Analysis

Figure B.12 and Figure B.13 present the training and testing metrics across epochs for 30 different configurations, with fixed learning rates of 1×10^{-5} and 1×10^{-4}, respectively, and a range of λ values from 1×10^{-3} to 9×10^{-5}. The results indicate that increasing the number of epochs does not significantly impact accuracy or other performance metrics. Train accuracy remains relatively stable, while test accuracy exhibits fluctuations without clear improvement over time. Other metrics also display variability but do not show a consistent trend of enhancement as training progresses. This suggests that extending training beyond a certain point does not necessarily contribute to better performance, possibly due to model convergence or instability in certain configurations.

Figure B.11: Effect of the regularization parameter λ and learning rate.

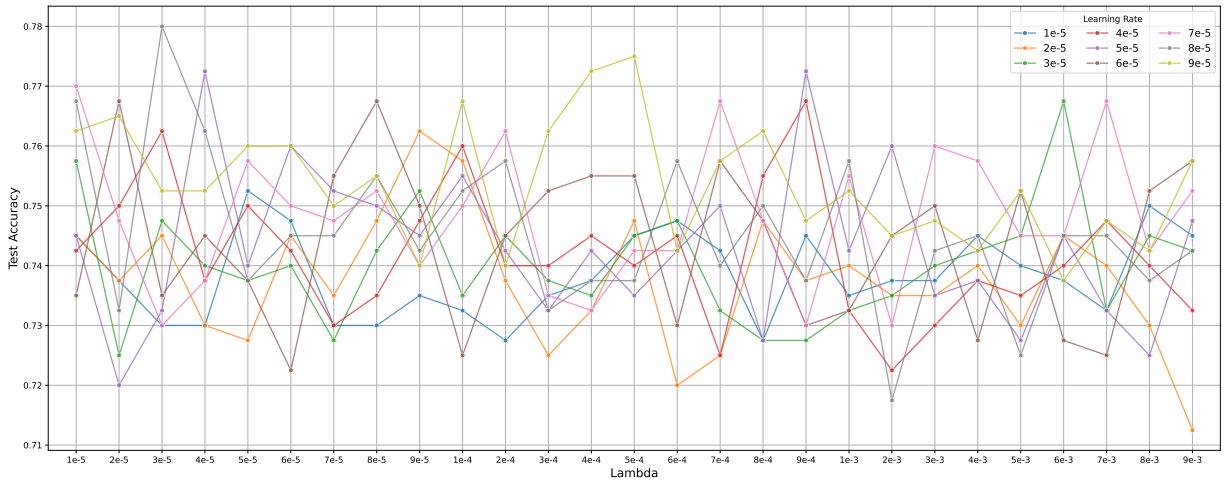


Figure B.11 illustrates the volatile and non-monotonic relationship between test accuracy and the hyperparameters for learning rate and λ. The results do not show smooth convergence; instead, accuracy fluctuates significantly with minor changes to the λ parameter. For example, a $8e-5$ learning rate peaks at 0.7800 accuracy, but only specifically near $\lambda = 3e{-}5$. This indicates that optimal performance relies on a specific, sensitive interaction between these hyperparameters rather than a broad, stable convergence point.

Figure B.12: Performance metrics across epochs with a fixed learning rate of 1×10^{-4}.

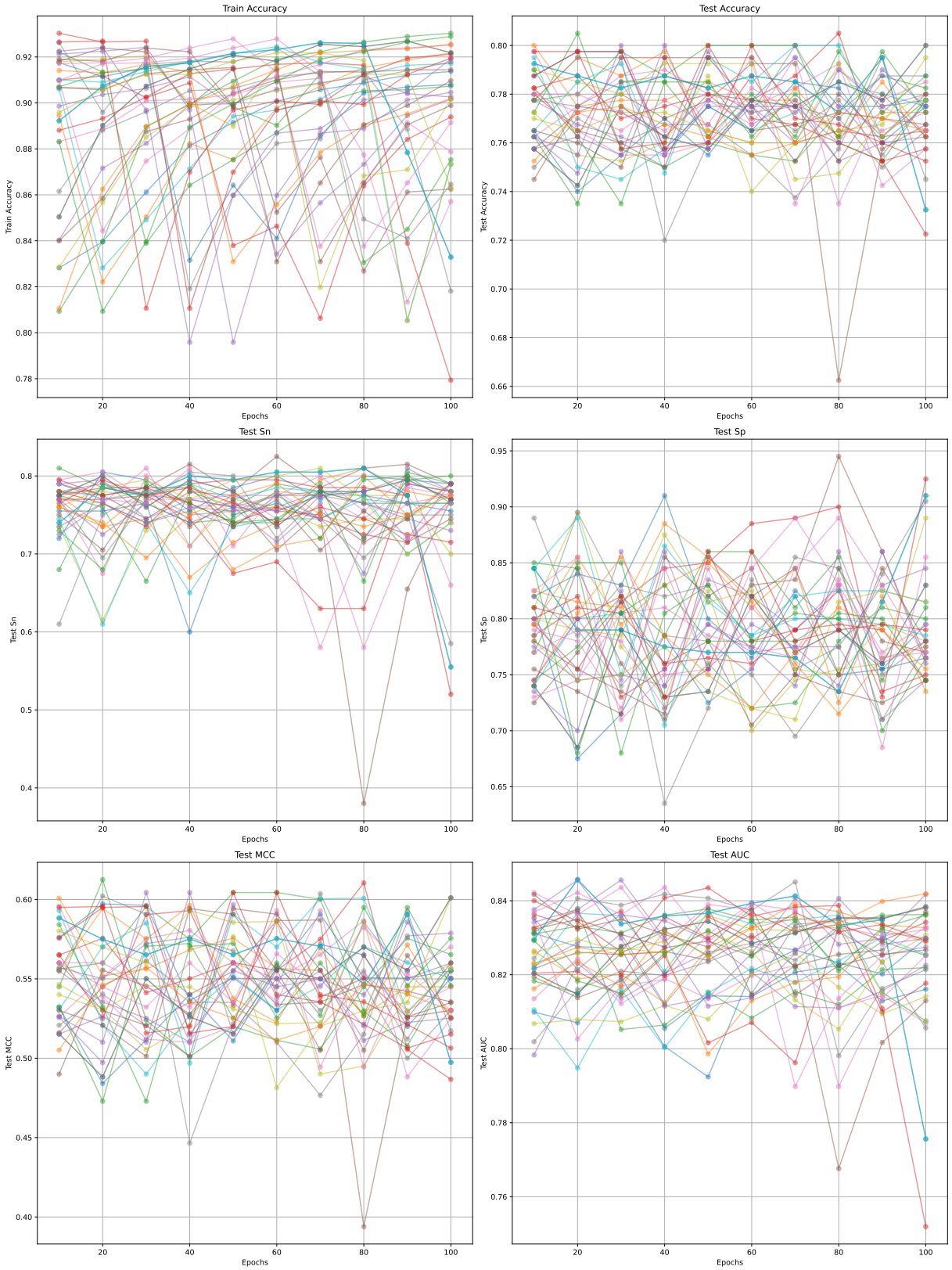


Figure B.13: Performance metrics across epochs with a fixed learning rate of 1×10^{-5}.