Genomic Data Classification via Universal Compression

Yasmine Omri

yomri@stanford.edu

Stanford University https://orcid.org/0009-0004-8444-2486

Naomi Sagan

Stanford University

Eugene Min

Stanford University

Heewoong Choi

Seoul National University https://orcid.org/0000-0002-4533-5715

Taesup Moon

Seoul National University

Tsachy Weissman

Stanford University

Article

Keywords: lossless compression, DNA classification, LZ78, Sequential Probability Assignment

Posted Date: April 9th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-6363017/v1

Additional Declarations: There is **NO** Competing Interest.

Genomic Data Classification via Universal Compression

Yasmine Omri[1*†], Naomi Sagan[1*†], Eugene Min[1*†], Heewoong Choi[2], Taesup Moon[2], Tsachy Weissman[1]

[1*]Electrical Engineering, Stanford University.
[2*]Electrical and Computer Engineering, Seoul National University.

*Corresponding author(s). E-mail(s): yomri@stanford.edu; nsagan@stanford.edu; eugenecm@stanford.edu;
Contributing authors: chw0501@snu.ac.kr; tsmoon@snu.ac.kr; tsachy@stanford.edu;
†These authors contributed equally to this work.

Abstract

Efficient and accurate DNA sequence classification is a crucial task in genomic data analysis. In this work, we construct a lightweight DNA classifier based on the LZ78 lossless universal compressor, and optimize its performance through hyperparameter tuning. This classifier outperforms the state-of-the-art DNABERT-2 model on the Genomic Understanding Evaluation suite, while drastically reducing computational costs. Unlike DNABERT-2, which requires two weeks of multi-GPU training, our classifier can be trained in about 30 minutes or less on a modern CPU with a fraction of the training data. It also offers up to 128× inference time speedup. These results highlight the potential of LZ78 for scalable and efficient genomics classification, particularly in resource-constrained environments. Additionally, we open-source our pipeline for compression-based classification. Future work aims to enhance its robustness and extend its applicability to more complex genomic tasks.

Keywords: lossless compression, DNA classification, LZ78, Sequential Probability Assignment

1 Introduction

In the context of genomic data analysis, DNA classification is the process of assigning a representative class to an input DNA sequence, composed of nucleotides from the alphabet A, C, G, or T. Protein classification, which may be performed at the amino acid level, is a closely related task. Efficient and accurate DNA classification is a crucial task in genomic data analysis. By providing a systematic framework for the taxonomy of species, DNA classification enables the study of biodiversity and the discovery of evolutionary relationships among organisms [1]. In medicine, it facilitates the identification of disease markers and the influence of genetic variations on drug responses, thus driving advancements in diagnostics, pharmacogenomics, and the development of safer more effective treatments [2]. Examples of DNA classification tasks include identification of functional gene regions such as promoters or enhancers, detecting genetic mutations, classifying viral variants, and taxonomic classification of organisms.

Numerous DNA classification techniques exist, including information theoretic methods, classical machine learning, and, in recent years, deep learning-based approaches. For instance, various ML algorithms have made considerable strides in DNA classification tasks, ranging from traditional classifiers to deep neural networks and billion-parameter large language models [1, 3].

Prior studies have explored various classifiers, including neural networks, KNNs, decision trees, SVMs [4], and CNN-based techniques [5] for DNA classification, including virus classification and disease detection. Deep learning models, such as CNNs, RNNs, LSTMs, and SSMs, have been employed on particularly advanced classification tasks, with applications in gene region identification, transcription factor binding site prediction, and viral genome classification [6, 7, 8, 9].

Recently, transformer-based models have driven major breakthroughs in computational genomics [3], setting new state-of-the-art performance in genomic data classification. Inspired by LLMs, Genomic Language Models (gLMs) leverage unsupervised pre-training on large genomic datasets followed by finetuning for DNA classification. Notable gLMs include AgroNT, GPN, GeneBERT, Nucleotide Transformers, DNABERT, and DNABERT-2 [10, 11, 12, 13, 14].

Although machine learning techniques can achieve high classification accuracies, they require orders of magnitude more compute than, *e.g.*, compression-based techniques. For example, although DNABERT-2, which maintains state-of-the-art performance in DNA classification, is considered to be parameter efficient, it remains very computationally expensive. During pre-training, where LLMs learn to encode or generate genomic language, DNABERT-2 was trained on tens of GBs of unlabeled data, and required 14 days of multi-GPU training.[1] In order to use the foundation model for classification, an additional phase of fine-tuning is required, which typically takes at least a few additional hours.

Thus, leveraging compression for DNA classification promises very attractive advantages in terms of both effectiveness and efficiency, particularly for compute-constrained settings.

[1] Specifically, 8 NVIDIA RTX 2080Ti GPUs, each of which contains 11GB of GDDR6 memory.

In particular, the ideas behind universal compression lend themselves well to genomic classification. The quintessential universal compressor is Lempel-Ziv 78 (LZ78) [15], which builds a prefix tree over the input data via an incremental parsing procedure (see Section 4 for details). LZ78 is universal in the sense that, for any input data, it asymptotically compresses at least as well as the best finite-state compressor. The related LZ77 [16] scheme compresses by searching for long repeated substrings. Though LZ77 lacks the strong universality results of LZ78, it performs well in practice and is extensively used in state-of-the-art compressors (*e.g.*, GZIP and ZSTD). [17] interprets the universality of LZ78 via a prefix-tree-based probability model. This probability model has formed the basis of many universal schemes, *e.g.*, in gambling [18], sequence prediction [19], and filtering [20].

From a probability modeling perspective, sequence classification is equivalent to constructing a probability model for each class, and subsequently determining which distribution best fits the test sequence. As such, universal compressors and probability models can induce universal classifiers by acting as measures of distance between two distributions. One key example is Ziv-Merhav cross-parsing [21], which constructs a universal relative entropy measure via the minimum number of phrases a test sequence can be parsed into using matches from the training data.

The idea of harnessing LZ78-based compression for genomic classification has previously appeared in the literature. [22] studies the performance of several variable-order Markov models on different next-symbol-prediction and sequence classification tasks, including protein classification. In this case, an LZ78-based classification outperformed the other methods considered over all classes. More recently, [23, 24] applies an LZ78-based normalized relative compression (NRC) measure to DNA classification, building an LZ78 prefix tree for the training samples of each class and measuring the number of phrases a test sequence parses into using each such tree. [25] takes a different approach to LZ78-based DNA analysis: a set of DNA sequences are all independently parsed into LZ78 phrases, and a graph is built where the edges represent adjacent phrases.

Our Method

In this work, we present a highly efficient lossless compression-based approach to DNA classification, highlighting its potential as a lightweight alternative to computationally intensive models like DNABERT-2. We leverage the LZ78 universal compression algorithm to construct an adaptive classification scheme, and demonstrate its promising performance and dramatic computational savings. In comparison to existing LZ78-based methods, e.g., [22, 24, 23], our classifier includes several methodological improvements that significantly boost accuracy. It leverages a rich family of universal sequential probability assignments (SPAs) based on LZ78 and Bayesian mixtures [26], which provides degrees of freedom that are not available in other LZ-based approaches. We provide a comprehensive hyperparameter sweeping framework to take advantage of these degrees of freedom. As shown in Figure 3, the optimal parameters are dataset-dependent, necessitating the hyperparameter sweep. We also introduce LZ78 ensemble inference, which we describe in detail in Section 4. Figure 5 demonstrates that this ensemble is key in achieving high classification accuracy. The novel

algorithmic enhancements and configurability we introduce to the LZ78 SPA enable us to achieve new state-of-the-art performance in genomic data classification.

We compare this LZ78-based classifier to DNABERT-2 [14], a transformer-based model that maintains state-of-the-art performance while being parameter-efficient. As with DNABERT-2, performance of the LZ78 DNA classifier is shown on 28 datasets from the Genomic Understanding Evaluation (GUE) suite [14]. The LZ78 classifier achieves comparable or superior accuracy on 89.2% of datasets while drastically reducing computational costs. Unlike DNABERT-2, which requires training for multiple weeks on significant GPU resources and memory, our LZ78 classifier is capable of training in under an hour (for all datasets combined) on a modern CPU using a fraction of the training data. It also offers up to 128× speedup in inference time and a dramatic decrease in training memory. These results highlight the potential of the LZ78 algorithm for scalable and efficient genomic data classification, particularly in resource-constrained environments.

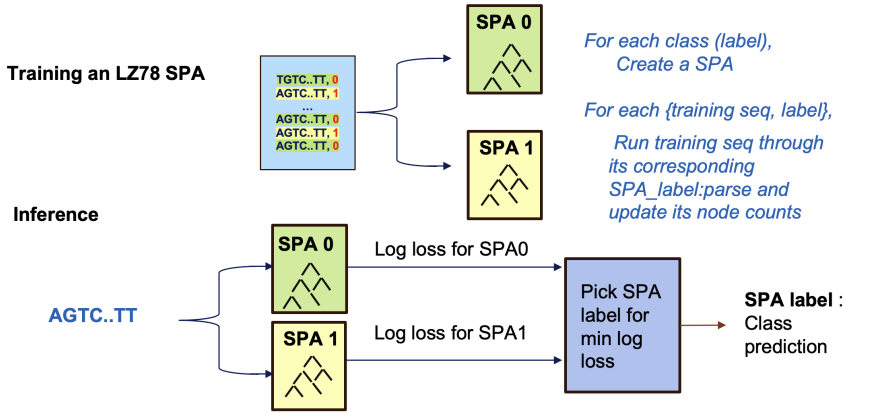
2 Results

A highly efficient compression-based scheme for DNA classification

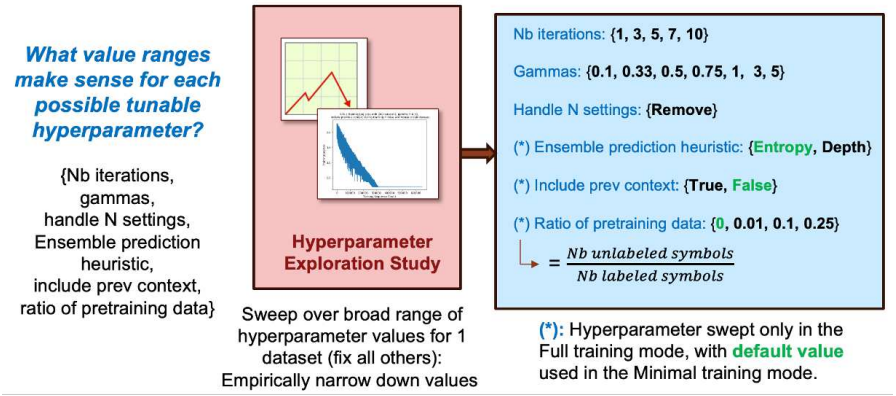
In this work, we demonstrate the effectiveness of using the LZ78 compression algorithm as a classifier for DNA data. We ground our work in the ability of a universal compressor to induce an equivalent sequential probability assignment (SPA) on a parsed sequence (from the theoretical results presented in [26] and summarized in Section 4). Intuitively, a compressor is able to learn and predict the patterns in a data sequence during the compression process. Then, this embedded prediction capability can be used to correlate new seen data with learned data, which is at the core of classification. The extent to which a probability model is able to predict or equivalently compress sequences is measured by the model's log loss. Given a new data sequence, the lower the log loss (for a compressor, the better the compression rate) achieved, the closer the new data sequence is to the learned data associated with that model. Specifically, for a stochastic sequence, the average log loss (or compression rate) achieved is equal to the entropy of the data sequence plus the relative entropy between the sequence and probability model. If the probability model perfectly captures the distribution of the sequence, then the relative entropy is zero and the average log loss matches the fundamental limit, *i.e.*, entropy of the sequence.

Figure 1a illustrates how to leverage this information theoretic equivalence between compression and sequential probability assignments to construct a classifier for genomic data. Given a classification task with α classes, the classifier for this task is constructed by α SPAs (or equivalently compressors). To construct this classifier, the SPAs are trained on labeled data. Each entry of the labeled data is comprised of a DNA sequence, along with its associated class. For each entry with training sequence *seq* and label ℓ, *seq* is passed through the SPA corresponding to label ℓ among the α possible compressors.

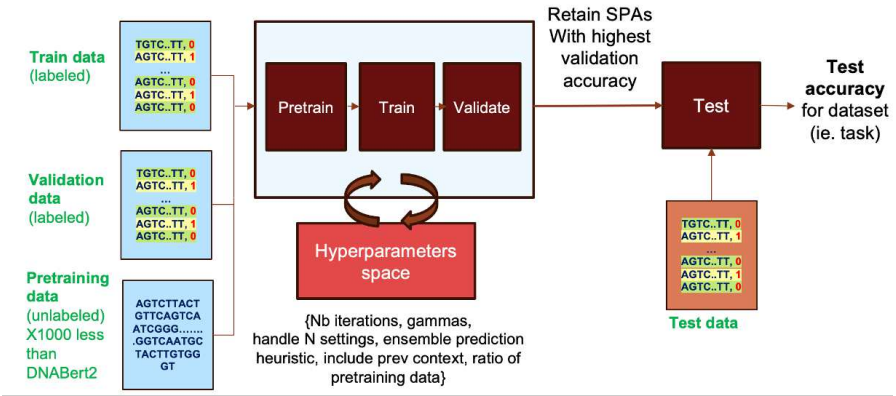
During inference, the goal is to assign a class to a new unlabeled sequence. To do so, the new DNA sequence is passed through all α trained SPAs. While parsing the sequence, each SPA creates a sequential probability model for each symbol, according



(a) Construction of an LZ78-based Classifier via Training Sequential Probability Assignments on Labeled Data.



(b) Hyperparameter Space Exploration for Training an LZ78-based Classifier.



(c) Training and Validation Framework for an LZ78-based Classifier (Pretraining only occurs in the Full training mode, and is skipped in the Minimal training mode.).

Fig. 1: Training and testing methodology for LZ78-based classification.

to Equation (1). The normalized log losses across all SPAs, incurred from this parsing process, are compared. A lower log loss means that the test sequence was closer to the training distribution for the SPA, and thus more likely to be in the associated class. The label associated with the SPA achieving the minimum log loss is outputted as the class for that sequence by the classifier.

A streamlined and open-sourced framework for training and validation

To achieve high accuracy, our LZ78-based classifier tunes a number of configurations that affect the training and the inference stages of the SPAs. We will refer to these configurations as the *hyperparameters* of the classifier. In particular, six hyperparameters were considered when tuning the LZ78 classifier: (1) the Dirichlet parameter γ, (2) inclusion of previous samples' context, (3) number of iterations (epochs), (4) ratio of unlabeled pre-training symbols over the number of training symbols from labeled data, (5) ensemble prediction heuristic, and (6) handling of nucleotide placeholders. We explain the significance of each of these hyperparameters in Section 4.

Given the large space of possible combinations and ranges of values for these hyperparameters, we first empirically determined reasonable ranges of values to sweep through a Hyperparameter Exploration Study, as we illustrate in Figure 1b and elaborate in Section 4. These ranges were kept small to maintain computational efficiency.

The framework to identify the best hyperparameters for a given dataset (and construct the classifier for that dataset accordingly) is illustrated in Figure 1c. The framework was inspired by the conventional AI training, validation, and testing framework. Under a specific hyperparameter combination, the classifier (one SPA for each class) is first pre-trained using unlabeled data, then each SPA is trained with the training data for the respective class. The trained classifier for each hyperparameter combination is evaluated on validation data, and the classifier with the highest validation accuracy is chosen as the final classifier for the dataset, and evaluated on test data to get test accuracy.

The hyperparameter combinations are explored through sweeping the hyperparameter space. We explored two training modes:

- **full hyperparameter sweep training mode**, where we sweep all hyperparameters (except for handling nucleotide placeholders, which we always remove)
- **minimal hyperparameter sweep training mode**, where we only sweep the values of the Dirichlet parameter (which we refer to as γ) and the number of iterations, and fix all other hyperparameters to default values as illustrated by Fig. 1b.

In fact, we found that the minimal hyperparameter sweep training mode provided similar and often superior accuracy to the full training mode (due to the full sweep overfitting to the validation set), with much faster training time (and slightly smaller training memory requirement). Therefore, we primarily present our minimal hyperparameter sweep training mode results in this manuscript, and refer the readers to Supplementary Section 1 for the results of our full training sweep.

Our open-sourced framework, including our training scripts, inference scripts, and saved models can be found on our GitHub repository (LZ-Genomics).

Competitive accuracy with a fraction of the training data and cost

From Table 2b and Figure 2c, the LZ78 classifier has comparable or higher accuracy than DNABERT-2⋄ (the enhanced version of DNABERT-2, which requires even further masked language modeling pre-training) for about 89.2% of the datasets, where comparable performance is defined as test accuracy within 5% of DNABERT-2⋄. The LZ78 classifier performs particularly well on problems in the EMP datasets, where we see increases of over 47% in accuracy.

This dramatic increase in accuracy over DNABERT-2 highlights that some genomic classification tasks are more naturally-suited to compression-based methods than deep learning techniques. Characterizing which tasks are more well-suited to information-theoretic classifiers is a direction for future exploration.

To this extent, there is no decisive pattern between increasing sequence length and accuracy, at least under the GUE benchmarks. For instance, the shorter prom_300 datasets of sequence length 300 perform more poorly than the longer EMP dataset of sequence length 500 and also more poorly than the shorter transcription factor datasets of sequence length 101. Although most of these benchmarks test binary classification tasks, the COVID dataset is a more challenging classification task with 9 classes and sequence lengths of about 1000. In this task, our LZ78 classifier still outperforms DNABERT-2⋄.

Across these 28 benchmarks, the LZ78 classifier performs significantly worse compared to its DNABERT-2 only for three datasets: "prom_300_notata", "prom_300_tata", and "splice", with accuracy degradations of 12.98%, 24.04%, and 23.46% respectively. In these cases, the computational efficiency of LZ78 (and the fact that it does not require GPU hardware) may make the accuracy degradation a reasonable trade-off. Further algorithmic enhancements to improve the performance of our LZ78 classifier remain for future work.

As can be shown in Fig. 3, the characteristics of various datasets may require different configurations of the hyperparameters used for training and inference. This further motivates our configurable LZ78 classification framework. Certain datasets may benefit from more regularization through a higher Dirichlet parameter. For example, the accuracy of most datasets increases with the Dirichlet parameter then plateaus at around $\gamma = 3$ whereas, for the COVID dataset, a smaller Dirichlet parameter is crucial for a much stronger classification accuracy. Similarly, datasets generally benefit from multiple training iterations (or epochs). However, too many epochs can result in overfitting, as can be seen for prom_300_all and prom_300_notata.

Computational Efficiency

A key advantage of the LZ78 classifier is its computational efficiency: it can be trained in a matter of minutes on a modern CPU, as opposed to days or weeks on several GPUs. Inference is also two orders of magnitude faster than DNABERT-2, running on the order of microseconds per symbol.

(a) Overview of dataset properties for the GUE benchmarks.

Species	Task	Num. Datasets	Number of Classes	Sequence Length
Human	Core Promoter Detection	3	2	70
	Transcription Factor Prediction	5	2	100
	Promoter Detection	3	2	300
	Splice Site Detection	1	3	400
Mouse	Transcription Factor Prediction	5	2	100
Yeast	Epigenetic Marks Prediction	10	2	500
Virus	COVID Variant Classification	1	9	1000

(b) Accuracy results (expressed as percent correct).

	mouse					tf				
	0	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**4**
DNABERT-2	56.26	84.77	79.32	66.47	52.66	71.99	76.06	66.52	58.54	77.43
DNABERT-2◇	64.23	**86.26**	81.28	73.49	50.8	69.12	71.87	62.96	55.35	74.94
LZ78 (ours)	**75.19**	85.1	**85.98**	**76.99**	**68.83**	**80.00**	**82.40**	**78.90**	**71.20**	**86.00**

	prom core			prom 300				EMP H4	
	all	**tata**	**notata**	**all**	**tata**	**notata**	**splice**	**H4**	**ac**
DNABERT-2	74.17	69.37	68.04	71.59	86.77	94.27	84.99	80.71	50.43
DNABERT-2◇	**76.18**	67.5	69.53	68.79	**88.31**	**94.34**	**85.93**	81.86	50.35
LZ78 (ours)	71.88	**75.04**	**73.96**	**78.72**	64.27	81.36	62.47	**88.77**	**89.88**

	EMP H3								
	H3	**K14ac**	**K36me3**	**K4me1**	**K4me2**	**K4me3**	**K79me3**	**K9ac**	**COVID**
DNABERT-2	78.27	52.57	56.88	50.52	31.13	36.27	67.39	55.63	71.02
DNABERT-2◇	80.17	57.42	61.9	53.0	39.89	41.2	65.46	57.07	68.49
LZ78 (ours)	**87.64**	**92.50**	**89.28**	**83.18**	**84.42**	**88.78**	**88.70**	**86.79**	**72.20**

(c) Radar plot comparison of LZ78 and the two DNABERT-2 variants.

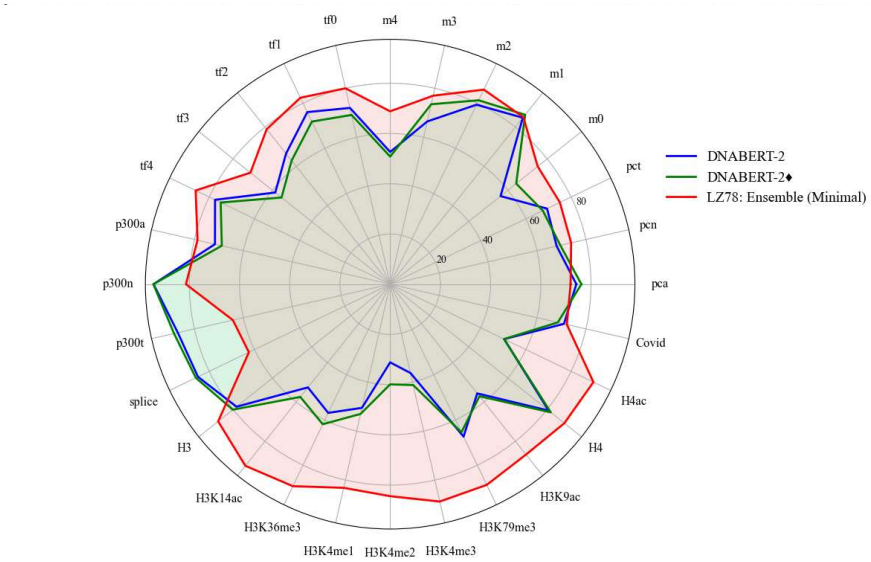
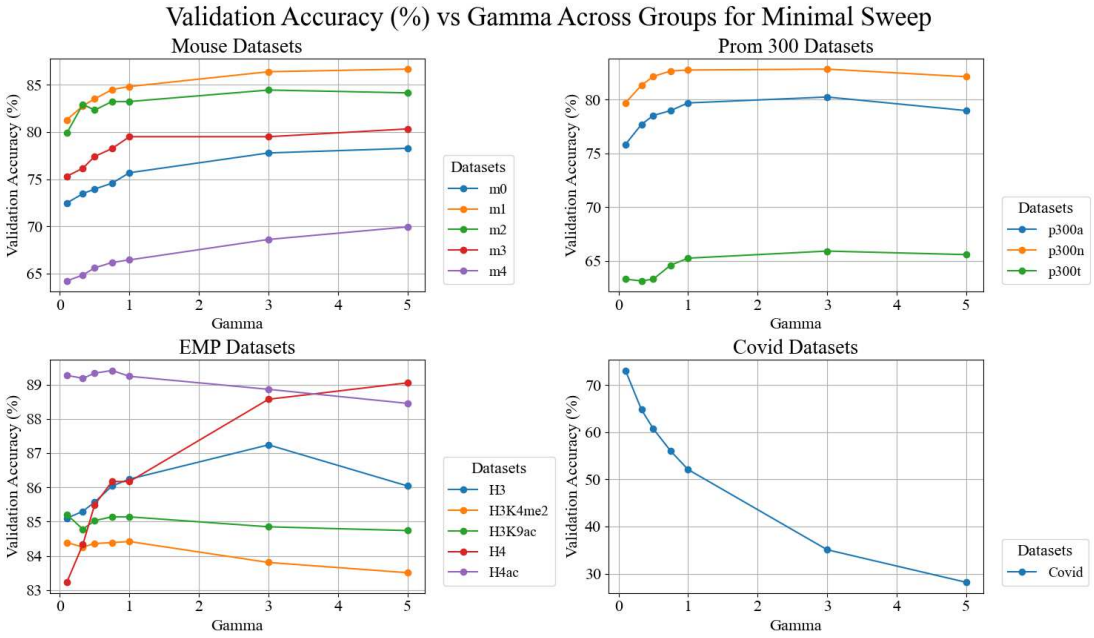
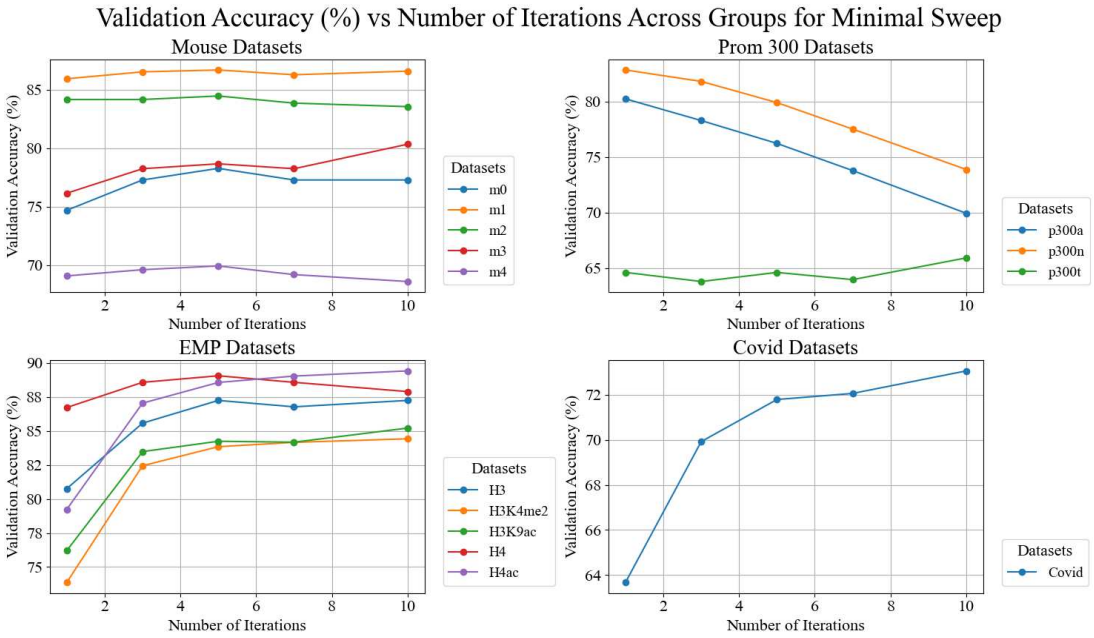


Fig. 2: Accuracy results on the GUE benchmark dataset (presented for the minimal hyperparameter sweep training mode)



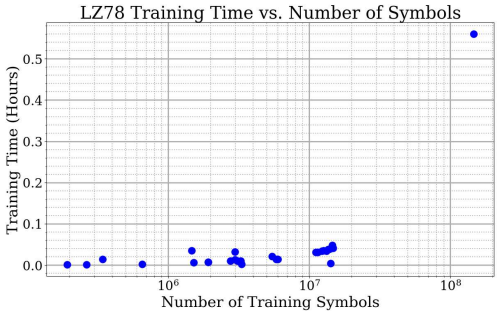
(a) Dataset Test Accuracy vs. Dirichlet Hyperparameter ("Gamma") Chosen by the Training/Validation Framework under Minimal Sweep.



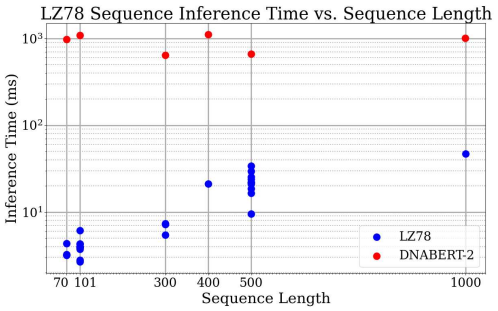
(b) Dataset Test Accuracy vs. Number of Iterations Chosen by the Training/Validation Framework under Minimal Sweep.

Fig. 3: Hyperparameter Trends: Dataset Accuracy Variations with Dirichlet (Gamma) Parameter and Number of Iterations under Minimal Sweep.

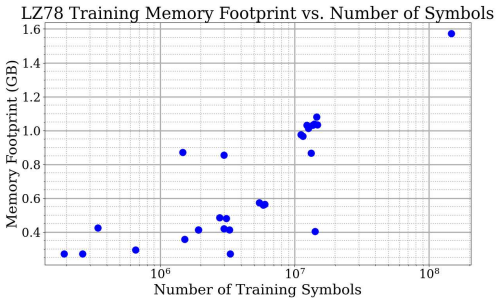
The efficiency of such compression-based methods can enable real-time DNA classification in edge settings, where low latency and reduced infrastructure costs are critical. By eliminating the reliance on expensive GPUs and extensive computational resources, LZ78 facilitates genomic analysis in remote or resource-limited settings, and can also preserve data privacy by processing locally. These attributes make LZ78, and other compression-based classifiers, a powerful solution for time-sensitive applications,[2] bridging the gap between computational genomics and real-world needs.



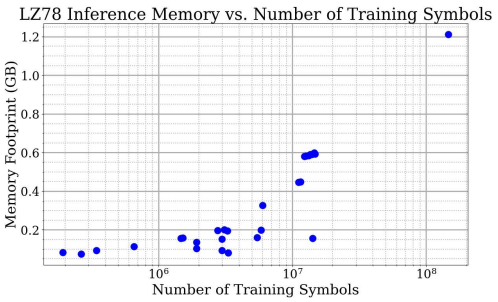
(a) Efficiency of LZ78 Classifier Training; average training time of ≈2.3 minutes per dataset on a modern CPU.



(b) Inference time speedup of Using an LZ78-Based classifier compared to DNABERT-2.



(c) LZ78 Classifier Training Memory.



(d) LZ78 Classifier Inference Memory.

Fig. 4: Computational and memory efficiency of the LZ78-based genomic classifier (presented for the minimal hyperparameter sweep training mode).

Training Time: LZ78 demonstrates significant advantages in training time compared to DNABERT-2. Our current LZ78 training script is highly optimized, and requires less than 6 minutes for datasets with 10^7 symbols and about 33 minutes for $1.1 * 10^8$ symbols when trained on an Intel Xeon Silver 4216 CPU @ 2.10GHz, as demonstrated by Figure 4a.

[2]Including, *e.g.*, point-of-care diagnostics, outbreak management, and precision medicine.

In stark contrast, DNABERT-2 demands a much longer training duration, requiring 14 days on 8 NVIDIA RTX 2080Ti GPUs for pre-training and additional few hours for fine-tuning the model for each classification task. This substantial gap highlights the lightweight computational requirements of LZ78, making it suitable for resource-constrained environments, where extended training durations are impractical.

Inference Time: LZ78 achieves a remarkable speedup during inference compared to DNABERT-2, as illustrated by Figure 4b. LZ78 processes symbols at an average rate of approximately 41.3, μs/symbol on an Intel Xeon Silver 4216 CPU @ 2.10GHz. In contrast, DNABERT-2 requires around 5.3ms/symbol when running on a high-performance NVIDIA RTX 4090 GPU. This translates to an impressive $\sim 128\times$ speedup for LZ78. Additionally, LZ78 operates on far more affordable and accessible hardware, such as standard CPUs, while DNABERT-2 relies on costly, specialized GPUs. This highlights LZ78's practicality for real-world applications, particularly in resource-constrained settings.

Memory Footprint: LZ78 is also far more memory-efficient than DNABERT-2 in training, as illustrated by Figure 4c. For training, LZ78 uses less than 1.6 GB of memory, even for the largest datasets, whereas DNABERT-2 was trained with a memory capacity of about 88 GB. The LZ78 classifier can be trained entirely on consumer hardware, or even edge devices. During inference, the memory requirement of our LZ78 classifier depends on the depths of the trees constructing the SPAs, which in turn depend on the richness of the training dataset. This is illustrated by Figure 4d. For DNABERT-2, the profiled inference memory requirement exhibits an average maximum GPU memory allocation of 500MB with a CPU RAM overhead of about 900MB, with little variance for sequence lengths ranging from 70 to 1000 symbols. Although the LZ78 classifier requires slightly more CPU RAM than DNABERT-2 for richer datasets, the memory usage is reasonable for modern CPUs.

Training Data: Our LZ78 classifier requires a fraction of the training data compared to deep learning models in general. For example, DNABERT-2 was pre-trained on ∼30GB of unlabeled data and fine-tuned on $\sim$ tens of MB of labeled data per dataset. In stark contrast, our LZ78 classifier only requires a few $\sim$ tens of MB of training data. This is because pre-training on unlabeled data is not needed (the case of minimal hyperparameter sweep training mode), and even when enabled (an option in the full hyperparameter sweep training mode), only a few MBs of unlabeled data are used. This drastic decrease in the classifier's need for data directly contributes to the small training memory footprint and time, but also highlights the outstanding expressive capability of our classifier.

On Backshifting, Ensemble Prediction, and Algorithmic Enhancements

Our LZ78 classifier underwent multiple algorithmic enhancements that allowed us to improve its classification accuracy. The hyperparameter space and its effects on the classification accuracy is one such contribution, where we identified a set of tunable parameters and enabled their configuration in our open-sourced training and inference framework. Another contribution is our backshifting and ensemble prediction feature. Since the sequential probability assignment (*ie,* prediction) of our classifier depends on the traversal of the compressor's tree, as we explain in Section 4, the classification

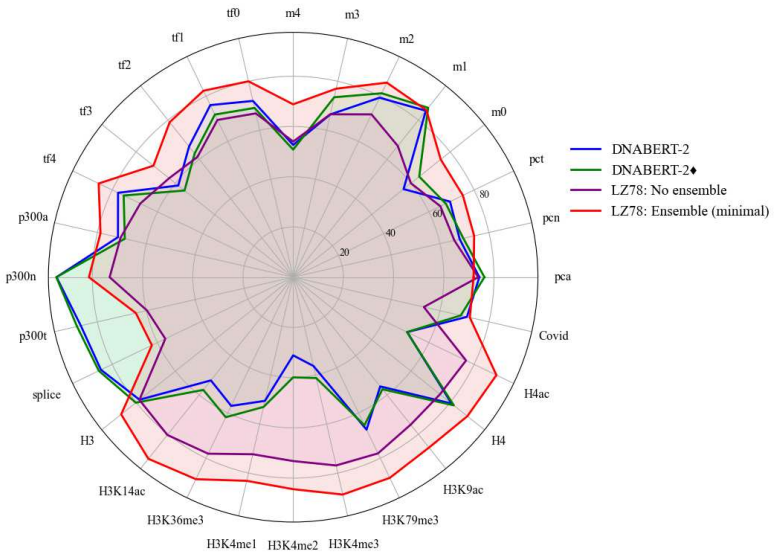


Fig. 5: The effect of backshifting and ensemble prediction on the accuracy of our LZ78 classifier (LZ78: No Ensemble is our LZ78 classifier with backshifting and ensemble prediction disabled).

accuracy may suffer from the classifier reaching suboptimal parts of the tree: a root, where it lacks previous context, or a leaf, where the probability distribution of the next symbol is uniform. Backshifting and ensemble prediction are features that we used to significantly enhance the accuracy of our classifier as shown in Fig. 5, as they allow the classifier to be repositioned in a more effective part of the tree by considering a subset of the phrase. We elaborate on these enhancements in Section 4. We also provide further discussion of our backshifting and ensemble prediction ablation study as well as a tree structure analysis in Supplementary Section 2.

3 Discussion

In this work, we presented an LZ78-based classification algorithm for genomic data, and compared its performance and computational efficiency against DNABERT-2. The results highlight significant advantages of LZ78 in terms of computational requirements, training memory, and inference speed while maintaining accuracy. Specifically, LZ78 sets new state-of-the-art accuracy in DNA classification, and achieves comparable or higher accuracy than DNABERT-2⋄ for 89.2% of the datasets in the GUE benchmark, demonstrating its effectiveness. Our classifier outperforms the state-of-the-art DNABERT-2⋄ by as much as 47% difference in accuracy *eg. epigenetic mark prediction (EMP) tasks*, while falling short on classification accuracy (a degradation in accuracy of over 5% compared to DNABERT-2) on only 3 datasets out of the 28-dataset suite.

LZ78 requires only minutes of training on a modern CPU, whereas DNABERT-2 demands over 14 days on 8 GPUs, emphasizing the computational efficiency of LZ78. During training, LZ78 uses less than 1.6 GB of memory, while DNABERT-2 consumes 88 GB. Moreover, LZ78 processes symbols at 41.3 microseconds per symbol using a standard CPU, achieving a remarkable 128× speedup over DNABERT-2. LZ78 can run on standard consumer CPUs, making it accessible and cost-effective, while DNABERT-2 relies on specialized hardware such as GPUs.

These results demonstrate that LZ78 provides a lightweight, scalable, and efficient solution for genomic data classification, particularly in resource-constrained environments. Although DNABERT-2 may offer improved accuracy for a small number of datasets, LZ78's efficiency and overall comparable or superior performance make it a practical alternative for a wide range of applications.

While we have demonstrated the potential of an LZ78-based classifier for genomic data, several avenues for further exploration remain. To the extent of improving LZ78 classifier performance, improvements can be made to the LZ78 SPA (Equation (1) from Section 4) itself, and to how the SPA is leveraged in classification. For instance, it could be interesting to explore adaptive SPA regularization, where the Dirichlet smoothing parameter is a function of the depth of the current node. Intuitively, nodes close to the root have less context to use for prediction and could benefit from more regularization than nodes deeper in the tree. This is a challenge we addressed by introducing backshifting and ensemble prediction, but it would be worthwile to continue exploring further enhancements such as adaptive regularization. Another important aspect to study is the LZ78-based classifier's limitations on context length. As the context length that is being used by the LZ78 SPA is equal to the depth of the current node, the maximum context length of any prediction is the depth of the tree. Whereas the tree grows infinitely deep with an infinite amount of data, a crucial future direction is to improve the context length as much as possible in the finite-sample regime.

In addition, motivated by results in Figure 2c, it would be interesting to combine LZ78 with neural networks in genomic classification and beyond to achieve accuracy commensurate with the maximum of the two methods. For example, validation can be used to determine which tasks a neural network is more well-suited for. For those tasks, a mixture of the SPA log losses and output class probabilities from the neural network can be used for classification. For example, for our case of DNA classification, tasks like EMP, where LZ78 achieves higher accuracy, the BERT classification can be skipped entirely for computational efficiency. This will be studied in future work.

4 Methods

Project Codebase

Our full codebase, including training scripts, inference scripts, and saved models can be found on our GitHub repository (LZ-Genomics). We hope our work can catalyze further advancements in genomic data analysis and beyond.

Universal Compression via Lempel-Ziv 78

A universal compressor, loosely speaking, is one that performs well on all finite-alphabet *individual sequences* (*i.e.*, any arbitrary, deterministic sequence where each symbol is from some finite set $\mathcal{A}$) of sufficient length. We denote an individual sequence $x^n = (x_1, x_2, \cdots, x_n)$, where $x_i \in \mathcal{A}$.

The LZ78 incremental parsing algorithm [15] is as follows for sequence x^n over alphabet $\mathcal{A}$:

1. Denote the list of phrases seen so far as $\mathcal{Z}$, which starts out as the empty list.
2. Repeat until the end of the sequence:
   (a) Starting after the previously-parsed phrase, find the prefix of x^n that consists of some phrase in $\mathcal{Z}$, plus one new symbol. This is a new phrase, which is added to $\mathcal{Z}$.
   (b) Encode the new symbol using $\lceil \log_2(|\mathcal{A}|) \rceil$ bits, and encode the index of $\mathcal{Z}$ corresponding to the beginning of this phrase using $\lceil \log_2(|\mathcal{Z}|) \rceil$ bits.

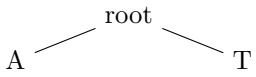
This, equivalently, can be viewed as the creation of the following prefix tree, where each node except the root corresponds to a phrase from $\mathcal{Z}$:[3]

1. Start off with a singular root node.
2. Repeat until the end of the sequence:
   (a) Starting at the root, traverse the prefix tree using the next symbols of x^n until we reach a leaf. Add a new node branching off of the leaf for the next symbol in the input sequence.
   (b) The new phrase in $\mathcal{Z}$ is defined as the slice of the input sequence used in traversing the tree and creating the new leaf.

As an example of building an LZ78 prefix tree, consider the following nucleotide sequence

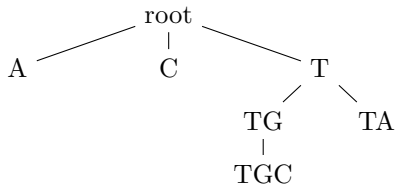
$$x^n = \texttt{ATTGCTGCTA}.$$

We start at the beginning of the sequence and add the first symbol, `A`, as a branch to the root node. `A` is then the first phrase. We then do the same thing with the next symbol, `T`, which becomes the second phrase. After encoding the first two phrases, the tree is as follows:



The next symbol is `T`, which is already present as a branch off of the root. So, we traverse from the root to the node `T` and move onto the next symbol, `G`, resulting in the phrase `TG`. By the process of LZ78 parsing, x^n is overall divided into the phrases `A,T,TG,C,TGC,TA`, forming the tree:

[3] This description, and the following example, are adapted from [26].



In the limit as the number of symbols in x^n goes to infinity, the compression ratio approaches $\frac{|\mathcal{Z}|\log_2|\mathcal{Z}|}{n\log_2|\mathcal{A}|}$. Consider an infinite-length sequence, denoted $\mathbf{x}$ (of which x^n is a prefix). The *finite-state compressibility* of this sequence, $\rho(\mathbf{x})$, is the fundamental limit of the class of compressors that depend only on a finite-state mechanism (*i.e.*, have finite memory, though the memory may be of an arbitrary size). For any infinite-length individual sequence, the LZ78 does at least as well as the best finite-state compressor: its compression ratio is upper-bounded by $\rho(\mathbf{x})$.

A LZ78-Based Universal Sequential Probability Assignment

In general, universal compression is tied to universal probability modeling in that good compression requires, implicitly or explicitly, a good probability model of the data being compressed. Intuitively, when compressing, more bits should be assigned to more "surprising" features in the data, whereas frequent or common patterns should be represented with fewer bits. Achieving this can be seen as distinguishing "high-probability" features from "low-probability" features.

A sequential probability assignment is a mapping from the prefix of a sequence, x^{t-1}, to a probability model on the next symbol, x_t. [26] shows that LZ78 induces a family of universal sequential probability assignments (SPAs) that are efficient to compute. Essentially, it is derived from the empirical distribution of symbols seen while traversing the current node of the LZ78 tree. The relevant form of this SPA is

$$q^{LZ78,\gamma}(x_t = a|x^{t-1}) = \frac{N(a|x^{t-1}, z_c(x^{t-1})) + \gamma}{\sum_{b\in\mathcal{A}} N(b|x^{t-1}, z_c(x^{t-1})) + \gamma A}, \tag{1}$$

where the notation being used is:

- $z_c(x^{t-1})$: the prefix of the current LZ78 phrase (up to and not including x_t). This is the node of the LZ78 tree that we reach after parsing x_{t-1}, or the root of the tree if x_{t-1} is the end of a phrase.
- $N(a|x^{t-1}, z_c(x^{t-1}))$: the number of times (up to index $t-1$) that we have seen the symbol $a \in \mathcal{A}$ while at node $z_c(x^{t-1})$ of the LZ78 tree.
- γ: this is a positive hyperparameter called the *Dirichlet smoothing parameter* (see [26] for more details). Smaller values of γ mean that the SPA tends more strongly towards the empirical distribution.

This form of the LZ78 SPA is referred to as the *LZ78 SPA induced by a Dirichlet prior*.

Theoretical results from [26] establish that:

1. The normalized self-entropy log loss of this SPA asymptotically approaches $\log_2|\mathcal{A}|$ times the LZ78 compression ratio.

2. On any individual sequence, the optimal log loss over the class of finite-state sequential probability assignments is equal to $\log_2 |\mathcal{A}|\rho(\mathbf{x})$.
3. The previous two statements, plus the universality of LZ78 compression, imply that the LZ78 SPA is universal (with respect to the class of finite-state SPAs). That is, its asymptotic log loss will be at most that of any finite-state SPA.

Refer to [26] for more details on the construction and theoretical results of this SPA.

Improvements to the Base LZ78 SPA

Although the LZ78 SPA is universal and therefore will do as well as the best finite-state SPA when applied to infinite data, if applied naively, its accuracy can suffer in the finite-data regime. This is because, at any point, the quality of the SPA $q^{LZ78,\gamma}$ (i.e., how closely it models the true distribution of symbol x_t, conditioned on x^{t-1}) depends on the location of the current node of the LZ78 tree. Specifically:

- If the node is too close to the root, then the LZ78 SPA is only using a short context of data before the current symbol. As a result, the SPA does not have access to potentially-predictive information from farther back in the past.
- If the node is too close to a leaf, then the node has not been visited that many times. Then, the counts $N(a|x^{t-1}, z_c(x^{t-1}))$, for $a \in \mathcal{A}$, are small and likely sparse: the SPA has not seen enough data at that node to make a good probability estimate.

We mitigate these issues via the following heuristics:

- **Backshift Parsing**: This is a technique to avoid making predictions too close to the root or a leaf of an LZ78 prefix tree, as proposed in [22].
  When we reach the root or the leaf of the tree, we attempt to "re-seed" the tree by taking a length-L_{back} context before the current symbol (i.e., $x_{t-L_{\text{back}}}^{t-1}$), and traversing from the root with $x_{t-L_{\text{back}}}^{t-1}$. If the traversal reaches a leaf at any point, we decrement the backshift context length ($L_{\text{back}} \leftarrow L_{\text{back}} - 1$) and try again, repeating until the backshift traversal does not reach a leaf.
- **Ensemble Prediction**: Instead of evaluating the SPA at a single point, we can get improved accuracy by taking a weighted average of SPA values, evaluated at different depths within the LZ78 tree.
  Assume that, at timepoint t in evaluating the SPA for sequence x^n, we are at node z of the LZ78 tree with depth d. We then construct a ensemble of target size N_{ens} as follows:

  1. We first form the ensemble, which is a set of nodes $\mathcal{Z}_{\text{ens}} \subseteq \mathcal{Z}$ at which we evaluate the LZ78 SPA. $\mathcal{Z}_{\text{ens}}$ starts out as the singleton set $\{z\}$. For each $k \in \{1, \ldots, d-1\}$, we traverse from the root with sequence x_{t-k}^{t-1}, reaching node z_k. This means that subsets of the current phrase are used to traverse the tree, by removing the left-most symbol for each new sub-phrase considered in the ensemble. Each tree traversal (resulting from considering a sub-phrase) results in a prediction, that gets considered along with other sub-phrase predictions to make the final overall prediction of the next symbol in the phrase. If no returns to the root occurred

in this parsing, z_k is added to $\mathcal{Z}_{\text{ens}}$. If, at the end of the process, $|\mathcal{Z}_{\text{ens}}| > N_{\text{ens}}$, the deepest nodes are retained.[4]

2. For each $z \in \mathcal{Z}_{\text{ens}}$, compute

$$q^{z,\gamma}(x_t = a|x^{t-1}) = \frac{N(a|x^{t-1}, z) + \gamma}{\sum_{b\in\mathcal{A}} N(b|x^{t-1}, z) + \gamma A}, \quad \forall a \in \mathcal{A}.$$

3. Perform a weighted average of $q^{z,\gamma}(x_t = a|x^{t-1})$ for $z \in \mathcal{Z}_{\text{ens}}$ to get the final SPA:

$$q^{\text{ens}}(x_t = a|x^{t-1}) = \sum_{z\in\mathcal{Z}_{\text{ens}}} w_z q^{z,\gamma}(x_t = a|x^{t-1}),$$

for some set of weights that sum to 1. There are two heuristics for the weighted average: a *depth-based* heuristic and an *entropy-based* heuristic.
Depth-based: Let $d(z)$ be the depth of node z. Then, the weights are

$$w_z = \frac{\hat{w}_z}{\sum_{z'\in\mathcal{Z}_{\text{ens}}} \hat{w}_{z'}}, \quad \hat{w}_z = \exp\left(\frac{d(z) - \min_{z'\in\mathcal{Z}_{\text{ens}}} d(z')}{\max_{z'\in\mathcal{Z}_{\text{ens}}} d(z') - \min_{z'\in\mathcal{Z}_{\text{ens}}} d(z') + 10^{-6}}\right),$$

where the 10^{-6} (and the 10^{-10} in the subsequent equation) is a small constant to prevent the denominator from becoming zero.
Entropy-based: Let $H(\cdot)$ be the Shannon entropy of a probability model, and $H_z = H\left(q^{z,\gamma}(\cdot|x^{t-1})\right)$. Then,

$$w_z = \frac{\hat{w}_z}{\sum_{z'\in\mathcal{Z}_{\text{ens}}} \hat{w}_{z'}}, \quad \hat{w}_z = \exp\left(-\frac{1}{2} \cdot \frac{H_z - \min_{z'\in\mathcal{Z}_{\text{ens}}} H_{z'}}{\max_{z'\in\mathcal{Z}_{\text{ens}}} H_{z'} - \min_{z'\in\mathcal{Z}_{\text{ens}}} H_{z'} + 10^{-10}}\right).$$

Intuition for weighted average heuristics: Both heuristics attempt to more heavily weight the "better" SPAs in the ensemble and put less weight on the "worse" SPAs. As we do not directly know which SPAs in the ensemble are closest to the true distribution of the data, we need to heuristically determine which SPAs to weight more. For the depth-based weighted average, we assume that deeper nodes produce better SPAs, because they have access to more information (a longer context). For the entropy-based average, we assume that SPAs with lower entropy are better. A high-entropy SPA is close to uniform, which is not typically very predictive. The high-entropy nodes tend to be those close to the root, which only use low-order information to make a prediction, or those that have not seen many symbols, where the Dirichlet smoothing parameter dominates the SPA value.

- **SPA Lower Bound**: If the SPA value is too close to 0 for any symbol, it can incur very large log loss. Although this can be correctly indicative of large relative entropy between the SPA and test sequence, it can also lead to spuriously large log losses. So, at all timepoints, we set $q^{LZ78,\gamma}(x_t = a|x^{t-1})$ to be at least ϵ_{LB}, scaling the rest of the values such that the SPA still sums to 1.

[4] Computationally, we iterate k backwards from $d-1$ to 1, stopping ensemble construction when $\mathcal{Z}_{\text{ens}}$ reaches the desired size.

These heuristics introduce hyperparameters, only one of which is added to the hyperparameter sweep. The rest of the parameters are set to the following defaults:

- **Backshift Parsing**: $L_{\text{back}} = 20$.
- **Ensemble Prediction**: $N_{\text{ens}} = 10$. The weighted average of the ensemble (*depth-based* versus *entropy-based*) is part of the hyperparameter sweep in the full hyperparameter sweep training mode, and is set to *entropy-based* in the minimal hyperparameter sweep training mode, as we further explain in Supplementary Section 1.
- **SPA Lower Bound**: $= 10^{-5}$.

Memory Efficiency of the LZ78 SPA

The simplest implementation of the LZ78 SPA is to directly implement a tree structure, where each node is an object that stores the number of times it has been traversed and a mapping to child nodes. Empirically, this incurs a large amount of overhead in the large number of small hashmaps (or an equivalent data structure to store branches in the tree) and node objects.

To maintain memory efficiency, we use an data structure inspired by the Lempel-Ziv-Welch compressor [27]. We implicitly assign each node an ID based on the order that it was added to the tree (the root always has ID 0, the first node added has ID 1, etc.). The LZ78 SPA is stored using two basic data structures:

1. An array of the number of times that each node was traversed, ordered by ID.
2. A hashmap mapping the tuple (parent node ID, symbol) to the ID of the corresponding child node.

Both data structures can be accessed and modified (including insertions) in amortized constant time, and avoid the overhead of many small objects.

Parallelization

Though the process of building an LZ78 prefix tree is inherently serial, there are several opportunities to leverage CPU parallelism. Inference over batches of sequences can be directly parallelized over sequences, which we apply to the validation component of the hyperparameter sweep (over 48 CPU threads). When classifying a single sequence, the computation of the SPA log loss can be parallelized over classes (by running inference on each SPA in parallel), which we apply to the inference time measurement. SPA training can also be parallelized across classes in the same manner. We do not implement this, as building the SPAs is fast compared to the validation process.

DNABERT-2 and the Genomics Understanding Benchmark

DNABERT-2 is a large language model that maintains state-of-the-art performance in DNA classification tasks on the Genomic Understanding Evaluation (GUE) suite. The GUE benchmarking suite is a multi-species genome classification suite curated by [14]. It includes 36 different datasets evaluating classification tasks on sequences ranging from 70 to 10000 symbols. 28 of these datasets have been open-sourced with sequence lengths ranging from 70 to 1000 symbols, and are used as the benchmarking

reference in this work. The 28 datasets from the GUE benchmarking suite used in the project are outlined in Table 2.

Compared to Nucleotide Transformer, DNABERT-2 is considered 3x more computationally efficient, as it achieves comparable performance to Nucleotide Transformer with 21x less parameters and about 92x less GPU pre-training time.

However, even with this significant computational efficiency improvement, DNABERT-2 and other large language model solutions remain very computationally expensive. During pre-training, where LLMs learn to represent genomic language, DNABERT-2 was trained on tens of GBs of unlabeled data, and required 14 days on 8 NVIDIA RTX 2080Ti GPUs, each of which contains 11GB of GDDR6 memory. In order to use the foundation model for classification, an additional phase of fine-tuning is required, which typically takes at least a few additional hours to cover all datasets. The model is comprised of 117M weights, each stored using 2 bytes, resulting in a memory requirement of 234MB for weight storage.

Profiling and Computing Platforms Used

Apart from aiming for high accuracies with the LZ78 classifier, a crucial goal is to develop a classifier that is computationally efficient as well. As such, we characterize the computational cost of the classifier is profiled during training and inference. The key efficiency metrics are training time, inference time, and memory footprint. These are compared to DNABERT-2.

The computational measurements for the LZ78 classifier were performed on an Intel Xeon Silver 4216 CPU @ 2.10GHz. The training and inference time were tracked using the `perf_counter` function from the `time` library in Python. The Python `memory_profiler` was used to estimate the total memory allocated by the program during runtime.

The total training time DNABERT-2 is obtained from [14], and the training memory footprint is estimated based on the number of GPUs used and the memory capacity of each. For characterizing the inference time per symbol and the peak memory usage during inference, a fine-tuned version of DNABERT-2 (available at [28]) was profiled. In order to use DNABERT-2 for classifying a dataset, the foundation model available on Hugging Face by [14] should be fine-tuned to classify the target dataset. The profiled version of DNABERT-2 was fine-tuned on the EMP H3 dataset, which means it is suited for correctly classifying the EMP H3 dataset (per the dataset accuracy). However, if the DNABERT-2 model was fine-tuned to classify each dataset, the model architecture would stay primarily unchanged. Typically, a softmax layer is added to the foundational model to predict the class, and this layer would change depending on the number of classes. However, the model architecture is otherwise maintained during the fine-tuning process, and only the values of the weights are updated. Given the high computational cost associated with fine-tuning all possible DNABERT-2 models, the available EMP H3 fine-tuned model was evaluated for a different variety of sequence lengths, with the understanding that it presents a good estimate for computational performance regardless of the specific dataset. DNABERT-2 was profiled on an NVIDIA RTX 4090 GPU, using `torch.cuda.memory_allocated` for GPU memory allocation

estimates, `memory_profiler` for CPU RAM estimates, and `time.perf_counter` for time measurements.

LZ78 Hyperparameter Space and Exploration Study

As specified in Section 2, we enhance our LZ78 classification scheme through a set of hyperparameters, most of which are learned during trained from a reduced configuration space. We defined six hyperparameters:

- **Dirichlet parameter** γ: the perturbation parameter used for computing the log loss when parsing data. The higher the Dirichlet parameter, the more the SPA is perturbed from the empirical distribution, *i.e.*, the more each node tends towards a uniform distribution.
- **Ensemble Prediction Heuristic**: determines whether entropy-based weighting or depth-based weighting is used for ensemble prediction, as explained in Section 4 in the subsection on *Improvements to the Base LZ78 SPA*.
- **Inclusion of previous samples' context:** when parsing the subsequent sample during training, or whether the SPA returns to the root in between separate sequences. Including the previous context is equivalent to concatenating the last phrase of each sequence to the beginning of the next sequence with the same label during training. Including the previous context means that all training samples are viewed as one long sequence, as opposed to separate samples.
- **Number of iterations (epochs)**: the number of times the labeled data is passed through the SPAs during training. Since LZ78 requires a long sequence length to converge to the true distribution, these iterations over the training data can filling more branches of the tree with the same data. It is reasonable to expect iterating over the training data to increase our classifier accuracy, as long as it is done with moderation to avoid overfitting.
- **Ratio of unlabeled pre-training symbols over the number of training symbols from labeled data:** this unlabeled data is used to initialize all SPAs with the same backbone. This hyperparameter was used to assess the impact of a pre-training stage on LZ78 classifier accuracy.
- **Handling of "N" placeholders:** some DNA data includes Ns (and sometimes other nucleotide placeholders), apart from A, G, C, T. These placeholders typically indicate uncertainty in the DNA sequence (*eg.* that any nucleotide can appear at a specific position. These nucleotide placeholders were simply omitted, as they were negligible in the training dataset. In more general cases where these nucleotide placeholders may be more prominent, an augmentation technique might prove beneficial.

The Dirichlet parameter and the ensemble prediction heuristic are primarily relevant during inference (so it can be tuned for validation or testing without re-building the SPAs), whereas the rest directly impact training.

In order to achieve high classification accuracy, it is important to make our classifier configurable, in order to enable higher expressive power based on the dataset. At the same time, this hyperparameter space needs to remain small in order to maintain low training complexity. To determine the ranges, a broad range of values were explored

for the Mouse 0 dataset, informing the process of narrowing the hyperparameter space small discrete sets of values per hyperparameter.

From the broader exploration of hyperparameters, it was found that larger numbers of epochs or iterations cause the training loss to plateau and the validation loss to degrade. As such, the number of epochs was constrained to 10. In addition, values of γ throughout the range between 0.1 to 5 achieved reasonable performance. For the pre-train to train symbols ratio, as the pre-training sequences were simply added to the training sets, high percentages of unlabeled data could result in underfitting of the training set. As such, the amount of pre-training data was swept from 0 to 0.25. Backshifting and ensemble prediction heuristics were formulated as algorithmic enhancements to our classifier to mitigate degradations caused by supotimal parts of the LZ78 tree.

5 Acknowledgements

We thank Itai Sharon for sharing a recent preprint of [24]. Our work was performed independently.

References

[1] Dr. Vinod Kimbahune Vishakha Nerkar. Deep learning approaches in genomic analysis: A review of dna sequence classification techniques. *International Journal of Scientific Research and Engineering Trends*, 2024.

[2] D. M. Roden, H. L. McLeod, M. V. Relling, M. S. Williams, G. A. Mensah, J. F. Peterson, and S. L. Van Driest. Pharmacogenomics. *Lancet*, 394(10197):521–532, 08 2019.

[3] Gonzalo Benegas, Chengzhong Ye, Carlos Albors, Jianan Canal Li, and Yun S. Song. Genomic language models: Opportunities and challenges, 2024.

[4] Faisal Hussain, Usman Saeed, Ghulam Muhammad, Nayyar Islam, and Gohar S. Sheikh. Classifying cancer patients based on DNA sequences using machine learning. *Journal of Medical Imaging and Health Informatics*, 9(3):436–443, 2019.

[5] Gaurav Mathur, Anjali Pandey, and Sandeep Goyal. A comprehensive tool for rapid and accurate prediction of disease using DNA sequence classifier. *Journal of Ambient Intelligence and Humanized Computing*, 14(10):13869–13885, 2023.

[6] F. Ben Nasr and A. E. Oueslati. CNN for human exons and introns classification. In *2021 18th International Multi-Conference on Systems, Signals & Devices (SSD)*, pages 249–254, Monastir, Tunisia, 03 2021.

[7] Zhen Shen Ying He Zhen-Heng Chen Jianqiang Li De-Shuang Huang Siguo Wang, Qinhu Zhang. Predicting transcription factor binding sites using DNA shape features based on shared hybrid deep learning architecture. *Molecular Therapy Nucleic Acids*, 24:154–163, 2021.

[8] A. Rex Macedo Arokiaraj S. Deepa Kanmani Chandran Venkatesan C. Suresh Gnana Dhas Hemalatha Gunasekaran, K. Ramalakshmi. Analysis of DNA sequence classification using CNN and hybrid models. *Computational and Mathematical Methods in Medicine*, 2021, 2021.

[9] E. Nguyen, M. Poli, M. Faizi, A. Thomas, M. Wornow, C. Birch-Sykes, S. Massaroli, A. Patel, C. Rabideau, Y. Bengio, S. Ermon, C. Ré, and S. Baccus. HyenaDNA: Long-Range Genomic Sequence Modeling at Single Nucleotide Resolution. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 43177–43201. Curran Associates, Inc., 2023.

[10] J. Mendoza-Revilla, E. Trop, L. Gonzalez, M. Roller, H. Dalla-Torre, B. P. de Almeida, G. Richard, J. Caton, N. Lopez Carranza, M. Skwark, A. Laterre, K. Beguir, T. Pierrot, and M. Lopez. A foundational large language model for edible plant genomes. *Communications Biology*, 7:835, 2024.

[11] Shentong Mo, Xi Fu, Chenyang Hong, Yizhen Chen, Yuxuan Zheng, Xiangru Tang, Yanyan Lan, Zhiqiang Shen, and Eric Xing. Multi-modal self-supervised pre-training for large-scale genome data. In *NeurIPS 2021 AI for Science Workshop*, 2021. Workshop Paper.

[12] H. Dalla-Torre, L. Gonzalez, J. Mendoza Revilla, N. Lopez Carranza, A. Henryk Grywaczewski, F. Oteri, C. Dallago, E. Trop, H. Sirelkhatim, G. Richard, et al. The Nucleotide Transformer: Building and Evaluating Robust Foundation Models for Human Genomics. *bioRxiv*, 2023.

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

[14] Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. Dnabert-2: Efficient foundation model and benchmark for multi-species genome, 2024.

[15] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[16] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[17] G. Langdon. A note on the ziv - lempel model for compressing individual sequences (corresp.). *IEEE Transactions on Information Theory*, 29(2):284–287, 1983.

[18] M. Feder. Gambling using a finite state machine. *IEEE Transactions on Information Theory*, 37(5):1459–1465, 1991.

[19] M. Feder, N. Merhav, and M. Gutman. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38(4):1258–1270, 1992.

[20] Tsachy Weissman, Erik Ordentlich, Marcelo J. Weinberger, Anelia Somekh-Baruch, and Neri Merhav. Universal filtering via prediction. *IEEE Transactions on Information Theory*, 53(4):1253–1264, 2007.

[21] J. Ziv and N. Merhav. A measure of relative entropy between individual sequences with application to universal classification. *IEEE Transactions on Information Theory*, 39(4):1270–1279, 1993.

[22] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, December 2004.

[23] Itai Sharon Group. Genezip. ISBRA 2022. https://github.com/SharonLab/GeneZip.

[24] Or Leibovich Yochai Meir and Itai Sharon. Using lossless compression algorithms to improve metagenomics binning and accelerate genome taxonomic classification. to appear at RECOMB 2025.

[25] Thomas Konstantinovsky and Gur Yaari. A novel approach to T-cell receptor beta chain (TCRB) repertoire encoding using lossless string compression. *Bioinformatics*, 39(7):btad426, 07 2023.

[26] Naomi Sagan and Tsachy Weissman. A family of LZ78-based universal sequential probability assignments. arXiv preprint. https://arxiv.org/abs/2410.06589, 2024.

[27] Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[28] T. YdyMy. DNABERT-2-117M-finetuned-human-150bp-dna-classification. https://huggingface.co/tydymy/DNABERT-2-117M-finetuned-human-150bp-dna-classification. Accessed: 2025-01-27.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SupplementaryfileNatureCS.pdf