Interpretable DNA Sequence Classification via Dynamic Feature Generation in Decision Trees

Nicolas Huynh
University of Cambridge

Krzysztof Kacprzyk
University of Cambridge

Ryan Sheridan
CU Anschutz Medical Campus

David Bentley
CU Anschutz Medical Campus

Mihaela van der Schaar
University of Cambridge

Abstract

The analysis of DNA sequences has become critical in numerous fields, from evolutionary biology to understanding gene regulation and disease mechanisms. While deep neural networks can achieve remarkable predictive performance, they typically operate as black boxes. Contrasting these black boxes, axis-aligned decision trees offer a promising direction for interpretable DNA sequence analysis, yet they suffer from a fundamental limitation: considering individual raw features in isolation at each split limits their expressivity, which results in prohibitive tree depths that hinder both interpretability and generalization performance. We address this challenge by introducing `DEFT`, a novel framework that adaptively generates high-level sequence features during tree construction. `DEFT` leverages large language models to propose biologically-informed features tailored to the local sequence distributions at each node and to iteratively refine them with a reflection mechanism. Empirically, we demonstrate that `DEFT` discovers human-interpretable and highly predictive sequence features across a diverse range of genomic tasks.

1 INTRODUCTION

DNA Sequence Analysis. DNA sequences represent the fundamental code of life, storing the genetic instructions essential for the development and functioning of all organisms. In recent years, machine learning approaches have emerged as powerful tools for building predictive models with DNA sequences. Supervised learning methods have proven effective in various genomic tasks, from classifying promoter regions (Le et al., 2019) and splice sites (Scalzitti et al., 2021; Albaradei et al., 2020) to predicting gene expression (Avsec et al., 2021a). However, predictive accuracy alone is *not* sufficient for scientific utility, as *transparency* is also important in biological research and clinical applications, where understanding the predictions is crucial for validation against known biological principles and discovery of biological features.

Limitations of Black Boxes. Despite their predictive performance, deep learning models (Linder et al., 2025; Brixi et al., 2025) traditionally suffer from a lack of interpretability — a limitation exacerbated by the scale of recent foundation models (Dalla-Torre et al., 2025). While post-hoc explanation methods, such as saliency maps (Simonyan et al., 2013) or visualization of attention maps (Avsec et al., 2021b) are frequently employed to probe these models, the faithfulness of post-hoc explanations to the model's predictions have been questioned, as different explanation methods can yield conflicting interpretations for the same prediction (Adebayo et al., 2018; Rudin, 2019).

Transparent Models. Motivated by this observation, we ask the question: *How do we design models that are inherently transparent, yet expressive for DNA sequence analysis?* In the search for models that are transparent by design, we draw inspiration from common practices with tabular data, where decision trees are a *ubiquitous* class of interpretable models (Murdoch et al., 2019). Crucially, they satisfy two interpretability properties that black-box models (even with post-hoc explanations) do not jointly possess (Lipton, 2018): *simulatability* and *decomposability*.

Axis-Aligned Trees. Axis-aligned trees (Breiman

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

et al., 1984; Quinlan, 1986, 2014) recursively partition the input space through binary decisions, based on the comparison between a specific feature and a learned threshold at every internal node. While they are widely used with tabular data (e.g. in finance or healthcare (Soleimanian et al., 2012)), their standard formulation is *flawed* in the context of DNA sequence analysis. First, axis-aligned trees must grow deep to capture complex interactions between multiple sequence positions, as each split can only consider a single nucleotide position, thereby compromising the interpretability and generalization performance of the resulting tree. Second, manually crafting high-level variables offers an alternative but remains limited by existing biological knowledge and ignores *local data characteristics* at each node during tree construction.

Our Framework. We address these limitations and answer our original question by introducing `DEFT` (Dynamic Engineering of Features in Trees), an *interpretable* and *expressive* tree-based model for DNA sequence classification. At each internal node, `DEFT` automatically discovers *high-level sequence features* that lead to discriminative splits, going beyond individual nucleotides. It is *adaptive*, since feature generation is informed by the local data characteristics at each leaf of the tree. Furthermore, it incorporates *domain knowledge* by favoring features that are biologically meaningful. `DEFT` achieves this by leveraging Large Language Models (LLMs) as adaptive feature generators, capitalizing on their in-context learning capabilities (Brown et al., 2020) and domain knowledge acquired through extensive pretraining (Achiam et al., 2023). At each internal node, the LLM generates both an interpretable semantic representation and executable code for the proposed feature, guided by the partial tree structure and task-specific metadata. This generation process is embedded into an evolution-inspired optimization scheme in which the LLM iteratively refines candidate features through a *self-reflection* mechanism based on the node splitting scores. We evaluate `DEFT` across a broad range of tasks, showing that `DEFT` constructs decision trees with high-level, highly predictive features, outperforming other tree-based methods and rivaling black-box models while remaining *transparent*.

Our contributions. (1) Conceptually, we propose `DEFT`, an interpretable model for DNA sequence classification that combines the transparency of decision trees with automated feature generation during tree construction. **(2) Technically**, we leverage Large Language Models (LLM) as adaptive feature generators that exploit local node context and task-specific metadata. We also employ an evolution-inspired optimization scheme in which the LLM iteratively refines candidate features through a reflection mechanism. **(3) Experimentally,** `DEFT` reveals *interpretable* sequence features which are highly *predictive* across diverse genomic tasks.

2 RELATED WORKS

We summarize strands of research related to our work, with more details given in Appendix A.

Machine Learning for DNA Sequence Analysis. Machine learning methods have become essential tools for analyzing DNA sequences, with approaches ranging from classical models to deep neural networks. Traditional methods like position weight matrices and k-mer-based models (Stormo, 2000; Ghandi et al., 2014) provide interpretable results but often lack expressivity. More recently, deep learning architectures, particularly convolutional and transformer networks (Linder et al., 2025; Brixi et al., 2025), have demonstrated strong predictive performance across various genomic tasks such as transcription factor binding prediction (Alipanahi et al., 2015; Zeng et al., 2016; Avsec et al., 2021b) and splice site prediction (Scalzitti et al., 2021; Albaradei et al., 2020; Wang et al., 2019). However, these powerful models typically operate as black boxes, making it difficult to interpret their predictions. Although there exist various post-hoc interpretability methods (Simonyan et al., 2013), they are not guaranteed to be faithful to the models they aim to explain (Rudin, 2019; Adebayo et al., 2018).

Decision Trees. Decision trees are typically constructed greedily, finding at each node an optimal feature-threshold pair for splitting (Breiman et al., 1984; Quinlan, 1986, 2014). These conventional methods are restricted to splits based on raw features, providing axis-aligned trees with limited expressivity. Exact optimization methods (Aglin et al., 2020; Lin et al., 2020) also suffer from this bottleneck and can only be used in restricted settings. Multivariate decision trees (Murthy et al., 1994; Zhu et al., 2020) allow splits to consider multiple features simultaneously (e.g. linear combinations of features for oblique trees). However, a common drawback of these methods is that they consider constrained feature spaces, are very challenging to optimize especially in high-dimensional settings, and assume continuous features.

Applications of LLMs. Recent works have leveraged LLMs for diverse tasks, including code evolution (Lehman et al., 2023; Brownlee et al., 2023), optimization (Yang et al., 2024; Liu et al., 2024), and feature engineering (Han et al., 2024; Hollmann et al., 2024; Nam et al., 2024). However, (Han et al., 2024; Hollmann et al., 2024; Nam et al., 2024) have primarily

focused on tabular datasets, while our work addresses DNA sequence analysis. Moreover, they do not account for the characteristics of the downstream model, whereas our approach integrates feature generation directly into tree induction, making it *adaptive* to the local characteristics of nodes, which is a key contribution of `DEFT`. A recent line of works (Fallahpour et al., 2025) use LLMs to reason over biological modalities. The reasoning signal is a multi-step natural-language chain that takes a question in text and produces an answer, with explicit chain-of-thought (CoT). However, there is no guarantee that the CoT is faithful to the actual predictions. This contrasts `DEFT`, where the rationale is not a justification for a final prediction, but an internal CoT used to propose a feature at a given node; the final prediction is made by a decision tree that composes features.

3 BACKGROUND

We aim to build models that are transparent by design for DNA sequence analysis. To make this concrete, we require the following criteria (Lipton, 2018):

Simulatability. A biologist should be able, in reasonable time, to carry out the computations needed to produce a prediction for a single example by following the model step by step.

Decomposability. Each part of the model (inputs/features, parameters, and intermediate computations) has a clear, human-meaningful semantics, and its contribution to the final prediction can be inspected in isolation (e.g. splits or rules correspond to specific sequence patterns).

Simulatability and decomposability are practically useful: they let the practitioner trace predictions to specific sequence features, check them against known biology, and plan targeted perturbations (e.g., CRISPR). While post-hoc methods can offer some insight on black boxes, they do *not* satisfy both simulatability and decomposability. For example, feature attribution methods (Simonyan et al., 2013), which are widely used with CNNs or Transformers, do not provide the explicit rules or computational steps the model used for its prediction. These considerations suggest using models that are interpretable by construction. Decision trees fit this category: their paths encode explicit rules, and their splits localize feature contributions.

3.1 Top-down Tree Induction for Classification

Before describing our method, we recall the standard top-down tree induction procedure. Given a feature space $\mathcal{X} \subset \mathbb{R}^d$ and an output space $\mathcal{Y}$, decision tree induction is the process of learning a predictor $t : \mathcal{X} \to \mathcal{Y}$ from a dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of samples in $\mathcal{X} \times \mathcal{Y}$. The predictor t is described by a tree structure which recursively partitions $\mathcal{X}$ into disjoint regions through feature-threshold splits at internal nodes, and each leaf region is associated with a prediction in $\mathcal{Y}$.

Top-down Construction. Decision trees are typically constructed top-down, with methods like `CART` (Breiman et al., 1984) and `ID3` (Quinlan, 1986) greedily building the tree one node at a time with axis-aligned splits. These methods select the optimal split at a given node by minimizing a score based on a node impurity measure Q (e.g., Gini index, misclassification error, or information gain). Given a subset $\mathcal{D}$ of $\mathcal{D}_{\text{train}}$, a feature map $f : \mathcal{X} \to \mathbb{R}$, and a threshold τ, we denote by

$$\mathcal{D}_{L,(f,\tau)} = \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in \mathcal{D}, f(\mathbf{x}) \leq \tau\}$$
$$\mathcal{D}_{R,(f,\tau)} = \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in \mathcal{D}, f(\mathbf{x}) > \tau\}$$

the subsets formed by splitting $\mathcal{D}$ with the feature map f at threshold τ. Furthermore, let s be a scoring function based on the impurity measure Q, such that:

$$s(f, \tau, \mathcal{D}) = w^l_{f,\tau} Q(\mathcal{D}_{L,(f,\tau)}) + w^r_{f,\tau} Q(\mathcal{D}_{R,(f,\tau)}) \quad (1)$$

where the weights $w^l_{f,\tau} = \frac{|\mathcal{D}_{L,(f,\tau)}|}{|\mathcal{D}|}$ and $w^r_{f,\tau} = \frac{|\mathcal{D}_{R,(f,\tau)}|}{|\mathcal{D}|}$ account for the relative sizes of the splits.

For any coordinate index $i \in [d]$, we denote by $\phi_i : \mathcal{X} \to \mathbb{R}$ the projection along the i-th feature. The induction process then finds the optimal pair of feature and threshold (i^*, τ^*) for $\mathcal{D}$ by solving:

$$(i^*, \tau^*) \in \arg \min_{(i,\tau) \in [d] \times \mathbb{R}} s(\phi_i, \tau, \mathcal{D}) \quad (2)$$

Equipped with this optimization procedure, top-down approaches progressively grow the tree, starting from a single root node containing the training set $\mathcal{D}_{\text{train}}$. Given a leaf v in the partially constructed tree corresponding to the subset $\mathcal{D} \subset \mathcal{D}_{\text{train}}$ which satisfies all splitting conditions from root to v, top-down approaches find an optimal split (i^*, τ^*) for $\mathcal{D}$ using the criterion defined in Equation (2). The leaf v then spawns two child nodes containing $\mathcal{D}_{L,(\phi_{i^*},\tau^*)}$ and $\mathcal{D}_{R,(\phi_{i^*},\tau^*)}$ respectively, and the induction process continues with the updated tree, until a stopping criterion is met (e.g. when a maximum depth is reached).

3.2 Limitations of Axis-Aligned Trees for DNA Sequence Analysis

The transparency of decision trees with a limited depth (Murdoch et al., 2019), coupled with their predictive performance, explains their widespread use with *tabular data*. However, naively leveraging traditional

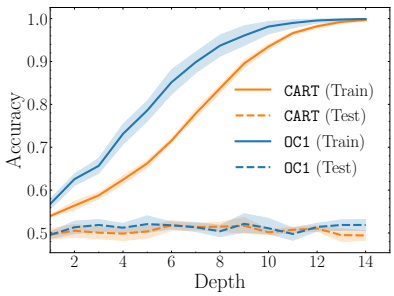


Figure 1: **Limitations of conventional trees.** Training and test accuracies versus tree depth for motif detection, mean and confidence intervals at 95% reported for 5 seeds. Conventional decision trees must grow deep to achieve a high training accuracy, and they do not learn patterns which generalize at test time.

axis-aligned decision trees for *DNA sequence analysis* presents several key challenges, which we now detail.

Limited Expressivity of Raw Sequence Features. Decision trees can be trained on raw DNA sequence features after basic preprocessing, such as one-hot encoding of nucleotides or treating them as ordinal variables (Hastie et al., 2009). However, Equation (2) reveals a fundamental limitation: the induction process considers *positions in isolation* at each node, unable to capture interactions between different sequence positions simultaneously. This limitation creates a tension between *expressivity* and *interpretability*. Since each split considers only a single position, trees must grow deep to capture complex sequence patterns that depend on multiple nucleotide positions. This compromises the interpretability that makes decision trees appealing in the first place, since predictions are described by long sequences of splitting conditions. Furthermore, this complexity also impacts generalization performance, as deeper trees might capture spurious patterns which do not generalize well at test time.

An example. We illustrate these limitations with a simple example. Given a dataset of DNA sequences, we consider the task of predicting whether or not the motif "TATA" is present, which requires considering multiple sequence positions simultaneously. As we show in Figure 1, `CART` struggles with this seemingly simple task. While it can achieve high training accuracy by growing deep enough to memorize specific position-by-position patterns, the test accuracy remains poor. This is a critical limitation since many biologically meaningful patterns emerge from the joint consideration of multiple nucleotides positions, with examples including secondary structure formation or binding site motifs. While multivariate trees (Murthy et al., 1994; Zhu et al., 2020) offer improved expressivity by splitting on linear combinations of raw features, the space of features they can explore is limited, potentially missing out on more complex combinations that could lead to better generalization. We show this in Figure 1 with `OC1` (oblique decision tree induction (Murthy et al., 1994)) achieving better training performance than `CART`, but it does not find patterns which generalize at test time.

Constraints of Manual Feature Engineering. A potential solution to the aforementioned limitation is to manually craft higher-level features that capture known biological patterns and sequence motifs. However, this approach faces two fundamental limitations. First, it remains inherently constrained by current *biological knowledge*, potentially missing important but undiscovered sequence patterns—a significant drawback when studying problems whose underlying biological mechanisms are not yet understood. Second, engineered features, such as k-mers features (Ghandi et al., 2014), are typically designed in a model-agnostic way, lacking *adaptivity* to local data characteristics that emerge during tree induction, and which should inform feature generation.

4 METHOD

The limitations of conventional decision trees detailed in Section 3 —namely their restricted expressivity when using raw features, and the constraints of manual feature engineering—show the need for a novel framework for interpretable DNA sequence analysis. To address these challenges directly, we introduce `DEFT`, which combines the *transparency* of tree structures while ensuring *expressivity* through automatically generated sequence features which yield discriminative splits during tree construction.

Adaptive Feature Generation with LLMs. Searching over the space $\mathbb{R}^{\mathcal{X}}$ of possible feature maps at every leaf node during tree induction is highly non-trivial, due to its combinatorial nature and the high dimensionality of DNA sequences. Instead, our key insight is to leverage Large Language Models (LLMs) as adaptive feature generators, capitalizing on their prior knowledge acquired through pretraining and their in-context learning capabilities. Furthermore, they are powerful hypothesis generators (Wang et al., 2023), making them particularly suitable for exploring the combinatorial space of feature maps.

Method Overview. Our method `DEFT` performs tree induction in a top-down manner, progressively growing the tree starting from the root node. At every leaf node of the partially constructed tree, `DEFT` generates an initial set of candidate feature maps through a two-step process, first producing semantic descriptions and then generating corresponding executable code for feature computation. `DEFT` then employs an evolution-

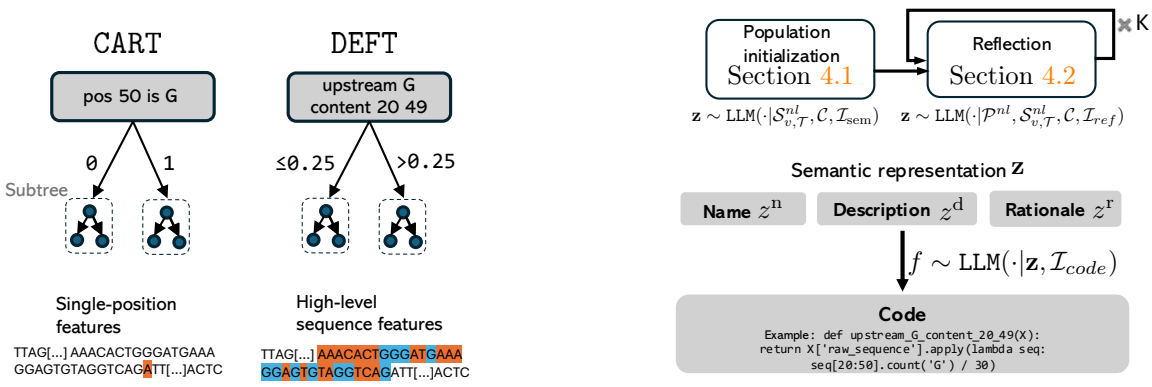


Figure 2: DEFT is a tree-based method for interpretable DNA sequence analysis. **Left**: DEFT discovers high level sequence features that can consider multiple positions simultaneously, contrasting CART. For example, the feature upstream_G_content_20_49 operates on a window of positions highlighted in orange (non-G nucleotides) and blue (G nucleotides). **Right**: It leverages LLMs to generate candidate features at each node in a two-step process: first outputting semantic representations, then converting them into Python executable code. DEFT conditions the generation of features on the partial tree structure and task-specific metadata.

inspired optimization scheme where the LLM iteratively refines the candidate features through self-reflection to optimize the splitting criterion. In what follows, we detail each of these components in turn.

4.1 Initializing the Population of Candidate Features

Given a partially constructed tree $\mathcal{T}$ and a leaf node v associated with the subset $\mathcal{D} \subset \mathcal{D}_{\text{train}}$, we seek a feature map $f^* : \mathcal{X} \to \mathbb{R}$ that is discriminative for $\mathcal{D}$. Unlike traditional top-down approaches that only search over the set $\{\phi_i\}_{i=1}^d$ of raw feature maps (cf. Equation (2)), DEFT explores a richer space of feature maps in $\mathbb{R}^{\mathcal{X}}$ by considering the unique characteristics of $\mathcal{D}$. It generates initial candidate features through a two-step process, first generating semantic representations describing the candidate features, and then obtaining executable code to compute these candidate features.

Step 1. Obtaining Semantic Representations. DEFT first generates M semantic representations $\{\mathbf{z}_j\}_{j=1}^M$, where each triplet $\mathbf{z}_j = (z_j^{\text{r}}, z_j^{\text{n}}, z_j^{\text{d}},)$ provides a human-interpretable specification of a candidate feature: z_j^{r} provides a rationale to ground the feature in potential biological relevance or observed data patterns (e.g., *"GC-rich regions often indicate regulatory elements"*), z_j^{n} provides a concise name (e.g., *"GC content"*), and z_j^{d} provides a precise description for unambiguous implementation (e.g., *"percentage of G and C nucleotides in the first 20 positions"*). The goal of the rationale z_j^{r} is to anchor the LLM's feature proposal by requiring an explicit justification, encouraging the generation of features that are not only syntactically valid but also plausible within the given biological context. To generate semantic representations suited to the characteristics of $\mathcal{D}$, we incorporate information from the partial tree structure $\mathcal{T}$ and the leaf v. Specifically, we represent the path from the root node to the leaf v in $\mathcal{T}$ as a sequence of splitting conditions $\mathcal{S}_{v,\mathcal{T}}$ defined by $\mathcal{S}_{v,\mathcal{T}} = \{(f_l, \tilde{\mathbf{z}}_l, o_l, \tau_l)\}_{l=1}^L$ where L is the path length, and each tuple consists of a feature map $f_l : \mathcal{X} \to \mathbb{R}$, its semantic representation $\tilde{\mathbf{z}}_l$, a comparison operator $o_l \in \{\leq, >\}$, and a threshold τ_l. We then serialize this node context in natural language to obtain a prompt $\mathcal{S}_{v,\mathcal{T}}^{nl}$, using the few-shot template: "$\{\tilde{z}_l^n\}$ $\{o_l\}$ $\{\tau_l\}$ $(\{\tilde{z}_l^d\})$". In addition to the node context, we also guide the generation with a task context $\mathcal{C}$ in natural language that describes the input space $\mathcal{X}$, output space $\mathcal{Y}$, and prediction task. We then sample semantic representations of candidate features from the LLM as $\mathbf{z}_j \sim \texttt{LLM}(\cdot|\mathcal{S}_{v,\mathcal{T}}^{nl}, \mathcal{C}, \mathcal{I}_{\text{sem}})$, where $\mathcal{I}_{\text{sem}}$ contains the generation instructions that specify the expected format.

Step 2. Obtaining Executable Code. For each semantic representation $\mathbf{z}_j$, we generate an executable implementation of the corresponding feature by prompting the LLM to translate the natural language representation into Python code as $f_j \sim \texttt{LLM}(\cdot|\mathbf{z}_j, \mathcal{I}_{code})$, where $\mathcal{I}_{code}$ contains instructions for producing valid Python code that computes the feature value for any input in $\mathcal{X}$.

We then define the initial population of candidate features $P = \{(f_j, \mathbf{z}_j)\}_{j=1}^K \cup \{(\phi_i, \bar{\mathbf{z}}_i)\}_{i=1}^d$, where we incorporate the raw features and their associated semantic representations in addition to the LLM-generated features.

4.2 Iterative Improvement with Reflection

While the contextual information provided by $S_{v,\mathcal{T}}$ and $\mathcal{C}$ guides the generation of the initial candidate features, it may not be sufficient to obtain highly discriminative features. We therefore propose an evolution-inspired optimization scheme that iteratively improves feature quality through LLM-based reflection.

Given a population P of features and their semantic representations for a node with dataset $\mathcal{D}$, we first evaluate the discriminative power of the features by computing for each $(f, \mathbf{z}) \in P$ the score $\eta = \min_{\tau \in \mathbb{R}} s(f, \tau, D)$, where s depends on an impurity measure Q (cf. Equation (1)). We then create a few-shot prompt $\mathcal{P}^{nl}$ by serializing each feature $(f, \mathbf{z})$ with its score η following the template *"Score:* $\{\eta\}$*, Feature name:* $\{z^n\}$*, Feature description:* $\{z^d\}$*, Feature code:* $\{f\}$*"*.

To obtain better features, we define a set of instructions $\mathcal{I}$ comprising two distinct prompt instructions: one for *exploration*, which directs the LLM to propose features distinct from P, and one for *exploitation*, which guides the LLM to analyze and refine patterns from the highest-performing in-context features. These instructions also incorporate constraints regarding the interpretability of the generated features, an aspect that we investigate in Section 5.3. For each instruction $\mathcal{I}_{ref} \in \mathcal{I}$, we generate a set of M semantic representations as $\mathbf{z}'_m \sim \texttt{LLM}(\cdot|\mathcal{P}^{nl}, \mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{ref})$. Each $\mathbf{z}'_m$ is then transformed into executable code, yielding a population P'. The M solutions from $P \cup P'$ with lowest scores are selected to form the next population, and this optimization process repeats for K iterations. Finally, we select the feature that achieves the minimum score in the final population as the splitting feature for the current node v. Since we include the raw features $\{\phi_i\}_{i=1}^d$ in the initial population, we have the guarantee that the feature selected by `DEFT` at the end of the optimization procedure will be at least as discriminative as the feature that an axis-aligned tree would select.

We summarize the different steps for feature generation in Algorithm 1 in the Appendix, and provide details on the prompts in Appendix B. Once the tree is fully constructed, predictions for new samples are obtained by traversing the tree from the root to a leaf node, as detailed in Algorithm 2, and comparing at each node the feature value against the node's learned threshold, analogous to conventional decision tree inference.

Computation Cost. A discussion of the computation cost can be found in Section B.2.5. In brief, the cost scales linearly with the number of reflection steps K. Furthermore, since feature generation at the same tree depth is independent, the process is highly parallelizable.

5 EXPERIMENTS

In this section, we evaluate `DEFT` across diverse genomics tasks. In Section 5.1, we benchmark predictive performance against tree-based baselines and black-box models, showing that `DEFT` generates highly predictive sequence features. In Section 5.2, we analyze the features learned by `DEFT`, introducing a taxonomy that reveals how they adapt to dataset-specific characteristics. In Section 5.3, we show that `DEFT` enables control of the balance between interpretability and discriminative power of the discovered features. In Section 5.4, we perform ablation studies that provide insight into the key mechanisms contributing to `DEFT` 's performance. Code to reproduce our experiments can be found at `https://github.com/nicolashuynh/deft`.

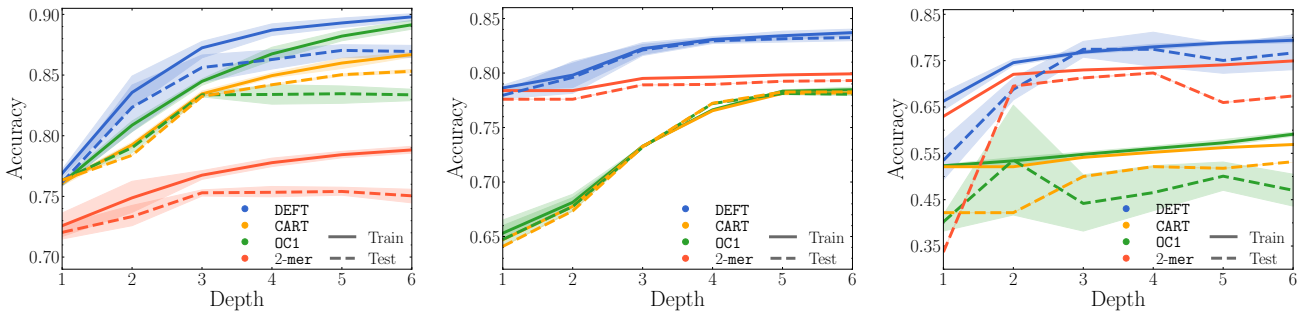
Datasets. We evaluate `DEFT` on a diverse set of genomics tasks: *Pol II Pausing* (Mayer et al., 2017), predicting RNA polymerase II pausing from DNA sequence context; *Promoters* (Grešová et al., 2023), identifying human non-TATA promoters; and *Enhancers* (Ernst et al., 2016), classifying enhancer activity from massively parallel reporter assay constructs. More details on the datasets can be found in Appendix B.

Experimental details. In what follows, we use `gpt-4o` version `1001` as the underlying LLM. We use a population size $M = 10$ and $K = 20$ reflection steps per node. The Gini index serves as the splitting criterion at each node and we set a minimum number of samples per leaf equal to 1% of the size of the training set to prevent overfitting. Appendix B provides more details on the hyperparameters and experimental setting, and code can be found in the supplementary material.

5.1 Predictive Performance of `DEFT`

Methodology. We first compare `DEFT` against interpretable tree-based baselines: 1) *CART* (Breiman et al., 1984), standard axis-aligned decision tree induction (with one-hot encoded nucleotides at each sequence position [1]) ; 2) *OC1* (Murthy et al., 1994), oblique decision trees; and 3) *CART* with 2-mer features, where counts of all possible dinucleotides serve as input features. All methods are evaluated across varying maximum tree depths $d \leq 6$, with a minimum of 1% of training samples per leaf to mitigate overfitting. We also consider a logistic regression baseline using 2-mers features. While the focus of this work is on developing *transparent* models for DNA sequence analysis, we contextualize `DEFT` 's performance by comparing it with non-transparent models: deep-learning models (*Convolutional Neural Networks*, *Transformers*), and ensembles (*XGBoost* with 2-mers).

[1] Treating nucleotides as ordinal variables empirically yielded inferior results to one-hot encoding.



(a) Pol II Pausing (b) Promoters (c) Enhancers

Figure 3: **Performance comparison against tree-based baselines.** Training and test accuracies across varying depths (mean ± 95% CIs over 5 seeds). `DEFT` consistently outperforms the tree-based baselines.

Table 1: **Performance comparison against black boxes and ensemble baselines.**

	Pol II Pausing		Promoters		Enhancers	
Method	Acc.	AUPRC	Acc.	AUPRC	Acc.	AUPRC
`CNN`	0.868 (0.007)	0.933 (0.006)	0.828 (0.005)	0.931 (0.002)	0.790 (0.065)	0.964 (0.006)
`Transformer`	0.876 (0.016)	0.942 (0.002)	0.835 (0.006)	0.933 (0.005)	0.704 (0.045)	0.952 (0.007)
`XGBoost`	0.759 (0.005)	0.848 (0.002)	0.822 (0.000)	0.925 (0.000)	0.777 (0.000)	0.965 (0.000)
`Log. Reg.`	0.769 (0.002)	0.857 (0.001)	0.796 (0.000)	0.901 (0.000)	0.730 (0.000)	0.945 (0.000)
`CART`	0.750 (0.007)	0.830 (0.003)	0.793 (0.000)	0.897 (0.000)	0.674 (0.000)	0.939 (0.000)
`DEFT`	0.869 (0.007)	0.912 (0.006)	0.833 (0.004)	0.926 (0.004)	0.767 (0.049)	0.948 (0.012)

Results. We compare in Figure 3 the training and test accuracies of `DEFT` and tree-based baselines across varying depths for 5 different seeds. `DEFT` consistently achieves superior performance on both training and test sets for the different depths. In Table 15, we compare `DEFT` against the non-transparent baselines, reporting test results (mean and std for 5 seeds) and showing that `DEFT` is competitive while remaining interpretable by design, closing the gap between transparent and black-box models. We also report additional results in Appendix C, e.g. other performance metrics, a comparative analysis with the Nucleotide Transformer (Dalla-Torre et al., 2025), results with an open-source LLM (`gpt-oss`), and performance gains obtained when ensembling `DEFT`'s trees. We also compare `DEFT` with dataset-specific deep-learning methods (PEPMAN (Feng et al., 2021) and DeePromoter (Oubounyt et al., 2019)).

5.2 What Features does `DEFT` Discover?

The performance gains reported in Section 5.1 are driven by the features that `DEFT` discovers. For concreteness, Appendix C shows a tree learnt by `DEFT` for the *Pol II pausing* task. Because `DEFT` is interpretable by design (each feature includes a human-readable description and executable code) we can directly examine what the model has learned through these semantic representations. This allows us to make the following observations:

Obs. 1—DEFT crosses categories. Across datasets, `DEFT` finds not only *composition windows* and *position checks*, but also features related to *layout/spacing* (spacing, block transitions) and *physics / epigenetics* (stacking energy, CpG spacing). These are higher-level abstractions tied to sequence structure, which leads to the following taxonomy:

Position checks: single/few bases at fixed sites.
e.g., `pos_50_is_G_and_pos_51_is_T` (Pol II), `pos_201_is_pyrimidine` (Prom.)

Composition windows: Percentage of GC/AT or counts over a span (local environment).
e.g., `upstream_G_content_20_49` (Pol II), `Interrupted_GC_Rich_6mer_Proportion` (Enh.)

Motifs: presence of specific k-mers (including palindromes).
e.g., `AP1_Motif_Variant_Proportion` (Enh.), `GAAG_CTTC_Palindromic_Proportion` (Enh.)

Layout / spacing: arrangement grammar (spacing, periodicity, GC ↔ AT block transitions).
e.g., `A_Flanked_CpG_Transition_Density` (Prom.), `Boundary_GC_AT_Transition_Density` (Enh.)

Physics / epigenetics — biophysical or CpG-based context proxies.

> *e.g.,* `upstream_base_pair_stacking_energy_10_49` (Pol II), `Potential_Methylation_Site_Density` (Prom.)

Obs. 2—Roots are higher-level. Root splits are typically *composition windows, layout/spacing, physics / epigenetics* (e.g., upstream G content 20–49; GC↔AT block transitions), integrating many positions. Deeper nodes then refine with *position checks* (e.g., pos 50 purine / 51 pyrimidine), consistent with progressive specialization along the tree.

Obs. 3—Dataset tendencies. For the *Pol II pausing* dataset, DEFT emphasizes *physics and composition* around the pause; in *Promoters*, it highlights *CpG context* and *layout/spacing*; and in *Enhancers*, it surfaces *motifs* (AP-1 variant, palindromes) plus *layout/spacing* over GC/AT blocks. This shows that DEFT adapts to the characteristics of each dataset.

5.3 DEFT Allows to Control the Balance Between Interpretability and Performance

Table 2: **Performance comparison** at different maximum tree depths, showing training and test accuracies for both methods. We report mean$_{(\text{std})}$ for 5 seeds.

	Train		Test	
d	DEFT	DEFT$_{\text{perf}}$	DEFT	DEFT$_{\text{perf}}$
1	$0.769_{(0.004)}$	$0.820_{(0.012)}$	$0.762_{(0.004)}$	$0.810_{(0.012)}$
2	$0.836_{(0.017)}$	$0.872_{(0.020)}$	$0.823_{(0.024)}$	$0.855_{(0.014)}$
3	$0.872_{(0.009)}$	$0.893_{(0.006)}$	$0.856_{(0.014)}$	$0.867_{(0.007)}$

Table 3: **Halstead metrics**: Volume, effort and difficulty (lower is better). Removing the intepretability constraints leads to more complex features.

Model	Hal. Vol.	Hal. Effort	Hal. Dif.
DEFT	15.5	15.5	0.875
DEFT$_{\text{perf}}$	19.7	17.3	1.0

Methodology. While the previous results demonstrate that DEFT generates high-level features that are discriminative (hence achieving both *interpretability* and *predictive* accuracy), we now show that DEFT also enables practitioners to control the balance between these objectives according to their requirements. To demonstrate this, we encourage the exploration and exploitation of the search space and remove the interpretability constraints from the reflection prompts in $\mathcal{I}$. These require that features be simple (the feature should be easy to understand and must not combine multiple complex phenomena), intuitive (the computational logic should be immediately clear to a biologist), and relevant (the feature must be relevant to the biological prediction task). This modification allows DEFT to focus solely on generating features that maximize predictive performance, regardless of complexity. We denote this as DEFT$_{\text{perf}}$.

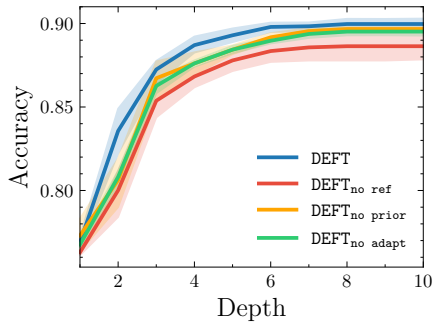


Figure 4: **Ablations.** Mean and confidence intervals at 95% for 5 seeds.

Results. We report the training and test accuracies for small tree depths in Table 2 (*Pol II pausing* task). While the gap between DEFT$_{\text{perf}}$ and DEFT reduces as d increases, there is a strong performance difference at depth 1. To provide intuition for this observation, we show in Appendix C an example of a feature discovered by DEFT$_{\text{perf}}$ at the root node, which exploits composite sequence patterns. We also compute Halstead complexity metrics (Halstead, 1977) for the features discovered by DEFT and DEFT$_{\text{perf}}$. The median results across the generated features are presented in Table 3 and confirm our intuition that removing the interpretability constraints leads to more complex features.

5.4 Ablations

Methodology. Having previously demonstrated that DEFT can explore efficiently the space of possible features to generate predictive features, we now ablate its different components to verify their contribution to the overall search performance. We conduct ablation experiments by removing the following key components: ▶ **Prior knowledge**: DEFT$_{\text{no prior}}$ removes the semantic information from the prompts for both population initialization and the reflection mechanism. This includes replacing the description of the task with generic information. ▶ **Adaptivity**: DEFT$_{\text{no adapt}}$ fits a CART model on a feature set consisting of the raw features and the features generated for the root node (which corresponds to the whole dataset). This contrasts DEFT which dynamically generates features at each node during tree induction and hence considers the local data characteristics. ▶ **Reflection.** DEFT$_{\text{no ref}}$ removes the reflection mechanism, relying solely on the LLM for population initialization.

Results. We report the search efficiency (training accuracy) for each of these ablations in Figure 4 (Pol II pausing), observing that these three components are necessary to achieve optimal search efficiency, in particular at smaller tree depths. Notably, the *reflection mechanism* (which leverages the LLM's in-context learning capabilities) plays a crucial role in navigating the search space of possible feature maps, also explaining the competitive performance of $\texttt{DEFT}_{\texttt{no prior}}$. We provide detailed analysis of the reflection's performance improvements in Appendix C.

6 DISCUSSION

In this work, we introduced DEFT, an interpretable tree-based model for DNA sequence analysis. In contrast to traditional decision trees which operate on raw features, DEFT automatically discovers high-level sequence features during tree induction. It leverages LLMs to navigate the search space of possible feature maps, exploiting both their prior knowledge and in-context learning abilities. Through comprehensive experiments on diverse genomic tasks, we demonstrate that DEFT discovers high-level and highly predictive features.

Limitations. As with traditional tree-based approaches, training data variations can influence the resulting tree — a characteristic inherent to decision tree induction that can be addressed through ensemble methods (e.g., bagging), although we note this comes at the cost of the interpretability of a single tree. Additionally, DEFT incurs higher computational costs per node compared to conventional trees due to LLM inference during feature generation (see Appendix B).

Future work. Future work may investigate the performance of DEFT on datasets with higher dimensionality or other domains with structured modalities (e.g. amino acids). Furthermore, while DEFT demonstrates strong empirical performance, establishing formal guarantees for the convergence of the evolutionary reflection mechanism constitutes an interesting research direction.

Acknowledgements

The authors would like to thank Tennison Liu and the four anonymous AISTATS reviewers for useful comments on an earlier version of the manuscript. NH is funded by Illumina, KK by Roche. This work was supported by Azure sponsorship credits granted by Microsoft's AI for Good Research Lab.

References

(2025). Gene expression omnibus.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. (2018). Sanity checks for saliency maps. *Advances in neural information processing systems*, 31.

Aglin, G., Nijssen, S., and Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3146–3153.

Albaradei, S., Magana-Mora, A., Thafar, M., Uludag, M., Bajic, V. B., Gojobori, T., Essack, M., and Jankovic, B. R. (2020). Splice2deep: An ensemble of deep convolutional neural networks for improved splice site prediction in genomic dna. *Gene*, 763:100035.

Alipanahi, B., Delong, A., Weirauch, M. T., and Frey, B. J. (2015). Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838.

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., Assael, Y., Jumper, J., Kohli, P., and Kelley, D. R. (2021a). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203.

Avsec, Ž., Weilert, M., Shrikumar, A., Krueger, S., Alexandari, A., Dalal, K., Fropf, R., McAnany, C., Gagneur, J., Kundaje, A., et al. (2021b). Base-resolution models of transcription-factor binding reveal soft motif syntax. *Nature genetics*, 53(3):354–366.

Bentley, D. (2014). Coupling mrna processing with transcription in time and space. *Nat Rev Genet*, 15:163–175.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984). *Classification and Regression Trees*. CRC Press.

Brixi, G., Durrant, M. G., Ku, J., Poli, M., Brockman, G., Chang, D., Gonzalez, G. A., King, S. H., Li, D. B., Merchant, A. T., et al. (2025). Genome modeling and design across all domains of life with evo 2. *BioRxiv*, pages 2025–02.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Brownlee, A. E., Callan, J., Even-Mendoza, K., Geiger, A., Hanna, C., Petke, J., Sarro, F., and Sobania, D. (2023). Enhancing genetic improvement mutations using large language models. In *International Symposium on Search Based Software Engineering*, pages 153–159. Springer.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cramer, P. (2019). Eukaryotic transcription turns 50. *Cell*, 179:808–812.

Dalla-Torre, H., Gonzalez, L., Mendoza-Revilla, J., Lopez Carranza, N., Grzywaczewski, A. H., Oteri, F., Dallago, C., Trop, E., de Almeida, B. P., Sirelkhatim, H., et al. (2025). Nucleotide transformer: building and evaluating robust foundation models for human genomics. *Nature Methods*, 22(2):287–297.

Ernst, J., Melnikov, A., Zhang, X., Wang, L., Rogov, P., Mikkelsen, T. S., and Kellis, M. (2016). Genome-scale high-resolution mapping of activating and repressive nucleotides in regulatory regions. *Nature biotechnology*, 34(11):1180–1190.

Face, H. (2023). huggingface.co/instadeepai/nucleotide-transformer-500m-human-ref.

Fallahpour, A., Magnuson, A., Gupta, P., Ma, S., Naimer, J., Shah, A., Duan, H., Ibrahim, O., Goodarzi, H., Maddison, C. J., et al. (2025). Bioreason: Incentivizing multimodal biological reasoning within a dna-llm model.

Feng, P., Xiao, A., Fang, M., Wan, F., Li, S., Lang, P., Zhao, D., and Zeng, J. (2021). A machine learning-based framework for modeling transcription elongation. *Proceedings of the National Academy of Sciences*, 118(6):e2007450118.

Fong, N., Sheridan, R. M., Ramachandran, S., and Bentley, D. L. (2022). The pausing zone and control of rna polymerase ii elongation by spt5: Implications for the pause-release model. *Molecular cell*, 82(19):3632–3645.

Ghandi, M., Lee, D., Mohammad-Noori, M., and Beer, M. A. (2014). Enhanced regulatory sequence prediction using gapped k-mer features. *PLoS computational biology*, 10(7):e1003711.

Grešová, K., Martinek, V., Čechák, D., Šimeček, P., and Alexiou, P. (2023). Genomic benchmarks: a collection of datasets for genomic sequence classification. *BMC Genomic Data*, 24(1):25.

Halstead, M. H. (1977). *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

Han, S., Yoon, J., Arik, S. O., and Pfister, T. (2024). Large language models can automatically engineer features for few-shot tabular learning. In *Forty-first International Conference on Machine Learning*.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Hollmann, N., Müller, S., and Hutter, F. (2024). Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Hu, X., Rudin, C., and Seltzer, M. (2019). Optimal sparse decision trees. *Advances in Neural Information Processing Systems*, 32.

Jonkers, I. and Lis, J. (2015). Getting up to speed with transcription elongation by rna polymerase ii. *Nat Rev Mol Cell Biol*, 16:167–177.

Le, N. Q. K., Yapp, E. K. Y., Nagasundaram, N., and Yeh, H.-Y. (2019). Classifying promoters by interpreting the hidden information of dna sequences via deep learning and combination of continuous fasttext n-grams. *Frontiers in bioengineering and biotechnology*, 7:305.

Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., and Stanley, K. O. (2023). Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pages 331–366. Springer.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. (2022). Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. (2020). Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR.

Linder, J., Srivastava, D., Yuan, H., Agarwal, V., and Kelley, D. R. (2025). Predicting rna-seq coverage from dna sequence as a unifying model of gene regulation. *Nature Genetics*, pages 1–13.

Lipton, Z. C. (2018). The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57.

Liu, F., Xialiang, T., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., and Zhang, Q. (2024). Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Forty-first International Conference on Machine Learning*.

Mayer, A., Landry, H. M., and Churchman, L. S. (2017). Pause & go: from the discovery of rna polymerase pausing to its functional implications. *Current opinion in cell biology*, 46:72–80.

Mooney, R., Zhu, J., Saba, J., and Landick, R. (2025). Nusg-spt5 transcription factors: Universal, dynamic modulators of gene expression. *J Mol Biol*, 437:168814.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080.

Murthy, S. K., Kasif, S., and Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32.

Nam, J., Kim, K., Oh, S., Tack, J., Kim, J., and Shin, J. (2024). Optimized feature generation for tabular data via llms with decision tree reasoning. *arXiv preprint arXiv:2406.08527*.

Nguyen, E., Poli, M., Faizi, M., Thomas, A., Wornow, M., Birch-Sykes, C., Massaroli, S., Patel, A., Rabideau, C., Bengio, Y., et al. (2023). Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *Advances in neural information processing systems*, 36:43177–43201.

Noe Gonzalez, M., Blears, D., and Svejstrup, J. (2021). Causes and consequences of rna polymerase ii stalling during transcript elongation. *Nature Reviews Molecular Cell Biology*, 22:3–21.

Oubounyt, M., Louadi, Z., Tayara, H., and Chong, K. T. (2019). Deepromoter: robust promoter predictor using deep learning. *Frontiers in genetics*, 10:286.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1:81–106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215.

Scalzitti, N., Kress, A., Orhand, R., Weber, T., Moulinier, L., Jeannin-Girardon, A., Collet, P., Poch, O., and Thompson, J. D. (2021). Spliceator: multi-species splice site prediction using convolutional neural networks. *BMC bioinformatics*, 22:1–26.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMlR.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Soleimanian, F., Mohammadi, P., and Hakimi, P. (2012). Application of decision tree algorithm for data mining in healthcare operations: a case study. *Int J Comput Appl*, 52(6):21–26.

Stormo, G. D. (2000). Dna binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Veloso, A., Kirkconnell, K., Magnuson, B., Biewen, B., Paulsen, M., Wilson, T., and Ljungman, M. (2014). Rate of elongation by rna polymerase ii is associated with specific gene features and epigenetic modifications. *Genome Res*, 24:896–905.

Verwer, S. and Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632.

Vrajitoru, D. (2000). Large population or many generations for genetic algorithms? implications in information retrieval. In *Soft computing in information retrieval: Techniques and applications*, pages 199–222. Springer.

Wang, R., Wang, Z., Wang, J., and Li, S. (2019). Splicefinder: ab initio prediction of splice sites using convolutional neural network. *BMC bioinformatics*, 20:1–13.

Wang, R., Zelikman, E., Poesia, G., Pu, Y., Haber, N., and Goodman, N. D. (2023). Hypothesis search: Inductive reasoning with language models. *arXiv preprint arXiv:2309.05660*.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. (2024). Large language models as optimizers.

Zeng, H., Edwards, M. D., Liu, G., and Gifford, D. K. (2016). Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127.

Zhang, J. and Landick, R. (2016). A two-way street: Regulatory interplay between rna polymerase and nascent rna structure. *Trends Biochem Sci*, 41:293–310.

Zhou, J. and Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934.

Zhou, Z., Ji, Y., Li, W., Dutta, P., Davuluri, R. V., and Liu, H. (2024). Dnabert-2: Efficient foundation model and benchmark for multi-species genomes.

Zhu, H., Murali, P., Phan, D., Nguyen, L., and Kalagnanam, J. (2020). A scalable mip-based method for learning optimal multivariate decision trees. *Advances in neural information processing systems*, 33:1771–1781.

CHECKLIST

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] Details are provided in Appendix B.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] We discuss computational complexity in Appendix B.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] We provide code in the supplementary material.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] We provide code in the supplementary material.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] We provide experimental details in Appendix B.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes] We provide those in the main text.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] We provide those in Appendix B.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes] We provide those in Appendix B.
   (b) The license information of the assets, if applicable. [Yes] We provide those in Appendix B.
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Material

A EXTENDED RELATED WORKS

Our work intersects with several areas in machine learning and computational biology, which we detail below.

Machine Learning for DNA Sequence Analysis. The analysis of DNA sequences using machine learning has seen significant advances in recent years. Traditional approaches rely on position weight matrices and k-mer based models (Stormo, 2000), which provide interpretable results but often lack the expressivity to capture complex sequence patterns. More recently, deep learning architectures have demonstrated superior predictive performance across various genomic tasks. Convolutional neural networks have proven particularly effective for DNA sequence analysis, with (Alipanahi et al., 2015) showing their ability to learn regulatory motifs and predict transcription factor binding sites. This line of work has been extended through architectures like DeepSEA (Zhou and Troyanskaya, 2015), which can identify sequence patterns at multiple spatial scales. Transformer-based models have further advanced the field, with works like Enformer (Avsec et al., 2021a) or the Nucleotide Transformer (Dalla-Torre et al., 2025) demonstrating the ability to capture dependencies in genomic sequences. While these deep learning approaches achieve remarkable performance, their black-box nature and their large scale ($500M$ parameters for the Nucleotide Transformer) makes their predictions non transparent, which does not answer our original question: *How do we design models that are inherently transparent, yet expressive for DNA sequence analysis?*

Interpreting Black Boxes in DNA Sequence Analysis. The need for interpretability in genomic applications has led to various approaches to explain machine learning models. Methods like integrated gradients (Sundararajan et al., 2017), DeepLIFT (Shrikumar et al., 2017), visualization of attention maps (Avsec et al., 2021b), have been widely used to probe these black boxes. However, these post-hoc interpretation methods have several limitations. They explain individual predictions rather than providing global model understanding, and the extracted patterns may not faithfully represent the model's predictions (Rudin, 2019). In contrast, `DEFT` is inherently *transparent*, using a tree structure with human-understandable features.

Decision Trees. Decision trees are valued for their interpretability and ability to capture nonlinear patterns. Greedy algorithms sequentially grow trees with a top-down approach. Popular methods in this class of algorithms are `CART` (Breiman et al., 1984), `ID3` (Quinlan, 1986) and `C4.5` (Quinlan, 2014). Another branch of methods use combinatorial optimization techniques to search for sparse, optimal trees, e.g. branch and bound (Lin et al., 2020) and dynamic programming (Aglin et al., 2020). Notable works include `BinOCT` (Verwer and Zhang, 2019), `DL85` (Aglin et al., 2020), `OSDT` (Hu et al., 2019), and `GOSDT` (Lin et al., 2020). A common limitation of these methods is their reliance on single-feature splits. Hence some works have explored ways to enhance tree expressivity, for example through oblique splits (Murthy et al., 1994). However, these methods explore a restricted search space (e.g. linear combinations of features), and are very difficult to optimize given the non-differentiability of the objective function, which is exacerbated by the high dimensionality of genomic datasets.

LLMs in Scientific Applications. Recent work has demonstrated the potential of large language models (LLMs) in scientific applications beyond natural language processing. LLMs have shown strong capabilities in tasks like mathematical reasoning (Lewkowycz et al., 2022), code generation (Chen et al., 2021). Particularly relevant to our work are approaches using LLMs for hypothesis generation in scientific discovery (Wang et al., 2023) and genetic algorithms (Liu et al., 2024). The use of LLMs for feature engineering is an emerging area, with recent works exploring their potential for tabular data (Han et al., 2024; Hollmann et al., 2024). However, these approaches typically treat feature generation as a standalone preprocessing step, while `DEFT` integrates feature discovery in the tree induction process, allowing for adaptivity based on local data characteristics. Furthermore, while (Han et al., 2024; Hollmann et al., 2024) mostly construct features based on compositions of simple arithmetic operations (e.g. $+$, $-$, $\times$) applied to continuous features, `DEFT` can discover high-level features which take into account the sequential nature of the data (motifs, counts...).

B EXPERIMENTAL DETAILS

Code Release. Code can be found at https://github.com/nicolashuynh/deft.

Compute Resources. All the experiments were conducted on a machine equipped with a 18-Core Intel Core i9-10980XE CPU, and a NVIDIA GeForce RTX 3080.

B.1 Details on the Datasets

Data Processing. `DEFT` can operate on the original raw features without any preprocessing, since it generates code representations that can take as input dataframes. However, `CART` can only process continuous or ordinal features. Therefore, for `CART` we one-hot encode each position in the sequences, yielding a total of $404 = 4 \times 101$ features. This approach outperformed the alternative of treating nucleotides as ordinal variables (computing the ordering based on the label proportions (Hastie et al., 2009)), which is why we adopt it in Section 5.

B.1.1 Pol II Pausing

The Polymerase II pausing task is based on a dataset (Fong et al., 2022) publicly available on the GEO platform (GEO, 2025), with accession number `GSE202749` (licensed under a Creative Commons Attribution 4.0 International License).

Background on Pol II Pausing. We first give some background information on RNA polymerase II pausing. RNA Polymerase II (Pol II) is responsible for transcribing protein-coding and many non-coding genes. Its transcription cycle includes an elongation phase where the RNA chain grows (Cramer, 2019). This growth is not continuous; Pol II intrinsically pauses at specific sites, temporarily halting nucleotide addition. These pauses are fundamental to gene regulation, as the average transcription velocity—varying from ≤ 0.5 to ≥ 5 kb/min (Noe Gonzalez et al., 2021; Jonkers and Lis, 2015)—is primarily governed by pause frequency and duration. This speed, in turn, kinetically couples to co-transcriptional events such as RNA splicing, polyadenylation, and chromatin modifications (Bentley, 2014). The importance of pausing is underscored by the universal conservation of Spt5/NusG, a protein dedicated to its regulation (Mooney et al., 2025). While methods like NET-seq have precisely mapped reproducible pause sites, the full set of sequence determinants remains unknown. Pausing likely involves extensive contacts within the transcription elongation complex (Pol II, DNA, RNA, elongation factors), as well as influences from RNA secondary structure, DNA sequence-dependent nucleosome positioning, and the stability of RNA-DNA hybrids and dsDNA (Veloso et al., 2014; Zhang and Landick, 2016).

Description. Following the description of (Fong et al., 2022), this dataset was collected using an enhanced version of NET-seq (eNET-seq), which maps RNA Polymerase II pausing at single-base resolution in human HCT116 cells. The standard NET-seq protocol was modified with optimized MNase digestion, decapping enzymes, and unique molecular identifiers (UMIs) for accurate quantification. The resulting data captures the precise positions of paused RNA Polymerase II complexes across the genome by sequencing the $3'$ ends of nascent RNA transcripts that are protected from MNase digestion by the polymerase. Pause sites in the dataset are defined using three specific quantitative criteria: each pause site must have an eNET-seq signal that exceeds the mean signal of its surrounding 200 bp window by more than 3 standard deviations, contain at least 5 reads at the precise pause position, and have a minimum of 5 additional reads within the surrounding window.

Dataset Statistics. The DNA sequences have length 101, and the labels are defined with respect to the central position in each sequence (indexed as position 50). We summarize the statistics of the dataset in Table 4. The random seeds in the experiments on the Pol II dataset control the training set, while the test set remains fixed.

Table 4: Pol II pausing dataset characteristics

Characteristic	Value
Total number of samples	6000
Training set size	4000
Test set size	2000
Dimensionality of the sequences	101
Label distribution (training set)	Pausing: 51%, Non-pausing: 49%

B.1.2 Human non-TATA Promoters

The identification of human non-TATA promoters task uses the dataset from the benchmarks suite (Grešová et al., 2023) under the Apache-2.0 license. It involves classifying 251bp genomic sequences as either non-TATA promoters (positive class; sequences spanning -200bp to +50bp around a transcription start site lacking a TATA-box) or non-promoter regions (negative class; random 251bp fragments from human gene regions located after the first exons).

Intuitively, this promoter classification task differs from the Pol II pausing task in several key aspects. The Pol II pausing task focuses on identifying relatively short, localized sequence motifs (within a 100bp window). In contrast, the non-TATA promoter task requires the features to model a broader, more diverse set of distributed sequence elements in the absence of the strong, canonical TATA-box signal.

We give the characteristics of the dataset in Table 5.

Table 5: Non-TATA Promoters dataset characteristics

Characteristic	Value
Total number of samples	36131
Training set size	27097
Test set size	9034
Dimensionality of the sequences	251
Label distribution (training set)	1: 54%, 0: 46%

B.1.3 Enhancers

The enhancer classification task uses MPRA data from (Ernst et al., 2016) (under MIT license). We use the data corresponding to the K562 cell line, minimal promoter, and average replicate. We define the binary labels via a thresholding approach, where positive samples are such that their activity is two standard deviations above the mean (calculated from the training set), and negative samples are such that their activity is two standard deviations below the mean. We also subsample the dataset to make the label distribution balanced. The characteristics of the data can be found in Table 6.

Table 6: Enhancers dataset characteristics

Characteristic	Value
Total number of samples	16390
Training set size	16108
Test set size	282
Dimensionality of the sequences	145
Label distribution	1: 50%, 0: 50%

B.2 Details on the Method

B.2.1 Algorithms: Feature Generation and Inference

Algorithm 1 describes the *feature generation process* at any node v, while Algorithm 2 describes how to get predictions from the trees at *inference time*.

Algorithm 1 Feature generation at node v

Require: Node v, tree $\mathcal{T}$, subset $\mathcal{D}$, task context $\mathcal{C}$, population size M, number of iterations K

$\mathcal{S}^{nl}_{v,\mathcal{T}} \leftarrow$ serialized sequence of splitting conditions from root to v
$P \leftarrow \emptyset$ {Initial population}
for $j = 1$ to M **do**
  $\mathbf{z}_j \sim \texttt{LLM}(\cdot|\mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{sem})$ {Semantic rep.}
  $f_j \sim \texttt{LLM}(\cdot|\mathbf{z}_j, \mathcal{I}_{code})$ {Executable code}
  $P \leftarrow P \cup \{(f_j, \mathbf{z}_j)\}$
end for
$P \leftarrow P \cup \{(\phi_i, \bar{\mathbf{z}}_i)\}_{i=1}^{d}$ {Add raw features}
Compute scores $\eta = \min_{\tau \in \mathbb{R}} s(f, \tau, \mathcal{D})$ for all f in P
for $k = 1$ to K **do**
  Create few-shot prompt $\mathcal{P}^{nl}$ from P and scores
  for $\mathcal{I}_{ref} \in \mathcal{I}$ **do**
    for $m = 1$ to M **do**
      $\mathbf{z}'_m \sim \texttt{LLM}(\cdot|\mathcal{P}^{nl}, \mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{ref})$
      $f'_m \sim \texttt{LLM}(\cdot|\mathbf{z}'_m, \mathcal{I}_{code})$
    end for
    $P' \leftarrow P' \cup \{(f'_m, \mathbf{z}'_m)\}_{m=1}^{M}$
    Compute scores for all features in P'
  end for
  $P \leftarrow \texttt{Top-M}(P \cup P')$ {Selection on scores}
end for
return $(f^*, \mathbf{z}^*, \tau^*) = \arg\min_{(f,\mathbf{z}) \in P, \tau} s(f, \tau, \mathcal{D})$

Algorithm 2 Prediction with a fitted tree

Require: Fitted tree t, input sample $\mathbf{x} \in \mathcal{X}$

$v \leftarrow$ root node of t
while v is not a leaf node **do**
  Let (f_v, τ_v) be the feature and threshold associated with node v.
  $f_v(\mathbf{x}) \leftarrow$ Compute feature value for sample $\mathbf{x}$ using executable code of f_v.
  if $f_v(\mathbf{x}) \leq \tau_v$ **then**
    $v \leftarrow$ left child of v
  else
    $v \leftarrow$ right child of v
  end if
end while
Let $\hat{y}_v$ be the prediction associated with leaf node v.
return $\hat{y}_v$

B.2.2 LLM Hyperparameters

We detail in Table 7 the hyperparameters of the LLM used throughout our experiments.

Table 7: LLM parameters

Parameter	Value
Model name	`gpt-4o`
Version	`1001`
Temperature	1
Top p	0.95

`DEFT` uses a rejection mechanism where invalid features (e.g. which cannot be automatically parsed) are discarded.

While our implementation and experiments utilize `gpt-4o` (version 1001) as the underlying LLM, we note that `DEFT` is model-agnostic and can be readily adapted to work with any LLM that demonstrates capabilities in natural language understanding and code generation. For instance, we instantiate `DEFT` with the `gpt-oss` model in Section C.7.

B.2.3 `DEFT`-specific Hyperparameters

In our experimental section, `DEFT` uses a population size $M = 10$, and a number of reflections $K = 20$. As in traditional genetic algorithms, larger populations would lead to a more thorough search (Vrajitoru, 2000), however in practice it is limited by the size of the context window of the LLM and the token limit rate. Furthermore, more reflection steps would also lead to a more thorough exploration of the feature space, but at the cost of a higher number of LLM inferences. This motivates our choice of hyperparameters, which allows a good trade off between these computational considerations and search performance.

`DEFT` uses the Gini index as the splitting criterion, defined as:

$$Q(\mathcal{D}) = 1 - \left(\frac{\sum_{(x,y)\in\mathcal{D}} \mathbb{1}(y=0)}{|\mathcal{D}|} \right)^2 - \left(\frac{\sum_{(x,y)\in\mathcal{D}} \mathbb{1}(y=1)}{|\mathcal{D}|} \right)^2 \tag{3}$$

B.2.4 Prompts

Prompt Structure. Each prompt for feature generation contains:

1. the *task context* $\mathcal{C}$: describes the input space, the label, the characteristics of the dataset, and the tree induction task
2. the *node context* $\mathcal{S}^{nl}_{v,\mathcal{T}}$: lists the sequence of splitting conditions from the root node to the current node v
3. *interpretability instructions*: prevents composite features combining multiple mechanisms
4. *task-specific instructions*: for example, instructions for exploration and exploitation in reflection

In addition to that, the reflection prompts contain in-context features along with their scores.

Examples of Prompts. In what follows, we provide examples of prompts (Pol II pausing task) for population initialization in Listing 1, reflection (exploration in Listing 2, exploitation in Listing 3) and code generation in Listing 4.

```
Your goal is to help with the task of growing a decision tree to predict RNA
    polymerase pausing.  This is a dataset about RNA polymerase pausing. Given a
    current node that we want to split in the tree, you will  construct a new
    feature based on the original DNA sequence, such that this new feature  is
    discriminative for the prediction task of classifying RNA polymerase pausing.
    The raw  feature is the DNA sequence of length 101 centered on the pause site.
    Positions from 0 to 49 included correspond to the upstream region. Position 50
    corresponds to the site. Positions from 51 to 100 included are in the
    downstream region. The possible  nucleotide values are A, C, G, T.

The features are:
raw_sequence: text (average length: 101.0 characters)

 The dataset has 1644 samples.

<Beginning Splitting conditions from root to current node>
upstream_G_content_20_49 smaller than 0.250 (Calculate the proportion of guanine
    (G) nucleotides in the upstream region from positions 20 to 49. Count the
    number of G nucleotides in this region and divide by 30 to get the proportion.)
<End of Splitting conditions from root to current node>
```

```
Leverage your biology expertise and your creativity to generate a good feature
    (name, description, justification). This feature can be simple. Importantly,
    take into account the contextual history given above, which defines the
    sequence of splitting conditions from the root node up to the current node.
    Another important point: The feature should be interpretable. The feature
    should capture a single, clear biological mechanism. Use only basic sequence
    properties that a biologist could understand and find intuitive. Avoid
    combining multiple biological mechanisms into one feature, which would
    introduce complexity. Be very explicit in your description on how you compute
    the feature. Return the feature in the following JSON format:
{"rationale": "rationale", "description": "your_feature_description", "name":
    "your_feature_name"}
Only return the JSON object, with no additional text.
```

Listing 1: **Example prompt for population initialization**

```
Your goal is to help with the task of growing a decision tree to predict RNA
    polymerase pausing.  This is a dataset about RNA polymerase pausing. Given a
    current node that we want to split in the tree, you will  construct a new
    feature based on the original DNA sequence, such that this new feature  is
    discriminative for the prediction task of classifying RNA polymerase pausing.
    The raw  feature is the DNA sequence of length 101 centered on the pause site.
    Positions from 0 to 49 included correspond to the upstream region. Position 50
    corresponds to the site. Positions from 51 to 100 included are in the
    downstream region. The possible  nucleotide values are A, C, G, T.

The features are:
raw_sequence: text (average length: 101.0 characters)

 The dataset has 1644 samples.

Given a current node that we want to split in the tree, you will construct a new
    feature based on the original DNA sequence, such that this new feature is
    discriminative  for the prediction task of classifying RNA polymerase pausing.
    The feature should be: 1. Biologically interpretable - based on possible
    mechanisms of transcription and intuitive  for biologists 2. Computationally
    clear - specify exact positions and calculation methods,  3. Complementary to
    previous splitting conditions from the root to current node. I am going to give
    you an initial population of features with their respective  scores (lower is
    better). The feature you generate should be as different as possible from the
    initial population in order to explore new ideas.

Another important point: The feature should be interpretable. To assess
    interpretability, please  consider the following aspects: 1) Simplicity: the
    should be simple and easy to understand.  An interpretable feature should not
    be overly complex (and not involve multiple complex  phenomena). Hence features
    which incorporate too many multiple distinct components are not  simple and
    should not be generated. 2) Intuitiveness: The feature should be intuitive and
    easy to explain to a biologist 3) Relevance: the feature should be relevant to
    the  biological prediction task.

Return three things: 1. Rationale: describe step by  step how you came up with this
    feature, taking into account both the splitting conditions  and the population
    of candidate features. If you feature involves a physical or biological
    mechanism, say why it is relevant. 2. Description: precise calculation method
    3. Name:  clear and descriptive. Return in the following JSON format:
{"rationale": "your_rationale", "description": "your_feature_description", "name":
    "your_feature_name"}
Only return the JSON object, with no additional text. Do not include "json" in
    front of it

<Beginning of the population of features>
Here is the list of features along with their score:
Feature 1
Score: 0.2667
 Feature name: upstream_GC_content_10_29
 Feature description: Calculate the proportion of guanine (G) and cytosine (C)
     nucleotides in the upstream region from positions 10 to 29. Count the number
     of G and C nucleotides in this region and divide by 20 to get the proportion.
 Feature code: def add_upstream_GC_content_10_29(X):
    def calculate_gc_content(seq):
        upstream_region = seq[10:30]
        gc_count = upstream_region.count('G') + upstream_region.count('C')
        return gc_count / 20

    return X['raw_sequence'].apply(calculate_gc_content)
...
Feature 10
Score: 0.195
 Feature name: pos_50_is_G_and_pos_51_is_T
 Feature description: Check if position 50 in the raw sequence is G and position 51
```

```
    is T. Return 1 if both conditions are true, otherwise 0.
 Feature code: def construct_feature(X):
    return X['raw_sequence'].apply(lambda seq: 1 if len(seq) > 51 and seq[50] ==
        'G' and seq[51] == 'T' else 0)
<End of the population of features>

<Beginning Splitting conditions from root to current node>
upstream_G_content_20_49 smaller than 0.250 (Calculate the proportion of guanine
    (G) nucleotides in the upstream region from positions 20 to 49. Count the
    number of G nucleotides in this region and divide by 30 to get the proportion.)
<End of Splitting conditions from root to current node>
```

Listing 2: **Example prompt for reflection (exploration)**

```
Your goal is to help with the task of growing a decision tree to predict RNA
    polymerase pausing.  This is a dataset about RNA polymerase pausing. Given a
    current node that we want to split in the tree, you will  construct a new
    feature based on the original DNA sequence, such that this new feature  is
    discriminative for the prediction task of classifying RNA polymerase pausing.
    The raw  feature is the DNA sequence of length 101 centered on the pause site.
    Positions from 0 to 49 included correspond to the upstream region. Position 50
    corresponds to the site. Positions from 51 to 100 included are in the
    downstream region. The possible  nucleotide values are A, C, G, T.

The features are:
raw_sequence: text (average length: 101.0 characters)

 The dataset has 1644 samples.

Given a current node that we want to split in the tree, you will construct a new
    feature based on the original DNA sequence, such that this new feature is
    discriminative  for the prediction task of classifying RNA polymerase pausing.
    The feature should be: 1. Biologically interpretable - based on possible
    mechanisms of transcription and intuitive  for biologists 2. Computationally
    clear - specify exact positions and calculation methods,  3. Complementary to
    previous splitting conditions from the root to current node. I am going to give
    you an initial population of features with their respective  scores (lower is
    better). Very important: First identify common ideas of the top performing
    solutions in the population. Then base your feature on these common ideas and
    simplify them to get one good feature, but do not simply combine these common
    ideas (do not just add them for example, that would create too much complexity).
Another important point: The feature should be interpretable. To assess
    interpretability, please  consider the following aspects: 1) Simplicity: the
    should be simple and easy to understand.  An interpretable feature should not
    be overly complex (and not involve multiple complex  phenomena). Hence features
    which incorporate too many multiple distinct components are not  simple and
    should not be generated. 2) Intuitiveness: The feature should be intuitive and
    easy to explain to a biologist 3) Relevance: the feature should be relevant to
    the  biological prediction task.

Return three things: 1. Rationale: describe step by  step how you came up with this
    feature, taking into account both the splitting conditions  and the population
    of candidate features. If you feature involves a physical or biological
    mechanism, say why it is relevant. 2. Description: precise calculation method
    3. Name:  clear and descriptive. Return in the following JSON format:
{"rationale": "your_rationale", "description": "your_feature_description", "name":
    "your_feature_name"}
Only return the JSON object, with no additional text. Do not include "json" in
    front of it

<Beginning of the population of features>
Here is the list of features along with their score:
Feature 1
Score: 0.2667
 Feature name: upstream_GC_content_10_29
 Feature description: Calculate the proportion of guanine (G) and cytosine (C)
```

```
    nucleotides in the upstream region from positions 10 to 29. Count the number
    of G and C nucleotides in this region and divide by 20 to get the proportion.
 Feature code: def add_upstream_GC_content_10_29(X):
    def calculate_gc_content(seq):
        upstream_region = seq[10:30]
        gc_count = upstream_region.count('G') + upstream_region.count('C')
        return gc_count / 20

    return X['raw_sequence'].apply(calculate_gc_content)
...
Feature 10
Score: 0.195
 Feature name: pos_50_is_G_and_pos_51_is_T
 Feature description: Check if position 50 in the raw sequence is G and position 51
    is T. Return 1 if both conditions are true, otherwise 0.
 Feature code: def construct_feature(X):
    return X['raw_sequence'].apply(lambda seq: 1 if len(seq) > 51 and seq[50] ==
        'G' and seq[51] == 'T' else 0)
<End of the population of features>


<Beginning Splitting conditions from root to current node>
upstream_G_content_20_49 smaller than 0.250 (Calculate the proportion of guanine
    (G) nucleotides in the upstream region from positions 20 to 49. Count the
    number of G nucleotides in this region and divide by 30 to get the proportion.)
<End of Splitting conditions from root to current node>
```

Listing 3: **Example prompt for reflection (exploitation)**

```
Your goal is to generate a python code to construct a feature, based on a dataframe.

 You should build this feature using the following original features:
raw_sequence: text (average length: 101.0 characters)

The feature you should generate has the following characteristics:
Feature name: pos_50_is_G_and_pos_51_is_T
 Feature description: Check if position 50 in the raw sequence is G and position 51
    is T. Return 1 if both conditions are true, otherwise 0.

The raw feature is the DNA sequence of length 101 centered on the pause site.
    Positions from 0 to 49 included correspond to the upstream region. Position 50
    corresponds to the site. Positions from 51 to 100 included are in the
    downstream region. The possible  nucleotide values are A, C, G, T. Give
    instantly executable code without example usage. Only return the Python
    function,  with no additional text. The name of the argument should be 'X'.
    Only output one function,  this is very important. The function should only
    return the new feature column. Start your  output with the string 'def
    {Function Name}(X):' do not include 'python' in front of it
```

Listing 4: **Example prompt for code generation**

B.2.5 Computational Overhead

Compared to conventional decision tree induction (e.g. `CART`), `DEFT` incurs an additional computational overhead which mostly comes from the LLM inference time at each node during tree construction, both for generating the initial population of candidate features and during the reflection mechanism. As such, the cost scales linearly with the number of reflection steps K. For example, each split for tree induction on the Pol II pausing task took on average 353 seconds. We note that the overall run time could be greatly reduced by parallelizing the construction of the splits at the same depth, as the total cost would then scale linearly in the depth, rather than on the number of nodes. This constitutes an interesting avenue for future work, but it falls outside the scope of the current work as our key contributions are conceptual and methodological (i.e. a method to embed feature generation into tree induction for DNA sequence analysis).

We emphasize that this computational overhead allows `DEFT` to discover novel features which are highly

discriminative, hence requiring fewer splits to achieve comparable performance to conventional trees. Hence, the comparison between DEFT and CART must consider two axes: *computational cost* and *search space complexity*. While CART's search space is highly constrained (it can only evaluate splits along the predefined raw features), DEFT explores an effectively infinite search space of potential features, and the higher per-node cost is directly exchanged for greater performance (cf Figure 3). Finally, assuming sufficient compute is available for inference, the overall runtime could be significantly reduced through parallelization: since the feature generation process for nodes at the same depth is independent, the tree's branches can be constructed in parallel.

Finally, it is important to note that DEFT provides valuable *insights* for the practitioner by *automatically* discovering *human-interpretable features* that can capture high level sequence patterns, going beyond single-position splits.

Remark on Computational Cost with Respect to Sequence Length. The computational cost of DEFT depends on the complexity of the Python code generated for the features at each split. While the LLM cost for generating the feature descriptions can realistically be assumed to be independent of sequence length (within its context window, and when not providing in-context samples), the cost of executing the generated feature code on a sequence of length L varies. Crucially, DEFT can generate features whose computational complexity scales linearly with L (e.g. counting motifs, calculating GC content in a window). For instance, the "upstream G content 20-49" feature involves a simple string scan over a fixed-size or proportional window, resulting in $O(L)$ or even $O(1)$ complexity if the window size is constant relative to L. This contrasts with methods like standard Transformers whose self-attention mechanism often leads to quadratic complexity with sequence length. Therefore, if DEFT predominantly generates features with linear or sub-linear complexity, its application can potentially be significantly more scalable for very long sequences compared to models with inherent quadratic or higher-order scaling with L. The practitioner can also guide the LLM (via prompts during the reflection step) to favor computationally simpler features if scalability for long sequences is a primary concern.

B.3 Details on the Baselines

B.3.1 OC1

OC1 (Murthy et al., 1994) is a method to build oblique decision trees (*multivariate* trees). Unlike axis-aligned trees that split nodes based on a single feature, OC1 creates splits using linear combinations of multiple features (i.e. oblique hyperplanes). It identifies these hyperplanes by first finding the best axis-parallel split and then employing a randomized hill-climbing algorithm. This algorithm iteratively perturbs the hyperplane's coefficients to improve the Gini criterion, and we consider 50 random restarts to escape local optima.

B.3.2 k-mers Features

K-mers are short, fixed-length (k) subsequences of DNA. In Section 5.1, the input DNA sequences are featurized by counting the occurrences of all possible k-mers (e.g., for $k = 2$, "AA", "AC", "AG", "AT", "CA", etc.). These counts then serve as input features to a decision tree. While k-mers can capture local sequence motifs without explicit prior biological knowledge, they constitute a form of manual feature engineering.

B.3.3 Deep Learning Baselines

In Section 5.1, we tune the hyperparameters of the CNN and the Transformer models using a grid search over a set of hyperparameters which we show in Table 8 (CNN) and Table 9 (Transformer). We optimize the validation AUPRC (with a 0.9/0.1 train/validation split). For model training, we perform early stopping with a patience of 5, and set the maximum number of epochs to 50.

Table 8: CNN grid-search hyperparameters and their ranges.

Hyperparameter	Values
Conv1 filters	{32, 64}
Conv1 kernel size	{2, 12, 16}
Conv2 filters	{64, 128}
Dense units	{64, 128}
Dropout rate	{0, 0.2, 0.5}
Learning rate	$\{1 \times 10^{-3}, 1 \times 10^{-4}\}$
Batch size	{32, 128}

Table 9: Transformer grid-search hyperparameters and their ranges.

Hyperparameter	**Values**
Hidden dimension	{32, 64, 128}
Attention heads	{2, 4}
Encoder layers	{1, 2, 4}
Dropout rate	{0, 0.2, 0.5}
Learning rate	$\{1 \times 10^{-3}, 1 \times 10^{-4}\}$
Batch size	{32, 128}

C ADDITIONAL RESULTS

Table 10: **Summary of additional results.**

Section / Reference	**Summary**	**Results**
Section C.1	Example decision tree discovered by `DEFT` for Pol II pausing.	Fig. 5
Section C.2	Feature found without interpretability constraints.	—
Section C.3	Unregularized `CART` increasingly overfits as depth grows.	Fig. 6
Section C.4	F1, precision, recall, and AUPRC across datasets.	Fig. 7
Section C.5	`DEFT` vs. Nucleotide Transformer (fine tuned with LoRA) for context.	Tab. 12
Section C.6	Specialized CNN/attention baselines: `PEPMAN` (Pol II), `DeePromoter` (Promoters).	Tab. 13, Tab. 14
Section C.7	`DEFT` instantiated with an open-source LLM (`gpt-oss`).	Tab. 15,
Section C.8	Ensembling `DEFT` trees on the same split boosts Acc./AUPRC.	Tab. 17
Section C.9	Reflection mechanism analysis.	Fig. 8
Section C.10	Evaluation of the interpretability of the features	
Section C.11	Comparison with two foundation models	Tab. 18

C.1 Example of Tree Discovered by `DEFT`

We show an example of a tree discovered by `DEFT` for the Pol II pausing task in Figure 5.

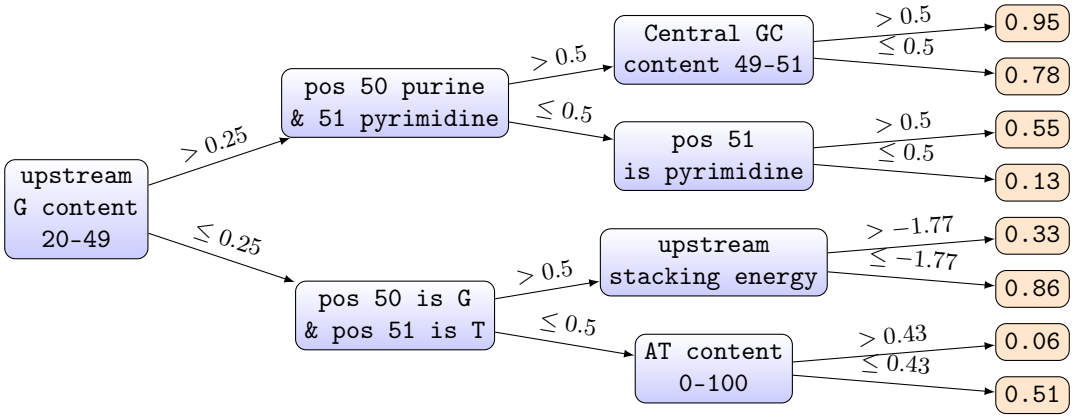


Figure 5: **Decision tree constructed by DEFT.** DEFT discovers high-level sequence features. We also report the leaves' predictions. Values along the edges correspond to the threshold at each split.

C.2 Example of Feature Without Interpretability Constraints

We provide an example of a feature found by DEFT for the Pol II pausing task when we remove the interpretability constraints from the reflection prompts (see Section 5.3 for more details).

This feature illustrates how relaxing interpretability constraints leads to the discovery of composite sequence patterns: it combines position-specific G densities upstream with broader GCT content downstream, achieving higher discriminative power through a more sophisticated feature compared to the ones shown in Figure 5. While the feature is more complex, it is worth noting that the semantic and code representations obtained with DEFT still provide transparency into the feature computation, facilitating both analysis and potential simplification by domain experts.

Feature name: `upstream and segmented downstream GCT density`

Feature description: "Calculate the density of G nucleotides in six upstream segments (positions 25-28, 29-33, 34-38, 39-43, 44-47, and 48-49) by dividing the count of G nucleotides by 4, 5, 5, 5, 4, and 2 respectively. At the pause site (position 50), consider the presence of a G nucleotide as 1. Additionally, calculate the density of G, C, and T nucleotides in three downstream segments (positions 51-63, 64-75, and 76-100) by dividing the combined count of G, C, and T nucleotides by 13, 12, and 25 respectively. Combine these densities to get a comprehensive measure."

C.3 CART Without Complexity Penalty

Methodology. In contrast to Section 5.1, where we set the minimum number of samples per leaf to 1% of the training set size for CART to prevent overfitting, we now evaluate an unregularized baseline $\texttt{CART}_{\texttt{no reg}}$ with zero minimum samples per leaf.

Results. As shown in Figure 6 for the Pol II pausing task, while $\texttt{CART}_{\texttt{no reg}}$ achieves increasingly higher training accuracy at greater tree depths, this comes at the cost of deteriorating test accuracy, an indication of overfitting.

C.4 Other Classification Metrics

We report in Figure 7 the *F1 score*, *precision*, *recall* and *AUPRC* for DEFT and the tree-based baselines. The results corroborate our findings from Section 5.1, demonstrating that DEFT identifies sequence features that are highly predictive both in and out of sample.

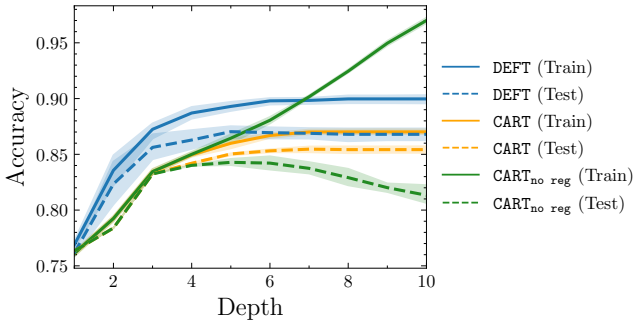
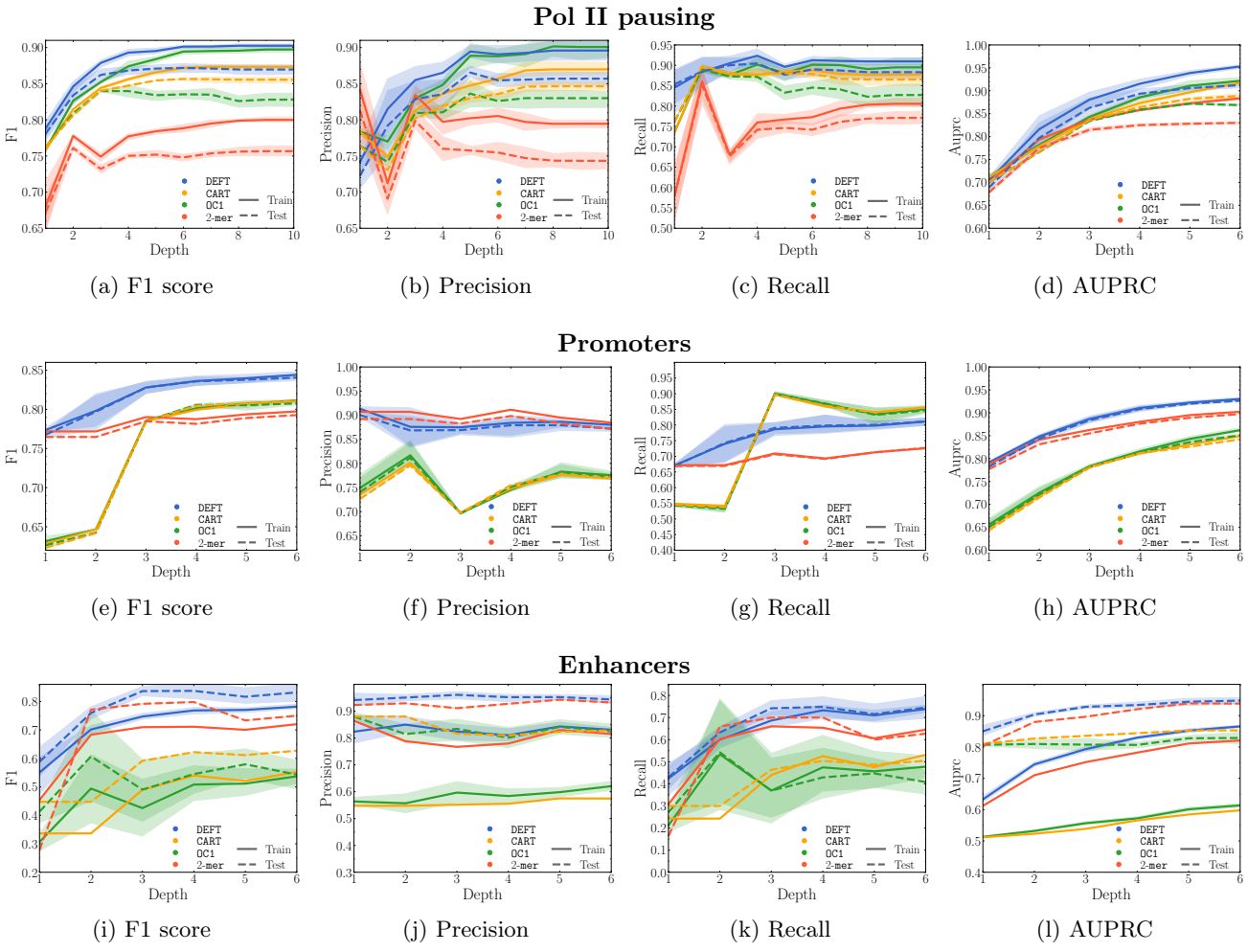


Figure 6: **CART trees can overfit.** Deep trees constructed with `CART` overfit the training set when there is no explicit regularization mechanism. We report the mean and confidence intervals at the 95% level for 5 seeds.



(a) F1 score (b) Precision (c) Recall (d) AUPRC

(e) F1 score (f) Precision (g) Recall (h) AUPRC

(i) F1 score (j) Precision (k) Recall (l) AUPRC

Figure 7: **Performance comparison across datasets (rows) and metrics (columns).** Each plot shows mean with 95% CIs over 5 seeds.

C.5 Comparison with the Nucleotide Transformer

While the primary focus of this work is on developing *transparent* models for DNA sequence analysis, we provide a comparison with a foundation model, the Nucleotide Transformer (Dalla-Torre et al., 2025), in order to contextualize DEFT 's performance. It is important to note that large foundation models like the Nucleotide Transformer, while often achieving state-of-the-art predictive accuracy, operate as *black boxes*, which contradicts the objective of this paper, i.e. designing models which are inherently transparent.

Methodology. For this comparison, we use the pre-trained `nucleotide-transformer-500m-human-ref` model (Face, 2023), which has approximately 500 million parameters. To adapt it for our classification tasks, we employ Low-Rank Adaptation (LoRA) (Hu et al., 2022) for fine-tuning, and we target the query and value matrices within its attention layers for LoRA adaptation. Key hyperparameters for fine-tuning are detailed in Table 11. The number of training steps is set to the length of the training set divided by the batch size, multiplied by the number of epochs (2), and we take the model which achieves the best validation AUPRC. The performance was then evaluated on the corresponding test sets.

Table 11: Nucleotide Transformer fine-tuning hyperparameters.

Hyperparameter	Value
LoRA rank (r)	1
LoRA alpha (α)	32
LoRA dropout	0.1
Optimizer	AdamW
Learning rate	5×10^{-4}
Train batch size	8

Results. The results, alongside DEFT and CART for reference, are presented in Table 12.

Table 12: **Performance comparison with the Nucleotide Transformer.**

	Pol II Pausing		Promoters		Enhancers	
Method	Acc.	AUPRC	Acc.	AUPRC	Acc.	AUPRC
CART (2-mers)	0.750 (0.007)	0.830 (0.003)	0.793 (0.000)	0.897 (0.000)	0.674 (0.000)	0.939 (0.000)
DEFT	0.869 (0.007)	0.912 (0.006)	0.833 (0.004)	0.926 (0.004)	0.767 (0.049)	0.948 (0.012)
Nucleotide Transformer	0.885 (0.012)	0.948 (0.009)	0.872 (0.007)	0.960 (0.004)	0.771 (0.035)	0.919 (0.040)

As anticipated, the Nucleotide Transformer, leveraging its extensive pre-training on large genomic datasets, achieves strong predictive performance. However, DEFT can significantly bridge the performance gap between CART and this large-scale deep learning approach. Crucially, DEFT achieves this improved accuracy while retaining the inherent transparency of decision trees and providing insights through its dynamically generated, high-level features. This suggests that DEFT offers a valuable trade-off, enhancing predictive power without sacrificing the interpretability essential for scientific discovery and hypothesis generation in DNA sequence analysis.

C.6 Dataset-Specific Baselines

Specialized models for each of the prediction tasks are typically variants of CNNs or attention-based networks (models that we used as baselines in our experiments in Section 5). In this section, we consider the specialized baselines PEPMAN (Feng et al., 2021) for the *Pol II pausing* task, and DeePromoter (Oubounyt et al., 2019) for the *Promoters* task. Note that these methods are not competitors to DEFT since they are not interpretable by design, but we report their performance to contextualize DEFT. We use the authors' implementations available on Github (https://github.com/fpy94/PEPMAN and https://github.com/egochao/DeePromoter) and we provide the results in the following tables:

Table 13: Pol II pausing task

Model	Accuracy	AUPRC
`CART` (2-mers)	0.750 (0.007)	0.830 (0.003)
`DEFT`	0.869 (0.007)	0.912 (0.006)
`PEPMAN`	0.883 (0.008)	0.944 (0.007)

Table 14: Promoter identification task

Model	Accuracy	AUPRC
`CART` (2-mers)	0.793 (0.000)	0.897 (0.000)
`DEFT`	0.833 (0.004)	0.926 (0.004)
`DeePromoter`	0.802 (0.007)	0.914 (0.005)

As we can see, `DEFT` is competitive with respect to these deep-learning baselines, but it differs from these methods by being inherently *interpretable*.

C.7 Instantiating `DEFT` with another LLM

`DEFT` is a modular framework, and it is not tied to a single LLM. While we used `gpt-4o` in Section 5 for its strong performance, `DEFT` can be instantiated with any LLM, including stable, open-source alternatives. To demonstrate this, we reran our experiments using the open-source model `gpt-oss`. We report the test accuracy and AUPRC in the following table.

Table 15: **Performance comparison with an open-source LLM.**

	Pol II Pausing		Promoters		Enhancers	
Method	Acc.	AUPRC	Acc.	AUPRC	Acc.	AUPRC
`CART` (2-mers)	0.750 (0.007)	0.830 (0.003)	0.793 (0.000)	0.897 (0.000)	0.674 (0.000)	0.939 (0.000)
`DEFT` (`gpt-4o`)	0.869 (0.007)	0.912 (0.006)	0.833 (0.004)	0.926 (0.004)	0.767 (0.049)	0.948 (0.012)
`DEFT` (`gpt-oss`)	0.864 (0.009)	0.908 (0.006)	0.829 (0.001)	0.921 (0.002)	0.724 (0.031)	0.938 (0.010)

Table 16: LLM parameters

Parameter	**Value**
Model name	`gpt-oss 120b`
Temperature	1
Top p	0.95
Reasoning effort	Medium

As shown, `DEFT` with `gpt-oss` still significantly outperforms the `CART` baseline and remains competitive with `DEFT` instantiated with `gpt-4o`, demonstrating the framework's robustness and adaptability.

C.8 Ensembling Trees Found by `DEFT`

The randomness in the LLM's sampling process is advantageous for two reasons. It is a crucial component of the reflection mechanism, as it enables an efficient exploration across the infinite search space of possible features (cf. Section C.9). Furthermore, we can take advantage of this stochasticity to *boost predictive performance* through ensembling.

We show this by ensembling the predictions of three trees found by `DEFT` on the *same split* for each dataset, and report the results in Table 17.

Table 17: **Ensembling trees found by DEFT improves performance.** Results reported for a given split per dataset.

	Pol II Pausing		Promoters		Enhancers	
Method	Acc.	AUPRC	Acc.	AUPRC	Acc.	AUPRC
DEFT	0.870	0.917	0.830	0.926	0.713	0.943
DEFT (Ensemble)	0.884	0.937	0.848	0.942	0.809	0.969

We note though that this performance gain comes at the cost of the interpretability of a single tree.

C.9 Analysis of the Reflection Mechanism

Methodology. We examine the reflection mechanism's effectiveness in exploring the complex search space of potential feature maps. For each node, we compute the mean of the scores of the features generated at each reflection iteration. We then normalize these scores across nodes to compute an averaged normalized score for each reflection iteration.

Results. Figure 8 presents the normalized scores across reflection iterations for the Pol II pausing task, demonstrating that the reflection mechanism is essential for refining the initial candidate population. This aligns with our findings from Section 5.4, where $\texttt{DEFT}_{\texttt{no ref}}$ exhibits lower performance compared to DEFT. This analysis also confirms that DEFT does not rely on LLM memorization to achieve good performance. Indeed, if this was the case, DEFT would be able to achieve optimal performance without the reflection mechanism, which is invalidated by Section 5.4.

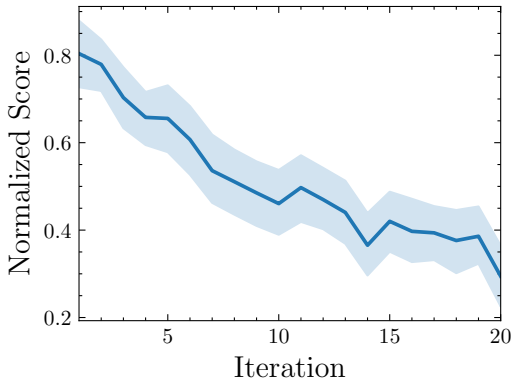


Figure 8: The reflection mechanism effectively refines the features.

C.10 Expert Evaluation

Setup. We extracted the top 20 features discovered by DEFT for the Pol II pausing task and asked a biologist specializing in transcriptional regulation to score each feature on a 1–5 scale along two axes:

1. **Faithfulness of rationale to feature and task**: How accurately does the rationale describe what the feature's code computes and why that computation is relevant for Pol II pausing?
2. **Biological interpretability**: To what extent does the proposed mechanism correspond to a single, coherent biological phenomenon and appear biologically plausible?

Results. The expert judged that the rationales almost always correctly described the executable feature and articulated a task-relevant mechanism, yielding a score of 4.89 ± 0.46 out of 5 on faithfulness. The expert also found that DEFT largely proposed features plausibly aligned with determinants of pausing and transcriptional mechanics (e.g. GC-rich regions associated with pause stability or nucleosome positioning), with a score of 4.58 ± 0.67 out of 5, confirming the trustworthiness of DEFT.

C.11 Comparison with Foundation Models

We empirically compare DEFT with DNABERT-2 (Zhou et al., 2024) and HyenaDNA (Nguyen et al., 2023) and report the results in Table 18.

Table 18: **Comparison with DNABert-2 and HyenaDNA.**

	Pol II Pausing		Promoters		Enhancers	
Method	Acc.	AUPRC	Acc.	AUPRC	Acc.	AUPRC
DNABERT-2	0.795 (0.006)	0.881 (0.005)	0.937 (0.007)	0.967 (0.007)	0.855 (0.025)	0.979 (0.003)
HyenaDNA	0.777 (0.018)	0.868 (0.005)	0.909 (0.009)	0.932 (0.008)	0.808 (0.045)	0.965 (0.009)
CART	0.750 (0.007)	0.830 (0.003)	0.793 (0.000)	0.897 (0.000)	0.674 (0.000)	0.939 (0.000)
DEFT	0.869 (0.007)	0.912 (0.006)	0.833 (0.004)	0.926 (0.004)	0.767 (0.049)	0.948 (0.012)

On Pol II pausing, DEFT outperforms both DNABERT-2 and HyenaDNA while remaining fully interpretable. On Promoters and Enhancers, DNABERT-2 and HyenaDNA achieve higher absolute scores, as expected for large pre-trained models, but DEFT substantially closes the gap between CART and these black boxes.

D Broader Impacts

This work introduces DEFT, an interpretable framework for DNA sequence analysis. Its positive impacts include accelerating biological research through the discovery of human-understandable sequence features, leading to greater trust in AI models applied to life sciences. While the use of LLMs involves considerations regarding computational resources, the generation of human-interpretable features inherently facilitates expert review and validation. This opens the door for a collaborative human-AI approach, helping responsibly navigating the analysis of diverse genomic data.