seqLens: optimizing language models for genomic predictions

Mahdi Baghbanzadeh[1], Brendan Mann[1], Keith A. Crandall[1], Ali Rahnavard[1*]

[1]Department of Biostatistics and Bioinformatics, Computational Biology Institute, The George Washington University, Washington, 20052, DC, USA.

*Corresponding author(s). E-mail(s): rahnavard@gwu.edu;
Contributing authors: mbagh@gwu.com; bmann@gwu.com; kcrandall@gwu.edu;

Abstract

Understanding genomic sequences through the lens of language modeling has the potential to revolutionize biological research, yet challenges in tokenization, model architecture, and adaptation to diverse genomic contexts remain. In this study, we investigated key innovations in DNA sequence modeling, treating DNA as a language and applying language models to genomic data. We gathered two diverse pretraining datasets: one consisting of 19,551 reference genomes, including over 18,000 prokaryotic genomes (115B nucleotides), and another more balanced dataset with 1,354 genomes, including 1,166 prokaryotic and 188 eukaryotic reference genomes (180B nucleotides). We trained five byte-pair encoding tokenizers and pretrained 52 DNA language models, systematically comparing different architectures, hyperparameters, and classification heads. We introduce `seqLens`, a family of models based on disentangled attention with relative positional encoding, which outperforms state-of-the-art models in 13 of 19 benchmarking phenotypic predictions. We further explore continual pretraining, domain adaptation, and parameter-efficient fine-tuning methods to assess trade-offs between computational efficiency and accuracy. Our findings demonstrate that relevant pretraining data significantly boosts performance, alternative pooling techniques enhance classification, and larger tokenizers negatively impact generalization. These insights provide a foundation for optimizing DNA language models and improving genome annotations.

Keywords: DNA Sequencing, Language Models, Bioinformatics, Metagenomics, Genomics

1 Main

DNA is a language [1–3] with many unexplored aspects due to the challenges associated with experimentally validating the function of genetic elements on a genome-wide scale. Genomics and metagenomics [4] have transformed primarily due to rapid advancements in high-throughput sequencing (HTS) technologies [5]. These technologies have catalyzed the era of big data in genomics, generating vast amounts of genomic information with precision and detail [6–8]. Improvements in HTS technologies have increased accessibility and cost-effectiveness in sequencing nucleic acids from complex microbial ecosystems [9]. However, a remaining challenge lies in extracting meaningful insights from this data deluge in a computationally efficient manner, which is where machine learning (ML) and deep learning (DL) techniques have emerged as invaluable tools [10–13].

The application of DL to DNA sequences initially targeted specific biological problems. Early studies explored neural network implementations for various tasks, including splice site prediction [14], modeling genetic responses to specific medications [15], identifying chromatin accessibility prediction [16], transcription factor binding sites [17], and performing taxonomic profiling [1, 18, 19]. In contrast to these studies, which focused on training separate models for specific tasks, language models offer a more generalized approach. DNA foundation models are first pretrained on large-scale genomic sequences, allowing them to develop a broad understanding of genomic patterns and structures. Once pretrained, a single model can be adapted to a variety of specific tasks through fine-tuning, eliminating the need to train separate models for each individual application [2, 3, 20, 21]. This flexibility makes language models a powerful tool for a wide range of genomic analyses. DNABERT [2] is a pretrained bidirectional encoder model based on the bidirectional encoder representations from transformers (BERT) architecture [22], trained with the human genome and designed to capture nucleotide context and improve genomic DNA sequence interpretation. Although DNABERT showed promising results for predicting gene regulatory elements such as promoters and transcription factor binding sites, its original implementation had technical limitations, including overlapping k-mer tokenization and restricted handling of long input sequences [3]. The overlapping k-mer tokenization, as discussed in the DNABERT-2 study, is not ideal for masked language modeling because each token (k-mer) contains information from adjacent tokens, leading to significant information leakage [3]. Despite these challenges, DNABERT demonstrated generalization across species, highlighting the potential for further advancements in DNA language models. DNABERT-2 builds on the original DNABERT model by addressing the limitations of k-mer tokenization, which caused inefficiencies in computation and sample processing [3]. By replacing k-mer with byte-pair encoding (BPE) [23], DNABERT-2 improves tokenization efficiency and scalability, allowing better handling of longer input sequences and reducing time and memory costs. This refined model enhances the ability to decode genomic sequences with greater computational efficiency and model performance [3]. This tokenization method was then implemented in other studies such as GROVER [21] that trained a BERT model with the same architecture only using the human genome assembly GRCh37 as pretraining dataset. The Nucleotide Transformer models [20], developed to enhance molecular phenotype prediction from DNA

sequences, leverage data from 3,202 human genomes and 850 genomes across diverse species. Their models demonstrated the ability to transfer learning between tasks and outperform specialized methods on multiple prediction tasks, even in low-data settings. Nucleotide Transformer v2 enhances efficiency by incorporating rotary positional embeddings, Gaussian error linear units (GELUs), multi-species datasets, and longer input sequences (2,048 tokens instead of 512), enabling it to process longer genomic contexts and improving performance [20]. However, the nucleotide transformer models use a tokenizer based on non-overlapping *6*-mers. This tokenizer processes sequences from left to right in *6*-mers. The letter "N" is excluded from k-mer groupings, so whenever an "N" appears or the sequence length is not a multiple of six, nucleotides are tokenized individually [24]. This differs from the BPE tokenizer, which segments sequences into the longest and most frequent available k-mers, ranging from single nucleotides to larger subwords. The authors of the Nucleotide Transformer models [20] also introduced a comprehensive set of benchmark datasets, focusing on the human genome, to evaluate the generalizability of the DNA language models. These datasets include tasks related to *chromatin profiles*, *regulatory elements*, and *splicing*. We also extended this list by adding two new datasets of *genome origin*, providing a standardized framework for assessing model performance across diverse genomic applications.

In this study, we develop innovative DNA sequence decoding and prediction strategies using advanced language models. To facilitate the understanding and decoding of DNA sequences, we employ encoder architectures, namely DeBERTa [25] and ESM [26, 27], and introduce `seqLens` models (Figure 1a). We selected the DeBERTa-v2 architecture for its strong selection strategy for smaller models or limited data scenarios and because it achieves superior performance with half the training data compared to other models [25]. The disentangled attention mechanism, which separately encodes content and position in DeBERTa, specifically allows it to model token relationships more effectively with fewer parameters [25]. We also explored the ESM model with rotary positional encoding [28]. The ESM model was originally designed for protein structure prediction and design, but implementing this encoder architecture with rotary positional encoding demonstrated promising increased performance in the nucleotide-transformer study [20].

To train the `seqLens` models, we gathered two pretraining datasets. First, a dataset includes 19,551 representative genomes from the National Center for Biotechnology Information (NCBI) database. This dataset predominantly includes bacterial genomes (over 18,000). The second dataset is an extended version of the multi-species pretraining dataset used in the Nucleotide Transformers [20] study by adding 499 archaea genomes and 5 eukaryotic genomes (see subsection 8.1). The benchmarking results on various applications indicate that model performance is significantly influenced by tokenization methods, architectural design, parameter size, training steps, and the domain of pretraining data (see subsection 2.2)

To tokenize DNA sequences, we employed a dynamic approach using BPE [29], which allows flexible token lengths (Figure 1). Our experiments demonstrate that dynamic tokenization accelerates model convergence and results in a lower training and validation loss than the k-mer-based tokenization (subsection 2.1). We implemented

and trained `seqLens` models based on DeBERTa-v2 [25] (with and without disentangled attention) and ESM architectures [20, 26, 27], using a context length of 512 and various hyperparameter configurations (see Figure 1a, Figure B2, and Appendix B for details). Our results show that incorporating disentangled attention with relative positional encoding is particularly effective, as this architectural choice improves the `seqLens` models' ability to capture relevant features from DNA sequences (see subsection 2.2).

For optimized fine-tuning, we investigated the low-rank adaptation (LoRA) approach, a parameter-efficient fine-tuning method that significantly reduces the number of trainable parameters, making it well-suited for resource-constrained environments [30]. However, our analysis identifies a critical trade-off: while LoRA excels in computational efficiency, it consistently underperforms compared to full fine-tuning in terms of model performance. These findings suggest that the efficiency gains achieved through LoRA come at the cost of reduced performance, particularly in tasks requiring nuanced understanding or complex feature extraction. We also explored alternatives to conventional use of the `[CLS]` token in classification models and investigated three approaches: 1) mean pooling, 2) max pooling, and 3) concatenation of `[CLS]`, mean pooling, and max pooling (`CLS|Mean|Max`) (Figure 1b). Our results indicate that the `[CLS]` token does not consistently yield the best performance and in some cases, passing the mean pooling or `CLS|Mean|Max` can improve the performance of the classifier. We also found that max pooling was less effective for classification tasks compared to the other methods (see subsection 2.8). Finally, we analyzed the vector representations of DNA sequences through the model and assessed how fine-tuning shifts these representations. Using clustering metrics, we found that max pooling produces vector representations that are more aligned with the actual sequence categories. Fine-tuning further enhances this alignment, demonstrating its importance for downstream performance. Lastly, we explored continual pretraining (Figure 1b) and demonstrated that, when applied with domain-specific data, it substantially enhanced model performance on downstream tasks, facilitating effective adaptation to specialized domains (see subsection 2.8). Since biological datasets often suffer from limited annotated data, making it impractical to train language models solely on small datasets, domain adaptation presents a viable approach to leveraging pretrained models effectively. Based on these findings, bioinformaticians can adapt a generic DNA model to their specific domain using unannotated genomic data from their area of interest (e.g., taxonomic focus such as fungi, and/or genomic focus such as antibiotic resistance). Following domain adaptation, the model can then be fine-tuned on the limited annotated data available, improving its performance on domain-specific tasks.

Our study underscores the importance of dynamic tokenization, pretraining datasets, advanced architectural features such as disentangled attention, and continual pretraining in enhancing the performance of genomic language models. By systematically analyzing these factors, we provide a robust framework for developing future DNA language models that are more effective, adaptable, and capable of addressing complex biological questions. We trained and benchmarked `seqLens` DNA language models in various configurations, and one of the `seqLens` models outperformed the state-of-the-art models in 13 of 19 benchmarking tasks. In addition, we

investigated different fine-tuning strategies, classification methods, and vector representation approaches that can be used in future studies. These insights establish a solid foundation for developing robust DNA language models that are applicable to a broad range of biological tasks.

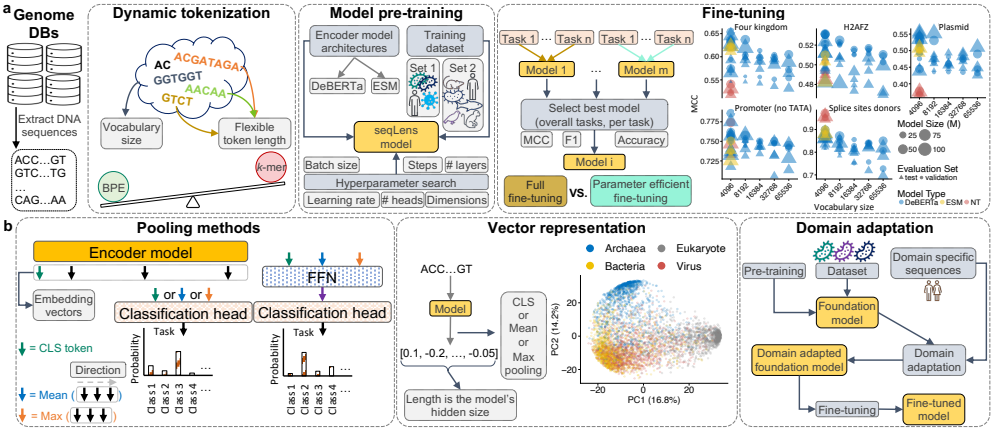

Fig. 1: **End-to-end DNA sequence modeling: core and auxiliary strategies. a, first row:** The workflow begins with genome collection and dataset preparation, followed by BPE tokenization and model training using masked language modeling. Models are benchmarked on downstream tasks, and fine-tuning strategies are compared. **b, second row:** Various embedding pooling methods are evaluated for classification, along with genome vector representations and fine-tuning effects. Domain adaptation is also explored by continuing pretraining on a second dataset to assess performance impact.

2 Results

2.1 Byte-pair encoding tokenization improves convergence and performance in DNA language models

Tokenization is a crucial preliminary step in processing text sequences, as it defines how inputs are structured for models [31]. In our study, we selected the BPE approach to develop our tokenizers, a choice informed by established research in natural language processing [32] and previous investigations in applications of language modeling [29] in DNA [3, 21]. BPE is particularly suitable for handling large vocabularies and capturing subword information, making it a robust choice for diverse language data.

To comprehensively explore tokenizer behavior, we experimented with five different vocabulary sizes: 4,096, 8,192, 16,384, 32,768, and 65,536. We sampled 1 million sequences from the training dataset to train the tokenizers. We detail the key characteristics observed for each tokenizer after training in Table A1 and Figure A1a. For example, the 4096 tokenizer is like a dictionary with 4096 vocabularies, consisting of tokens such as {A, C, ..., TCCTGGCC, TAAACCC}, with approximately 53% of

them being *7*-mers (tokens of length 7) and the maximum length of tokens in this tokenizer is *10*-mers with frequency of 0.05% (Table A1). The varying maximum token lengths and frequency distributions highlight the trade-offs between vocabulary size and token representation efficiency. In general, larger vocabularies tend to produce longer tokens (Table A1), which can significantly impact the frequency distribution of token lengths, as illustrated in Figure A1. This behavior has a direct influence on the efficiency of model training and inference, affecting both computational costs and the model's ability to understand language [33].

The BPE tokenizer's ability to dynamically tokenize DNA sequences provides an edge in capturing complex data patterns, leading to better model performance, while the robust implementation ensures computational efficiency. To evaluate the efficiency of the BPE tokenizer, we conducted a series of experiments using the Nucleotide Transformer multi-species models with 50M and 100M parameters, denoted as `NT-50` and `NT-100`, respectively [20, 24]. Our objective was to compare the BPE tokenizer against a fixed-length *6*-mer tokenizer that was introduced in the original study of these models. Both models were trained under identical conditions, including the same parameter configuration, dataset, and computational resources (GPU nodes). We evaluated the models by measuring the loss on the validation set and the total floating point operations (TFLOP), as shown in Figure A1b and Figure A1c.

The validation loss results (Figure A1b) demonstrate an advantage for the BPE tokenizer. When using the BPE tokenizer, both the `NT-50` and `NT-100` models achieved significantly lower validation losses compared to the scenario in which they utilized the fixed-length *6*-mer tokenizer. Furthermore, the models showed faster convergence with the BPE tokenizer, indicating a more efficient learning process. This suggests that the flexible, variable-length nature of the BPE tokenizer provides a better fit for the data, capturing more relevant patterns that aid the learning process. In terms of computational efficiency, the BPE tokenizer exhibited a higher TFLOP value compared to the fixed-length tokenizer, as illustrated in Figure A1c. It is also worth mentioning that we trained the models for 150K steps or a maximum time of 7 days. Since models with the *6*-mer tokenizer take longer to train, they did not reach the maximum number of steps in the given time. The results highlight the effectiveness of the BPE tokenizer in both model accuracy and training efficiency.

2.2 Smaller tokenizers lead to better generalization

We trained a set of models ranging from 15M to 111M parameters using various architectures, setups, and approaches (see Appendix B). The architectures and model sizes of the `seqLens` models were based on the DeBERTa-v2 study and models used in the Nucleotide Transformers [20] study (`ESM`). To benchmark the models, we included two pretrained models developed by the Nucleotide Transformers team: `NT-50` and `NT-100`. These two models were selected due to their strong performance and compatibility with our computational resources.

There is no universally best-performing model (Figure C3), although `seqLens` models generally have a better score on test sets. Across datasets, models based on smaller vocabulary sizes generally outperform larger ones on the test set. As vocabulary size increases, model performance tends to decrease, although not uniformly (Figure 2a,

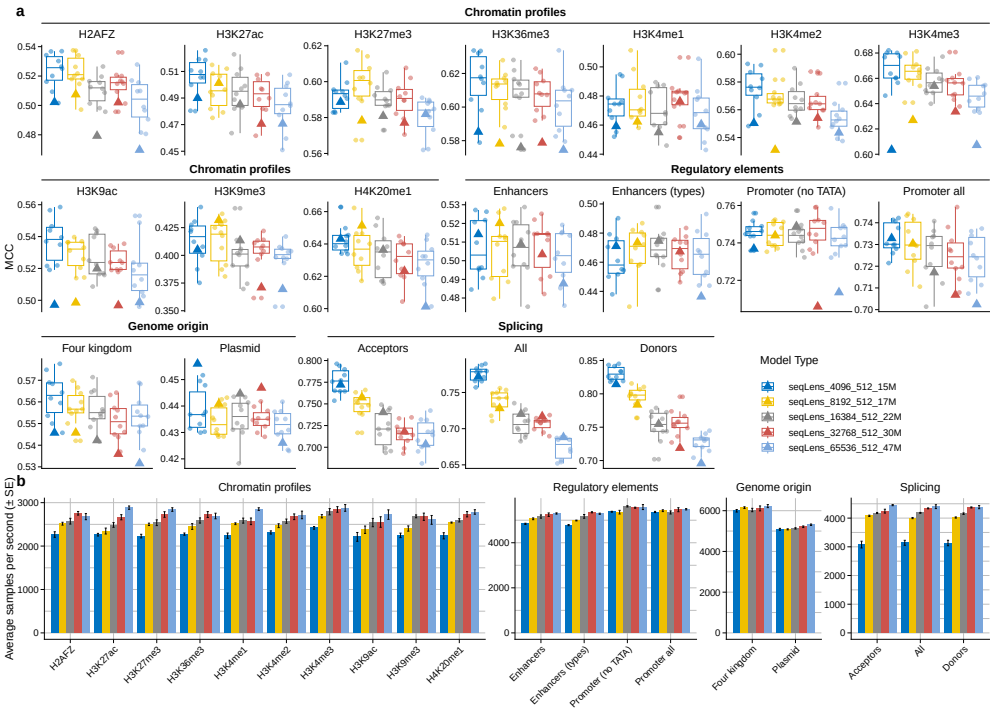

Fig. 2: **Validation and test scores of models across benchmarking datasets.** **a**: The figure compares the performance of the models with the same parameter configuration but different tokenizer vocabulary sizes. All the models have 4 *layers*, 4 *attention heads*, *intermediate size* of 2048, and *hidden size* 512. Boxplots are based on the validation scores (circles), and the triangle represents the models' performance on the test set. **b**: The figure compares the processing time of the models by comparing the average samples per second.

Figure C4a, and Figure C5a). This can be attributed to two factors: the increased number of token embeddings requiring tuning and the reduced likelihood of tokens being sufficiently observed during pretraining. For example, for a given model with a *hidden dimension* of 512, each new vocabulary adds 512 new embedding weights to the model. So, the difference in parameter size of a model based on the 4,096 and the 65,536 tokenizers is more than $(65536 - 4096) > 30M$ embedding weights. Also, if we consider a uniform chance of observing a token, the chances to train the weights of a token using the first tokenizer relative to the last tokenizer are roughly $65536/4096 \approx 16$ times more. Consequently, the weights of all tokens may not receive sufficient tuning due to the fixed number of training steps.

Larger tokenizers segment input sequences into longer tokens, resulting in shorter overall sequence lengths and, consequently, faster processing times (Figure 2b, Figure C4b, and Figure C5b). This effect is particularly noticeable in tasks such as *chromatin profiles*, where raw sequences are 1 Kbp in length, and *splicing* tasks, where

the input sequences are 600 bp long. The reduction in sequence length due to larger tokenization units enhances computational efficiency, especially for longer genomic sequences.

2.3 Disentangled attention enhances seqLens model performance across most genomic tasks

One group of the seqLens models we trained involved varying parameter sizes of DeBERTa-v2 and an investigation into the effect of disentangled attention. Considering the performance on the test set, the small model outperforms its counterpart in 17 of 19 tasks, with exceptions in two *chromatin profiles* tasks (H3K27me3 and H3K4me1) (Figure 3a). The small architecture differs from the other model with 46M parameters by implementing disentangled attention and having 12 attention heads (instead of 6). Similarly, the base model outperforms the other model with 89M parameters (its non-disentangled attention equivalent) in 15 of 19 tasks, with exceptions in three *chromatin profiles* tasks (H3K4me1, H3K4me2, and H3K4me3) and the splicing acceptors task (Figure 3). These results highlight the potential benefits of incorporating disentangled attention, particularly for most tasks, although certain *chromatin profiles* and *splicing* tasks remain challenging. Figure 3a also allows for a comparison among the xsmall, small, and base models. The base model demonstrates superior performance in 10 of 19 tasks, including the four kingdoms and plasmid detection datasets. The xsmall model performs better in 5 tasks, showcasing that smaller models can sometimes excel in specific scenarios.

With the same tokenizers, larger models process sequences more slowly than smaller models (Figure 3b). For example, the 15M model processes samples, on average, 2.5 times faster than the 89M model. Additionally, disentangled attention led to slower sample processing, with the 89M model being, on average, about 20% faster than the 89M-at-base model.

2.4 seqLens models achieve strong performance across genomic tasks

We trained two of the seqLens models with configurations similar to NT-50 and NT-100 but with a context length of 512. By maintaining similar training datasets, tokenizers, and infrastructures, this approach enables a more direct comparison of architectures. In both the four kingdoms and plasmid detection datasets, the ESM-based seqLens models outperformed the NT models, likely due to the influence of training data Figure 4. However, overall, Nt-100 outperformed its equivalent model in 15 tasks, and Nt-50 outperformed its equivalent in 13 tasks. Since the backbone of these models is the same, except for the tokenizer, context length, pretraining dataset, and training steps, these results underscore the importance of pretraining dataset composition and its interaction with model architecture and tokenizer design in determining downstream task performance.

Conversely, 47M-at-small outperformed seqLens_esm_4096_512_55M (similar to NT-50) in 14 of 19 tasks (Figure 4). Exceptions included two chromatin profiles tasks and all three splicing tasks. When comparing 47M-at-small with

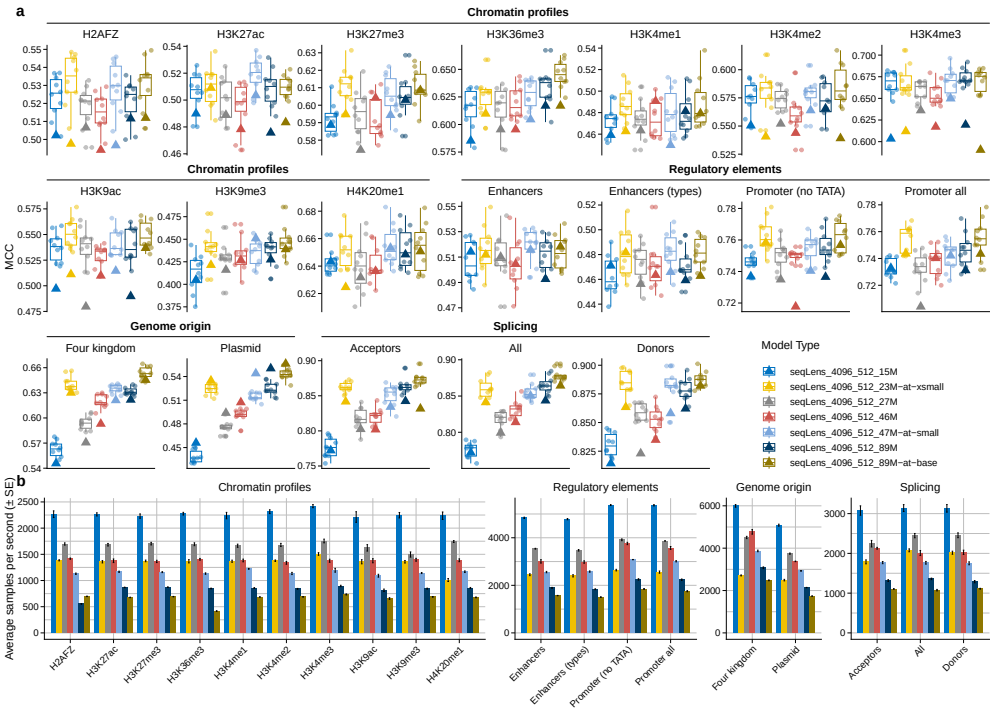

Fig. 3: **Benchmarking results of `seqLens` models with varying parameter sizes and the effect of disentangled attention. a**: The figure compares the performance of seven `seqLens` models on 19 tasks, highlighting the impact of disentangled attention. The boxes are based on the validation scores, and the test score is marked with a triangle. The `small` model outperforms its counterpart without disentangled attention in 17 of 19 tasks. Similarly, the `base` model outperforms its non-disentangled equivalent in 15 of 19 tasks. Comparisons among `xsmall`, `small`, and `base` models show that the `base` model performs best in 10 of 19 tasks. Boxplots are based on the validation scores (circles) and the triangle represents the models' performance on the test set. **b**: The figure compares the processing time of the models by comparing the average samples per second.

`seqLens_esm_4096_512_97M` (similar to `NT-100`), which has nearly twice the parameters, the smaller model still outperformed the larger model in 12 of 19 tasks. Exceptions included two `chromatin profiles` tasks, all three `splicing` datasets, the `promoter all` dataset, and the `four kingdoms` dataset (where the performance difference was marginal: 0.621 vs. 0.622). For the larger models, `89M-at-base` outperformed `seqLens_esm_4096_512_55M` in 16 of 19 tasks. Exceptions were observed in the `H3K4me3` task and two `splicing` datasets (`acceptors` and `all`). Moreover, comparing `89M-at-base` with `seqLens_esm_4096_512_97M`, the former achieved better results in 12 of 19 tasks. Exceptions included two `chromatin profiles` tasks (`H3K4me1` and `H3K4me3`), the `promoter all` task, and two `splicing` datasets (`acceptors`: 0.832 vs.

0.835, and `donors`: 0.882 vs. 0.883). The only difference between these two types of models is their architecture. These results demonstrate the superiority of the DeBERTa-inspired `seqLens` models over the ESM models in generalizability and the ability to obtain better results in test datasets across benchmarking tasks.

Also, we trained a model similar to the configuration of the `89M-at-base` with an extended version of the NT pretraining datasets. This comparison will also help us understand the effect of the pretraining dataset. In 13 tasks `89M-at-base-multi` outperforms `NT-100` and `89M-at-base` and outperforms `NT-50` in 12 tasks (Figure 4). With this comparison, we can see the effect of pretraining dataset on the performance of the model on the downstream tasks.

These results highlight that the DeBERTa-inspired `seqLens` models often outperform ESM-based models across a majority of tasks despite having fewer parameters. The observed performance differences are not uniform, as ESM-based models excel in a few specific datasets, particularly those focused on splicing and chromatin profiles. This suggests that architectural differences and training dynamics, such as context length and parameter tuning, play critical roles in task-specific performance. Also, we observed that generally, the `seqLens` models with DeBERTa-v2 architecture provide better test scores across different tasks.

2.5 Full parameter training outperforms LoRA in most tasks

Fine-tuning large pretrained models for downstream tasks can be computationally intensive and financially prohibitive, often comparable to the original pretraining process [30, 34]. To address this challenge, we implemented LoRA, an optimization strategy that significantly reduces computational complexity [30]. LoRA works by freezing pretrained model weights and introducing trainable low-rank decomposition matrices into the model layers, thereby significantly reducing the number of trainable parameters.

In this study, we applied LoRA to two pretrained models, `NT-100` and `89M-at-base`, using a rank of 8 and a scaling factor of 32. Notably, the results demonstrate that full parameter training consistently (17 of 19 tasks) outperforms LoRA fine-tuning across most scenarios (Figure 5). These performance differences may be attributed to several factors, including our specific choice of rank and scaling factor. Prior research has already suggested that LoRA fine-tuning might inherently lead to some performance reduction [35].

2.6 CLS token performance is exceeded by alternative pooling methods

A common approach for fine-tuning models in the BERT family models for classification or regression tasks involves utilizing the `[CLS]` hidden state from the final layer. This `[CLS]` token is then passed through a feed-forward layer to make predictions [36]. A comprehensive study investigating various fine-tuning strategies for BERT models examined the utility of outputs from different layers and concluded that the last layer is generally the most informative for such tasks [37]. We then extend our investigation by experimenting with the hidden states of the final layer for all non-padding tokens.

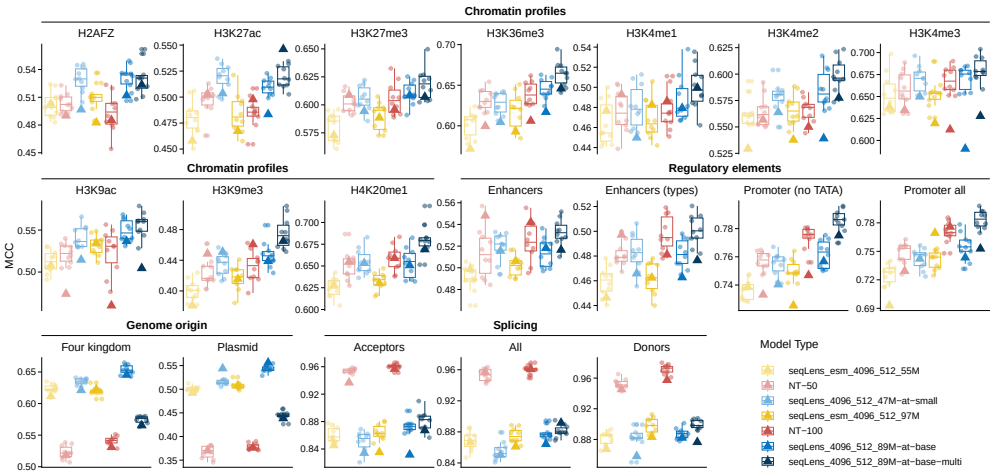

Fig. 4: Comparison of validation and test scores for DeBERTa-inspired, ESM-based models, and NT models. The figure illustrates the performance of the DeBERTa-inspired models, NT models, and ESM-based models, which have similar configurations to `NT-50` and `NT-100` but are trained with the same setup as the DeBERTa-inspired models. Validation scores are represented as individual points within each box, while test scores are marked with triangles. Across 19 tasks, DeBERTa-inspired models (`small` and `base`) generally outperform ESM-based models. `base-multi` outperforms the other models in most cases. The results demonstrate that the DeBERTa-inspired models outperform ESM-based models, highlighting the importance of architectural choices and task-specific dynamics. Boxplots are based on the validation scores (circles), and the triangle represents the models' performance on the test set.

Specifically, we explore three alternative strategies to using the `[CLS]` token: 1) mean pooling, 2) max pooling, and 3) concatenating the `[CLS]` token with the mean and max values (`CLS|Mean|Max`). Approaches 1 and 2 are widely used in sentence embedding applications and have been shown to outperform using only the `[CLS]` token in many scenarios [38, 39], motivating us to investigate their applicability in DNA language models. These approaches aim to capture richer contextual representations from the model. The experimental setup and results for these strategies are detailed in subsection 8.9.

Among the three proposed approaches, `Mean` (best in 6 tasks) and `Concat` (best in 5 tasks) generally demonstrate better performance than the `Max` approach (Figure 6a). Notably, `Concat` shows clear advantages in tasks such as `H3K36me3`, `promoter all`, `promoter (no TATA)`, `splice sites acceptors`, and `splice sites donors`. However, using the `[CLS]` token alone yields the best classification performance in 8 tasks, highlighting its effectiveness in certain scenarios (Figure 6a). These findings suggest that, while the model is often capable of effectively summarizing the necessary information within the `[CLS]` token for classification tasks, exploring alternative summarization strategies for the last hidden layer remains worthwhile. (Figure 6b) also

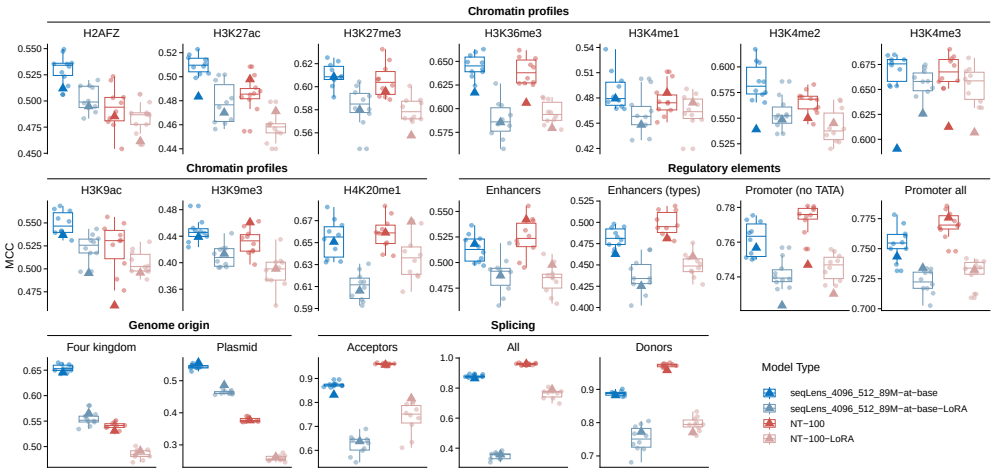

Fig. 5: Performance comparison of full parameter training and LoRA fine-tuning.. Each panel shows the results of the `NT-50` and `seqLens_4096_512_89M-at-base` models with two different approaches of full model and parameter efficient (LoRA) fine-tuning. Boxplots are based on the validation scores (circles), and the triangle represents the models' performance on the test set.

shows no clear distinction between the sample processing of the models, meaning that these different approaches do not drastically change the computation time.

2.7 `seqLens` models provide informative genome representations

Encoder models output a vector of size equal to the hidden dimension d for each input token. If an input sequence is tokenized into n tokens, the model generates a vector for each of these tokens, including special tokens such as `[CLS]`. A common way to summarize information within an input sequence is to pass the sequence through the model and then use its vector representations [38, 39]. In this study, we selected the `base` model that we trained to explore its vector outputs on two datasets: `four kingdoms` and `plasmid detection`. This analysis was performed for both the pre-trained model and the fine-tuned model to examine the differences in performance and vector representations after fine-tuning. To visualize the representations, we used the uniform manifold approximation and projection (UMAP) transformations [40] of the vectors produced by the models (Figure 7a,b) and also the first two principal components (Figure D6). For each scenario, we summarized the vectors from the last hidden state of the model using `[CLS]` token, mean pooling, and max pooling. These transformations were applied to both the pretrained model (representing a general understanding of DNA sequences) and the fine-tuned model. For the fine-tuned models, we also selected the last hidden state prior to the classification head for analysis. Fine-tuning resulted in significant improvements in vector representations, with better separation and clustering of data points (Figure 7a,b).

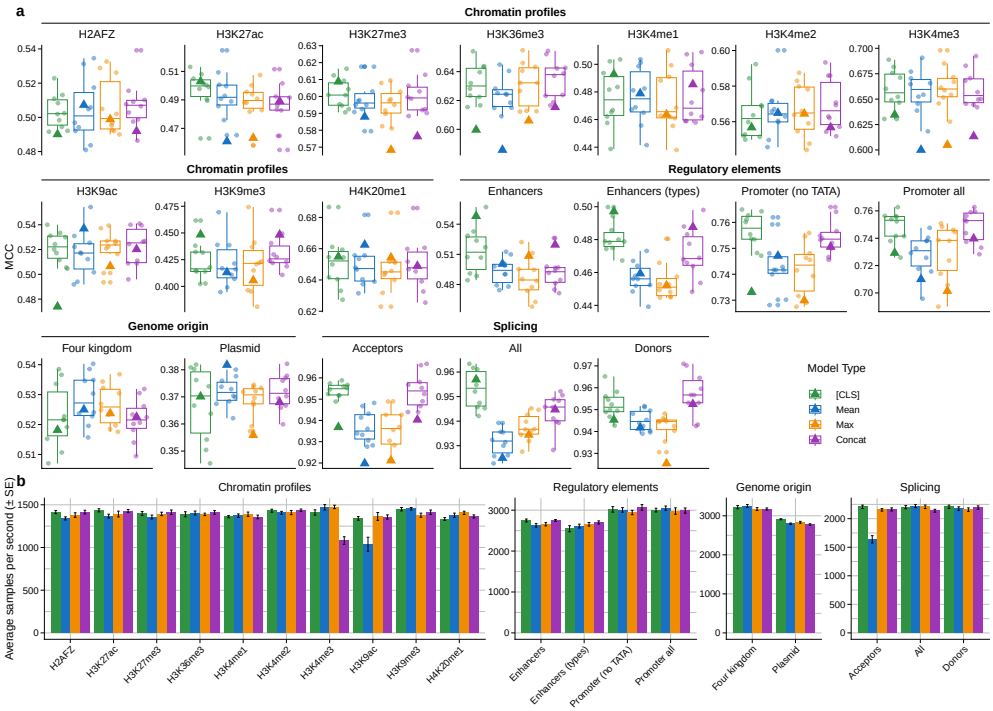

Fig. 6: **Comparison of classification performance across different fine-tuning strategies. a**: The figure presents results for models using the `[CLS]` token, mean pooling (`Mean`), max pooling (`Max`), and the concatenated `CLS|Mean|Max` approach (`concat`). Boxplots are based on the validation scores (circles), and the triangle represents the models' performance on the test set. **b**: The figure compares the processing time of the models by comparing the average samples per second.

Fine-tuning enables the model to translate sequences with similar labels into similar vector spaces and provide task-specific embeddings. In contrast, the pretrained model generates vector representations in a more general format. These vectors are likely influenced by sequence similarity or other latent patterns in the DNA data. The summary vectors produced by the pretrained model aim to encode all potentially useful information, ensuring versatility across a wide range of downstream tasks, rather than focusing on a single specific task. To quantitatively measure this improvement, we employed several clustering metrics, including adjusted Rand index (ARI), adjusted mutual information (AMI), normalized information distance (NID), and normalized mutual information (NMI) (Figure 7c). In both datasets, the max pooling method consistently achieved the highest scores across all clustering metrics, indicating its superior ability to capture meaningful vector representations from input sequences. One notable observation is the difference between the datasets. In the `four kingdoms` dataset, we observed a detectable signal between the vectors of the pretrained model and the true kingdom labels, whereas no such signal was present for

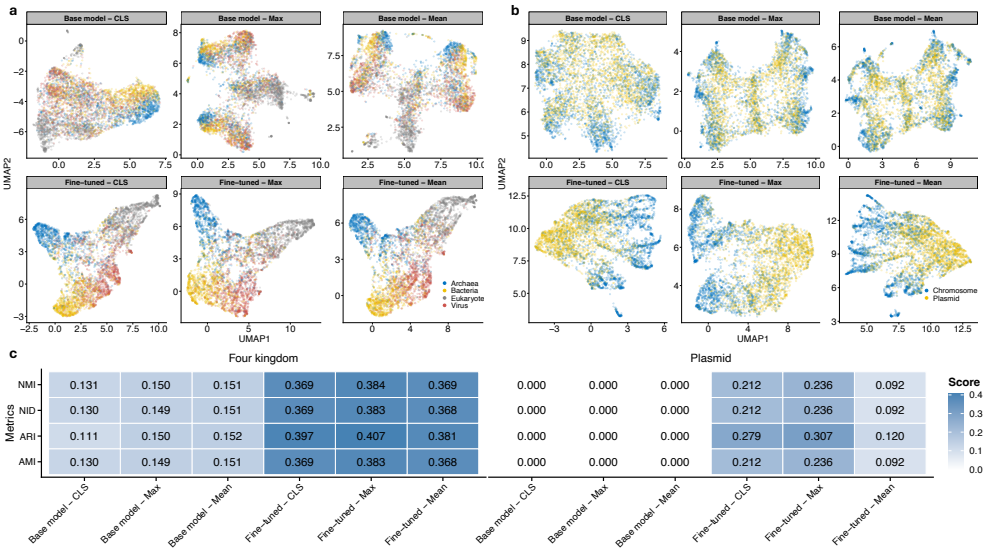

Fig. 7: **Analysis of vector representations from the `base` model on the `four kingdoms` and `plasmid detection` datasets.** UMAP projections of vector representations for pretrained and fine-tuned models, summarized using three methods: `[CLS]` token, mean pooling, and max pooling. Fine-tuning leads to clearer clustering and better separation in both datasets. **a**, results for the `four kingdoms` dataset; **b**, results for the `plasmid detection` dataset; `four kingdoms` Clustering metric scores, adjusted rand index (ARI), adjusted mutual information (AMI), normalized information distance (NID), and normalized mutual information (NMI) for each summarization method on pretrained and fine-tuned models. Max pooling achieves the highest scores uniformly, indicating its superior ability to capture meaningful sequence representations.

the `plasmid detection` dataset (Figure 7c). This suggests that the general understanding of genomes enables the model to translate genomes from different kingdoms into distinct vector spaces, but this representation does not discriminate between plasmid samples from bacterial chromosome samples. Additionally, in this case, the mean pooling approach outperformed the other methods, further emphasizing the impact of pooling strategies on representation quality (Figure 7c).

2.8 Domain-adaptation boosts task-specific model performance

In most cases of working with annotated sequences, the challenge of limited sample sizes can hinder the implementation of predictive models [41]. This is because such models typically require large datasets to learn hidden patterns within the data effectively and to fine-tune their internal weights. A promising solution to this issue is to adapt the model to the domain through continual pretraining before fine-tuning it for the specific downstream task at hand [42, 43]. To investigate the impact of continual domain-adaptive pretraining (DAP-training), we selected a model

(seqLens_4096_512_46M we call it 46M model here) that had been pretrained. We then trained this model with an additional 150K training steps with two different datasets and the pretraining dataset. The datasets used for DAP-training were as follows: a prokaryote-only genome (Mp), a eukaryote-only genome (Me), and the same general-purpose training dataset (Ms). Further details about these datasets and training are provided in subsection 8.10. After performing DAP-training, we evaluated the models on the same benchmarking datasets as in subsection 2.2. These benchmarking datasets include a set of human-only genome tasks, a bacterial genome (focused on plasmid detection) task, and a general multi-species task (four kingdoms). With this setup, we investigate the effect of exposing a pretrained model to different domains.

The performance of Me is generally higher than that of the 46M model (13 tasks) and Ms (12 tasks). However, Me is outperformed in the general tasks of four kingdoms and plasmid detection (Figure 8a). This difference becomes even more pronounced when comparing Me with Mp, emphasizing the advantages of using a eukaryote-specific dataset for pretraining in human-centric tasks. For the four kingdoms and plasmid detection tasks, Me consistently performs worse than all three other models (Figure 8). Overall, Me achieves the best test results in 11 of 19 tasks compared to the other models. These findings suggest that domain adaptation to eukaryotic genomes while improving the domain-specific performance, does not generalize well to tasks involving broader taxonomic diversity or prokaryote-specific features (see Figure 8a).

The performance of Mp, as expected, is generally lower than that of the 46M model and Ms in the human-only genome and four kingdoms tasks (Figure 8a). This outcome reflects the limitations of adapting the model specifically to prokaryote-only data for these tasks. However, Mp shows a notable exception in the plasmid detection task, where it outperforms the 46M model in both validation and test sets and surpasses Ms in the test set. This result highlights the utility of prokaryote-specific domain adaptation for tasks that closely align with its training data (see Figure 8a). The performance of Ms surpasses the 46M model in 12 tasks. This indicates that continual pretraining on the same general-purpose dataset (Ms) enhances the base model's capabilities without compromising its versatility across diverse tasks (see Figure 8a).

We also examined the token embeddings in the models and observed that the UMAP representations of token embeddings align with token lengths (Figure 8b). This suggests that token length plays a role in how embeddings are structured within the models and that models were able to detect it through training. Additionally, an analysis of the Euclidean distances between token embeddings of models from the 46M model revealed that Me has the largest distance from 46M, while Ms exhibits the most similar token embeddings to 46M (Figure 8c). This suggests that exposing the model to a completely different domain dataset causes the largest shift in token embeddings compared to other scenarios.

The observed performance trends highlight the critical importance of selecting appropriate datasets for domain-adaptive pretraining (DAP-training). The superior performance of Me in human-only genome tasks demonstrates the benefits of aligning the pretraining dataset with the target domain. However, its poor performance in other tasks underscores the potential risks of overfitting to a narrow domain, resulting in reduced generalization capabilities. Similarly, Mp excels in the plasmid detection

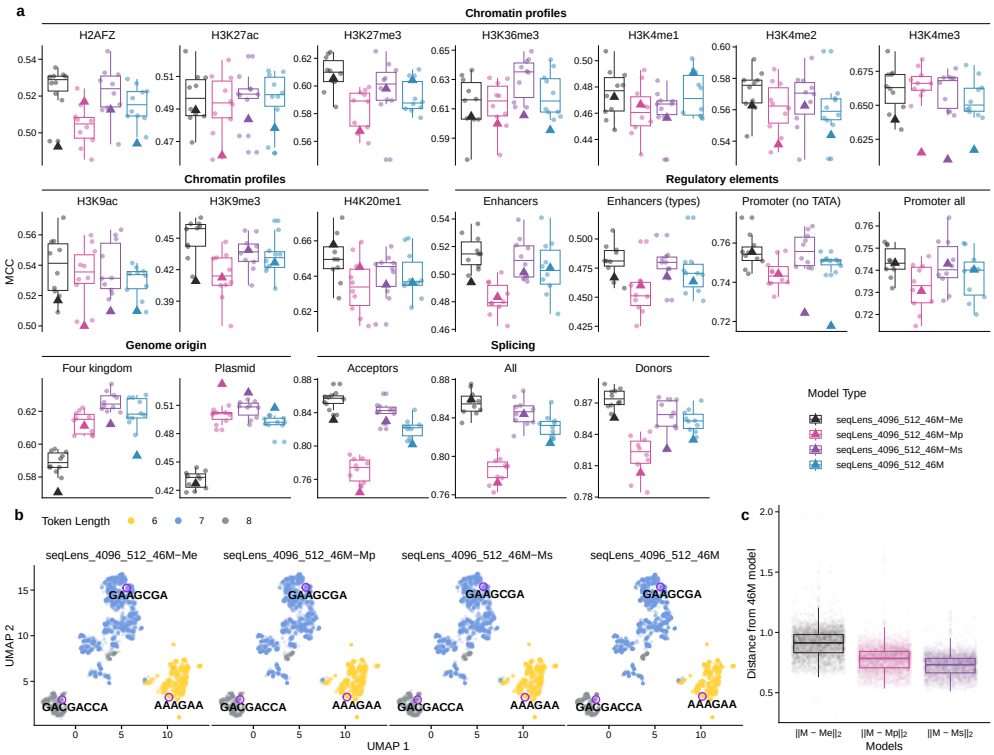

Fig. 8: **Effect of Domain-Adaptive Pretraining (DAP-training) on Model Performance Across Tasks. a**: The `base-model` underwent an additional 150K training steps with three datasets: prokaryote-only genome (`Mp`), eukaryote-only genome (`Me`), and the original general-purpose training dataset (`Ms`). The resulting models were evaluated on benchmarking datasets. Boxplots are based on the validation scores (circles), and the triangle represents the models' performance on the test set. s **b**: The figure compares the 2d representation of token embeddings of models. Three sample tokens were marked to highlight the changes in representations. **c**: The distribution of token embedding distances from the `base-model`.

task, showcasing the value of domain specificity for niche applications. However, its limited effectiveness in broader or unrelated tasks highlights the inherent trade-offs between specialization and generalization. In contrast, `Ms` achieves a balance between specialization and versatility. Continuing to pretrain on the original diverse dataset consolidates the base model's strengths while preserving adaptability across various tasks. These findings suggest that while DAP-training can significantly enhance performance in domain-specific tasks, it must be carefully tailored to mitigate overfitting and maintain generalization for diverse downstream applications. Overall, the results illustrate the significant potential of DAP-training to enable models to adapt effectively to niche applications where annotated data is limited. By leveraging generic domain data, DAP-training enhances the model's capacity to capture relevant patterns

and improve performance on targeted tasks. These findings underscore the flexibility of pretrained models to be fine-tuned for specific requirements, demonstrating that carefully curated domain data can serve as a valuable supplement when task-specific data is scarce.

3 Discussion

DNA sequences can be conceptualized as a complex language, rich with unseen patterns that hold the key to understanding fundamental biological insights [1–3]. Recent advancements in language modeling technology have revolutionized the ability to decipher these hidden patterns, offering faster, more efficient, and versatile methods compared to traditional alignment-based approaches [2, 3, 20]. Unlike conventional techniques that focus primarily on individual regions of DNA, language models provide a novel framework for exploring intricate interactions among regions, uncovering a deeper layer of genomic complexity. This paradigm shift aligns with the sequential nature of many biological techniques, such as mass spectrometry for protein analysis, suggesting that language models could play a transformative role in advancing our understanding of genomics [26, 27].

In this study, we investigated the impact of key features of language models on their performance in downstream genomic tasks. Our findings highlight several significant aspects that can enhance the design and utility of DNA language models. We demonstrated that a dynamic tokenizer, such as BPE, outperforms fixed-length k-mer-based tokenizers. Dynamic tokenization provides flexibility in capturing meaningful patterns and enables models to converge faster during training while achieving lower loss values [3]. This adaptability makes it a preferable choice for genomic language models. We also demonstrated that the BPE tokenizer processes text significantly faster than the native Python k-mer tokenizer. This speed-up can be attributed to the enhanced implementation efficiency of the BPE tokenizer, which leverages a high-performance library developed in Rust [44]. The Rust-based implementation is optimized for speed, enabling rapid tokenization and contributing to improved model efficiency. Despite the higher TFLOP requirements, this accelerated tokenization process likely plays a role in the superior performance observed with BPE-based models.

We also explored the DeBERTa-v2 architecture, particularly its implementation of disentangled attention, and compared it to equivalent models with the same parameter sizes. Our results show that architectures with disentangled attention consistently outperform their counterparts in most tasks and indicate a strong potential for scalability and effectiveness in larger implementations. We also investigated models with the same architecture but different vocabulary sizes. Our analysis revealed that models with smaller tokenizers tend to generalize better in downstream tasks. While a larger vocabulary can capture more nuanced linguistic patterns by introducing more distinct tokens, it also demands greater computational resources. Additionally, with a larger vocabulary, each token is seen less frequently on average during training, which may limit the model's ability to learn robust representations [33].

Additionally, we evaluated methods for leveraging language models in classification tasks. The conventional approach of using the `[CLS]` token from the last hidden state

[36] does not consistently produce the best results. In fact, in more than half the cases, alternative methods such as mean pooling or concatenating the `[CLS]` token with mean and max pooling (`CLS|Mean|Max`) yield superior performance. This highlights the importance of exploring diverse strategies for summarizing model representations [37]. We also investigated LoRA as a parameter-efficient fine-tuning method and found that while it reduces the computational costs of fine-tuning, it comes at the expense of model performance on downstream tasks compared to full parameter training. While the LoRA approach is highly beneficial for large language models [30] in terms of computing efficiency, our results indicate that for relatively small language models, the computational gains are not as significant as full model training is still achievable on a single GPU. Instead, we observe a loss of information that impacts overall model effectiveness. Future work could explore optimizing LoRA's hyperparameters to minimize these performance gaps and better balance efficiency with accuracy.

Moreover, our findings highlight the importance of fine-tuning in refining vector representations for genomic data. While pretrained models provide general-purpose embeddings, fine-tuning enables the model to better capture task-specific distinctions, leading to improved clustering and separation of sequences. The differences observed between the `four kingdoms` and `plasmid detection` datasets suggest that pretrained embeddings inherently reflect broad genomic relationships but may lack the specificity needed for certain classification tasks. Additionally, the effectiveness of different pooling strategies underscores the need to carefully select summarization methods based on the nature of the task [38, 39].

Another critical aspect we investigated was the effect of continual pretraining. Our experiments reveal that even with limited exposure to domain-specific data, continual pretraining can significantly enhance model performance. This stage allows researchers to adapt language models to specific domains by providing generic domain data before fine-tuning for downstream tasks [43]. This adaptability underscores the practicality and value of language models in scenarios where annotated domain-specific data are scarce. Future research should aim to optimize this approach further, exploring strategies to balance specialization with generalization. This includes investigating pretraining methods [43], adaptive fine-tuning techniques [45], and multi-domain training [46] frameworks to maximize the utility of pretrained models across diverse and challenging domains.

Despite these advancements, there remains significant room for improving DNA language models. One promising avenue is the development of smarter tokenizers. A guided tokenizer that incorporates biological insight into its tokenization process could substantially enhance model performance. For example, a tokenizer capable of segmenting DNA sequences into biologically meaningful units or contextually relevant patterns may improve the model's ability to capture complex genomic information. Further research could also explore hybrid approaches that combine the strengths of different architectures or tokenization methods. Advances in pretraining strategies, such as multi-domain pretraining or biologically informed masking techniques [47], could further optimize DNA language models for diverse and challenging tasks.

4 Data availability

The `seqLens` models and the datasets we used in this study are available at https://huggingface.co/omicseye. The scripts that we used to get the data from the NCBI's website, data preparation, model training, benchmarking, and visualizations are also available at https://github.com/omicsEye/seqLens.

5 Acknowledgments

This work was supported by the National Science Foundation under grant no. 2109688 to A.R. and K.A.C. We thank Ali R. Taheriyoun for the feedback on the manuscript text, and Ajinkya Patil and Matthew Mollerus for their assistance with the applications data.

6 Author contributions

M.B. and A.R. conceived the study and designed the models. M.B., A.R., and K.A.C. developed the analytical research framework. A.R. and K.A.C. supervised the study's execution and guided data interpretation and result structuring. M.B. implemented and evaluated the models. All authors contributed to discussing the results. M.B. and A.R. wrote the manuscript. B.M. and K.A.C. reviewed and revised it, providing critical feedback that shaped the final version.

7 Competing interests

AR and KAC are inventors of a pending patent related to the work presented. The other authors declare no competing interests.

8 Methods

Developing advanced DNA language models requires a multifaceted approach encompassing several critical methodological components. In this manuscript, we present a comprehensive framework for creating and optimizing masked language models specifically tailored for DNA sequence analysis. Our methodology addresses key challenges in computational genomics through a systematic exploration of several interconnected research dimensions:

1. **Data Preparation and Curation**: We meticulously gathered and preprocessed diverse DNA sequence datasets to ensure robust and representative training and evaluation.
2. **Tokenization Strategies**: Recognizing the fundamental importance of sequence representation, we conducted an in-depth investigation of BPE tokenization, exploring various vocabulary sizes and their implications for model performance.
3. **Model Architecture Design**: We explored transformer-based architectures and assessed their capabilities for DNA language models.
4. **Vector Representation Techniques**: We explored different methods for representing DNA sequences as dense vector spaces, including:

- Examining different pooling strategies (CLS token, mean pooling, max pooling)
- Analyzing vector transformations using Uniform Manifold Approximation and Projection (UMAP)
- Quantitatively assessing representation quality through clustering metrics

5. **Fine-Tuning Methodologies**: We implemented sophisticated adaptation techniques to enhance model performance on specific downstream tasks:

- Low-Rank Adaptation (LoRA) for parameter-efficient fine-tuning [30]
- Domain adaptation through continual pretraining
- Innovative approaches to aggregating last hidden state vectors for classification tasks

In the subsequent sections, we provide a detailed exposition of each of these methodological components, offering insights into our approach to developing state-of-the-art DNA language models.

8.1 Training dataset

For pretraining and validation, we curated a dataset comprising all 19,551 representative genomes sourced from the National Center for Biotechnology Information (NCBI) database by February 23, 2024, including 18,268 bacterial, 647 archaeal, 577 fungal, 40 viral, and 1 human reference genome. Then, to create samples, we created chunks of 3200 long nucleotide sequences with 100 bases overlapping. The resulting train set has 27,831,882 samples, the validation set has 3,478,985, and the test set has 3,478,986 samples.

For the second dataset, which is a more diverse dataset in terms of the tree of life, we extended the multi-species pretraining dataset that was collected in the Nucleotide Transformers [20] study and extended it with 499 archaea genomes and 5 eukaryotic genomes (see section 4).

8.2 Tokenization

Tokenization is the first step in language modeling [31]. In language models, a given text sequence is split into tokens (tokenization), and then the model processes these tokens as data units. In natural language, these tokens can be sequences of characters, including complete words, punctuation marks, or even subwords. However, there are no specific words or separators in analyzing DNA sequences, so not all the natural language approaches are applicable. We used BPE [29] as a tokenization method to optimize sequence encoding. This method is more robust to small changes in DNA sequences, as reads can come from different parts of a sequence or mutations can happen only on a single base pair [3]. In the DNABERT-2 study [3], the authors provide examples of the superiority of this method to the traditional k-mer approach.

We trained 5 different tokenizers with vocabulary sizes of 4096, 8192, 16384, 32768, and 65536. The corpus consists of 1M samples from the training dataset. Since the tokenizers have different sets of tokens and more vocabulary leads to longer tokens, tokenization results in a sequence of tokens of different lengths.

8.3 Model architecture

8.3.1 Masked language modeling

Masked language modeling (MLM) is a technique used in natural language processing to train language models by predicting the most likely tokens at masked positions within a sequence [25, 36]. In MLM, a portion of the input tokens is replaced with a special [MASK] token, and the model is tasked with predicting the original tokens based on the context provided by the remaining, unmasked tokens. For example, given an input training sequence of n tokens $S = \{t_1, \ldots, t_n\}$, 15% of these tokens are randomly selected and masked. The training objective of the model M is to predict these masked tokens. Each prediction yields a probability distribution over the model's vocabularies of the model and the training objective is to minimize the cross-entropy loss between the predicted probability distribution and the true distribution of the masked tokens. This approach allows the model to learn bidirectional sequence representations, capturing dependencies between tokens in both forward and backward directions [25, 36]. MLM has proven particularly effective in domains where sequential data is prevalent, such as protein research, where sequences of amino acids are treated as words, and genomics, where sequences of nucleotides can similarly be processed as sentences composed of k-mers [20, 26, 27]. By leveraging MLM, language models can generate rich, context-aware embeddings that significantly enhance the understanding and analysis of complex biological sequences [3, 20].

Given a training sequence $S = \{t_1, \ldots, t_n\}$, let $\mathcal{M} \subset \{1, \ldots, n\}$ denote the set of indices of the masked tokens, representing 15% of the sequence. The model M predicts the probability distribution over the vocabulary for each masked token t_i, where $i \in \mathcal{M}$:

$$P(t_i \mid S_{\setminus \mathcal{M}}) = M(S_{\setminus \mathcal{M}}), \quad \text{for } i \in \mathcal{M}$$

Here, $S_{\setminus \mathcal{M}}$ is the input sequence with the masked tokens replaced by special mask tokens. The training objective is to minimize the cross-entropy loss between the predicted probability distribution $P(t_i \mid S_{\setminus \mathcal{M}})$ and the true distribution t_i for each masked token:

$$\mathcal{CE} = -\sum_{i \in \mathcal{M}} \log P(t_i \mid S_{\setminus \mathcal{M}})$$

This loss function sums over the *log* probabilities of the true tokens at each masked position, which the model attempts to maximize.

8.3.2 Language model architectures

For the architectures of the base model, we selected Decoding-enhanced BERT with disentangled attention (DeBERTa-v2) model architecture [25] and Evolutionary Scale

Modeling (ESM) with rotary positional encoding [28]. DeBERTa-v2 is a state-of-the-art model developed by Microsoft that uses a disentangled attention mechanism and enhanced masked decoder. DeBERTa introduces a disentangled attention mechanism that separates the attention to content and position information. Unlike BERT, which represents each word with a single vector combining content and position embeddings, DeBERTa uses two separate vectors to encode a word's content and position. The attention weights are calculated using disentangled matrices based on content and relative positions of words, reflecting that word (token) dependencies rely on both content and proximity. This mechanism captures the stronger dependency between closely related words (tokens) compared to words in different contexts and better understands the input's positional relationships and content dependencies. The enhanced mask decoder in DeBERTa-v2 also tries to improve the training efficiency and model performance. It allows the model to predict masked tokens during the pretraining phase better. Also, DeBERTa-v2 employs parameter-sharing techniques across different layers of the network. This reduces the overall number of parameters, making the model more efficient while maintaining high performance.

In DeBERTa, the calculation of the cross-attention score between tokens i and j involves both their content and relative positional information. Specifically, H_i and $P_{i|j}$ represent the content and the relative positional embedding of token i with respect to token j, while H_j and $P_{j|i}$ represent the corresponding features for token j. The cross-attention score between tokens i and j is decomposed into four components as follows:

$$\begin{aligned} A_{i,j} &= \{H_i, P_{i|j}\} \times \{H_j, P_{j|i}\}^\intercal \\ A_{i,j} &= H_i H_j^\intercal + H_i P_{j|i}^\intercal + P_{i|j} H_j^\intercal + P_{i|j} P_{j|i}^\intercal \end{aligned} \tag{1}$$

This decomposition consists of four components:

1. **Content-to-Content:** $H_i H_j^\intercal$, which captures the interaction between the content features of tokens i and j.
2. **Content-to-Position:** $H_i P_{j|i}^\intercal$, which models how the content of token i interacts with the relative position of token j.
3. **Position-to-Content:** $P_{i|j} H_j^\intercal$, which captures the interaction between the relative position of token i and the content of token j.
4. **Position-to-Position:** $P_{i|j} P_{j|i}^\intercal$, which encodes the interaction between the relative positional features of tokens i and j.

These four components work together to provide a richer representation of the relationships between tokens by incorporating both their content and relative positions [25].

Relative position encoding can be implemented by using only the *Content-to-Content* and *Content-to-Position* terms from Equation 1. However, as proposed in DeBERTa study [25], including the *Position-to-Content* term is helpful to the model for fully capturing the interactions between token content and their relative positions,

ensuring a more comprehensive modeling of attention weights. The same study proposed that with the first three terms of Equation 1, there is no need to include the fourth term [25].

The ESM models have two versions, and we are implementing the architecture of the ESM-2. These models are mainly developed for protein sequences, variant effect prediction, and protein structure prediction, and their main difference from other BERT family models is the ways that these models pass the last layer of the BERT encoder block to folding or classification heads of the model [26, 27]. ESM-1 used a BERT-style transformer architecture with absolute positional encodings, which were either static or learned, to provide token position information [26]. In ESM-2, the architecture remains transformer-based but introduces Rotary Position Embeddings (RoPE) [48] instead of absolute positional encodings. RoPE allows the model to extrapolate beyond the context window it was trained on. While RoPE increases the computational cost by multiplying query and key vectors with sinusoidal embeddings, it improves model quality in smaller models [27]. However, it has been reported that the performance gains diminish as model size and training time increase [27]. Here we present a refined explanation with proper notation, dimensions, and structure:

The Nucleotide Transformers v2 models adopt the architecture of the ESM-2 model, with a key modification in the feed-forward layer. In the original ESM-2 implementation, the feed-forward layer is defined as follows [49]:

$$\begin{aligned} H_1 &= \mathrm{GeLU}(W_1 h + b_1) \\ H_2 &= \mathrm{Dropout}(W_2 H_1 + b_2) \end{aligned}$$

Where $W_1 \in \mathbb{R}^{\text{hidden_size} \times \text{intermediate_size}}$, $b_1 \in \mathbb{R}^{\text{intermediate_size}}$, $W_2 \in \mathbb{R}^{\text{intermediate_size} \times \text{hidden_size}}$, $b_2 \in \mathbb{R}^{\text{hidden_size}}$, $h \in \mathbb{R}^{\text{hidden_size}}$ represents the input to the feed-forward layer, and H_2 is the output. This design enables intermediate feature transformation through a higher-dimensional space defined by the *intermediate_size* parameter, enhancing the model's capacity to learn complex representations. The first transformation applies a linear projection followed by a *GeLU* activation and the second transformation applies another linear projection with a dropout layer. However, in the Nucleotide Transformers v2 feed-forward layer is as follows [24]:

1. A linear transformation:

$$H_{\text{dense}} = W_{\text{dense}} h + b_{\text{dense}}$$

where $W_{\text{dense}} \in \mathbb{R}^{\text{hidden_size} \times 2 \cdot \text{intermediate_size}}$, $b_{\text{dense}} \in \mathbb{R}^{2 \cdot \text{intermediate_size}}$.

2. Split the output to apply a gated linear unit (*GLU*) followed by a *SiLU* activation function:

$$H_{\text{dense}} = [H_1 \parallel H_2], \quad H_1, H_2 \in \mathbb{R}^{\text{intermediate_size}}$$

$$H_{\text{intermediate}} = \text{SiLU}(H_1) \odot H_2$$

where H_{dense} is split along the last dimension into two equal parts, H_1 and H_2 and $\odot$ represents element-wise multiplication.

3. The $H_{\text{intermediate}}$ is then passed through a linear layer with dropout and a residual connection to h. In this paper, the models based on the ESM architecture share the same implementation as the Nucleotide Transformers v2 in terms of the model architecture.

Similar to BERT, DeBERTa, and ESM both use MLM for pretraining, predicting masked words based on their surrounding context. For codes and available trained models, please see the data availability section.

Since we are sharing the `seqLens` model weights, the following naming convention was adopted in this study: `seqLens_TOKENIZER-SIZE_CONTEXT-LENGTH_MODEL-SIZE`.

- Models that implement disentangled attention include a `-at` suffix.
- Models derived directly from the DeBERTa-v2 architecture (`xsmall`, `small`, and `base`) retain their size as a suffix in the model ID.
- Models based on the `ESM` approach include `esm` in their IDs.

This consistent naming convention ensures clarity, traceability, and reproducibility when sharing and referencing the model weights.

8.4 Model training

We initialized the model weights for each DeBERTa configuration, exploring various combinations of hidden sizes, pooler sizes, attention heads, number of layers, intermediate sizes (feed-forward layers), learning rates, and batch sizes. To evaluate the impact of relative positional encoding, we trained the model under two configurations for some setups: one using the first three terms of Equation 1 with relative positional encoding and the other without relative positional encoding and relative position attention. This resulted in the investigation of 48 models with parameters ranging from 15M to 111M. Full details of the model configurations are provided in Appendix B. We also trained the `seqLens_4096_512_89M-at-base-multi` model, using the second pretraining dataset with context length 512, batch size 1024, and learning rate of 1e-4. We trained this model for 300K which lead to roughly 150B training tokens. For the ESM models, we adhered to the architectures of the Nucleotide transformers v2 50M and 100M models, using the 4096 tokenizer and max input length of 512, batch size of 1024, and learning rate of 5e-4.

For each model (combinations of architecture, tokenizers, and parameters), we pretrained using MLM approach, where we sampled a few tokens in each sequence for masking with a probability of 15%. Of the masked tokens, 80% were replaced with the tokenizer's mask token ([MASK]), 10% were substituted with a random word, and the remaining 10% were left unchanged. The input size for each model was 512 tokens ([CLS] + 510 + [EOS]). Each model was scheduled for 150,000 steps of training, though some failed to converge due to hardware issues, inadequate learning

rates, or other unknown reasons Appendix B. We used the AdamW optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and a weight decay of 0.1. We applied L_2 regularization only to the attention layers to improve the training process. Each training step consisted of 512, 1024, or 2048 samples (see Appendix B), resulting in approximately 768 million samples (about 3 epochs, batch size of 512) in total. We used a cosine learning rate scheduler with 5% warm-up steps (7,500 steps) and learning rates of 5e-4, 1e-4, and 5e-5. For training, we utilized a cluster of 8 Nvidia Tesla V100 SXM2 16GB GPUs (see data and code availability section). To load the training data, we used a buffer size of five times the accumulation step to avoid repeated data and introduce more randomization into the training process. We evaluated the model after each 1000 steps with 800 batches of the validation set.

8.5 Effect of BPE tokenizer

The Nucleotide Transformers v2 models utilize a *6*-mer tokenization approach [20]. To evaluate the impact of this tokenization method on model performance, we re-initialized the Nucleotide Transformers multi-species v2 models, the `v2-50M` and `v2-100M`, and trained them from scratch using two different tokenizers: our custom-trained tokenizer with a vocabulary size of 4096 and the original tokenizer from the Nucleotide Transformers study. For training, we used a batch size of 512 and maintained the same training and validation datasets, and hyperparameters as previously outlined (see subsection 8.1). We monitored training loss, evaluation loss, and model performance in terms of computational efficiency, specifically measuring the time per iteration.

8.6 Benchmarking datasets

8.6.1 Genome benchmark dataset

The *nucleotide_transformer_downstream_tasks* dataset represents a comprehensive genomics benchmark comprising 18 downstream tasks, exclusively derived from human samples [50]. The dataset features a mix of binary and multi-class classification tasks, carefully curated from high-quality sources. It offers diverse genomic element classifications across various scales, ranging from 300bp to 1kb sequence lengths, covering histone marks, enhancer elements, promoter regions, and splice sites. We are using the revised version of the dataset that addresses previous methodological limitations by replacing randomly generated negative samples with existing genomic sequences and implementing proper chromosome held-out test sets. The tasks include classifications of histone modifications, enhancer types (tissue-specific and tissue-invariant), promoter elements (with and without TATA-box motifs), and splice site annotations, providing a robust and consistent framework for genomic machine learning research.

8.6.2 Genome origin datasets

To extend the benchmarking beyond the human genome, we prepared two additional datasets: one for plasmid/chromosome detection and another for the classification of short reads into four categories: prokaryote, eukaryote, archaea and viruses.

The `plasmid detection` dataset consists of a balanced sample of 100,000 sequences, split into 95,000 for training and 5,000 for testing. To generate these reads, we utilized the plasmid and chromosome database introduced in the DeepPlasmid study [51], which can be accessed here.

For simulating reads from the contigs, we employed the `SimLoRD` read simulation tool [52] with default parameters and a target read length of 300 base pairs (bp). The command used was:

```
simlord --read-reference "$input_file" \
        -n "$number_of_samples" \
        -fl "$read_length" \
        out/"$output_file" \
        --no-sam --without-ns
```

The `four kingdoms` dataset focuses on classifying short reads as prokaryote, eukaryote, archaea, and virus. For this dataset, we simulated reads using reference genomes of representative species from each domain, as well as the human reference genome, to ensure coverage of diverse genomic structures. The reads were simulated with a length of 150 bp using the `ART` read simulator [53]. The simulation command was:

```
art -ss HS25 -i "$input_genome" -o "$output_simulated" \
    -l 150 -f "$fold_coverage" -na --rndSeed 1 \
    --quiet -p -m 400 -s 60
```

This dataset is also balanced, containing a total of 80,000 samples. We divided the data into 75,000 samples for training and 5,000 for testing.

8.7 Benchmark

To perform extensive benchmarking, we conducted a 10-fold cross-validation using the training set of each dataset. To ensure reproducibility and consistent fold generation, we set a random seed to control for equal comparison. We reported the best performance for each fold based on the validation Matthews Correlation Coefficient (`MCC`). After completing the cross-validation, the model with the best validation performance was selected to evaluate on the test set. In addition to accuracy metrics, we also measured the evaluation speed in terms of "samples per second" and "steps per second" to compare the computational efficiency of the models.

For the benchmarking, we utilized two Nucleotide Transformers multi-species models, `v2-50M` and `v2-100M`. This selection was driven by our computational resources and the fact that, at the time of initiating this project, these models demonstrated strong performance in benchmarking tasks performed by the developers, often surpassing other implementations. We also included two ESM models, which we trained using our training dataset and a custom tokenizer with a vocabulary size of 4096. For models based on the DeBERTa architecture, where we explored the impact of various

hyperparameters (such as learning rate, batch size, and the use of relative positional encoding), we selected the configuration with the lowest validation loss for inclusion in the benchmarking study. For instance, if we trained a model M_1 with two different learning rates, $lr \in \{lr_1, lr_2\}$, only the version with the lower validation loss was used in the final evaluation. Additionally, we included models with continual pretraining (see subsection 8.10) and models with various classification heads (see subsection 8.9) to assess the impact of these modifications on performance.

8.8 Low-rank adaptation

Low-Rank Adaptation (LoRA) is an efficient fine-tuning technique for large pretrained models [30]. Traditional fine-tuning involves updating all the parameters of a model, which can be computationally expensive and memory-intensive, especially for large-scale models. LoRA addresses this challenge by freezing the pretrained model's original weights and introducing small, trainable low-rank matrices to specific layers. During training, only these low-rank matrices are updated which results in a significant reduction of the number of trainable parameters while maintaining the model's performance on downstream tasks. LoRA is particularly well-suited for resource-constrained environments and applications requiring frequent fine-tuning across diverse tasks. It also reduces storage overhead since only the low-rank matrices need to be saved, making it easier to switch between tasks without retraining the entire model. Furthermore, by adding a negligible computational overhead, LoRA enables fine-tuning large models with limited hardware.

Here we investigated the effect of fine-tuning with LoRA by employing a rank of 8 and a scaling factor of 32. We targeted the query and value modules of the attention mechanism, as these are critical components for model performance and efficiency. For our experiments, we utilized two models: `NT-100` and `seqLens_4096_512_89M-at-base`. After integrating LoRA into these models, we adopted the same benchmarking setup described in subsection 8.7, ensuring consistency in evaluation metrics, datasets, and hyperparameter configurations. This allowed us to systematically assess the impact of LoRA fine-tuning on task performance, scalability, and computational efficiency.

8.9 Effect of different classification heads

For this analysis, we selected the `NT-50` model. In classification tasks, the typical approach involves using the `[CLS]` token from the final layer of the transformer as a summary representation of the input sequence. This token is then passed to a classification head, which performs a mapping from the hidden size d to the number of classes.

To explore the impact of different classification strategies, we implemented three different classification heads:

Mean pooling: In this approach, we compute the average of all non-padding token embeddings in the final layer. Let $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n\}$ be the set of hidden state vectors for each token, where $\mathbf{h}_i \in \mathbb{R}^d$ and d is the hidden size, and n is the number of input tokens. The average pooling is defined as:

$$\mathbf{h}_{\text{mean}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

This mean vector $\mathbf{h}_{\text{mean}}$ is then fed into the classification head.

Max pooling: Here, we take the maximum value across all non-padding token embeddings in the final layer for each hidden dimension. The max pooling operation is defined as:

$$\mathbf{h}_{\text{max}} = \max_{i=1}^{n}\mathbf{h}_i$$

where the maximum is computed element-wise across the hidden size dimensions. The resulting vector $\mathbf{h}_{\text{max}}$ is used as the input to the classification head.

Concatenated Head (`[CLS]` — Mean — Max): In this approach, we concatenate three different representations—the `[CLS]` token, the average-pooled vector (excluding the `[CLS]` token), and the max-pooled vector—into a single vector (`Mean` and `Max` are based on non-padding tokens). If $\mathbf{h}_{\text{CLS}}$ is the hidden state of the `[CLS]` token, the combined input to the classification head is:

$$\mathbf{h}_{\text{concat}} = [\mathbf{h}_{\text{CLS}}; \mathbf{h}_{\text{mean}}; \mathbf{h}_{\text{max}}] \in \mathbb{R}^{3d}$$

where $[\cdot;\cdot;\cdot]$ denotes vector concatenation. In this case, the input size to the classification head is $3 \times d$, accounting for the combined dimensions.

Each of these classification heads was implemented to assess its impact on the performance of the model during benchmarking. The results were then compared to understand how the choice of classification head influences the model's classification performance metrics (see subsection 8.7).

8.10 Continual domain adaptive pretraining

To measure the effect of continual pretraining, we selected one of the models, `seqLens_4096_512_46M`, and continued pretraining as follows. We began by loading the pretrained weights while re-initializing the cosine learning rate scheduler, treating it as if the training was starting from scratch.

To evaluate the impact of this continual pretraining on model performance in benchmarking tasks, we generated two datasets using `SimLoRD` [52]. The first dataset consisted exclusively of eukaryotic samples, comprising 600K samples from the human genome, 600K samples from mouse genome (*Mus musculus*), and samples from fungal genomes (a total of 7,280,000 training samples). The second dataset contained only prokaryotic and archaeal samples, including samples 20M samples from 2000 bacterial and 1M samples 100 archaeal species (a total of 20,950,000 samples).

We continued pretraining the selected model on these two datasets for 150K steps and also conducted another continuation of pretraining where we used the original training dataset that the model was trained on. In this second scenario, the learning rate scheduler was re-initialized at the start of the process, but otherwise, the training method remained unchanged. Since the learning rate is initialized, we have two separate 150K steps of training, and this is different from training with 300K steps.

After completing training, we included the three resulting models in the benchmarking analysis (see subsection 8.7).

References

[1] Mock, F., Kretschmer, F., Kriese, A., Böcker, S. & Marz, M. Taxonomic classification of DNA sequences beyond sequence similarity using deep neural networks. *Proc. Natl. Acad. Sci. U. S. A.* **119**, e2122636119 (2022).

[2] Ji, Y., Zhou, Z., Liu, H. & Davuluri, R. V. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics* **37**, 2112–2120 (2021).

[3] Zhou, Z. *et al.* DNABERT-2: Efficient foundation model and benchmark for multi-species genome. *arXiv [q-bio.GN]* (2023).

[4] Liu, S. *et al.* Analysis of metagenomic data. *Nat. Rev. Methods Primers* **5** (2025).

[5] Churko, J. M., Mantalas, G. L., Snyder, M. P. & Wu, J. C. Overview of high throughput sequencing technologies to elucidate molecular pathways in cardiovascular diseases. *Circ. Res.* **112**, 1613–1623 (2013).

[6] Reuter, J. A., Spacek, D. V. & Snyder, M. P. High-throughput sequencing technologies. *Molecular cell* **58**, 586–597 (2015).

[7] Horner, D. S. *et al.* Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Briefings in bioinformatics* **11**, 181–197 (2010).

[8] Mardis, E. R. Next-generation dna sequencing methods. *Annu. Rev. Genomics Hum. Genet.* **9**, 387–402 (2008).

[9] Bahram, M. *et al.* Metagenomic assessment of the global diversity and distribution of bacteria and fungi. *Environ. Microbiol.* **23**, 316–326 (2021).

[10] Moore, J. H., Asselbergs, F. W. & Williams, S. M. Bioinformatics challenges for genome-wide association studies. *Bioinformatics* **26**, 445–455 (2010).

[11] Yang, A. *et al.* Review on the application of machine learning algorithms in the sequence data mining of DNA. *Front Bioeng Biotechnol* **8**, 1032 (2020).

[12] Routhier, E. & Mozziconacci, J. Genomics enters the deep learning era. *PeerJ* **10**, e13613 (2022).

[13] Deep learning for genomics. *Nat. Genet.* **51**, 1 (2019).

[14] Albaradei, S. *et al.* Splice2deep: An ensemble of deep convolutional neural networks for improved splice site prediction in genomic dna. *Gene* **763**, 100035

(2020).

[15] Kuenzi, B. M. *et al.* Predicting drug response and synergy using a deep learning model of human cancer cells. *Cancer cell* **38**, 672–684 (2020).

[16] Liu, Q., Xia, F., Yin, Q. & Jiang, R. Chromatin accessibility prediction via a hybrid deep convolutional neural network. *Bioinformatics* **34**, 732–738 (2018).

[17] Zheng, A. *et al.* Deep neural networks identify sequence context features predictive of transcription factor binding. *Nature machine intelligence* **3**, 172–180 (2021).

[18] Liang, Q., Bible, P. W., Liu, Y., Zou, B. & Wei, L. DeepMicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genom Bioinform* **2**, lqaa009 (2020).

[19] Wichmann, A. *et al.* Metatransformer: deep metagenomic sequencing read classification using self-attention models. *NAR Genomics and Bioinformatics* **5**, lqad082 (2023).

[20] Dalla-Torre, H. *et al.* The nucleotide transformer: Building and evaluating robust foundation models for human genomics. *bioRxiv* (2023). URL https://www.biorxiv.org/content/early/2023/01/15/2023.01.11.523679.

[21] Sanabria, M., Hirsch, J., Joubert, P. M. & Poetsch, A. R. Dna language model grover learns sequence context in the human genome. *Nature Machine Intelligence* **6**, 911–923 (2024).

[22] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding (2019). URL https://arxiv.org/abs/1810.04805. 1810.04805.

[23] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units. *arXiv [cs.CL]* (2015).

[24] InstaDeepAI. Nucleotide transformer v2 - 50m multi-species model. https://huggingface.co/InstaDeepAI/nucleotide-transformer-v2-50m-multi-species (2024). Accessed: 2024-11-20.

[25] He, P., Liu, X., Gao, J. & Chen, W. Deberta: Decoding-enhanced bert with disentangled attention (2021). URL https://arxiv.org/abs/2006.03654. 2006.03654.

[26] Meier, J. *et al.* Language models enable zero-shot prediction of the effects of mutations on protein function. *bioRxiv* (2021). URL https://www.biorxiv.org/content/early/2021/11/17/2021.07.09.450648.

[27] Lin, Z. *et al.* Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv* (2022). URL https://www.biorxiv.org/content/early/2022/07/21/2022.07.20.500902.

[28] Su, J. *et al.* Roformer: Enhanced transformer with rotary position embedding (2023). URL https://arxiv.org/abs/2104.09864. 2104.09864.

[29] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units (2016). URL https://arxiv.org/abs/1508.07909. 1508.07909.

[30] Hu, E. J. *et al.* Lora: Low-rank adaptation of large language models (2021). URL https://arxiv.org/abs/2106.09685. 2106.09685.

[31] Webster, J. J. & Kit, C. Tokenization as the initial phase in NLP (1992).

[32] Brown, T. B. *et al.* Language models are few-shot learners (2020). URL https://arxiv.org/abs/2005.14165. 2005.14165.

[33] Dagan, G., Synnaeve, G. & Rozière, B. Getting the most out of your tokenizer for pre-training and domain adaptation (2024). URL https://arxiv.org/abs/2402.01035. 2402.01035.

[34] Prottasha, N. J. *et al.* Parameter-efficient fine-tuning of large language models using semantic knowledge tuning. *Sci. Rep.* **14**, 30667 (2024).

[35] Biderman, D. *et al.* Lora learns less and forgets less (2024). URL https://arxiv.org/abs/2405.09673. 2405.09673.

[36] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv [cs.CL]* (2018).

[37] Sun, C., Qiu, X., Xu, Y. & Huang, X. How to fine-tune bert for text classification? (2020). URL https://arxiv.org/abs/1905.05583. 1905.05583.

[38] Reimers, N. & Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks (2019). URL https://arxiv.org/abs/1908.10084. 1908.10084.

[39] Reimers, N. & Gurevych, I. Making monolingual sentence embeddings multilingual using knowledge distillation (2020). URL https://arxiv.org/abs/2004.09813. 2004.09813.

[40] McInnes, L., Healy, J., Saul, N. & Großberger, L. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software* **3**, 861 (2018). URL https://doi.org/10.21105/joss.00861.

[41] Chang, E. *et al.* Revisiting sample size determination in natural language understanding (2023). URL https://arxiv.org/abs/2307.00374. 2307.00374.

[42] Ke, Z. *et al.* Continual pre-training of language models (2023). URL https://arxiv.org/abs/2302.03241. 2302.03241.

[43] Gururangan, S. *et al.* Don't stop pretraining: Adapt language models to domains and tasks (2020). URL https://arxiv.org/abs/2004.10964. 2004.10964.

[44] Face, H. Tokenizers: Fast and customizable tokenization library (2023). URL https://github.com/huggingface/tokenizers. Accessed: November 26, 2024.

[45] Stollenwerk, F. Adaptive fine-tuning of transformer-based language models for named entity recognition (2022). URL https://arxiv.org/abs/2202.02617. 2202.02617.

[46] Schafhalter, P. *et al.* Scalable multi-domain adaptation of language models using modular experts (2024). URL https://arxiv.org/abs/2410.10181. 2410.10181.

[47] Althammer, S., Buckley, M., Hofstätter, S. & Hanbury, A. Linguistically informed masking for representation learning in the patent domain (2021). URL https://arxiv.org/abs/2106.05768. 2106.05768.

[48] Su, J. *et al.* RoFormer: Enhanced transformer with rotary position embedding. *arXiv [cs.CL]* (2021).

[49] Face, H. Transformers - esm model implementation. https://github.com/huggingface/transformers/blob/v4.46.3/src/transformers/models/esm/ (2024). Accessed: 2024-11-20.

[50] InstaDeep AI. Nucleotide transformer downstream tasks revised dataset. Hugging Face Dataset (2024). URL https://huggingface.co/datasets/InstaDeepAI/nucleotide_transformer_downstream_tasks_revised. Revised downstream tasks dataset for Nucleotide Transformer.

[51] Andreopoulos, W. B. *et al.* Deeplasmid: deep learning accurately separates plasmids from bacterial chromosomes. *Nucleic acids research* **50**, e17–e17 (2022).

[52] Stöcker, B. K., Köster, J. & Rahmann, S. Simlord: simulation of long read data. *Bioinformatics* **32**, 2704–2706 (2016).

[53] Huang, W., Li, L., Myers, J. R. & Marth, G. T. Art: a next-generation sequencing read simulator. *Bioinformatics* **28**, 593–594 (2012).

Appendix A Tokenizers

Here, we present the information on the trained tokenizers and the tokenization comparison study.

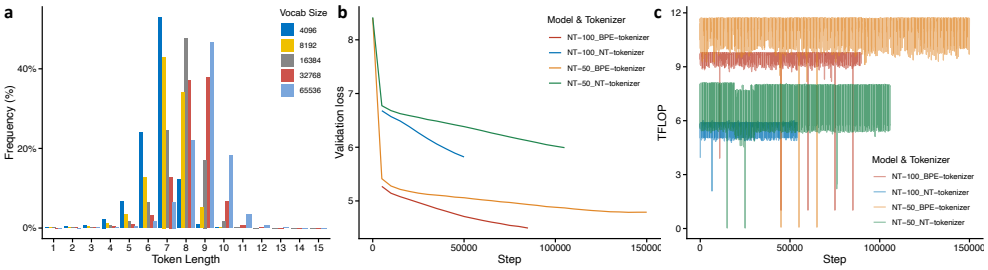

Fig. A1: **Performance and Efficiency Analysis of Tokenizers**. **a**, token frequency distribution across varying vocabulary sizes, showing the impact of different token lengths on representation; **b**, validation loss comparison between models using the BPE tokenizer and the fixed-length *6*-mer tokenizer, demonstrating faster convergence and improved performance with the BPE tokenizer; **c**, computational efficiency measured in TFLOP for the BPE and *6*-mer tokenizers, highlighting the higher computational demand of the BPE tokenizer due to its more sophisticated implementation.

Vocabulary Size	**Max Token Length (Frequency)**	**Most Frequent Token Length (Frequency)**	**Average Token Length**
4,096	10 (0.05%)	7 (52.93%)	4.9
8,192	20 (0.01%)	7 (42.98%)	5.3
16,384	21 (0.01%)	8 (47.55%)	5.6
32,768	25 (0.003%)	9 (37.88%)	6.0
65,536	40 (0.003%)	9 (46.54%)	6.4

Table A1: Summary of tokenization statistics across different vocabulary sizes, showing the maximum token length, the most frequent token length with their frequencies in the vocabulary set of the tokenizer, and average token length of tokenized inputs. For example, token length 7 is the most frequent length in the tokenizer with 4,096 vocabularies and this token length has 52.93% frequency in the vocabulary set. On average, the tokenized input lengths for the 4,096 tokenizer is 4.9 characters per token.

Appendix B Model pretraining

We initialized a set of models with varying parameter sizes for each tokenizer. For example, each tokenizer was used to create a model configuration with 4 layers, 4 attention heads, a hidden size of 512, and an intermediate size of 2048. The number

of layers, attention heads, hidden size, and intermediate size influence the architecture of a typical encoder model Figure B2a. Each model was trained for 150K steps using batch sizes of 256, 512, and 1024, learning rates of 5e-4, 1e-4, 5e-5, and 5e-1, and different attention mechanisms (Table B2). This resulted in training 48 different combinations of models and hyperparameters with DeBERT-v2 architecture. The runtime for the pretraining of each combination with Nvidia Tesla V100 SXM2 16GB GPUs took two to seven days.

Each model was trained using one or a subset of the combinations of the hyperparameters, as detailed in Table B2. We also investigated the `xsmall`, `small`, and `base` parameter sets that were introduced in the DeBERTa-v2 study [25]. Figure B2b shows the validation loss of these three setups with the 4096 tokenizer. The final validation loss at the end of training is closely tied to the model size, with larger models generally achieving lower losses. We validated loss for the `small` model with the 4096 tokenizer across different hyperparameter combinations (Figure B2c). Our results indicate that the best loss for the `small` model was achieved with a learning rate of 5e-4 and a batch size of 1024 samples, underscoring the importance of proper hyperparameter tuning. Figure B2d compares the training performance of two models: one with 15M parameters (left panel) and another with 46M parameters (right panel). For these models, we explored the effects of various learning rates, batch sizes, and the use of disentangled attention. Although the goal was not to comprehensively analyze the impact of disentangled attention, for these two specific models, the best setup was achieved using standard attention. However, for the `base` model with the 4096 tokenizer, the best performance was observed with the model employing disentangled attention (validation loss: 4.932), outperforming the equivalent setup without disentangled attention (validation loss: 5.287). The optimal setup for this model was a learning rate of 1e-4 and a batch size of 512. To build the `seqLens` models, we also investigated other configurations from the DeBERTa-v2 models. For a detailed list of models subsection 8.3 and Table B2). After training, the best-performing configuration for each of the model sizes (e.g., the 15M parameter model) was selected based on evaluation loss. These selected models then progressed to the next phase, which involved benchmarking their performance on downstream tasks (see subsection 8.4).

Table B2: Models parameters and pretraining loss results.

#Vocab-abularies	#Layers	#Attention heads	Intermediate size	Hidden size	#Para-meters	Relative position attention	#Steps	Learning rate	Batch size	Loss
4096	4	4	2048	512	15M	-	150K	5e-4	512	5.223
4096	4	4	2048	512	15M	C|P & P|C|C[1]	150K	5e-4	512	5.266
4096	4	4	2048	512	15M	C|P & P|C	150K	1e-4	512	5.364
4096	4	4	2048	512	15M	C|P & P|C	150K	5e-5	512	5.799
4096	8	8	2048	512	27M	-	150K	5e-4	512	5.138
4096	8	8	2048	512	27M	C|P & P|C	150K	1e-4	512	5.224
4096	8	8	2048	512	27M	C|P & P|C	150K	5e-5	512	5.805
4096	6	6	3072	768	46M	-	150K	5e-4	512	5.079
4096	6	6	3072	768	46M	C|P & P|C	150K	1e-4	1024	5.169
4096	6	6	3072	768	46M	C|P & P|C	150K	5e-5	256	5.430
4096	12	12	3072	768	89M	-	-	5e-4	512	FTC
4096	12	12	3072	768	89M	-	150K	1e-4	512	5.287
4096	12	12	3072	768	89M	-	150K	1e-4	1024	5.026
4096	12	12	3072	768	89M	C|P & P|C	-	1e-4	1024	5.026
4096	12	12	3072	768	89M	C|P & P|C	145K	1e-4	512	4.932
4096	12	12	3072	768	89M	C|P & P|C	70K	5e-5	1024	5.113
4096	8	8	4096	1024	106M	-	-	5e-4	1024	FTC
4096	8	8	4096	1024	106M	-	145K	1e-4	512	5.105
4096	12	6	1536	384	23M	C|P & P|C	150K	5e-4	256	5.080
4096	12	6	1536	384	23M	C|P & P|C	135K	5e-4	1024	5.018
4096	12	6	1536	384	23M	C|P & P|C	150K	5e-5	256	5.318
4096	6	12	3072	768	47M	C|P & P|C	145K	5e-4	512	4.984
4096	6	12	3072	768	47M	C|P & P|C	130K	5e-4	1024	4.941
4096	6	12	3072	768	47M	C|P & P|C	130K	5e-4	1024	4.941
4096	6	12	3072	768	47M	C|P & P|C	145K	1e-4	512	5.099
8192	4	4	2048	512	17M	-	150K	5e-4	512	5.646
8192	8	8	2048	512	30M	-	150K	5e-4	512	5.531
8192	6	6	3072	768	50M	-	150K	5e-4	512	5.549
8192	12	12	3072	768	92M	-	-	5e-4	512	FTC
8192	12	12	3072	768	92M	-	105K	1e-4	512	5.670
8192	12	12	3072	768	92M	-	105K	1e-5	1024	5.652
8192	8	8	4096	1024	111M	-	-	5e-4	512	FTC
8192	8	8	4096	1024	111M	-	145K	1e-4	512	5.503
8192	8	8	4096	1024	111M	-	145K	5e-5	512	5.334
8192	12	6	1536	384	25M	C|P & P|C	150K	5e-4	512	5.534
8192	12	6	1536	384	25M	C|P & P|C	150K	5e-5	512	5.731
16384	4	4	2048	512	22M	-	150K	5e-4	512	6.060
16384	8	8	2048	512	34M	-	150K	5e-4	512	5.958
16384	6	6	3072	768	56M	-	150K	5e-4	512	5.917
16384	12	6	1536	384	28M	C|P & P|C	150K	5e-4	512	5.944
16384	12	6	1536	384	28M	C|P & P|C	150K	5e-5	256	6.170
32768	4	4	2048	512	30M	-	150K	5e-4	512	6.446
32768	8	8	2048	512	43M	-	150K	5e-4	512	6.310
32768	6	6	3072	768	69M	-	150K	5e-4	512	6.303
32768	12	6	1536	384	34M	C|P & P	150K	5e-4	512	6.322
65536	4	4	2048	512	47M	-	150K	5e-4	512	6.876
65536	8	8	2048	512	59M	-	150K	5e-4	512	6.855
65536	6	6	3072	768	94M	-	150K	5e-4	512	6.593

1: C|P: Content-to-position, P|C: Position-to-content.

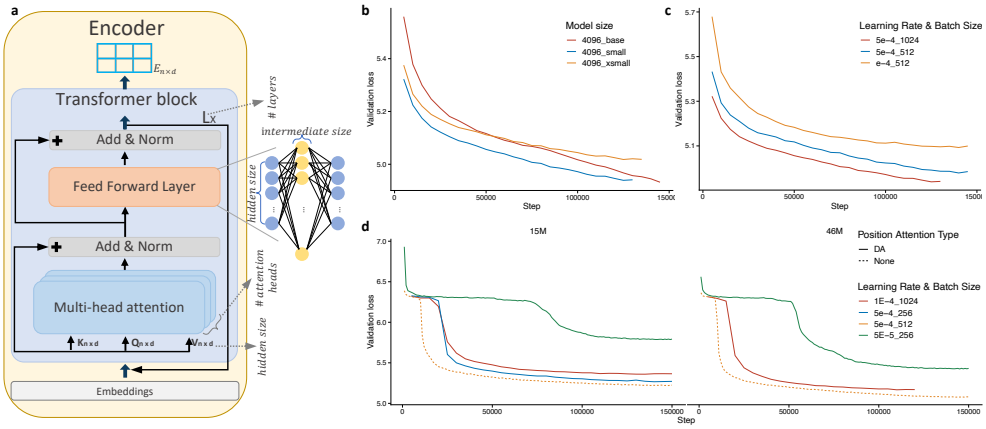

Fig. B2: **Training dynamics and hyperparameter impacts on model performance. a**, An overview of model architecture components, illustrating how the number of layers, attention heads, hidden size, and intermediate size influence the design; **b**, Validation loss curves for the `xsmall`, `small`, and `base` configurations of the DeBERTa-v2 model using the 4096 tokenizer, showing that larger models generally achieve lower losses; **c**, Validation loss curves for the `small` model with the 4096 tokenizer, highlighting that the best performance is achieved with a learning rate of 5e-4 and a batch size of 1024; **d**, Training loss comparisons for models with 15M parameters (left) and 46M parameters (right), including variations with disentangled attention (DA).

Appendix C Benchmarking results of models

Here, we present the results of model benchmarking on downstream tasks, including validation and test score comparisons across benchmarking datasets. These plots allow for a direct comparison of different architectures and an investigation into the impact of tokenizers with varying vocabulary sizes within the same architecture.

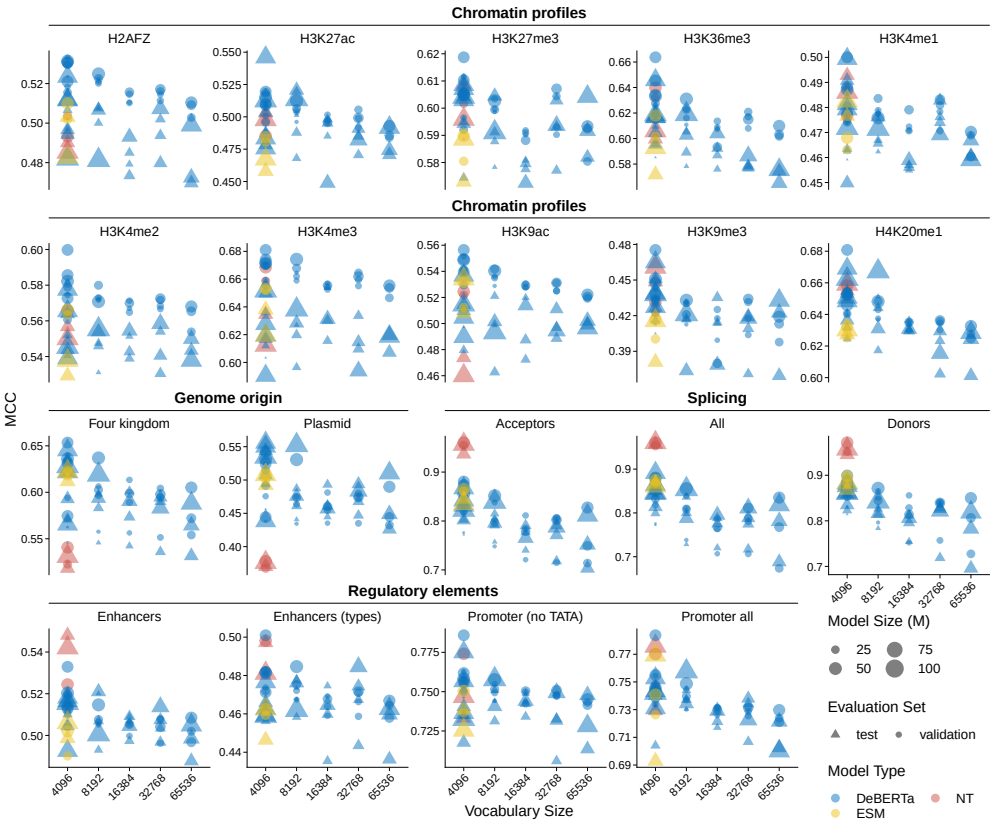

Fig. C3: **Validation and test scores of models across benchmarking datasets.** The figure compares the performance of models trained in this study (based on DeBERTa and ESM architectures) with Nucleotide Transformer models (`NT-50` and `NT-100`). Validation scores are the average of the scores for each fold. Results demonstrate no universally best-performing model, as smaller models occasionally outperform larger ones across datasets. A negative correlation is observed between tokenizer size and model performance, with larger vocabularies generally leading to lower performance. Models trained on bacterial genomes in this study perform better on the `four kingdoms` and `plasmid detection` datasets, while Nucleotide Transformer models, pretrained on datasets with greater eukaryotic representation, perform better on human genome-specific tasks such as chromatin profiles.

Figure C4 and Figure C5 illustrate a comparison between models of the same architecture using tokenizers with different vocabulary sizes. The benchmarking results indicate that, in most cases, models benefit from tokenizers with smaller vocabulary sizes, leading to improved performance.

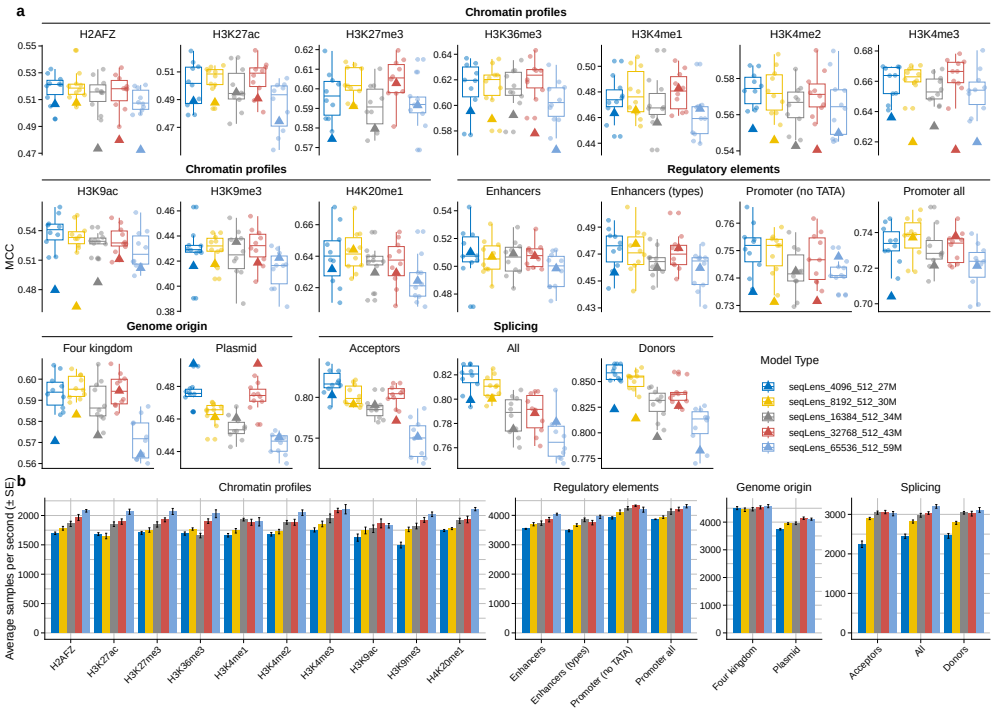

Fig. C4: **Validation and test scores of models across benchmarking datasets.** **a**: The figure compares the performance of the models with the same parameter configuration but different tokenizer vocabulary sizes. Boxplots are based on the validation scores (circles) and the triangle represents the models' performance on the test set. **b**: The figure compares the processing time of the models by comparing the average samples per second.

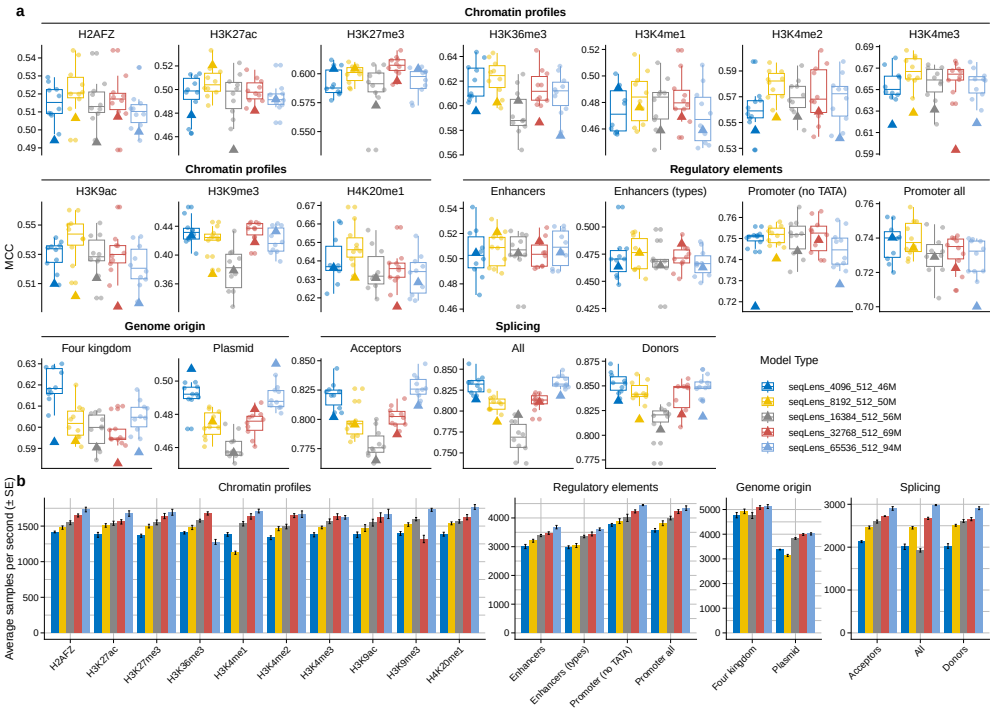

Fig. C5: **Validation and test scores of models across benchmarking datasets.** **a**: The figure compares the performance of the models with the same parameter configuration but different tokenizer vocabulary sizes. Boxplots are based on the validation scores (circles) and the triangle represents the models' performance on the test set. **b**: The figure compares the processing time of the models by comparing the average samples per second.

Appendix D Vector representation of DNA sequences

This appendix presents a detailed analysis of the vector representations generated by the models discussed in the main text. To explore the structure and properties of these representations, we conducted a Principal Component Analysis (PCA) on the output vectors and visualized the first two principal components (PCs) through pair plots. Each figure provides a comprehensive view of how these components interact and distribute for specific datasets and summarization methods.

The analysis covers both pretrained and fine-tuned models across different datasets (`four kingdoms` and `plasmid detection`) and summarization methods (`[CLS]` token, mean pooling, and max pooling). These visualizations help to uncover patterns, clustering behavior, and separability in the vector spaces, highlighting the impact of fine-tuning on embedding quality and task alignment.

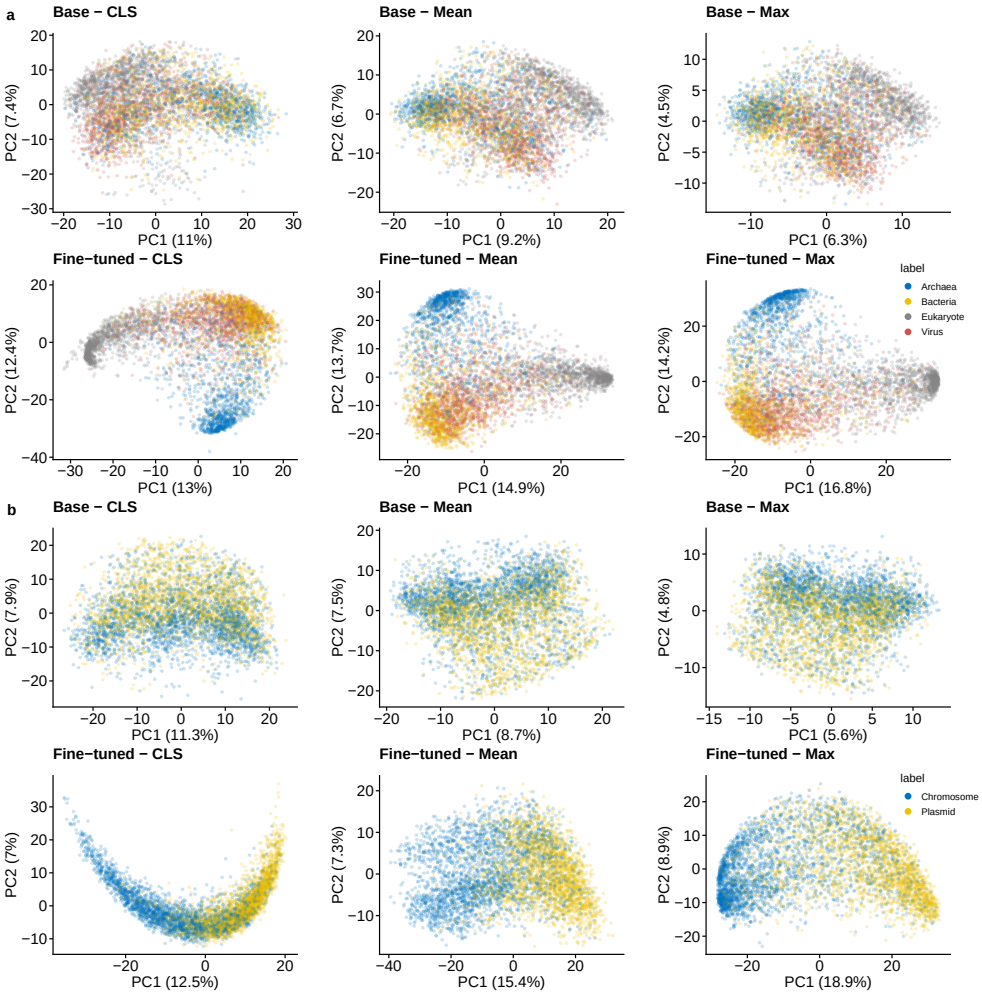

Fig. D6: Pair plot of the first two principal components (PCs) of the vector representations from the models on the `four kingdoms` and `plasmid detection` dataset. Each panel shows the interaction between two PC 1 and PC 2, revealing patterns and clustering in the embedding space.