Decision Tree Induction with Dynamic Feature Generation: A Framework for Interpretable DNA Sequence Analysis

Nicolas Huynh *[1], **Krzysztof Kacprzyk**[1], **Ryan Sheridan**[2], **David Bentley**[2], **Mihaela van der Schaar**[1]
[1]University of Cambridge, [2]University of Colorado Anschutz Medical Campus

Abstract

The analysis of DNA sequences has become increasingly critical in numerous fields, from evolutionary biology to understanding gene regulation and disease mechanisms. While machine learning approaches to DNA sequence classification, particularly deep neural networks, achieve remarkable performance, they typically operate as black boxes, severely limiting their utility for scientific discovery and biological insight. 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. Through a comprehensive case study on RNA polymerase II pausing prediction, we demonstrate that `DEFT` discovers human-interpretable sequence features which are highly predictive of pausing, providing insights into this complex phenomenon.

1 Introduction

DNA sequences represent the fundamental code of life, storing the genetic instructions essential for the development and functioning of all organisms. The analysis of DNA sequences has become increasingly critical in numerous fields, including medical diagnostics, evolutionary and molecular biology, and personalized medicine. 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; Umarov et al., 2019) and splice sites (Scalzitti et al., 2021; Albaradei et al., 2020) to predicting gene expression and regulatory motifs (Avsec et al., 2021a).

Despite their predictive performance, many machine learning approaches to DNA sequence analysis - particularly deep learning models - suffer from a critical limitation: their lack of interpretability. This opacity poses significant challenges in biological research and clinical applications, where understanding the predictions is crucial for validation against known biological principles, discovery of novel biological mechanisms, and experimental design guidance.

In contrast to black boxes, decision trees offer inherent interpretability for supervised learning tasks (Breiman et al., 1984; Quinlan, 1986; 2014). These models recursively partition the input space through a series of binary decisions, based on the comparison between a specific feature and a learned threshold at every internal node. Their hierarchical structure provides transparency, as each prediction can be traced by a path from root to leaf. However, while decision trees are commonly used in domains such as finance or healthcare (Soleimanian et al., 2012), using them for the analysis of DNA sequences comes with several challenges. When operating on raw nucleotide features, trees must

*Correspondence: `nvth2@cam.ac.uk`

grow deep to capture complex interactions between multiple sequence positions, as each split can only consider a single position, thereby compromising the interpretability and generalization performance of the resulting tree. Although manually crafting high-level variables offers an alternative, this approach remains limited by existing biological knowledge and ignores local data characteristics at each node during tree construction, which should guide feature generation.

In this work, we address these limitations and propose `DEFT` (Dynamic Engineering of Features in Trees), an *interpretable* and *expressive* tree-based model for DNA sequence classification. `DEFT` operates with a top-down tree induction process that progressively grows the tree starting from a root node. At each internal node, `DEFT` automatically discovers high-level sequence features that lead to discriminative splits. It is *adaptive*, since feature generation is informed by the local data characteristics at each leaf of the tree during induction. Furthermore, it incorporates *domain knowledge*, thereby improving search efficiency by favoring features that are biological 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 prior 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. We also employ 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 put `DEFT` in action through a case study on RNA polymerase II (Pol II) pausing, a mechanism related to critical regulatory processes including co-transcriptional splicing (De La Mata et al., 2003; Alexander et al., 2010), and transcription termination (Gromak et al., 2006), but whose underlying mechanisms remain incompletely understood. In particular, we demonstrate that `DEFT` discovers biologically meaningful features which accurately characterize Pol II pausing patterns.

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,** we conduct a case study on Pol II pausing, where `DEFT` reveals *interpretable* sequence patterns which are highly *predictive* of this regulatory process.

2 Background

In this section, we describe decision trees for classification and examine their limitations in the context of DNA sequence analysis.

2.1 Top-down tree induction for classification

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 value 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). Formally, 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:

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

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 split (i^*, τ^*) for $\mathcal{D}$ by solving:

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

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).

2.2 Limitations of decision trees for DNA sequence analysis

Decision trees are valuable in many domains given their inherent transparency (Rudin, 2019). Indeed, their predictions can be simply traced through a sequence of splitting conditions starting from the root node down to the leaves. This transparency makes them appealing for DNA sequence analysis, where understanding the predictions of a model is crucial for validating predictions against biological principles, discovering novel biological mechanisms, and guiding experimental design. However, leveraging traditional 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 GC content is higher than a given threshold. The GC content is defined as the percentage of nucleotides in the sequence that are either G (guanine) or C (cytosine), hence it requires considering multiple sequence positions simultaneously. As we show in Figure 1, conventional decision trees struggle with this seemingly simple task. While they can achieve high training accuracy by growing deep enough to memorize specific position-by-position patterns, their 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.

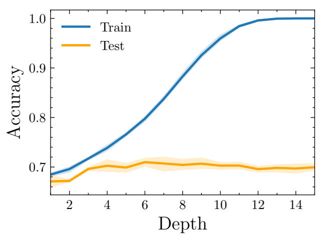
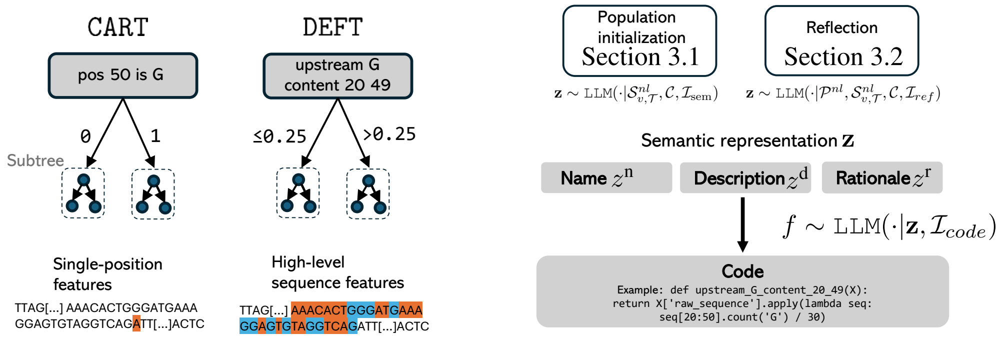


Figure 1: **Limitations of conventional trees.** Training and test accuracies versus maximum tree depth for GC content classification.

Constraints of manual feature engineering. A potential solution to the aforementionned 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


CART
pos 50 is G
0
1
Subtree
Single-position features
TTAG[...] AAACACTGGGATGAAA
GGAGTGTAGGTCAGATT[...]ACTC
DEFT
upstream G content 20 49
≤0.25
>0.25
High-level sequence features
TTAG[...] AAACACTGGGATGAAA
GGAGTGTAGGTCAGATT[...]ACTC
Population initialization Section 3.1
$\mathbf{z} \sim \texttt{LLM}(\cdot|\mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{\text{sem}})$
Reflection Section 3.2
$\mathbf{z} \sim \texttt{LLM}(\cdot|\mathcal{P}^{nl}, \mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{ref})$
Semantic representation $\mathbf{z}$
Name z^{n}
Description z^{d}
Rationale z^{r}
$f \sim \texttt{LLM}(\cdot|\mathbf{z}, \mathcal{I}_{code})$
Code

```
Example: def upstream_G_content_20_49(X):
return X['raw_sequence'].apply(lambda seq:
seq[20:50].count('G') / 30)
```

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, constrasting 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.

undiscovered sequence patterns—a significant drawback when studying problems whose underlying biological mechanisms are not yet understood. Second, engineered features 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.

3 Method

In this section, we introduce DEFT, an interpretable tree-based model for DNA sequence analysis that addresses the limitations of conventional decision trees detailed in Section 2. Our approach combines the *transparency* of tree structures while ensuring *expressivity* through automatically generated sequence features which yield discriminative splits during tree construction. This makes DEFT's predictions readily interpretable through decision paths involving semantically meaningful features.

Adaptive feature generation with LLMs. Naively searching over the space $\mathbb{R}^{\mathcal{X}}$ of possible feature maps at every leaf node during tree induction is 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. Indeed, large language models pretrained on Internet-scale data, such as GPT4 (Achiam et al., 2023), Claude (Anthropic, 2024), and PALM2 (Anil et al., 2023), have demonstrated marked proficiencies in many tasks including natural language (Brown et al., 2020), code programs (Chen et al., 2021) and optimization (Yang et al., 2024). Furthermore, they are powerful hypothesis generators (Wang et al., 2023), making them particularly suitable for exploring the vast 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-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.

3.1 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 $\mathbf{z}_j = (z_j^{\mathrm{n}}, z_j^{\mathrm{d}}, z_j^{\mathrm{r}})$ provides a human-interpretable specification of a candidate feature: z_j^{n} provides a concise name (e.g., *"GC content"*), z_j^{d} details its computation (e.g., *"percentage of G and C nucleotides in the first 20 positions"*), and z_j^{r} explains its biological relevance (e.g., *"GC-rich regions often indicate regulatory elements"*). 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 following 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 $\mathbf{z}_j \sim \texttt{LLM}(\cdot|\mathcal{S}_{v,\mathcal{T}}^{nl}, \mathcal{C}, \mathcal{I}_{\mathrm{sem}})$, where $\mathcal{I}_{\mathrm{sem}}$ 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.

3.2 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 Appendix C.1. For each instruction $\mathcal{I}_{ref} \in \mathcal{I}$, we generate a set of M semantic representations as $\mathbf{z}'_m \sim \texttt{LLM}(\cdot|\mathcal{P}^{nl}, \mathcal{S}_{v,\mathcal{T}}^{nl}, \mathcal{C}, \mathcal{I}_{ref})$. 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 achieving the minimum score in the final population as the splitting feature for the current node v.

We summarize the different steps for feature generation in Algorithm 1. This routine is performed at every leaf node of the partially constructed tree, until a stopping criterion is met (e.g. maximum depth reached). We also provide more details on the prompts in Appendix B.2.

4 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) provide interpretable results but often lack expressivity. More recently, deep learning architectures, particularly convolutional and transformer networks, have demonstrated superior 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.

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; 1993). These conventional methods are restricted to splits based on raw features, providing axis-aligned trees with limited expressivity. While oblique decision trees (Murthy et al., 1994) enhance expressivity by considering linear combinations of features at each node, they still explore a restricted feature space, are 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.; Hollmann et al., 2024; Nam et al., 2024). However, (Han et al.; 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 the tree induction process, making it adaptive to the local characteristics of nodes.

5 A Case Study on RNA Polymerase II Pausing

We put `DEFT` in action with a case study on the classification of RNA polymerase II pausing using DNA sequences. The rest of this section is structured as follows. In Section 5.1, we analyze the features discovered by `DEFT`, revealing its ability to capture biologically meaningful concepts related to Pol II. In Section 5.2, we demonstrate that these features accurately discriminate pausing sites from non-pausing sites. In Section 5.3, we perform ablation studies that provide insight into the key mechanisms contributing to `DEFT` performance. We provide additional results in Appendix C.

RNA polymerase II pausing. We base our case study on the task of predicting RNA polymerase II pausing (Mayer et al., 2017). RNA polymerase II (Pol II) is the essential nuclear enzyme that catalyzes the transcription of protein-coding genes into messenger RNA in eukaryotic cells. During transcription elongation, Pol II temporarily interrupts its forward movement along the DNA template at specific positions, slowing down the rate of RNA synthesis. This mechanism is related to critical regulatory processes including co-transcriptional splicing (De La Mata et al., 2003), nascent RNA folding (Hein et al., 2014), transcription termination (Gromak et al., 2006), and recruitment of regulatory factors (Core & Adelman, 2019). Some approaches have revealed sequence motifs associated with pausing (Mayer et al., 2017; Fong et al., 2022), but the complete ”grammar” of pause is not fully characterized. This makes Pol II pausing an ideal case study for `DEFT`, as it represents a complex problem where interpretable machine learning could provide new biological insights.

Dataset. We analyze enhanced Native Elongating Transcript sequencing (eNET-seq) data from HCT116 human cells, which maps Pol II transcription at single-base resolution. Transcriptional pause sites are identified where the signal exceeds 3 standard deviations above the local mean and contains at least 5 reads. From the genome-wide data, we construct a balanced dataset of 6,000 DNA sequences, randomly splitting them into a training set of 4,000 sequences and a held-out test set of 2,000 sequences. For each DNA sequence, we define a binary label based on the presence of a pause site at the center position of the sequence (position 50). We provide detailed information about data processing and characteristics in Appendix B.1.

Experimental details. In what follows, we use `gpt-4o` version `1001` as the underlying LLM. We use a population size $M = 10$ and perform $K = 20$ 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

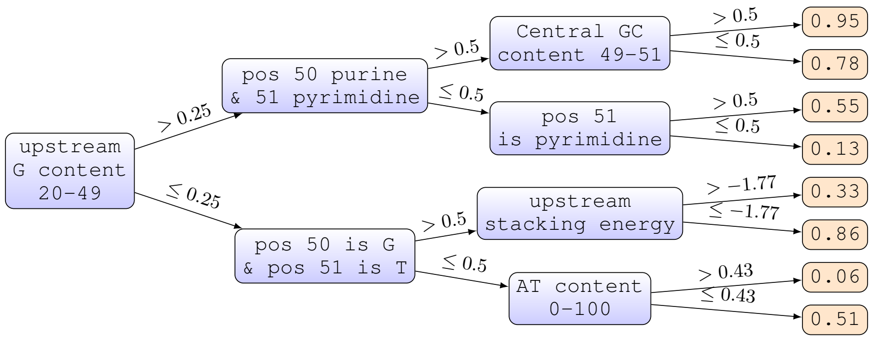


Figure 3: **Decision tree constructed by `DEFT`.** `DEFT` discovers high-level sequence features. We also report the leaves' predictions.

1% of the size of the training set to prevent overfitting. Appendix B.2 provides more details on the hyperparameters and experimental setting.

5.1 `DEFT` DISCOVERS BIOLOGICALLY MEANINGFUL FEATURES

We begin the case study by analysing the features discovered by `DEFT` during tree induction.

Analysis. Figure 3 shows a tree of depth 3 constructed by `DEFT`. Each node displays a feature name, which is part of the semantic representations, with corresponding threshold values labeled at the edges. We first observe that `DEFT` discovers high-level features based on the DNA sequences which integrate information across multiple nucleotide positions simultaneously, going beyond single-nucleotide information. For example, the feature at the root node computes the *G* content on a specific window of the sequences. Its semantic representation is the following:

Feature name: `upstream G content 20-49`

Feature description: "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."

Rationale: "Based on the top-performing features, it is evident that GC content, particularly the presence of guanine (G) nucleotides in specific regions, plays a significant role in RNA polymerase pausing. Notably, the features `upstream G content`, `upstream G content 30-49`, and `G count upstream 10bp` focus on G content in the upstream region and have shown high relevance. Given this, I propose a feature that captures the proportion of guanine nucleotides in a slightly broader and different upstream window (positions 20 to 49)."

The biological significance of this feature in the upstream region likely relates to guanine's chemical properties and its potential to form alternative DNA structures like G-quadruplexes (Lipps & Rhodes, 2009) when present in high density. The specific positioning of this window (1-31 bases upstream of potential pause sites) suggests that G-rich sequences in this region may influence Pol II dynamics, possibly through local structural changes in the DNA template or by creating binding sites for regulatory factors that affect pausing. This is in agreement with (Turowski et al., 2020) that found that high GC content in the transcription bubble correlates with slow Pol II elongation.

Furthermore, a key advantage of the decision tree structure in `DEFT` is its ability to progressively refine sequence classification as we go deeper into the tree. For instance, while upstream G content serves as a discriminative feature for the entire dataset, `DEFT` reveals a distinct subset of sequences with high upstream G-content, characterized by a purine at position 51 (1 base pair downstream of the site), and which consists almost exclusively of *non-pause sites*.

Take-away 1. `DEFT` discovers biologically meaningful features which can capture patterns across multiple positions.

5.2 DEFT FINDS TREES WHICH FAITHFULLY CHARACTERIZE POL II PAUSING

Methodology. Having demonstrated DEFT's ability to discover biologically meaningful DNA sequence features during tree induction, we now verify if they faithfully characterize Pol II pauses. To do so, we evaluate the predictive performance of trees found by DEFT, comparing it against trees which use the raw sequence positions as features (CART). For this baseline, we use one-hot encoded nucleotides at each position as features. While nucleotides could alternatively be treated as ordinal variables, this approach empirically led to inferior results compared to one-hot encoding. We assess both DEFT and CART across varying tree depths $d \leq 10$, with CART also using a minimum number of samples per leaf equal to 1% of the size of the training set to mitigate overfitting (results without this regularization can be found in Appendix C.2).

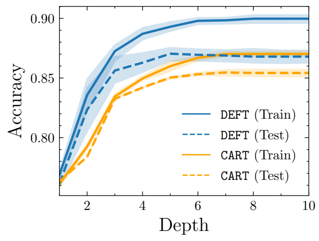


Figure 4: **Performance comparison.** Training and test accuracies across varying depths for DEFT and CART. DEFT achieves superior predictive performance by generating highly discriminative features. We report the mean and confidence intervals at the 95% level for 5 seeds.

Results. We compare in Figure 4 the training and test accuracies of DEFT and CART across varying tree depths for 5 different seeds. DEFT consistently achieves superior performance on both training and test sets for the different tree depths. The predictive power of DEFT demonstrates that its ability to discover high-level sequence features leads to more effective discrimination between pause and non-pause sites than position-specific nucleotide splits alone. We report additional performance metrics (F1 score, precision, and recall) in Appendix C.3, which further support these findings.

Take-away 2. DEFT's ability to discover and leverage high-level DNA sequence features leads to more accurate Pol II pause site prediction compared to traditional approaches restricted to single-nucleotide splits.

5.3 ABLATIONS

Methodology. We conduct ablation experiments by removing the following key components: ▶ **Prior knowledge**: $\text{DEFT}_{\text{no prior}}$ 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**: $\text{DEFT}_{\text{no adapt}}$ 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.** $\text{DEFT}_{\text{no ref}}$ removes the reflection mechanism, relying solely on the LLM for population initialization.

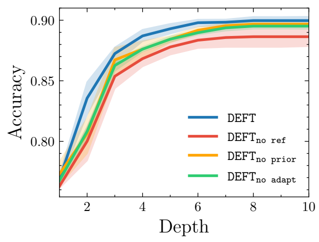


Figure 5: **Ablation study.** We report the training accuracies for the different ablations across varying tree depths. We report the mean and confidence intervals at the 95% level for 5 seeds.

Results. We report the training accuracy for each of these ablations in Figure 5, observing that these three components are necessary to achieve optimal search efficiency. Notably, the reflection mechanism (which leverages the LLM's in-context learning capabilities) plays a crucial role in navigating the complex search space of possible feature maps, also explaining the competitive performance of $\text{DEFT}_{\text{no prior}}$. We provide detailed analysis of the reflection mechanism's performance improvements in Appendix C.4.

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 a comprehensive case study on Pol II pausing classification, we demonstrate that `DEFT` discovers biologically meaningful features that accurately characterize the pauses. Future work may investigate the performance of `DEFT` on datasets with higher dimensionality or other domains with structured modalities (e.g. amino acids). Finally, `DEFT` could be extended to a human-in-the-loop setting, where the discovery of features by LLMs would be guided by domain experts.

REFERENCES

Gene expression omnibus. URL `https://www.ncbi.nlm.nih.gov/geo/`.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3146–3153, 2020.

Somayah Albaradei, Arturo Magana-Mora, Maha Thafar, Mahmut Uludag, Vladimir B Bajic, Takashi Gojobori, Magbubah Essack, and Boris R Jankovic. Splice2deep: An ensemble of deep convolutional neural networks for improved splice site prediction in genomic dna. *Gene*, 763:100035, 2020.

Ross D Alexander, Steven A Innocente, J David Barrass, and Jean D Beggs. Splicing-dependent rna polymerase pausing in yeast. *Molecular cell*, 40(4):582–593, 2010.

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

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

AI Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 2024.

Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18 (10):1196–1203, 2021a.

Žiga Avsec, Melanie Weilert, Avanti Shrikumar, Sabrina Krueger, Amr Alexandari, Khyati Dalal, Robin Fropf, Charles McAnany, Julien Gagneur, Anshul Kundaje, et al. Base-resolution models of transcription-factor binding reveal soft motif syntax. *Nature genetics*, 53(3):354–366, 2021b.

Leo Breiman, Jerome Friedman, Charles J Stone, and RA Olshen. *Classification and Regression Trees*. CRC Press, 1984.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Alexander EI Brownlee, James Callan, Karine Even-Mendoza, Alina Geiger, Carol Hanna, Justyna Petke, Federica Sarro, and Dominik Sobania. Enhancing genetic improvement mutations using large language models. In *International Symposium on Search Based Software Engineering*, pp. 153–159. Springer, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Leighton Core and Karen Adelman. Promoter-proximal pausing of rna polymerase ii: a nexus of gene regulation. *Genes & development*, 33(15-16):960–982, 2019.

Manuel De La Mata, Claudio R Alonso, Sebastián Kadener, Juan P Fededa, Matıas Blaustein, Federico Pelisch, Paula Cramer, David Bentley, and Alberto R Kornblihtt. A slow rna polymerase ii affects alternative splicing in vivo. *Molecular cell*, 12(2):525–532, 2003.

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

Natalia Gromak, Steven West, and Nick J Proudfoot. Pause sites promote transcriptional termination of mammalian rna polymerase ii. *Molecular and cellular biology*, 26(10):3986–3996, 2006.

Sungwon Han, Jinsung Yoon, Sercan O Arik, and Tomas Pfister. Large language models can automatically engineer features for few-shot tabular learning. In *Forty-first International Conference on Machine Learning*.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

Pyae P Hein, Kellie E Kolb, Tricia Windgassen, Michael J Bellecourt, Seth A Darst, Rachel A Mooney, and Robert Landick. Rna polymerase pausing and nascent-rna structure formation are linked through clamp-domain movement. *Nature structural & molecular biology*, 21(9):794–802, 2014.

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36, 2024.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *Advances in Neural Information Processing Systems*, 32, 2019.

Nguyen Quoc Khanh Le, Edward Kien Yee Yapp, Nagarajan Nagasundaram, and Hui-Yuan Yeh. 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, 2019.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366. Springer, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pp. 6150–6160. PMLR, 2020.

Hans J Lipps and Daniela Rhodes. G-quadruplex structures: in vivo evidence and function. *Trends in cell biology*, 19(8):414–422, 2009.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Forty-first International Conference on Machine Learning*, 2024.

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

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

Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. Optimized feature generation for tabular data via llms with decision tree reasoning. *arXiv preprint arXiv:2406.08527*, 2024.

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

J Ross Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

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

Nicolas Scalzitti, Arnaud Kress, Romain Orhand, Thomas Weber, Luc Moulinier, Anne Jeannin-Girardon, Pierre Collet, Olivier Poch, and Julie D Thompson. Spliceator: multi-species splice site prediction using convolutional neural networks. *BMC bioinformatics*, 22:1–26, 2021.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pp. 3145–3153. PMlR, 2017.

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

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

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

Tomasz W Turowski, Elisabeth Petfalski, Benjamin D Goddard, Sarah L French, Aleksandra Helwak, and David Tollervey. Nascent transcript folding plays a major role in determining rna polymerase elongation rates. *Molecular cell*, 79(3):488–503, 2020.

Ramzan Umarov, Hiroyuki Kuwahara, Yu Li, Xin Gao, and Victor Solovyev. Promoter analysis and prediction in the human genome using sequence-based deep learning models. *Bioinformatics*, 35 (16):2730–2737, 2019.

Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 1625–1632, 2019.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D Goodman. Hypothesis search: Inductive reasoning with language models. *arXiv preprint arXiv:2309.05660*, 2023.

Ruohan Wang, Zishuai Wang, Jianping Wang, and Shuaicheng Li. Splicefinder: ab initio prediction of splice sites using convolutional neural network. *BMC bioinformatics*, 20:1–13, 2019.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024. URL `https://arxiv.org/abs/2309.03409`.

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

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

A Extended Related Works

Our work intersects with several research 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 & 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) demonstrating the ability to capture long-range dependencies in genomic sequences. While these deep learning approaches achieve remarkable performance, their black-box nature makes their predictions non transparent. `DEFT` addresses this limitation by providing an interpretable framework that maintains high predictive power.

Interpreting black boxes in DNA sequence analysis. The need for interpretability in genomic applications has led to various approaches to explain machine learning models. Attribution methods like integrated gradients (Sundararajan et al., 2017) and DeepLIFT (Shrikumar et al., 2017) have been widely used to identify important nucleotides in deep learning predictions. 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, our approach builds interpretability directly into the model, 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, 1993). 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 & 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 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.; 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.; 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.

B Experimental Details

Code release. Code wil be released upon acceptance.

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 dataset.

We base the case study conducted in Section 5 on a dataset (Fong et al., 2022) publicly available on the GEO platform (GEO), with accession number `GSE202749`.

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. We extract a subset of the original dataset, randomly sampling 6000 samples from the control group (corresponding to the condition $+Spt5$). 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 1.

Table 1: 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	Pausing: 51%, Non-pausing: 49%

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, 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.2 Details on the method.

B.2.1 Algorithm

We provide the algorithm for feature generation at a node v in Algorithm 1.

B.2.2 LLM hyperparameters

We detail in Table 2 the hyperparameters of the LLM used throughout our experiments.

Table 2: 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.

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 $K$
1: $\mathcal{S}^{nl}_{v,\mathcal{T}} \leftarrow$ serialized sequence of splitting conditions from root to $v$
2: $P \leftarrow \emptyset$ {Initial population}
3: for $j = 1$ to $M$ do
4:     $\mathbf{z}_j \sim \texttt{LLM}(\cdot|\mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{sem})$ {Semantic rep.}
5:     $f_j \sim \texttt{LLM}(\cdot|\mathbf{z}_j, \mathcal{I}_{code})$ {Executable code}
6:     $P \leftarrow P \cup \{(f_j, \mathbf{z}_j)\}$
7: end for
8: $P \leftarrow P \cup \{(\phi_i, \bar{\mathbf{z}}_i)\}_{i=1}^{d}$ {Add raw features}
9: Compute scores $\eta = \min_{\tau \in \mathbb{R}} s(f, \tau, \mathcal{D})$ for all $f$ in $P$
10: for $k = 1$ to $K$ do
11:     Create few-shot prompt $\mathcal{P}^{nl}$ from $P$ and scores
12:     for $\mathcal{I}_{ref} \in \mathcal{I}$ do
13:         for $m = 1$ to $M$ do
14:             $\mathbf{z}'_m \sim \texttt{LLM}(\cdot|\mathcal{P}^{nl}, \mathcal{S}^{nl}_{v,\mathcal{T}}, \mathcal{C}, \mathcal{I}_{ref})$
15:             $f'_m \sim \texttt{LLM}(\cdot|\mathbf{z}'_m, \mathcal{I}_{code})$
16:         end for
17:         $P' \leftarrow P' \cup \{(f'_m, \mathbf{z}'_m)\}_{m=1}^{M}$
18:         Compute scores for all features in $P'$
19:     end for
20:     $P \leftarrow \texttt{Top-M}(P \cup P')$ {Selection on scores}
21: end for
22: return $(f^*, \mathbf{z}^*, \tau^*) = \arg\min_{(f,\mathbf{z}) \in P, \tau} s(f, \tau, \mathcal{D})$
```

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.

B.2.3 Other details

In our experimental section, `DEFT` uses a population size $M = 10$, and a number of reflections $K = 20$. Furthermore, `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 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 trees, DEFT incurs an additional computational overhead which 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. However, this allows DEFT to discover novel features which are highly discriminative, hence requiring fewer splits to achieve comparable performance to conventional trees. This is evidenced by DEFT's superior accuracy (as shown in Figure 4), where it achieves better performance than CART given the same depth. Hence, the LLM-guided feature discovery effectively trades increased per-node computation for a more focused exploration of the feature space. 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.

C ADDITIONAL RESULTS

C.1 DEFT ALLOWS TO CONTROL THE BALANCE BETWEEN INTERPRETABILITY AND PERFORMANCE

The results in Section 5 demonstrate that DEFT discovers biologically meaningful features that are discriminative for Pol II pausing, hence achieving both *interpretability* and *predictive* accuracy. We now illustrate how our framework enables practitioners to control the balance between these objectives according to their requirements. Specifically, we show that DEFT can find composite features that yield trees with enhanced predictive performance at small tree depths.

Methodology. To explore this, we remove the interpretability constraints from the reflection prompts in $\mathcal{I}$. This modification allows DEFT to focus solely on generating features that maximize predictive performance, regardless of complexity. We denote this configuration as $\texttt{DEFT}_{\texttt{perf}}$.

Results. We report the training and test accuracies for small tree depths in Table 3. While the gap between $\texttt{DEFT}_{\texttt{perf}}$ and DEFT reduces as d increases, we can see a strong performance difference at depth 1. To provide intuition for this observation, we show an example of a feature discovered by $\texttt{DEFT}_{\texttt{perf}}$ at the root node of the tree.

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."

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 one shown in Section 5.1. While the feature is more complex, it is worth noting that the semantic

Table 3: Performance comparison at different maximum tree depths, showing training and test accuracies for both methods.

	Train		Test	
d	DEFT	DEFT$_{\texttt{perf}}$	DEFT	DEFT$_{\texttt{perf}}$
1	$0.769_{(0.003)}$	$0.820_{(0.011)}$	$0.762_{(0.003)}$	$0.810_{(0.011)}$
2	$0.836_{(0.015)}$	$0.866_{(0.025)}$	$0.823_{(0.021)}$	$0.849_{(0.022)}$
3	$0.872_{(0.008)}$	$0.886_{(0.015)}$	$0.856_{(0.013)}$	$0.862_{(0.014)}$

and code representations obtained with DEFT still provide transparency into the feature computation, facilitating both analysis and potential simplification by domain experts.

C.2 CART without complexity penalty

Methodology. In contrast to Section 5.2, where we set the minimum number of samples to 1% of the training set size for CART to prevent overfitting, we now evaluate an unregularized baseline CART$_{\texttt{no reg}}$ with zero minimum samples per leaf.

Results. As shown in Figure 6, while 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.

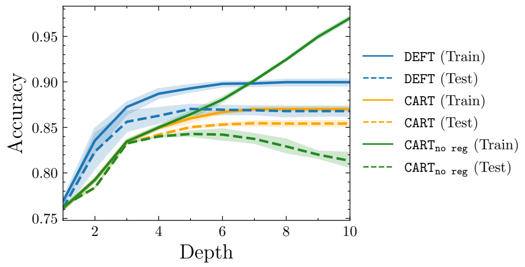
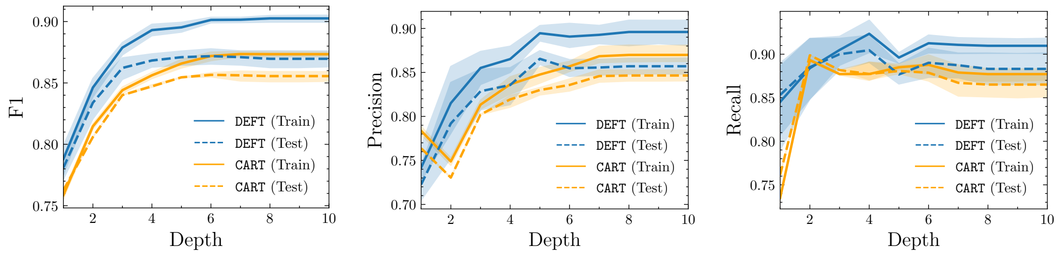


Figure 6: **CART trees can overfit.** Deep trees constructed with CART overfit the training set when there is no explicit regularization mechanism. We report the mean and confidence intervals at the 95% level for 5 seeds.

C.3 Other classification metrics

We report in Figure 7 the *F1 score*, *precision* and *recall* for both DEFT and CART. The results corroborate our findings from Section 5.2, demonstrating that DEFT identifies sequence features that are more predictive of pausing behavior both in and out of sample, thus describing Pol II pausing more faithfully.



(a) F1 Score (b) Precision (c) Recall

Figure 7: **Performance comparison.** We report F1 score, precision and recall. We report the mean and confidence intervals at the 95% level for 5 seeds.

C.4 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, demonstrating that the reflection mechanism is essential for refining the initial candidate population. This aligns with our findings from Section 5.3, where `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 5.3.

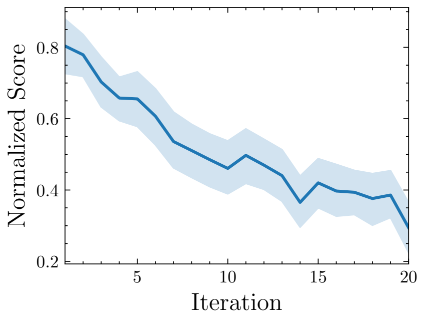


Figure 8: The reflection mechanism effectively refines the features.