Binary Antimicrobial Resistance Prediction from PBP4 Gene Sequences: Comparing Encoder-Only Transformer Models

Suvinava Basak[1], Terril Joel Nazareth[1]

Abstract

Antimicrobial resistance (AMR) is a major global health threat, underscoring the need for rapid, accurate resistance prediction methods. This study presents a comprehensive comparison of different transformer models for binary AMR prediction using *pbp4* gene sequences from *Staphylococcus aureus* (against *cefoxitin*). We evaluated multiple transformer architectures, including a custom transformer built from scratch, sophisticated pre-trained transformer models like DNABERT-6 (under three different settings: full fine-tuning, frozen pre-trained backbone with trainable head, and with parameter-efficient Low-Rank Adaptation (LoRA)) and Nucleotide Transformer. To handle class imbalance common in biological data, we employed class weighting alongside a special loss function, combining both max-margin loss and focal binary cross-entropy loss, during the training phase.

All models were trained and tested on datasets of 150 *S. aureus* sequences (containing 40 non-resistant and 110 resistant samples). Our custom transformer included genomic-specific tokenization and attention mechanisms similar to DNABERT-6. The best-performing model, DNABERT-6 (in full fine-tune mode), achieved an accuracy of 100%, precision 100%, recall 100%, F1-score 100%, and ROC-AUC 100% on held-out test data. Despite the superior performance, we believe that more training data will be helpful to draw an informed conclusion from the result. Our experiments highlighted the efficiency of LoRA fine-tuning for scalable, computationally inexpensive AMR prediction by achieving reasonable performance while utilizing only 0.66% of the total trainable parameters. The accuracy, precision, recall, F1-

[1]TU Braunschweig, Master's program in Data Science, Germany

score, and ROC-AUC achieved by LoRA-adapted DNABERT-6 is 93.33%, 100%, 91.67%, 95.65%, and 100% respectively.

Our results not only demonstrates the potential of transformer-based models for genomic AMR prediction, especially in parameter-efficient fine-tuning settings like LoRA, it also advances computational methods for rapid resistance detection, by providing a comprehensive comparison of modern transformer architectures, aiming to improve clinical decision-making and antibiotic stewardship.

Keywords: Antimicrobial resistance, Staphylococcus aureus, pbp4 gene, encoder-only transformers, DNABERT-6, Nucleotide Transformer

---

1. Introduction

Antimicrobial resistance (AMR) has emerged as a global health threat in the 21st century, ranked by the World Health Organization (WHO) among the top ten dangers to humanity [1]. In 2019, this deadly threat was responsible for at least 1.27 million global death and associated with nearly 5 million deaths [2, 3]. According to Center of Disease Control and Prevention (CDC), more than 2.8 million AMR infections occur each year in U.S, causing 35,000 death [4]. In addition to the loss of human lives, the economic loss is also severe. According to the prediction by The World Bank, AMR could cause US$ 1 trillion additional healthcare costs by 2050, and US$ 1 - 3.4 trillion gross domestic product (GDP) losses per year by 2030 [5].

Traditional antimicrobial susceptibility detection practices relies mainly on phenotypic methods, taking 24–72 hours to produce results. This delay often resulted in inappropriate empirical antibiotic usage, risking treatment failure and promoting resistance even higher. This highlights the urgency for a rapid, accurate resistance prediction method which motivated research into molecular methods using bacterial genetics to predict phenotypic resistance within hours.

Staphylococcus aureus is a major human pathogen causing infections ranging from minor skin issues to severe conditions like pneumonia, bacteremia, and endocarditis [6], being responsible for $> 50\%$ of hospital-acquired infections in the U.S.. Their resistance to β-lactam antibiotics like *cefoxitin* is

often used as a surrogate marker for methicillin-resistant strain of S. aureus (MRSA), which has 15-60% higher mortality rates than susceptible strains [7]. This higher resistance to β-lactam antibiotics in MRSA can be attributed to the mutations in the *pbp4* gene, which encodes protein responsible for bacterial cell wall synthesis and is a key β-lactam target. Mutations in *pbp4* can reduce antibiotic binding affinity, leading to resistance through decreased target susceptibility [8]. Genomic analysis of these species-specific mechanisms enables the development of more targeted resistance prediction models.

The advancement in deep learning has transformed computational biology, with transformer architectures excelling at modeling complex sequence dependencies [9]. Originally successful in natural language processing, models like Google's BERT (Bidirectional Encoder Representations from Transformers) demonstrated strong performance in capturing contextual relationships in sequences [10]. In genomics, recent breakthrough models like DNABERT-6 and Nucleotide Transformer showing how pre-trained transformers can uncover intricate sequence patterns often missed by traditional methods [11, 12]. Still, applying deep learning to AMR prediction brings specific challenges for good performance: genomic datasets are typically high-dimensional, limited in size, and often suffer from class imbalance, where one resistance class dominates the other. This can result in models that perform well overall but fail on critical cases, having severe consequences of false negatives in real-world situations.

Transfer learning has emerged as a powerful solution to data scarcity, allowing models pre-trained on large datasets to be adapted for tasks with limited labeled data [13]. In computational biology, it has proven effective, enabling models to learn broad biological patterns and be fine-tuned for downstream applications like resistance prediction. However, the best transfer learning strategy for genomic tasks is still an area of study, with methods ranging from full fine-tuning to parameter-efficient techniques that freeze most weights and adapt only specific components. Such parameter-efficient methods are particularly valuable in clinical settings, where limited resources and the need for efficient model deployment make full-scale fine-tuning impractical.

This study addresses the challenge of predicting antimicrobial resistance from *pbp4* gene sequences by systematically comparing several cutting-edge transformer architectures. We evaluate a custom transformer model tailored for

genomic classification alongside pre-trained models like DNABERT-6 and Nucleotide Transformer, using transfer learning strategies such as full fine-tuning, frozen feature extraction, and parameter-efficient methods. To address class imbalance, we incorporate advanced loss functions that combine max-margin objectives with focal loss.

2. Related Work

2.1. Early Machine Learning Approaches for AMR Prediction

Over the past two decades, machine learning for antimicrobial resistance prediction has come a long way. Early approaches used rule-based systems to create simple genotype-to-phenotype prediction mechanisms. These were limited because they required extensive manual selection of feature (depended on known resistance determinants) and could only capture simple sequence patterns. Then came traditional machine learning approaches like Random Forest, Support Vector Machine, and Logistic Regression. These methods were comparatively better than the traditional approaches, but still struggled to capture complex patterns in genomic data. The advent of deep learning marked a paradigm shift, enabling automatic feature learning directly from raw sequence data [14].

2.2. Deep Learning Architectures for Genomic Sequence Analysis

Deep neural networks was first introduced for genomic sequence analysis through convolutional neural networks (CNNs), which excelled at identifying motifs and local patterns. In the area of antimicrobial resistance, Arango-Argoty et al. first introduced DeepARG, a deep learning framework that outperforms traditional sequence alignment methods by reducing false negatives and improving recall in Antibiotic Resistance Gene (ARG) detection [15]. CNNs proved to be good at finding local patterns, but struggled to capture sequences. Recurrent neural networks (RNNs), especially bi-directional LSTMs with attention, addressed this by modeling sequence dependencies more effectively; however, they weren't effective to capture long-range dependencies in genetic sequences. Moreover, RNNs also suffered from the vanishing gradient problem when dealing with long sequences.

2.3. Transformer Architectures in Computational Biology

The transformer architecture, introduced by Vaswani et al. [9], revolutionized sequence modeling with its self-attention mechanism and parallel processing. Self-attention helped in capturing both local and distant genomic relationships. It allows the model to identify and weigh the importance of different parts of the input sequence by attending to itself. Success of transformer models in natural language processing, like BERT [10] and GPT [16], motivated its adaptation to various sequence analysis tasks in computational biology.

2.3.1. DNABERT and Genomic Language Models

DNABERT-6 was the first major adaptation of BERT for genomics [11], which used k-mer tokenization to treat overlapping nucleotide sequences as discrete tokens like words. Pre-trained on the human genome with Masked Language Modeling (MLM), DNABERT excelled across many genomic prediction tasks. Later, many more sophisticated architectures like DNABERT-2 was developed on top of DNABERT for better tokenization, more efficient attention and to boost performance further across tasks while maintaining computational efficiency.

2.3.2. Nucleotide Transformer and Multi-Species Models

Dalla-Torre et al. introduced Nucleotide Transformer, extending transformers to multi-species genomic analysis by pre-training on diverse genomes from multiple organisms [12]. This approach overcame species-specific model limits by learning universal patterns across taxonomic boundaries, boosting generalization and versatility in genomic tasks.

2.4. Transfer Learning Paradigms in Genomic Applications

Transfer learning gained popularity in NLP and computational biology due to the scarcity in labeled data [13]. It became possible to pre-train models on large genomic datasets and then to fine-tune them effectively for various downstream tasks, even outperforming training from scratch. In genomics, there are diverse strategies for transfer learning depending on the need of efficiency and accuracy.

Full fine-tuning, which involves updating all pre-trained model parameters for specific tasks, is very common in genomics but challenging for small datasets. Because of the high number of parameters and limited data, it often causes

overfitting and poor generalization. Careful scheduling of learning rate and proper regularization of the model can reduce the risk of overfitting and help adapt genomic transformers effectively, but performance will still be limited with very small datasets.

Alternative methods, using pre-trained models as fixed feature extractors and passing learned feature representations into task-specific classifiers worked well with limited data. Carefully selecting a few trainable layers and fine-tuning only those layers helped maintain trade-off between adaptation capability and overfitting risks.

2.5. Low-Rank Adaptation as a PEFT Method

The high computational and storage demands of fine-tuning large transformer models, often with millions or even billions of parameters, have led to the development of more parameter-efficient fine-tuning (PEFT) methods which can adapt to the pre-trained feature representation with the burden of heavy computational cost of full fine-tuning.

Low-Rank Adaptation (LoRA), proposed by Hu et al., offers a particularly efficient solution by adapting pre-trained models with minimal parameter updates [17]. Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique for large pre-trained models, where we inject two trainable low-rank matrices into existing weight layers. Instead of updating all the weights directly during the training phase, LoRA approximates the weight updates as the product of these two low-rank matrices. This means instead of updating the full weight matrices directly, we learn two smaller matrices A and B such that the weight update is approximated as $A \times B$. The idea is that weight updates during fine-tuning often have low intrinsic dimensionality, so we can capture most of the important changes with far fewer parameters (even less than 1% of total parameters), hence being more resource-efficient.

Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA approximates the update ΔW as a product of two low-rank matrices:

$$\Delta W = AB, \quad A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k)$$

Instead of updating W_0 directly, only A and B are learned during fine-tuning, significantly reducing the number of trainable parameters. The adapted

weight is:

$$W = W_0 + \Delta W = W_0 + AB$$

where r is the rank hyperparameter controlling the trade-off between model capacity and parameter efficiency.

2.6. Addressing Class Imbalance in Genomic Machine Learning

Class imbalance represents a major challenge in genomic dataset for machine learning applications. Traditional approaches to addressing class imbalance, including resampling have been adopted in various genomic applications, but they may not always perform well.

2.6.1. Advanced Loss Functions for Imbalanced Data

Recent efforts have developed advanced loss functions to handle class imbalance. In medical applications, we often care more about hard-to-classify cases. Focal loss, introduced by Lin et al., addresses this challenge by giving more attention on hard examples (those the model is struggling with) and down-weighing easy examples during training [18]. Proved to be helpful in handling class imbalance, especially in highly imbalanced datasets, focal loss was widely adapted for genomic data, improving classification performance.

2.6.2. Margin-Based Approaches

Max-margin loss encourages the model to not just classify correctly, but to classify with confidence. It does so by ensuring the samples from same class are closer together whereas the different classes as far apart as possible. This pushes the decision boundary away from the support vectors, creating a larger margin between classes. Cortes and Vapnik first introduced the max-margin approach in their revolutionary paper on support vector machines [19]. While they didn't explicitly named it max-margin loss in that paper, but the concept was central: it tries to find a decision boundary that maximizes the margin between classes. They idea has since been widely adapted to deep learning to improve class separation in feature space.

2.7. Genomic Sequence Representation and Tokenization

The representation of DNA sequences for machine learning models has evolved from simple one-hot encoding to advanced tokenization schemes. K-mer

based approaches, where sequences are segmented into overlapping subsequences of length k, have become popular by easily capturing local sequence patterns while maintaining computational efficiency. Later, more sophisticated tokenization methods (for example, Byte Pair Encoding (BPE)) was developed to discover optimal sequence representations during training.

3. Methodology

3.1. Encoder-only Transformer Architecture

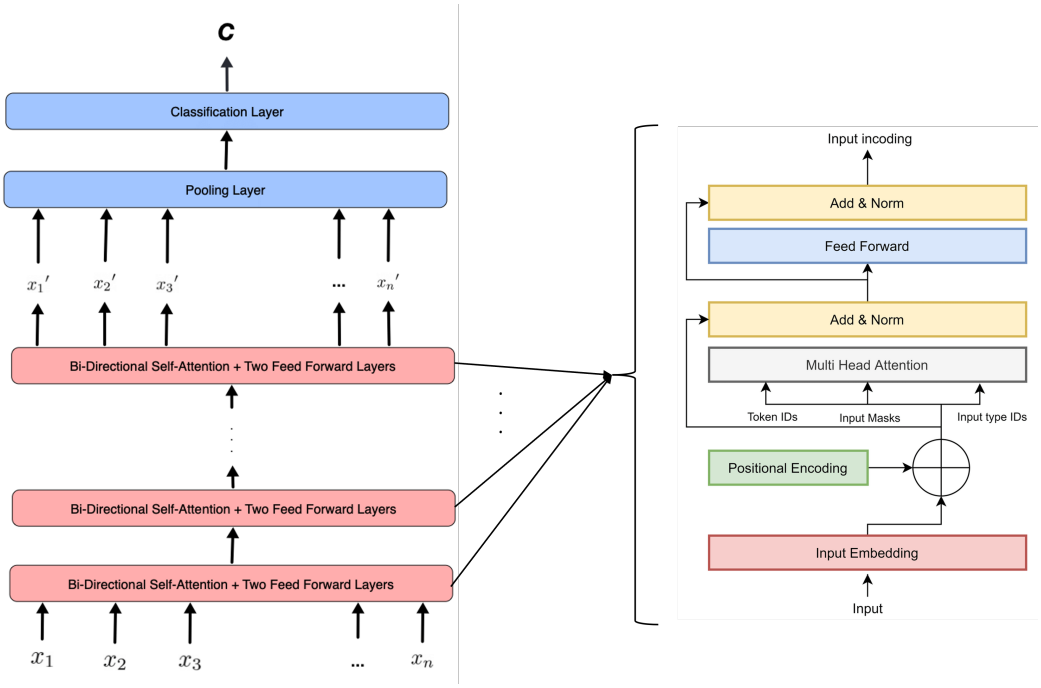


Figure 1: Sample architecture of encoder-only transformer[2]

We have built our transformers keeping in mind the architecture shown in figure 1. In our work, we rely on an encoder-only transformer architecture, based on the design introduced by Vaswani et al. in "Attention is All You Need" (2017) [9]. Unlike the original encoder-decoder setup used for sequence-to-sequence tasks, encoder-only models like BERT (and its adaptations such as DNABERT-6) are optimized for bidirectional representation

[2]Image source: left-image, right-image

learning from unlabeled sequence data.

Every transformer can have multiple encoder blocks (for example, 12 in DNABERT-6, 24 in Nucleotide Transformer), this is represented by the reddish-orange block in the left side of the figure, repeating multiple times. The right side of the image is a zoomed-in schematic of each encoder block, which contains two main components:

1. A **multi-head self-attention mechanism**, which enables the model to capture contextual relationships between tokens (e.g., DNA k-mers) at varying distances within the input sequence.
2. A **feed-forward neural network**, applied independently to each token representation.

Both components are wrapped with *residual connections* (Add) and *layer normalization* (Norm), promoting stable and efficient training. Inputs come from the previous encoder layer (or the embeddings for the first layer). Positional encodings are added to the input embeddings to preserve sequence order, which is otherwise lost due to the attention mechanism's permutation invariance.

DNABERT-6 follows this structure but adapts it for biological sequences by replacing word tokens with k-mer embeddings (usually 6-mers) derived from nucleotide sequences. This allows the model to learn biologically meaningful patterns relevant to downstream tasks like resistance prediction or motif discovery.

3.2. Dataset and Data Pre-processing

3.2.1. Dataset Description

Our study utilized genomic sequences of the *pbp4* gene from a clinically significant bacterial species: *Staphylococcus aureus* tested against *cefoxitin*. The *S. aureus* dataset comprised a total of 150 sequences with binary resistance labels (0: non-resistant, 1: resistant), each having a length of 1296 base pairs. The initial distribution showed significant class imbalance, with 40 non-resistant isolates (26.7%) and 110 resistant isolates (73.3%).

The entire dataset was divided into training set of 135 sequences (98 non-resistant and 37 resistant) and a test set of 15 sequences (12 non-resistant and 3 resistant).

3.2.2. Addressing Class Imbalance

The substantial class imbalance in our dataset presented a significant challenge for model training and evaluation. We introduced two different strategies to handle class imbalance based on the model in use.

For the lightweight custom transformer model, we used resampling techniques on the training data to make the number of samples in both classes same. So, now training data has total 196 samples with 98 samples for each class.

For more sophisticated pre-trained models like DNABERT-6 and Nucleotide Transformer, we implemented class weight to down-weight the samples from majority class and give higher importance to the samples from minor classes. We implemented a mathematically rigorous weighting scheme. Let the dataset contain n samples with class labels $y \in \{0, 1\}$, where 0 represents non-resistant and 1 represents resistant samples. The class distribution is quantified as n_0 and n_1 for non-resistant and resistant samples respectively, where $n_0 + n_1 = n$.

The balanced class weights are computed using the inverse frequency weighting formula, where 2 is the number of classes:

$$w_0 = \frac{n}{2 \times n_0} \quad \text{and} \quad w_1 = \frac{n}{2 \times n_1}$$

where w_0 and w_1 represent the weights for non-resistant and resistant classes respectively. This formulation ensures that $\sum_i w_i \times n_i = n/2$ for each class, providing balanced representation regardless of the original class distribution.

The positive weight tensor for model training is calculated as:

$$\text{pos\_weight} = -w_{\text{pos}} = \frac{w_1}{w_0} = \frac{n_0}{n_1}$$

This weighting increases the penalty for misclassifications of resistant isolates, which are underrepresented in the training data. The resulting BCE loss with logits is:

$$\mathcal{L}_{BCE} = -w_{\text{pos}} \cdot y \cdot \log(\sigma(x)) - (1 - y) \cdot \log(1 - \sigma(x))$$

where $y \in \{0, 1\}$ is the ground truth label, x is the raw model output (logit), and $\sigma(x)$ is the sigmoid activation.

This formulation addresses class imbalance by encouraging the model to give more importance to the minority class during training, thereby improving recall and F1-score for the resistant class. This strategy ensured that our model evaluation would be conducted on balanced datasets, providing reliable performance metrics while allowing the model to learn from the realistic class distribution.

3.2.3. Creating Validation Dataset

We split the training set using `scikit` library function to form a separate validation set with 20% of the training samples to evaluate the trained models after each epoch. However, we kept the original test dataset intact for the final evaluation.

3.3. Model Architectures

3.3.1. Custom Transformer Architecture

We developed a transformer model from scratch specifically designed for genomic sequence classification. The architecture incorporated several key components:

Vocabulary and Tokenization Strategy

Our custom tokenization method employed a k-mer ($k = 6$) based approach similar to DNABERT-6, where DNA sequences were segmented into overlapping subsequences of length 6. For example, a sequence of 'ACGTCGATG' will result in 4 6-mers: ['ACGTCG', 'CGTCGA', 'GTCGAT', 'TCGATG']. In general, a sequence of length l will generate $(l - k + 1)$ k-mers (including duplicates, if any). For our dataset, each sequence of length 1296 generated 1291 6-mers.

This generated k-mers were then converted to tokens using custom tokenization strategy. We created the vocabulary of all possible unique 6-mers and assigned them an unique numeric ID. In our dataset, we got 1946 unique tokens which created the vocabulary. We also added two special tokens: `<pad>:0` and `<unk>:1` for padding shorter sequences in a batch (since all inputs in a batch should have same number of tokens) and to represent k-mers that are not in the vocabulary respectively. For tokenization, we assigned

the corresponding numeric ID to each generated 6-mers. The generated tokens for each input sequence is nothing but a list of tokens (numeric ID). This tokenization strategy captures local sequence patterns while maintaining computational efficiency.

Transformer Architecture Details

The custom transformer consisted of:

1. *Embedding Layer*: Converting k-mer tokens to dense 64-dimensional vector representations
2. *Positional Encoding*: Sinusoidal positional embeddings to capture sequence order information
3. *Multi-Head Attention Layers*: 1 transformer block, containing:
   - Multi-head self-attention with 2 attention heads
   - Feed-forward networks with hidden dimension 64
   - Residual connections and layer normalization
4. *Dropout Layer*: Applied on the transformer outputs before feeding to the classification head.
5. *Classification Head*: A final linear layer mapping dropped transformer outputs to binary predictions:

   ```
   self.classifier = nn.Linear(embed_dim, num_classes)
   ```

Keeping in mind the limited size of the dataset, we kept the overall architecture of this custom transformer model simple to minimize the risk of overfitting and ensure the total number of parameters $(233,473)$ stays within reasonable limit for the model to learn the weights during training phase.

Model Configuration
In our custom transformer, we used:

1. Dropout rate of 0.4
2. `AdamW` optimizer with learning rate of $1e-3$ and weight decay rate of $1e-2$
3. we used `ReduceLROnPlateau` instead of normal `StepLR` as a learning rate scheduler with `mode='min', factor=0.3, patience=3, min_lr=1e-6`. This ensures that, in case of stagnancy for 3 epochs, the learning rate reduces by 0.3x factor until the minimum threshold.

4. We used *gradient clipping* to avoid challenge of exploding gradient.

3.3.2. Pre-trained DNABERT-6

We evaluated DNABERT-6 in three distinct configurations to explore different transfer learning strategies, as mentioned below. For k-mer generation we have used a custom function similar to the in-built function in DNABERT; however we couldn't reuse the in-built function in our project. For tokenization we have used the `Autotokenizer` from Hugging Face.

A. Full Fine-Tuning Configuration

In this setting, all parameters of the pre-trained DNABERT-6 model along with the parameters from the classifier head were made trainable, allowing the entire network to adapt to our specific resistance prediction task. The model architecture included:

- *Pre-trained Backbone*: DNABERT-6 with 89,191,681 parameters (including 769 parameters for the classifier head)
- *Classification Head*: A task-specific linear layer added on top of the pre-trained model:

  ```
  self.classifier = nn.Linear(hidden_size, num_classes)
  ```

- *Learning Rate Strategy*: Lower learning rates $5e-7$ for pre-trained parameters and $5e-6$ for classifier head to prevent catastrophic forgetting of pre-trained representations

B. Frozen Backbone Configuration

This approach treated DNABERT-6 as a fixed feature extractor, freezing all pre-trained 89,190,912 parameters while training only the classification head (only 769 parameters), significantly reducing the memory requirements and training time. Moderate learning rates $5e-4$ was used due to limited trainable parameters.

C. Low-Rank Adaptation (LoRA) Configuration

We implemented LoRA as a parameter-efficient fine-tuning technique, enabling model adaptation to our specific task with minimal parameter overhead.

LoRA Mathematical Framework

LoRA decomposes weight updates using low-rank matrices. The pre-trained weight matrix is $W_0 \in \mathbb{R}^{d \times k}$, which typically belongs to a linear (fully connected) layer. Here d is output dimension of the linear layer (no. of output feature) and k is the input dimension of the linear layer (no. of input features).

The adapted weight becomes:

$$W = W_0 + \Delta W = W_0 + BA$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ are trainable low-rank matrices with rank $r \ll \min(d, k)$. Instead of updating all the weights directly, LoRA approximates the weight updates as the product of two low-rank matrices. During training, W_0 remains frozen while we learn the two smaller matrices A and B, and the weight updates are approximated as $A \times B$. The adaptation is initialized with A drawn from a normal distribution and B initialized to zero, ensuring $\Delta W = 0$ at initialization.

As a result of the low-rank adaptation, total trainable parameters were reduced to $590,593$ out of total $89,781,505$ parameters (including the newly introduced low-rank matrices) - less than 1% of the full model.

LoRA Implementation Details

Our LoRA configuration included:

1. *Rank Parameter*: $r = 16$, large enough to capture important adaptations but small enough to prevent overfitting on our limited data
2. *Target Modules*: Applied to only "query" and "values" matrices in all transformer layers
3. *Scaling Factor*: $\alpha = 32$ for controlling adaptation strength

4. *Dropout*: LoRA-specific dropout rate of 0.1 was used additional to the transformer dropout rate of 0.3
5. *Learning Rate*: A learning rate of $1e-5$ for the base model and $1e-4$ for the classifier head was used

For all three settings, we used:

- Dropout rate: 0.3
- Simple linear classifier layer to classify the dropped features: `self.classifier = nn.Linear(hidden_size, num_classes)`
- Optimizer: `AdamW` with respective learning rate and weight decay rate of $1e-2$
- Learning Rate Scheduler: `ReduceLROnPlateau` with `mode='min', factor=0.3, patience=3, min_lr=1e-6`
- *Gradient clipping* to avoid challenge of exploding gradient

3.3.3. Pre-trained Nucleotide Transformer

We wanted to train another pre-trained transformer on the same model and compare the performance with that of DNABERT-6 in all settings. We employed `InstaDeepAI/nucleotide-transformer-500m-human-ref` model as our foundation for this purpose. This transformer-based model, specifically designed for genomic sequence analysis, provided a robust starting point for our fine-tuning approach on *cefoxitin* resistance prediction using *pbp4* gene sequences. In comparison to DNABERT-6, Nucleotide Transformer allows larger sequence length (1000 tokens), pretrained on diverse human reference genome and uses a different tokenization method.

Pretrained Model Specifications

The nucleotide transformer architecture consists of a sophisticated ESM-based transformer encoder with the following technical specifications:

- *Transformer Layers*: (24) deep transformer blocks
- *Hidden Dimension*: (1280) dimensional representations
- *Attention Heads*: (20) multi-head attention mechanisms per layer

- *Feed-Forward Networks*: Intermediate dimension of (5120)
- *Total Parameters*: Approximately (485) million parameters
- *Position Embeddings*: Maximum of (1002) tokens
- *Vocabulary*: (4107) unique nucleotide tokens optimized for genomic sequences
- *Sequence Length*: Maximum of (1000) tokens per input
- *Special Tokens*: `<pad>` (padding), `<mask>` (masked language modeling), `<unk>` (unknown sequences), and `<cls>` (classification)

In total, this architecture had a massive $485,701,868$ parameters - almost 6x of DNABERT-6 in full fine-tune mode.

Custom Classification Architecture

We developed a custom classification wrapper `NucleotideTransformerClassifier` that extends the pretrained nucleotide transformer for for our task. The wrapper maintains compatibility with the original transformer architecture while adding specialized components for classification:

- *Input Processing*: Extracts the first token representation from the final transformer layer with dimension (1280)
- *Regularization Layer*: Applies dropout with probability (0.1) to prevent overfitting on the limited dataset
- *Classification Head*: Linear transformation mapping from (1280) dimensions to (2) classes (resistant/susceptible):
  `self.classifier = nn.Linear(self.hidden_size, num_labels)`
- *Weight Initialization*: Classifier weights initialized with normal distribution ($\mu = 0.0$, $\sigma = 0.02$) and zero-initialized bias

The forward pass utilizes the first token representation as the sequence-level feature for classification, following established practices in transformer-based sequence classification. This approach captures global sequence information while maintaining computational efficiency for resistance prediction.

Fine-tuning Strategy

Given the limited dataset size (150 samples and the substantial parameter count of the pretrained model, we employed a strategic "top-n" unfreezing approach to balance transfer learning benefits with task-specific adaptation. We configured the unfreezing strategy as follows: `unfreeze_strategy="top_n"` with `unfreeze_layers=2`.

The layer-wise unfreezing configuration consisted of: frozen parameters for layers 0-21 (22 transformer layers) to preserve pretrained genomic representations, and unfrozen parameters for layers 22-23 (top 2 transformer layers) for task-specific sequence adaptation and the complete classification head $dropout + linearlayer$ for resistance prediction. This resulted in $39,357,442$ trainable parameters, which is $\approx 8\%$ of total parameters and $446,344,426$ frozen parameters 91.9% of total. This approach leverages the hierarchical nature of transformer representations, where lower layers capture general nucleotide patterns and higher layers learn task-specific features.

Optimization Configuration

We implemented a focused optimization strategy targeting only the unfrozen transformer layers, excluding the classification head from parameter updates due to experimental design considerations. The learning rate configuration used transformer parameters with learning rate $1e-5$ for the top two pretrained layers. We employed `Adam` optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e-8$. Our custom learning rate scheduler `NucleotideTransformerScheduler` follows the original nucleotide transformer training protocol with warm-up steps of 16000, maximum learning rate of $1e-4$, minimum learning rate of $5e-5$, and transformer ratio of 0.1. The scheduler implements a linear warm-up phase from minimum to maximum learning rate followed by square root decay, with adaptive termination when learning rate falls below $1e-6$.This approach stabilizes the gradient during warm-up phase and converges to a better solution during square root decay phase.

3.4. Loss Functions and Training Objectives

To address the challenges of class imbalance and improve feature separation, we implemented a combined loss function that incorporates both max-margin and focal binary cross-entropy components.

3.4.1. Max-Margin Loss

The max-margin loss aims to improve the geometric separation between classes in the learned feature space. For learned feature representations $\mathbf{f}_i$ with corresponding labels y_i, we compute:

$$\mathcal{L}_{margin} = \lambda_{margin} \cdot (\mathcal{L}_{intra} + \mathcal{L}_{inter} + \beta \mathcal{L}_{var})$$

where:

- $\mathcal{L}_{intra}$ is the *Intra-class Compactness*, which minimizes within-class distances using temperature scaling:

$$\mathcal{L}_{intra} = \frac{1}{|\mathcal{C}_0|} \sum_{\mathbf{f}_i, \mathbf{f}_j \in \mathcal{C}_0} \| \frac{\mathbf{f}_i}{\tau} - \frac{\mathbf{f}_j}{\tau} \|_2 + \frac{1}{|\mathcal{C}_1|} \sum_{\mathbf{f}_i, \mathbf{f}_j \in \mathcal{C}_1} \| \frac{\mathbf{f}_i}{\tau} - \frac{\mathbf{f}_j}{\tau} \|_2$$

- $\mathcal{L}_{inter}$ is the *Inter-class Separability*, which maximizes between-class distances using cosine similarity:

$$\mathcal{L}_{inter} = \max(0, \cos(\mu_0, \mu_1) + \text{margin})$$

  where, μ_0 and μ_1 are class centroids

- $\mathcal{L}_{var}$ is the *variance regularization*, which controls within-class variance:

$$\mathcal{L}_{var} = \text{Var}(\mathcal{C}_0) + \text{Var}(\mathcal{C}_1)$$

3.4.2. Focal Binary Cross-Entropy Loss

To address class imbalance, we employed focal loss, which down-weights well-classified examples:

$$\mathcal{L}_{focal} = -\alpha (1 - p_t)^\gamma \log(p_t)$$

where p_t is the model's estimated probability for the true class, α balances positive/negative examples, and γ focuses learning on hard examples.

3.4.3. Combined Loss Function

The total loss function combines both components:

$$\mathcal{L}_{total} = \mathcal{L}_{focal} + \mathcal{L}_{margin}$$

Specifically for Nucleotide Transformer, we employed a more aggressive variant of Focal Binary Cross-Entropy Loss with parameters $\alpha = 0.8$ and $\gamma = 3.0$ to focus even more on learning on hard-to-classify examples, along with a *confidence regularization* with penalty 0.1 to penalize uncertain predictions and encourage confident decisions.

3.5. Training Configuration and Optimization

3.5.1. Hyperparameter Settings

Training configurations were optimized for each model architecture separately and carefully to ensure the model is able to learn enough parameters from the training data:

A. Custom Transformer

1. *Learning rate*: $1e-3 = 0.001$
2. *Batch size*: 2 [since the training set is already resampled and contains equal number of samples from both classes, a batch size of 2 has probability that each batch contains sample from every class]
3. *Number of epochs*: 50
4. *Optimizer*: AdamW with weight decay $1e-2 = 0.01$.

B. DNABERT-6 Configurations

1. *Full fine-tuning*: lr $= 5e-7$ for the base model and lr $= 5e-6$ for the classifier head, batch size $= 4$
2. *Frozen*: lr $= 5e-4$, batch size $= 4$
3. *LoRA*: lr $= 1e-5$ for the base model and lr $= 1e-4$ for the classifier head, batch size $= 4$

All variants were trained for 30 epochs and using AdamW optimizer with weight decay 0.01.

C. Nucleotide Transformer

1. *Learning rate*: $1e-5$
2. *Batch size*: 4
3. *Number of epochs*: 50
4. *Optimizer*: Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$
5. *Learning Rate Scheduler*: Custom Scheduler with warm-up
6. *Loss Function*: Aggressive Focal Loss + Confidence Regularization
7. *Unfreezing Strategy*: Top 2 transformer layers + frozen classification head
8. *Dropout Rate*: 0.1 (classification head)
9. *Weight Initialization*: Normal distribution ($\mu = 0.0$, $\sigma = 0.02$) for classifier

3.5.2. Regularization and Optimization Strategies

Different regularization and optimization strategies were adapted during the training phase of all model variants to ensure the model doesn't overfit and later generalize well to the held-out test set data:

- *Dropout*: Applied at rates of 0.3 to prevent overfitting
- *Gradient Clipping*: Maximum gradient norm of 1.0 to ensure training stability by avoiding the challenge of exploding gradient
- Optimizer: For Custom Transformer and DNABERT-6, `AdamW` optimizer with respective learning rate and weight decay rate of $1e-2$. For Nucleotide Transformer, `Adam` optimizer with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$
- *Learning Rate Scheduler*: `ReduceLROnPlateau` with `mode='min', factor=0.3, patience=3, min_lr=1e-6`. Custom Scheduler for Nucleotide Transformer similar to what it was used to pre-train.

3.6. Evaluation Metrics

Below performance metrics were used to evaluate all the trained models, where TP, TN, FP, FN represents the number of samples that are truly positive, truly negative, false positive and false negative respectively.

- *Accuracy*, denoting overall classification correctness:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Precision*, indicating positive predictive value, which is crucial for clinical applications:

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

- *Recall (Sensitivity).* denoting True Positive Rate, which is important for detecting resistance:

$$\text{Recall (Sensitivity or TPR)} = \frac{TP}{TP + FN}$$

- *F1-Score*: Harmonic mean of precision and recall:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- *ROC-AUC*: ROC (receiver operating characteristic) Curve plots the True Positive Rate (Recall) vs. False Positive Rate (FPR) at various threshold levels, whereas AUC (Area Under Curve) measures the entire two-dimensional area underneath the ROC curve.

$$\text{FPR} = \frac{FP}{FP + TN}$$

$$\text{AUC} = \int_0^1 \text{TPR}(x)dx$$

ROC-AUC doesn't have a closed-form formula; it's computed numerically from model scores using algorithms like the trapezoidal rule.

3.7. Computational Environment

Experiments related to Custom Transformer and Pre-trained DNABERT 6 were conducted on Google Colab and Experiment for Nucleotide on Kaggle using:

1. Hardware: Tesla T4 GPU with 14-15 GB memory
2. Software: Python 3.11.13, PyTorch 2.6.0, Transformers 4.52.4
3. Reproducibility: Fixed random seeds (42) across all experiments for consistent results

4. Results

4.1. Dataset Characteristics and Data Pre-processing Outcomes

Our study utilized two datasets: 150 *pbp4* gene sequences from *Staphylococcus aureus* isolates tested against *cefoxitin*. Table 1 shows the class distribution of the original data before any pre-processing and data splitting.

Dataset	**Split**	**Total**	**Non-Resistant**	**Resistant**
S. aureus	Training	135	37	98
	Test	15	3	12
		150	**40**	**110**

Table 1: Class distribution of original dataset

After the data loading, we employed two different strategies to handle class imbalance based on the model. For our custom transformer, which we built from scratch, we resampled the training data to make the number of non-resistant and resistant samples equal, whereas for DNABERT-6 and Nucleotide Transformer, we assigned class weights to both class to change their weightage during training phase. We also split the training set to form a separate validation set with 20% training samples, for evaluating the trained model after each epoch. The initial test set was held out without any change for the final evaluation after complete training; this enabled us avoid the problem of data leakage. Table 2 shows the class distribution of all the train datasets after our strategic data pre-processing.

Dataset	**Split**	**Total**	**Non-Resistant**	**Resistant**
Custom Transformer (Re-sampled)	Training	156	78 (50.00%)	78 (50.00%)
	Validation	40	20 (50.00 %)	20 (50.00%)
	Test	15	3 (20.0%)	12 (80.0%)
DNABERT-6, Nucleotide Transformer (Class-weight)	Training	108	30 (27.78%)	78 (72.22%)
	Validation	27	7 (25.93 %)	20 (74.07%)
	Test	15	3 (20.0%)	12 (80.0%)

Table 2: Class distribution of different datasets after strategic splitting

4.2. Model Architecture Specifications

The architectural specifications and parameter counts for all evaluated models are summarized in Table 3.

Model Configuration	**Total Params**	**Trainable Params**
Custom Transformer	233,473	233,473
DNABERT-6 Full Fine-tuning	89,191,681	89,191,681
DNABERT-6 Frozen Backbone	89,191,681	769
DNABERT-6 LoRA (r=16)	89,781,505	590,593
Nucleotide Transformer Partial Frozen	485,701,868	39,357,442

Table 3: Total and trainable parameter counts of all models

4.3. Test Set Performance Evaluation

After the model were trained and evaluated on the validation sets (during training) for several epochs, it was evaluated on held-out test sets. The comprehensive test set results are presented in Table 4.

Model	**Data**	**Acc.**	**Prec.**	**Rec.**	**F1**	**ROC-AUC**
Custom Transformer	Test	0.7333	1.000	0.6667	0.8000	1.0000
	Val	0.7333	1.000	0.6667	0.8000	1.0000
DNABERT-6 Full	Test	1.0000	1.0000	1.0000	1.0000	1.0000
	Val	0.8889	0.8696	1.0000	0.9302	0.6929
DNABERT-6 Frozen	Test	0.8000	0.8000	1.0000	0.8889	0.6667
	Val	0.7778	0.7692	1.0000	0.8696	0.5214
DNABERT-6 LoRA	Test	0.9333	1.0000	0.9167	0.9565	1.0000
	Val	0.9630	0.9524	1.0000	0.9756	0.8929
Nucleotide Transformer	Test	0.9330	1.0000	0.9170	0.9570	1.0000
	Val	0.9630	0.9520	1.0000	0.9760	0.9710

Table 4: Results of performance evaluation of all trained models on both test and validation set

4.4. Loss Function Analysis

A combination of focal binary cross-entropy loss and max-margin loss was employed across model configurations to address class imbalance and improve learning dynamics. The effectiveness of different loss components was analyzed during training. Final epoch loss breakdowns on training set are presented in Table 5.

Model	Total Loss	Focal BCE	Max-margin
Custom Transformer	0.2871	0.2467	0.0404
DNABERT-6 Full Fine-tune	1.3760	0.0255	1.3505
DNABERT-6 Frozen	4.6217	0.0086	4.6131
DNABERT-6 LoRA	1.3828	0.0115	1.3713

Table 5: Loss component breakdown on training set at final training epoch. For Nucleotide Transformer, we have used aggressive focal loss function with $\gamma = 3.0$ and $\alpha = 0.8$.

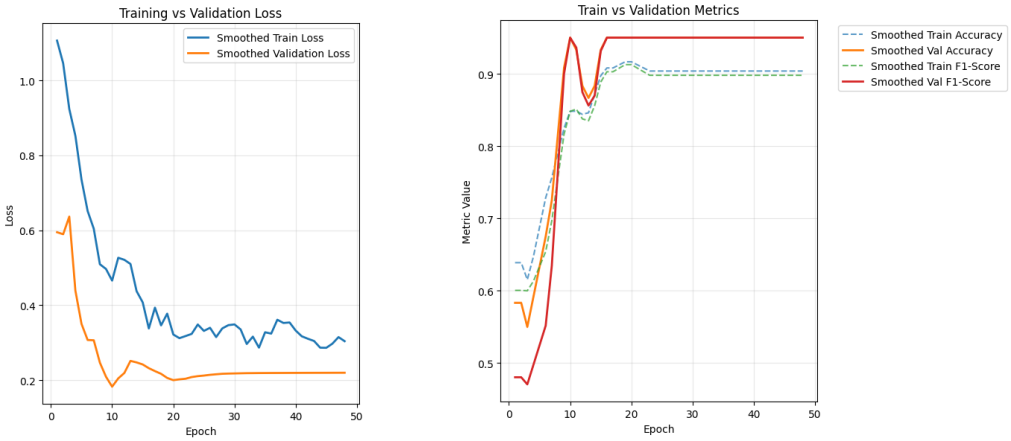
4.5. Classification Performance Analysis

Detailed classification performance metrics including confusion matrix analysis are presented below for each of the model variant.

4.5.1. Custom Transformer

Figure 2a shows the change in total loss (combination of focal binary cross-entropy and max-margin) for both train and validation data during training phase (over epochs). We can see the validation loss flattening early, indicating the model reached the "best generalization" point quickly. However, the training loss continues to go down with certain hiccups, might indicating a mild memorization. But the gap between training and validation loss is small. So likely there is no major overfitting – which should be because of the lightweight architecture.

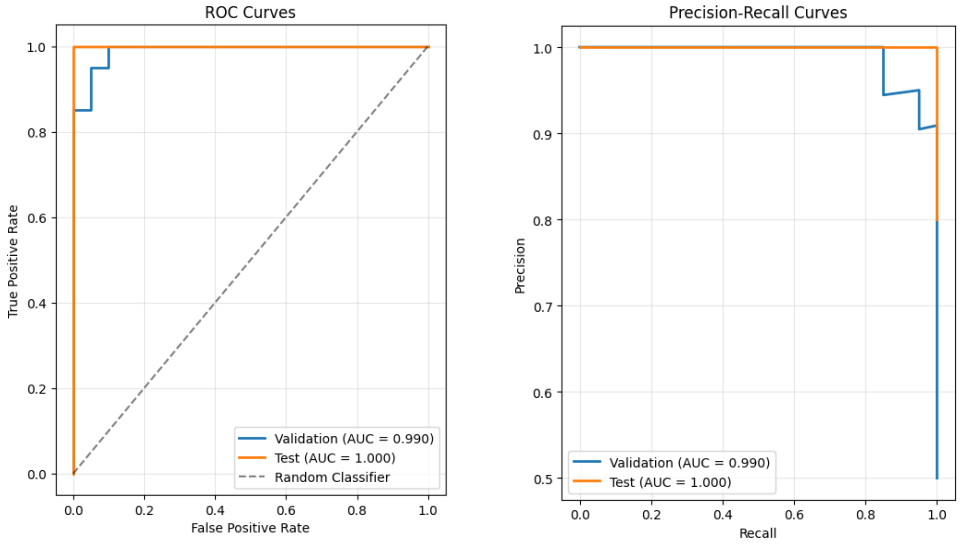
Figure 2b shows the change in evaluation metrics (accuracy, F1-score) for both train and validation data during training phase (over epochs). As we can see, both accuracy and F1-score start low and rapidly improve in the first 10 epochs, then stabilize around $0.9 - 0.95$. Both training and validation metrics following same pattern indicates good model generalization without major overfitting.



(a) Change of Training and Validation Loss over Epochs

(b) Change of Accuracy and F1 in both Train and Validation set during Training

Figure 2: Custom Transformer: Change of total loss and evaluation metrics (accuracy, F1-score) for both train and validation data during Training



(a) ROC-Curve

(b) Precision-Recall Curve

Figure 3: Custom Transformer: ROC curve and Precision-Recall curve

Figure 3 shows the ROC-curve and Precision-Recall curve. Both precision-recall curve and ROC curve shows near-perfect behavior, possibly because of the comparatively low number of trainable parameters, it could easily learn

all the patterns.

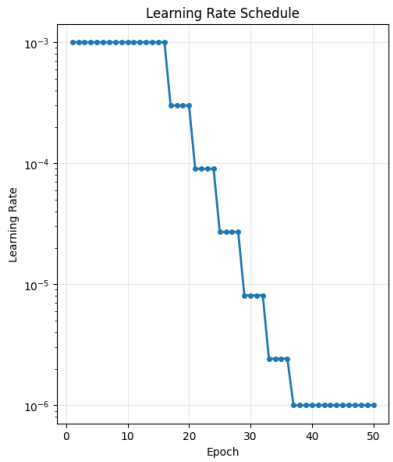


Figure 4: Custom Transformer: Change of Learning Rate during training

Figure 4 shows how the learning rate reduces over epochs due to the use of `ReduceLROnPlateau` as a learning rate scheduler. The learning rate was changed after 3 epochs and the minimum value was $1e-6$.

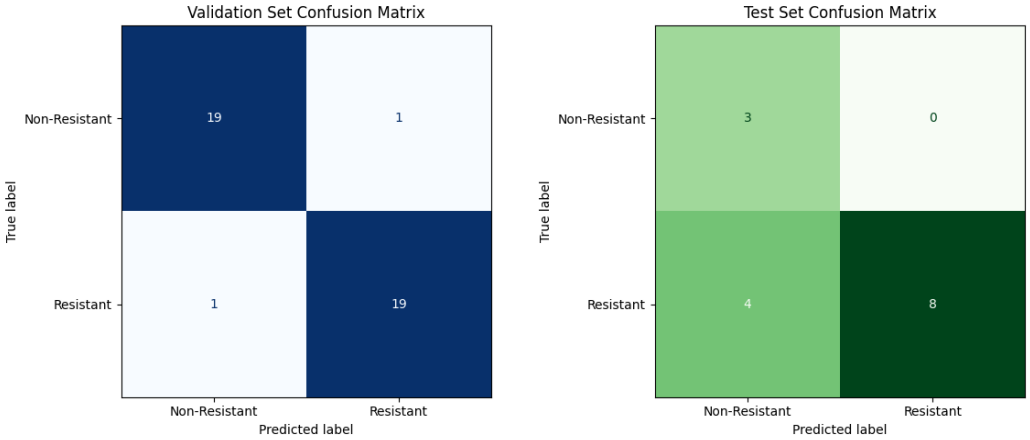
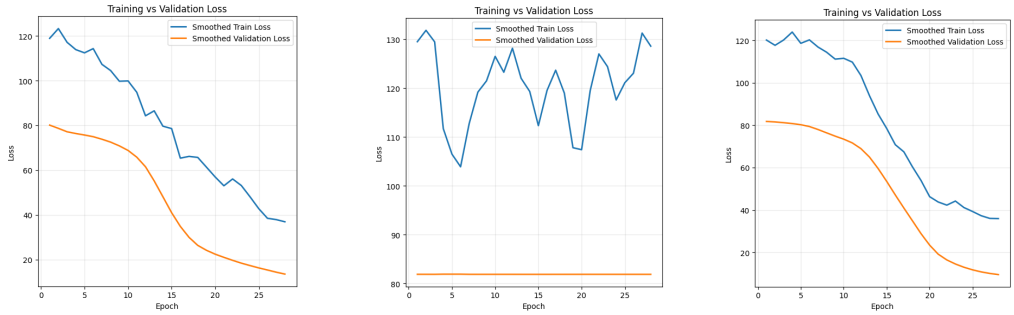


Figure 5: Custom Transformer: Confusion Matrix in both Validation and Test Set

Figure 5 shows the confusion matrix for both validation and test set after the completion of the training phase. We can see there are $\sim 25\%$ misclassifications, despite the PR and ROC curve shows good generalization. This

is because the confusion matrix uses a fixed threshold (0.5), while PR/ROC curves evaluate across all thresholds. Few of the misclassified samples may have probabilities very close to decision boundary. This suggests the model has good probability calibration and ranking ability, even though the default threshold of 0.5 isn't optimal for this specific dataset.

4.5.2. DNABERT-6 (in all different settings)

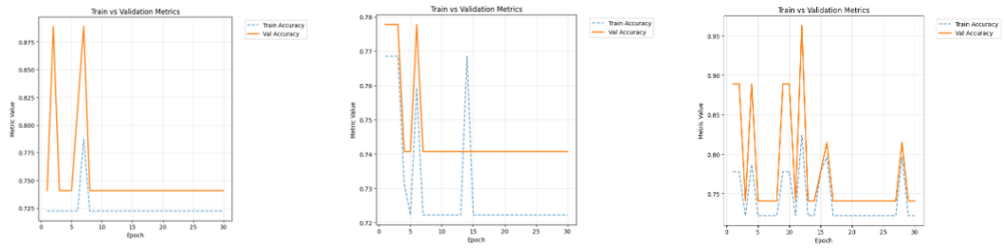


(a) Full fine-tune (b) Frozen mode (c) LoRA adapted

Figure 6: DNABERT-6: Change of total loss for both train and validation data during Training

Figure 6 shows the change of total loss in both training and validation set in three different settings of DNABERT-6. In full fine-tune mode and low-rank adapted mode, we see that both training loss and validation loss is reducing steadily until the end of the training phase. Moreover, the gap between them is also more or less stable, at least not converging, which possibly indicates a good learning. Whereas in frozen mode, we see an oscillating train loss with flat validation loss, which can be expected in frozen backbone with no representation learning as it gives fixed embeddings from the pre-training on a different task. And since we had a simple linear classifier head, most likely the model didn't have enough parameters to learn any meaningful pattern.

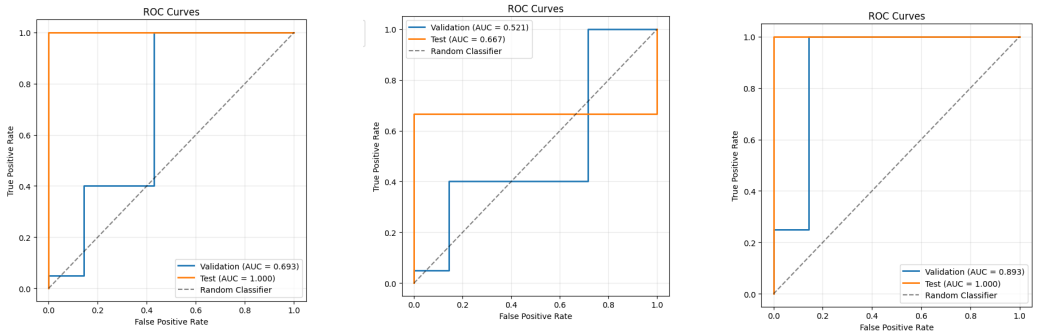
Figure 7 shows the change in accuracy in both training and validation dataset as the training progresses. In full fine-tune mode, we see early spikes in validation accuracy which may indicate the small dataset we have, as even a few sample flips may cause big accuracy swings. Moreover, both metrics are plateauing overall, which can be treated as a sign that the model is probably data-constrained, not underfitting or overfitting hard. In frozen mode, we



(a) Full fine-tune (b) Frozen mode (c) LoRA adapted

Figure 7: DNABERT-6: Change of accuracy for both train and validation data during Training

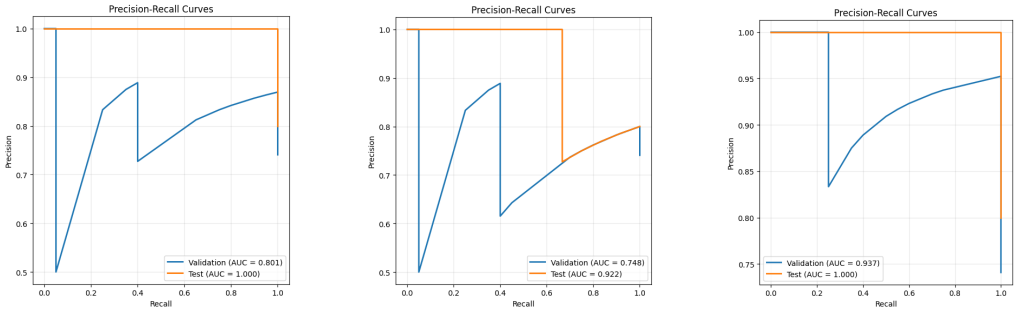
see that both training and validation accuracies are stuck at very low levels (0.72 – 0.74) with some erratic spikes but no sustained improvement. In the low-rank adapted mode as well, we see sudden spikes in both validation and test accuracy. This can be attributed to the Low-rank adaptation. As LoRA updates only a small slice of the full model and it introduces additional trainable projections, it can create sharp local minima or irregular gradients, eventually making the accuracy oscillating.



(a) Full fine-tune (b) Frozen mode (c) LoRA adapted

Figure 8: DNABERT-6: ROC curve in three different settings

Figure 8 and figure 9 show the ROC curve and precision-recall curve respectively for DNABERT-6 in all different settings. While both full fine-tune mode and low-rank adapted mode shows good results, in frozen mode ROC curve (figure 8b), we can see that it is barely performing better than a random classifier.



(a) Full fine-tune (b) Frozen mode (c) LoRA adapted

Figure 9: DNABERT-6: Precision-Recall curve in three different settings

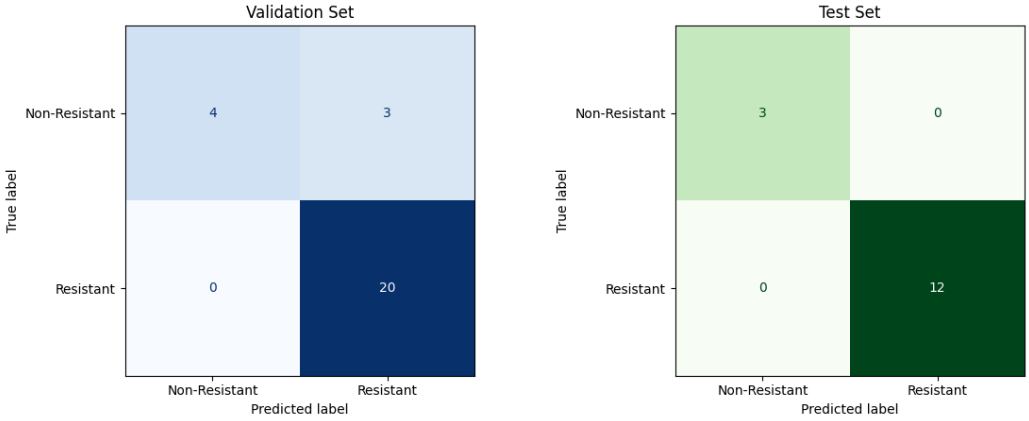


Figure 10: DNABERT-6: Confusion Matrix in full fine-tune mode

Figure 10, Figure 11, and Figure 12 represents the confusion matrix of DNABERT-6 in both validation and test dataset in full fine-tune mode, frozen mode, and LoRA-adapted mode respectively.

In full fine-tune mode (figure 10), DNABERT-6 performs perfectly, classifying all the test samples accurately while having a few misclassifications in validation set. While 100% accuracy on test set is desirable, there may be a challenge here. Since the test set is very small = 15, it is also likely that it predicts 100% with accuracy by chance and evaluating on a bigger test set will give us more confidence on how well our model learned from the fine-tuning.

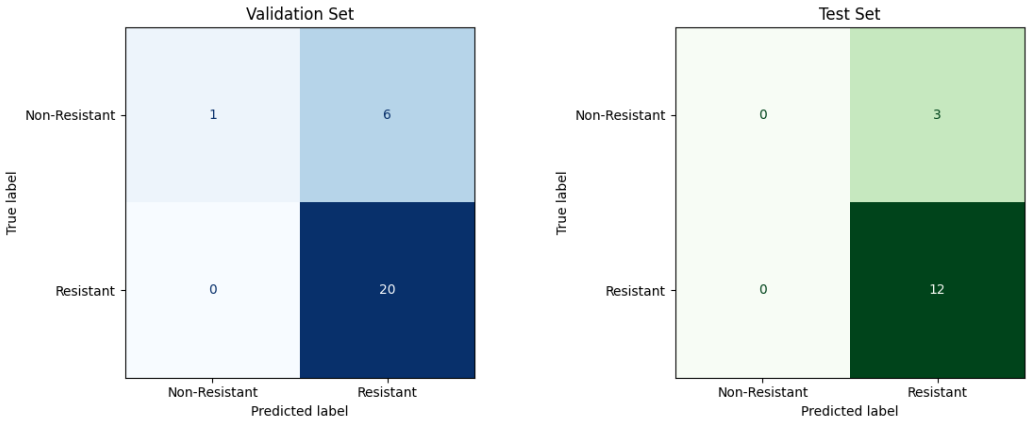


Figure 11: DNABERT-6: Confusion Matrix in frozen mode

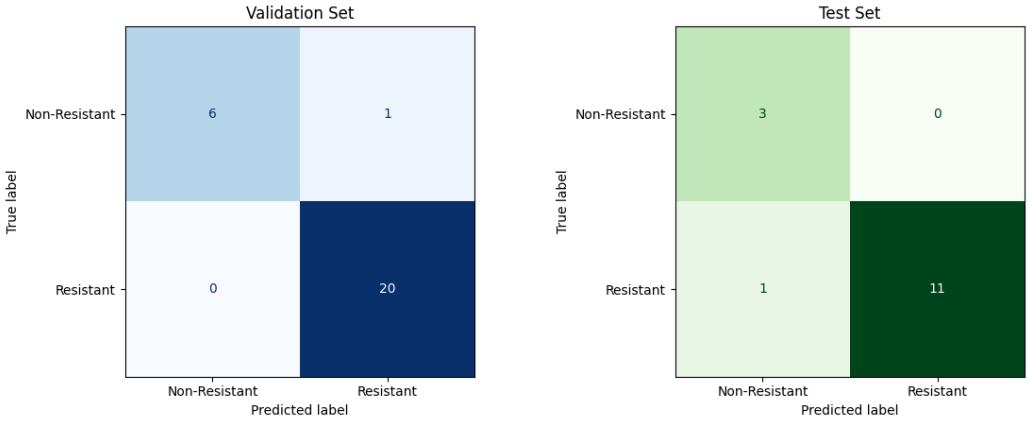


Figure 12: DNABERT-6: Confusion Matrix in LoRA adapted mode

In frozen mode (figure 11), we see a "collapsed prediction" - the model is essentially predicting one class for everything. This is a clear sign that there are too few trainable parameters for the model to learn meaningful patterns from our specific dataset.

This result in frozen mode actually makes sense. While DNABERT-6 was pre-trained on genomic data, it wasn't trained specifically for tasks like antimicrobial resistance prediction. The high-level features it learned during pre-training might not be directly applicable to our specific task without some adaptation of the deeper layers. The frozen approach works well in

natural language processing when the pre-training task is very similar to the target task, but in genomics, the gap between general DNA understanding and specific phenotype prediction might be too large to bridge with just a linear classifier. This validates the need to adapt the pre-trained model to our specific task in hand, but not at the cost of too much computational resource.

In low-rank adapted mode (figure 12), we see a balanced result. In both validation and test set, the model fails to misclassify only 1 sample, which is $\sim 3.7\%$ for validation set and $\sim 6.3\%$ in test set. This performance is reasonable and actually shows the efficiency of using parameter-efficient fine-tuning (PEFT) techniques like low-rank adaptation.

4.5.3. Nucleotide Transformer

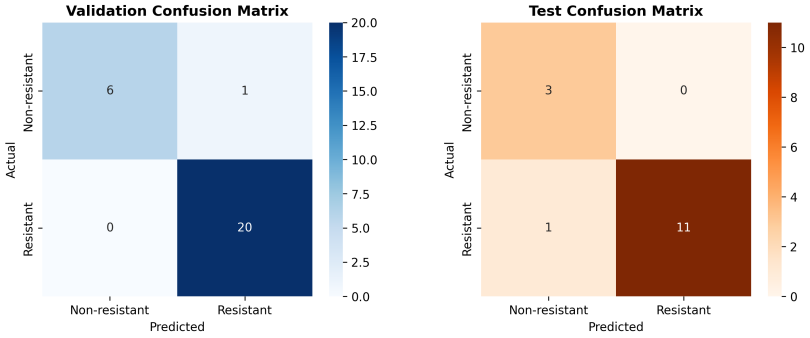
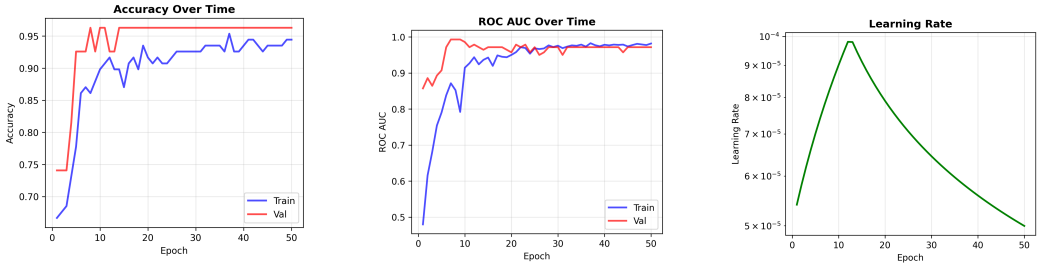


Figure 13: Nucleotide Transformer: Confusion Matrix in both Validation and Test Set

Figure 13 shows the confusion matrix for both validation and test set after the completion of the training phase. Just like DNABERT-6 in low-rank adapted mode, we see only 1 misclassified sample in both validation and test set.

Both accuracy in figure 14a and AUC in figure 14b show consistent improvement and plateau around epoch 12-14, indicating the model learns effectively. What's particularly interesting is that validation AUC starts at about 0.89 from epoch 1, indicating that the pre-trained features are immediately useful for our task. This is a strong sign that the pre-training was effective. In figure 14c, we can see that because of the custom learning rate scheduler using warm-up, the learning rate initially increases until epoch 12 when both accuracy and ROC-AUC achieves stability. Beyond that, the learning rate



(a) Accuracy over time (b) ROC AUC over time (c) Learning rate over time

Figure 14: Nucleotide Transformer

starts decaying till the last epoch.

5. Discussion

5.1. Principal Findings and Model Performance Comparison

Our comprehensive evaluation of multiple transformer architectures for antimicrobial resistance prediction identified DNABERT-6 (in full fine-tune mode) as the top performer, achieving an accuracy of 100% and F1-score of 1.00 for *pbp4*-based resistance prediction in *S. aureus*. However, given the small dataset that we had for training the model, this data seemed to be "too good to be true". Training the model on a larger and more robust dataset will possibly give us more confident and reliable result. The superior performance of the DNABERT-6 model can be attributed to its pre-trained genomic embeddings [11] and effective fine-tuning strategy, which balanced parameter adaptation to our specific task [13].

Parameter-efficient methods also performed well. LoRA emerged as a balanced solution, combining task-specific adaptability with the efficiency and stability of parameter-efficient tuning [17]. Its low-rank constraint acted as a regularizer, limiting overfitting while enabling sufficient adaptation, with the optimal rank 16 striking an effective trade-off, achieving and 93.33% accuracy and $\sim$ 96% F-1 score with just 590,593 trainable parameters (152x less than full fine-tune).

Nucleotide Transformer in partial frozen mode give reasonable performance achieving 93.30% accuracy and $\sim$ 0.96 F-1 score, which is much better than

DNABERT-6 (frozen mode) and comparable to LoRA-adapted DNABERT-6, which can be attributed to the fact that it allows larger sequence length, it is pretrained on diverse human reference genome and different tokenization method.

5.2. Class Imbalance Mitigation Strategies

Our approach to handle class imbalance, integrating max-margin and focal BCE loss with class weights, effectively addressed skewed ratio of $4 : 11$ in training dataset. Unlike standard classification losses focused on probability calibration, max-margin loss promotes class separation in feature space for more robust decision boundaries whereas focal loss gives more importance to hard-to-classify samples.

The combined formulation of max-margin loss, capturing intra-class compactness, inter-class separability, and variance regularization, was especially effective. Temperature scaling ($\tau = 0.1$) in the intra-class loss enabled precise control of compactness. The variance regularization term prevented intra-class fragmentation, with $\beta = 0.1$ empirically tuned to balance performance and representation quality. For the Nucleotide Transformer, aggressive focal loss ($\gamma = 3.0$) outperformed standard versions, and confidence regularization ($\lambda = 0.1$) reduced overconfident misclassifications, a critical situation in clinical contexts.

The introduction of specialized loss functions for genomic sequence classification also represents a methodological advancement over previous work, which typically relied on standard classification losses.

6. Conclusion

This study systematically evaluates and compares different modern transformer architectures for predicting antimicrobial resistance from genomic sequences, tackling a critical medical challenge [1, 3, 2]. By comparing custom and pre-trained models alongside novel class imbalance handling and parameter-efficient fine-tuning, we demonstrate notable improvements in prediction accuracy and computational efficiency for AMR classification.

6.1. Methodological Innovations and Broader Scientific Impact

Beyond antimicrobial resistance prediction, this work offers methodological advances for computational biology. Our systematic comparison of transfer learning strategies revealed that LoRA adapted methods can effectively trade-off between successful adaption of pre-trained embeddings to our specific task and the risk of overfitting. This is crucial where labeled data are scarce but pre-trained models exist.

Combining focal loss with max-margin objectives effectively tackled class imbalance, feature space structuring, and decision boundary refinement, offering a promising framework for imbalanced sequence classification [18, 19].

6.2. Clinical and Public Health Implications

Parameter-efficient methods benefit resource-limited settings by enabling effective resistance prediction with modest computational needs. This supports the global efforts to combat antimicrobial resistance through democratizing advanced diagnostics.

6.3. Limitations and Methodological Considerations

Several limitations should be noted. First, the dataset of 150 sequences for *S. aureus* is extremely limited, which typical for genomic studies but restricts generalization. In general, transformer architectures need thousands of data to learn meaningful patterns during the training phase. The high performance observed may not translate to larger, more diverse datasets or different bacterial species without additional validation.

Computational resources has been a constant challenge during the course of this work. Training even a single transformer architecture in full fine-tune mode, requiring the model to learn hundreds of millions of parameters require an extensive computational resource, which we lacked.

6.4. Future Research Directions

Several promising research directions emerge from this work:

Multi-modal Integration: Combining genomic sequence with protein structural information, gene expression profiles, or clinical metadata could help

accurate prediction of resistance mechanism, which were not apparent from sequence data alone.

Uncertainty Quantification: Incorporating uncertainty estimation could improve clinical decision-making by flagging low-confidence predictions, especially in borderline cases.

Ensemble Approaches: Exploring ensemble methods combining multiple transformer architectures may boost performance by leveraging complementary strengths and capturing diverse resistance patterns.

Acknowledgments

We thank **Prof. Dr. Alice McHardy** of Helmholtz-Zentrum für Infektionsforschung (HZI) for this seminar, giving us the opportunity to learn on real-life data and enhance our skills. We also extend our sincere gratitude to **Dr. Mohammad Hadi Foroughmand Araabi** and **Mr. Georgios Kallergis** for constant guidance and for providing several suggestions on possible improvement areas. We also thank them and Helmholtz-Zentrum für Infektionsforschung for providing us the dataset used in this work.

We acknowledge the use of computational resources provided by Google Colab and Kaggle platforms, which enabled the training and evaluation of the deep learning models presented in this work. The availability of these cloud-based computing environments was essential for conducting the experiments for this study, which required extensive computational resources.

We acknowledge the developers of PyTorch, Transformers library by Hugging Face, and the DNABERT-6 and Nucleotide Transformer models, which formed the foundation of our experimental framework.

We acknowledge the use of artificial intelligence tools, such as Claude (Anthropic) and Gemini API integrated in Google Colab environment. These tools have helped us in debugging code snippets, verifying mathematical formulas, clarifying technical concepts (e.g., Transfer Learning, Focal Loss etc.), structuring initial report (organizing order of different sections) and identifying relevant papers. However, all scientific judgments, interpretations, and

conclusions presented in the report are solely our responsibility.

Finally, we acknowledge the broader scientific community working on antimicrobial resistance research, whose foundational work had made several advancement in this field possible. The collaborative nature of open science continues to accelerate progress in addressing global health challenges.

References

[1] World Health Organization, Antimicrobial resistance: global report on surveillance (2014).
URL https://www.who.int/publications/i/item/9789241564748

[2] Antimicrobial Resistance Collaborators, Global burden of bacterial antimicrobial resistance in 2019: a systematic analysis, The Lancet 399 (10325) (2022) 629–655. doi:10.1016/S0140-6736(21)02724-0.
URL https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(21)02724-0/fulltext

[3] World Health Organization (WHO), Antimicrobial resistance (2023).
URL https://www.who.int/news-room/fact-sheets/detail/antimicrobial-resistance

[4] Centers for Disease Control and Prevention, Antibiotic resistance threats in the united states, 2019 (2019).
URL https://ndc.services.cdc.gov/wp-content/uploads/Antibiotic-Resistance-Threats-in-the-United-States-2019.pdf

[5] O. B. Jonas, A. Irwin, F. C. J. Berthe, F. G. Le Gall, P. V. Marquez, Drug-resistant infections: a threat to our economic future (vol. 2 of 2): final report (2017).
URL http://documents.worldbank.org/curated/en/323311493396993758

[6] S. Y. C. Tong, J. S. Davis, E. Eichenberger, T. L. Holland, V. G. Fowler, Staphylococcus aureus infections: Epidemiology, pathophysiology, clinical manifestations, and management, Clinical Microbiology Reviews 28 (3) (2015) 603–661. doi:10.1128/CMR.00134-14.
URL https://journals.asm.org/doi/10.1128/cmr.00134-14

[7] C. J. Fernandes, L. A. Fernandes, P. Collignon, A. G. on Antimicrobial Resistance, Cefoxitin resistance as a surrogate marker for the detection of methicillin-resistant staphylococcus aureus, Journal of Antimicrobial Chemotherapy 55 (4) (2005) 506–510. doi:10.1093/jac/dki052.
URL https://europepmc.org/article/med/15743899

[8] S. M. Hamilton, J. A. N. Alexander, E. J. Choo, L. Basuino, T. M. da Costa, A. Severin, M. Chung, S. Aedo, N. C. J. Strynadka, A. Tomasz, S. S. Chatterjee, H. F. Chambers, High-level resistance of staphylococcus aureus to -lactam antibiotics mediated by penicillin-binding protein-4 (pbp4), Antimicrobial Agents and Chemotherapy 61 (6) (2017). doi:10.1128/AAC.02727-16.
URL https://pubmed.ncbi.nlm.nih.gov/28373193/

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, I. Polosukhin, Attention is all you need, Vol. 30, 2017, pp. 5998–6008.
URL http://arxiv.org/abs/1706.03762

[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186. doi:10.18653/v1/N19-1423.
URL https://aclanthology.org/N19-1423/

[11] Y. Ji, Z. Zhou, H. Liu, R. V. Davuluri, DNABERT: pre-trained bidirectional encoder representations from transformers model for dna-language in genome, Bioinformatics 37 (15) (2021) 2112–2120. doi:10.1093/bioinformatics/btab083.
URL https://academic.oup.com/bioinformatics/article-pdf/37/15/2112/57195892/btab083.pdf

[12] H. Dalla-Torre, L. Gonzalez, J. Mendoza-Revilla, N. Lopez Carranza, A. H. Grzywaczewski, F. Oteri, C. Dallago, E. Trop, B. P. de Almeida, H. Sirelkhatim, G. Richard, M. Skwark, K. Beguir, M. Lopez, T. Pierrot, Nucleotide transformer: building and evaluating robust foundation

models for human genomics, Nature Methods 22 (2) (2025) 287–297. doi:10.1038/s41592-024-02523-z.
URL https://www.nature.com/articles/s41592-024-02523-z

[13] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2010) 1345–1359. doi:10.1109/TKDE.2009.191.
URL https://ieeexplore.ieee.org/document/5288526/

[14] Y. LeCun, Y. Bengio, G. E. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.
URL https://www.nature.com/articles/nature14539

[15] G. Arango-Argoty, E. Garner, A. Pruden, L. Heath, P. Vikesland, L. Zhang, Deeparg: A deep learning approach for predicting antibiotic resistance genes from metagenomic data, Microbiome 6 (23) (2018). doi:10.1186/s40168-018-0401-z.
URL https://microbiomejournal.biomedcentral.com/articles/10.1186/s40168-018-0401-z

[16] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training (2018).
URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, arXiv preprint arXiv:2106.09685 (2021).
URL https://arxiv.org/abs/2106.09685

[18] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, Focal loss for dense object detection, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2999–3007. doi:10.1109/ICCV.2017.324.
URL https://ieeexplore.ieee.org/document/8237586

[19] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297. doi:10.1007/BF00994018.
URL https://link.springer.com/article/10.1007/BF00994018