Genetic Engineering Attribution with Transformer Models

Joely Nelson[1], Marc Exposit Goy[2]

1 Introduction

1.1 The problem

Genetic engineering has greatly advanced biotechnology, but also has potential for misuse. Although containment strategies mitigate the risk of misuse, there is a need for more advanced surveillance approaches. Genetic Engineering Attribution (GEA) consists of identifying signatures characteristic to genetic designers, which could be used as a surveillance strategy to identify who produced a certain genetically engineered organism.

Previous work has relied on using deep learning to identify features in plasmid sequences that are characteristic of certain laboratories [8] [2] [9]. For instance, some antibiotic resistance genes or cloning methods may be used more frequently in one lab than in another, which allows to train models that identify which lab created a plasmid from its sequence alone.

This problem is related to this course's content because it focuses on understanding genetic information, like genes, codon usage, or cloning methods, from sequence alone. The techniques used for this are similar to the ones used in interpreting genomic information, like how to prepare the input data (raw sequence or extract features) or finding model architectures that understand the local and global context of the sequences.

1.2 Main hypothesis

Modern deep learning methods like transformers or equivariant models improve the accuracy in predicting the laboratory of origin from a plasmid DNA sequence with respect to previously described methods including recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

2 Related Work

The first study about GEA used the same dataset, and demonstrated the use of CNN to classify sequences with modest accuracy (48% accuracy and 70% top 10 accuracy) [8]. A follow up study employed a RNN to bring top 1 accuracy up to 70% and top 10 accuracy to 84% [2]. While the first CNN approach was not explainable at all, the RNN encoding method allowed the authors to identify important features and relate them to their function in the genetically engineered microorganism. An alternative approach focused on interpretability used multiple sequence alignment, instead of a deep learning model, and achieved slightly higher accuracy but much more clear interpretations [9].

More recently, releasing the data in form of a competition where teams had to train models to win prizes, resulted in models that practically solved the problem [6]. The winning models achieved top 1 accuracy of 80% and top 10 accuracy up to 95%. Most of these winning models were based on CNNs that processed plasmid regions up to 4kbp, with the exception of a k-mers based model that was the fastest to train. Some of the winners provided approaches to interpret their model's predictions. To the best of our knowledge, there are no reports of transformer models being applied to GEA.

3 Dataset

3.1 Data origins

The dataset consists of the DNA sequences of all genetically engineered plasmids deposited in the Addgene database up to July 27th 2018, which sums up to 81,834 entries. From this, only 63,017 entries (77%) are labeled and can be used in this work as part of the training and test set [6]. The dataset is already divided between features and labels. Each entry in the features data contains the sequence of the plasmid (variable length, string composed of A, T, G, C, or N characters), a unique numeric ID, and 39 binary features derived from the metadata associated to the sequence, including parameters such as the presence of certain resistance genes, the strain it was delivered in, the presence of common selection markers, and the species of destination. Each entry in the feature dataset contains a corresponding entry in the labels dataset, which consists of one-hot encoded categories for each of the 1,314 possible laboratories of origin.

The data was downloaded from the GEA competition on the DrivenData website before the competition ended [1], and is available for use for non-commercial purposes (see more information in the blog post [4]). Based on previous solutions to this problem, we chose to ignore the 39 binary features and use only elements of the sequence as part of our classifiers. We know that some models get high accuracy by processing the sequences into k-mers, while others are trained directly on the sequence without any processing, and that some use the binary features while some others don't [5]. Using this data is a great way to approach the problem because it has been used in previous work in this field, and hence it provides a great benchmark.

3.2 Data exploration

3.2.1 Labels

The number of labels per lab are not balanced (Figure 1). Some labs have as little as one instance in the dataset, and other have over 8000 instances. The average number of instances for a lab is 48 instances, and the median is 15. These class imbalances are likely to result in issues with classification when it comes to labs with less training instances.

For the transformers models, this issue was worked around by creating a classifier using the most abundant classes as a proof of concept.

3.2.2 Sequences

The sequence length is also very imbalanced (Figure 2). The shortest sequence length in the dataset is 20 nucleotides, whereas the longest is 60,009 nucleotides. The average sequence length is about 4839 nucleotides, and the median sequence length is 4741 nucleotides. This proved to cause issues when it came to incorporating the sequences into our networks, as we often did not have enough memory to accommo-

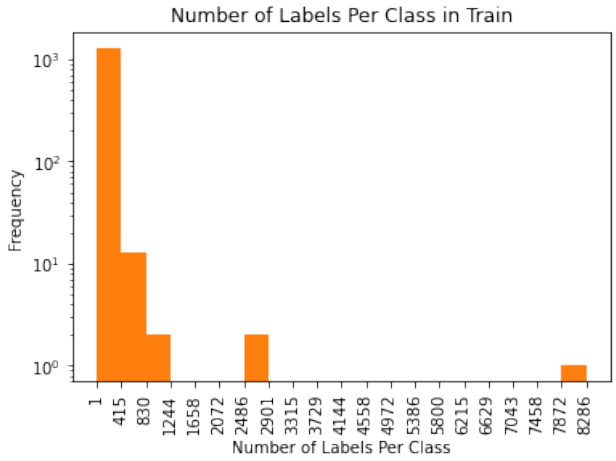

Figure 1: A histogram showing the frequency of a number of samples per lab in the training set.

date for sequences up to the max length, and instead had to cut sequences down.

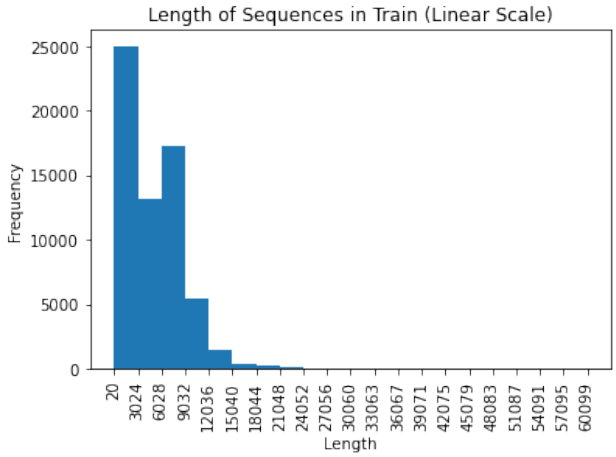

Figure 2: A histogram showing the frequency of sequence lengths in the training set.

4 METHODS

4.1 Convolutional Neural Network

A Convolution Neural Network (CNN) is a Deep Learning architecture which can capture relationships between nearby features in input to complete classification or regression tasks. CNNs are often applied to image learning tasks since it can capture relationships between local pixels. CNNs also have the potential to be applied to DNA sequences with 1-dimensional convolutions in order to learn relationships between nearby nucleotides or k-mers.

We chose to start by implementing CNNs as a proof of concept. By creating a CNN we could have a baseline for the transformer models, and get more experience creating neural networks for the problem space.

4.1.1 Data Preparation

Pad. Inputs to the CNN must all be the same length. Sequences shorter than the largest sequence need to be padded out. For each sequence, $< pad >$ tokens are added to the end of the sequence until it's the same length as the largest sequence in the dataset.

Cut. Due to the memory limitations of our research equipment, having each of the 63,017 padded out to the maximum length of 60,009 nucleotides was infeasible. Instead, each sequence was cut down to a shorter length. This was arbitrarily chosen to be cut to 1,000 - 4,000 nucleotides, depending on the experiment.

Tokenize. Each sequence would then be tokenized, either by its individual nucleotides or as as k-mers. In order to tokenize it as nucleotides, each DNA base is mapped to a different numerical token, and the sequence is filtered such that each nucleotide becomes the tokenized value. If the sequence is tokenized as a sequence of k-mers, each sequence is broken into its k-mers. Then, each kmer is mapped to a numerical token, and the sequence is filtered such that each k-mer is now the associated number.

Split into test and train. 90% of the labeled data was placed in the training set, and 10% was placed in the test data.

4.1.2 Architecture

The CNN architecture consisted of the following components (Figure 3):

- **Encoder**. An encoding layer designed to learn in an embedding from the tokenized input sequence. This layer has the potential to learn more information about the similarities and differences between the different tokens (either the individual nucleotides, or k-mers).
- **Convolutional Layers**. The architecture then consists of 3 convolutional layers made up of a 1D convolution, a RELU activation, and then a max pooling layer.
- **Fully connected and dropout**. The next layer consists of a fully connected layer and then a 50% dropout.
- **Softmax and classification**. The final layer consist of a softmax which will give a probability that the input sequence is coming from a particular lab.

To train, a cross entropy loss function was used, since these are common for classification tasks.

4.1.3 Hyperparameters

Consistent hyperparameters. Some hyperparameters were kept consistent during the experiments, such ass the weight decay, learning rate, and momentum. We also usually performed training with around 10 epochs since even this took a long time to train. If more time was allotted for this project, hyperparameter tuning in this area should be explored.

Kernel size. At first, a very small kernel size and stride was used (3), but this was updated to 20 after viewing the class slides on CNNs, which suggested a convolution size around this number. This greatly improved accuracy and decreased training time.

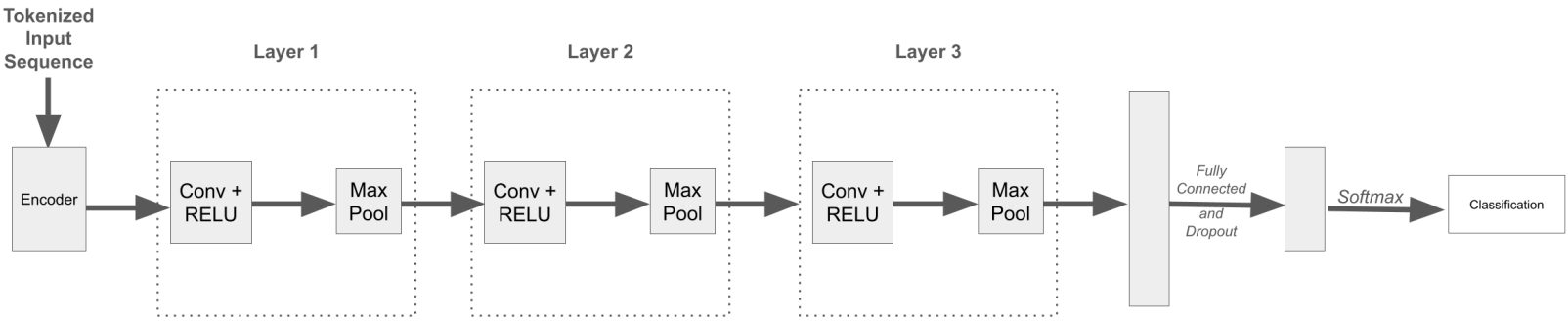

Figure 3: A diagram depicting the architecture of the CNN model.

Input sequence length. Input sequences of different sizes were explored, from 1,000 nucleotides to 2,000 nucleotides. This cutting was done due to limited computing resources. The computing resources we had were unable to handle large sequence length, and the longer the sequence, the longer training took.

K-mer size. We tried testing 1-mers and 3-mers.

4.1.4 Results

The table below provides a brief summary of the different approaches tried. See the code availability section and see the Simple CNN Models project for more details on each experiments architecture.

num	description	train acc	val acc
0	1000 nt input	33%	33%
1	2000 nt input	36%	34%
2	1000 nts input Smaller convolutional layers	27%	27%
3	1000 nts 3-mer input	35%	34%
4	1500 nts input 3-mer input	35%	34%

Overall this approach receieved a 27% - 36% accuracy depending on the experimental hyperparemeters. At first glance, this is not amazing, especially compared with winning competition submissions that were getting up to 80% accuracy. However, when compared to a null model (which simply draws labels from a probability distribution based on the training data), which only has a 2% accuracy, our CNN model seems to be getting signal. This has promise that we're on the right track for being able to replicate the results of the competition winners, although it would take siginificantly more time, computing power, and hyperparemeter tuning.

In general, larger input sequences tend to do better. This makes sense since these larger inputs will capture more information about the sequences. Additionally, k-mer models do better when compared with a non-kmer model that takes in a sequence of around the same size. This implies that using kmers as input has the potential to capture more information.

4.1.5 Limitations

Computing resources. Initially, we tried to work on the CNN in Google Collab, but we found that we were quickly running out of ram, and often the time it took to complete experiments took and unreasonable amount of time. The work for the CNN was done on a personal computer with limited computing power. This machine had 32 GB of Ram, a 12th Gen Intel Core i7 CPU, and an NVIDIA 3060Ti GPU. This machine, although much more powerful than an average laptop, was ill equipped to handle this computing problem, forcing us to limit input sequences and number of epochs performed. Training over 10 epochs took a median of around 5 hours for each experiment.

Hyperparameter space explored. There exists an exhaustive combination of CNN architectures, hyperparameters, and data processing methods that could be explored. Unfortunately, we only explored an extremely small subset of them. Future work involving CNNs would likely involve significant time into exploring hyperparameters and tuning.

4.2 Transformers

Transformers have been used extensively for language modeling. BERT is a popular transformer model that was trained on large parts of text for next sequence prediction and can be fine-tuned for specific applications. For GEA, we decided to use DNABERT, which uses BERT's architecture but was pre-trained on the human genome [7]. This model was made publicly available and used in binary classification tasks such as identifying if a sequence contains a promoter. Briefly, the model takes as input a sequence tokenized in 6-mers (they have been shown to have better accuracy than shorter k-mers), that is ran through 12 layers of self-attention. This generates an embedded sequence representation that is used as input for a fully connected layer that acts as a classifier.

DNABERT has only been used for short sequences (<500bp), but it can be adapted to longer sequences. Transformer models that use large regions of DNA sequence have been shown to overcome the limited receptive field of CNNs and do better in tasks such as predicting gene expression [3]. Our hypothesis is that adapting DNABERT to long sequences will improve the accuracy of current GEA models. In addition, DNABERT's attention weights have been used to provide clear interpretations of the sequence features that have more importance for sequence classification. Hence, it is a promising approach to

build an explainable model, which is critical to interpret the results and act on them in case GEA is used as a surveillance strategy.

4.2.1 Methods

Instead of building the ideal classifier from the beginning, we started simplifying the problem in multiple aspects to study how each of the challenges (sequence length, imbalanced data, etc.) affects model performance. In all cases, 50% of the sequences were passed as input to the model in their reverse complement orientation. The metrics used to evaluate the models included top 1 accuracy (% of test predictions whose label matches with the correct label), top 10 accuracy (% of predictions where the correct label is within the 10 most probable predicted labels) and Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC).

Binary classifier, short sequence. Since DNABERT was only used for binary classification in previous studies, we decided to test it only with the sequences of the 2nd and 3rd most abundant labels (labs of origin) in the dataset, which consist of approximately 2,500 sequences per label. Similarly, to stay consistent with previous studies with DNABERT, we used only 100 nucleotides segments of sequences as inputs. Within this constraints, we tested two different approaches: taking only the first 100nts of the plasmid sequence or randomly sampling 8 different 100nts subsequences for each input sequence. In both cases, the test set consisted of 1,000 sequences. The training set, instead, consisted of 4,400 sequences in the first case and 43,136 in the latter.

Multiclass classifier, long sequence. Then, we adapted DNABERT's code to support multiclass input and tested it with the 30 most abundant classes. To avoid problems with class imbalances we sampled 281 sequences for each of the 30 classes, which is the number of sequences in the least abundant class. This allowed us to study the effect of input sequence length. We trained three different models. The first one used a random subsequence of 100nts. The second one used 512nts, which is the maximum supported as input to the model. The third one used 1024nts, which were accommodated to the model by splitting it in two 512nts and concatenating their embedded representations in the final classification layer to make a single prediction. Since not all input sequences are longer than 1024nts, some sequences were padded to that length. Extending sequences to 2048nts resulted in memory problems. In all cases, 1,000 sequences were used for testing and 7,400 for training.

Full dataset, long sequence. Finally, we decided to test the model with sequences from all classes subsampling them to 512nts. The input dataset consists of thousands of sequences and training on all of them would have been time prohibitive given the available resources. To ensure that the model had a chance to be trained on sequences from all labels, we subsampled the initial distribution with a probability proportional to the number of sequences of the label in the dataset but with an additional term of 50,000 to minimize the imbalances ($P_{sampling}(i) = \#\text{seqs}_i + 50{,}000$, normalizing the sum to 1). The data was still imbalanced (some labels having 16 sequences and others just 2, with a mean of 7) but much less than the original. This resulted in 1,313 sequences (one per label) for testing and 7,009 for training.

4.2.2 Results

Binary classifier, short sequence. In the binary classifier case, the model reached high accuracy ($> 95\%$) and plateaued in about 500 steps (Figure 4A). Each step includes training with a batch of 32 sequences, so this indicates high accuracy after 16,000 example sequences. The *init* model considered only the first 100nts of each sequence, while the *rand* took 8 random 100nts subsequences for each sequence. Hence, 500 steps corresponds to about 3.6 epochs for the *init* model and only 0.37 epochs for the *rand* model. Note that comparing both models directly is not fair, since the input data for training and test sets is different in both of them. However, the *init* model indicates that the first 100nts of those two labels are different enough to recognize them and the transformer model can capture this. Similarly, the *rand* model indicates that random subsequences of the plasmid can also be used to accurately classify other random subsequences from the same plasmid.

Multiclass classifier, long sequence. Adapting DNABERT to multiclass classification with balanced data from 30 different labels allowed us to investigate in more detail the effect of sequence length. Curiously, the *short* model used only a random subsequence of 100nts per plasmid (like the previous binary models) did not perform well, with accuracy decreasing during training (Figure 4B-C). Instead, increasing the sequence length to 512nts in the *long* model or to 1024nts in the *xl* model resulted in top 1 and top 10 accuracy clearly greater (39.3% and 87.1%, respectively) than the baseline accuracy of a random classifier.

Interestingly, the *xl* model required more steps to start increasing the accuracy than the *long* model but gained accuracy faster after the initial delay. The *long* model ended up with a higher accuracy probably because it goes through each step twice as faster than the *xl* model, since the *xl* model relies on making two predictions per sequence. This indicates that the *long* model to gain more information from evaluating more training examples than the gain that the *xl* model has by looking at twice the length of the sequence. Thus, future training approaches might benefit from training with a higher number of short sequences instead of on a lower number of long sequences. Nevertheless, if time and resources allow it, a model trained with longer sequences might end up offering better accuracy.

Visualizing the ROC curves for the *long* (Figure 5) and *xl* (not included, similar to Figure 5) models showed that the classifying capacity of these models is decent, with some classes being accurately classified with higher confidence than others (Figure 6). Identifying the classes where the AUC is lower can be helpful to study their data in more detail and see if there are factors that affect its performance.

Full dataset, long sequence. Given the results in the previous section, we decided to train the model with sequences from all labels using an input sequence length of 512nts. However, model training was unstable and the accuracy of the model dropped to below that of a random classifier after 100 training steps (Figure 4). Studying the predictions made on the test set every 20 training steps we could see how predictions of the test set become more homogeneous during training until the model ends up predicting the same label for all test examples. Interestingly, the predicted label changes during training, indicating that weights are being adjusted in different ways de-

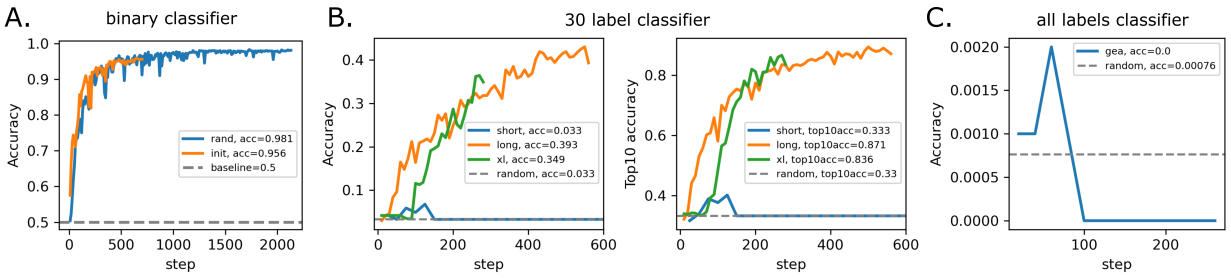

Figure 4: Comparing the accuracy of fine-tuned transformer models on the test set during training. (A) Top 1 accuracy of a binary classifier with 100nts as input sequence, either using the beginning of the sequence (init) or a random subsequence (rand). (B) Top 1 (left) and top 10 (right) accuracy of a multiclass classifier limited to the 30 most abundant labels, using a sequence input of 100nts (short), 512nts (long) or 1024nts (xl). (C) Top 1 accuracy of the classifier trained with 512nts sequences of all labels in the dataset.

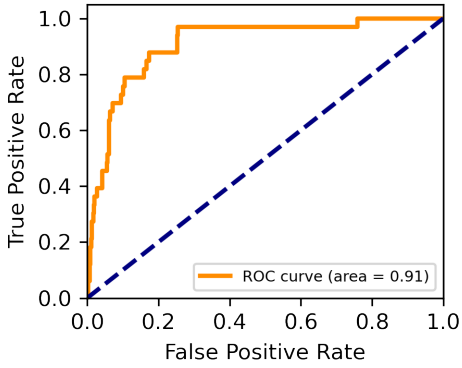

Figure 5: Receiver operating characteristic (ROC) curve for the long multiclass classifier operating on 30 labels. The curve is the average ROC across all 30 classes.

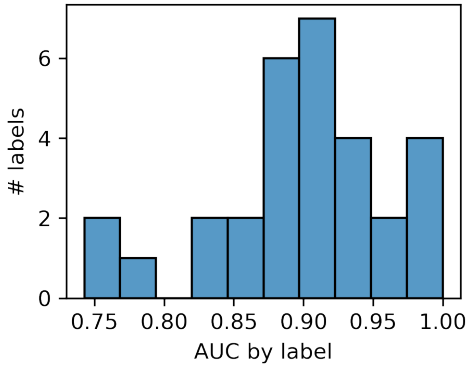

Figure 6: Distribution of area under the ROC curve (AUC) for the 30 labels of the long multiclass classifier.

pending on the data received in the training steps.

This effect was also observed in the *short* model that used 100nts for 30 classes. In that case, using longer sequences recovered the training ability of the model. It may be the case that scaling up to 1,313 classes requires extending the length of the sequences beyond 512nts as it was done here. However, this may cause memory issues and slow down the training process. Another possible solution may be increasing the batch size to incorporate more than 32 sequences in each training step or decreasing the learning rate.

5 Discussion

Finding resources to train the models was a limiting factor. This is generally the case with large deep learning practices. To mitigate its impacts, we decided to train on smaller datasets and mock examples before tackling the full problem. This, in the transformer case, enabled us to confirm that the model was indeed useful for sequence classification in the GEA context and that input sequence length and training strategy are key to build a model with the full dataset. Limited resources did not allow us to properly test the effect of important hyperparameters like the learning rate, batch size or input sampling strategies like the size of k-mers. Those parameters would likely let us achieve much higher accuracy that the one reported here.

Transformers showed great performance when using only 2 or 30 classes, but not for the full dataset. Hence, the initial hypothesis cannot be confirmed or rejected, since transformers were not explored enough to compare them fairly to the CNN approach. Nevertheless, the experience acquired in the process allows us to outline some clear future directions.

Fine-tuning transformer models typically results in high accuracy but it comes with a great cost. At every step, the large number of weights of the transformer model need to be readjusted, which takes a considerable amount of time. An alternative approach used in transformer-based classifiers is using the pre-trained model as is to featurize the input sequences into an embedded representation. This embedded representation can then be used to build a classifier on top of the transformer that predicts the label from the embedded representation. Since the weights of the transformer model are not adapted for the specific classification task, the embedded representation does not contain as much relevant information about the sequence than if the model was fine-tuned for the task, resulting in some loss of accuracy. However, training this kind of model is much faster than fine-tuning all the transformer because of the lower number of weights. With a limited amount of resources, it would be better to use the pre-trained transformer and build a classifier on top because it is

very likely that the gain in accuracy resulting from easier optimization and training of the classifier is greater than the loss in accuracy resulting from not having a fine-tuned transformer.

Another future direction of interest is using the scripts provided with the DNABERT model to visualize the importance of the features in the sequences. The importance of each nucleotide in the sequence can be readily obtained from the attention weights in the transformer layer, and visualizing them can point to certain sequence regions that might be relevant. Those sequences can then be matched to annotated sequences with BLAST to identify what kind of attributes are characteristic of a particular lab. Interpretability is key for the real deployment of the models. On a larger picture, the training data should also include examples of non-engineered sequences. This would allow the model to identify if a sequence has been really engineered or has natural origin, although multiple other factors like expert knowledge would be required to confirm these non trivial results.

In conclusion, our work illustrates a proof-of-concept for training transformer models on GEA data and comparing them to conventional CNN approaches. The lack of time and resources did not allow us to optimize a transformer model that learns from the full dataset. Nevertheless, smaller scale examples demonstrated the importance of input sequence length and hyperparameter optimization. In addition, the experience in training these models let us now see which approaches we would follow in the future to adjust our models to the resources we have, like avoiding fine-tuning and building classifiers based on transformer embedded sequence representations.

6 Code Availability

Code is available here: https://github.com/joely-nelson/GEA-CSE-599. The repository contains the trained model's checkpoints, data processing and training scripts, and jupyter notebooks used to analyze the data and generate the figures included in the text or used to support it. The README files contain information about where to find each element.

References

[1] Drivendata. genetic engineering attribution challenge. https://www.drivendata.org/competitions/63/genetic-engineering-attribution/, 2021.

[2] E. C. Alley, M. Turpin, A. Liu, et al. A machine learning toolkit for genetic engineering attribution to facilitate biosecurity. https://doi.org/10.1038/s41467-020-19612-0, 2020.

[3] Avsec, V. Agarwal, D. Visentin, et al. Effective gene expression prediction from sequence by integrating long-range interactions. https://www.nature.com/articles/s41592-021-01252-x, 2021.

[4] W. Bradshaw. Geac: Update on results data usage - genetic engineering attribution. https://community.drivendata.org/t/geac-update-on-results-data-usage/5824.

[5] C. Chung. Meet the winners of the genetic engineering attribution challenge. https://www.drivendata.co/blog/genetic-engineering-attribution-winners, 2021.

[6] O. M. Crook et al. Analysis of the first genetic engineering attribution challenge. https://arxiv.org/abs/2110.11242, 2021.

[7] Y. Ji, Z. Zhou, H. Liu, and R. V. Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. https://academic.oup.com/bioinformatics/article/37/15/2112/6128680, 2021.

[8] A. Nielsen and C. Voigt. Deep learning to predict the lab-of-origin of engineered dna. https://doi.org/10.1038/s41467-018-05378-z, 2018.

[9] Q. Wang, B. Kille, T. R. Liu, L. Elworth, and T. J. Treangen. Plasmidhawk improves lab of origin prediction of engineered plasmids using sequence alignment. https://www.nature.com/articles/s41467-021-21180-w, 2021.