Logical Distillation of Transformer Encoders

Matteo Forasassi*
TU Wien, Vienna, Austria
matteo.forasassi@tuwien.ac.at

Thomas Gärtner
TU Wien, Vienna, Austria
thomas.gaertner@tuwien.ac.at

Thomas Lukasiewicz
TU Wien, Vienna, Austria
University of Oxford, UK.
thomas.lukasiewicz@tuwien.ac.at

Sagar Malhotra*
TU Wien, Vienna, Austria
sagar.malhotra@tuwien.ac.at

Abstract

We present an algorithm that distills transformer encoders into logical formulas. Interpretable surrogates for transformers are important for safety and scientific knowledge discovery. Recent work extracts sparse latent features from transformer activations, but these features alone do not express how information is composed across a sequence. Building on theoretical connections between transformer expressivity and linear temporal logic (LTL), we introduce *iterated temporal decision trees* (ITDTs), a model that can express any LTL formula as a sequence of decision trees. Each decision tree builds on the previous ones, producing LTL formulas of increasing temporal depth. Our algorithm distills a transformer encoder into an ITDT. Empirically, our algorithm recovers succinct and accurate formulas on established LTL learning benchmarks and distills interpretable rules for multiple protein sequence datasets from ESM-2-T6, an established protein language model.

1 Introduction

We present a methodology for distilling a *transformer encoder* model [1] into interpretable logical formulas. Interpreting transformer models learned from real-world data is a fundamental objective for their safe deployment [2] and holds significant potential for scientific knowledge discovery [3]. However, transformer models are inherently black-box; their computed representations are neither human nor machine-readable [4]. Methods such as sparse autoencoders can reveal sparse, interpretable features, but they do not provide a systematic framework for expressing compositional rules that capture how features interact across context [5–7]. Recent work has identified formal languages that transformers can express [8–10]. We draw insights from these results to devise a logic based model, and an algorithm to distill transformers into logical formulas.

Theoretical analyses of transformer expressivity [8, 9] show that transformer encoders can express *linear temporal logic* (LTL) formulas [11]. For a given LTL formula with temporal depth k, these works build a transformer encoder with k layers, where the i^{th} transformer layer allows expressing subformulas with temporal depth up to i. Using these insights, we introduce a decision tree based model called *iterated temporal decision tree (ITDT)*, and an algorithm for distilling it from transformers. An ITDT is a stack of decision trees whose decision variables are temporal formulas. Like the transformer constructions in Barcelo et al. [8], each ITDT layer adds one unit of temporal depth. This shared layerwise structure lets us distill ITDTs from transformers: each transformer layer supervises one ITDT tree, and each later tree uses temporally enriched features from the previous tree.

*Corresponding authors. Code available at https://github.com/MatteoForasassi/LogDist-Transformers

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

We test ITDTs on an established benchmark for LTL specification learning [12–14] and multiple real-world protein sequence datasets [15–18]. Our results show that when the ground-truth formula is in LTL, then the distillation procedure is able to learn an ITDT with almost no loss in accuracy. For protein sequence datasets, we distill a protein language model, ESM-2-T6 [19], often with minimal loss in accuracy. Our ablations show that both layer-wise transformer supervision and temporal features are advantageous for ITDT learning. Finally, an LLM-as-expert analysis further shows that the learned rules are interpretable and broadly consistent with biochemical priors.

2 Related Work

Our work primarily builds on formal analysis of transformer expressivity and is relevant to research in interpretability of neural networks.

Transformer expressivity. Multiple works have analyzed the expressivity of transformer encoders and their connection to formal languages [8–10, 20–27]. The inductive bias of ITDTs and our distillation algorithm is guided by proof ideas from Barcelo et al. [8] and Yang et al. [9]. Together, these works connect transformer encoders to linear temporal logic (LTL) [11] and its counting extension, LTL(C,+). They construct families of hard-attention transformer encoders that recognize exactly the relevant LTL-definable languages and establish a tight correspondence between transformer layers and temporal depth. This layer-wise correspondence directly motivates the layered construction of ITDTs. These results mainly concern hard-attention transformers, whereas practical encoders use soft attention. Recent work narrows this gap by showing that soft attention can simulate hard attention [28] and express fragments of LTL [10].

Mechanistic Interpretability. Mechanistic interpretability (MI) seeks to explain neural-network behavior by reverse-engineering internal components such as neurons, attention heads and circuits [2, 29, 30]. For transformers, MI includes attention-head analysis and pruning [31, 32], interpretations of feed-forward layers as key-value memories [33], and causal methods such as activation patching, causal tracing, and circuit discovery [34–36]. Recent sparse-probing and sparse-autoencoder methods identify interpretable neurons or latent features, including in protein language models [3, 5–7, 37–40]. Our objective is complementary to that of MI. MI describes the transformer rather than the data, whereas we distill a formula that identifies which sequence-level pattern drives a prediction. Such a formula can be read directly by an expert and parsed, queried or manipulated using formal methods.

Expressivity based interpretability. The idea of using expressivity of the machine learning model for interpreting it has been extensively investigated in graph neural network literature [41–44]. However, to the best of our knowledge no similar attempts have been made for transformers. The most relevant work to this paper in this domain is by Pluska et al. [44]. The key idea of creating iterated decision trees with increasing quantifier depth (resp. temporal depth) is inspired by their work. However, iterated decision trees in [44] are limited to the logic of graph neural networks and cannot be ported to string data for transformers.

3 Background notation and results

We use $[n]$ to denote the set $\{0, \ldots, n-1\}$. An *alphabet* is a set of symbols. Given a finite alphabet Σ, a *string* is an ordered sequence of symbols from Σ. We write Σ^+ for the set of all finite non-empty strings over Σ. Thus, any string $w \in \Sigma^+$ can be written as $w = w_0 w_1 \cdots w_{n-1}$ for some $n \geq 1$. For $i \in [|w|]$, we write (w, i) for the i-th position of w. We will use Boolean connectives $\wedge, \vee$ and $\neg$ with their usual semantics. Given a finite set of logical formulas Δ, we will use the notation $\mathsf{Bool}(\Delta)$ to denote the *Boolean closure* of Δ, i.e., the smallest set of semantically distinct formulas containing Δ and closed under application of Boolean connectives.

3.1 Linear temporal logic on strings

We use LTL over finite non-empty strings. An LTL formula is recursively defined as follows:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid X\varphi \mid (\varphi U \varphi), \qquad p \in \Sigma.$$

Let $w = w_0 \cdots w_{n-1} \in \Sigma^+$ and $i \in [n]$. Atoms are interpreted by $(w, i) \models p$ iff $w_i = p$, and Boolean connectives have their usual semantics. The next operator (X) is interpreted as follows:

$$(w, i) \models X\varphi \quad \text{iff} \quad i + 1 < n \text{ and } (w, i + 1) \models \varphi.$$

We also use the until (U) operator:

$$(w, i) \models \varphi U \psi \quad \text{iff} \quad \exists j \in [n] \Big(i < j \wedge (w, j) \models \psi \wedge \forall k \, (i \leq k < j \Rightarrow (w, k) \models \varphi) \Big).$$

A string satisfies φ, written $w \models \varphi$, iff $(w, 0) \models \varphi$. Temporal depth $\mathsf{td}(\varphi)$ is defined by $\mathsf{td}(p) = \mathsf{td}(\top) = 0$, Boolean connectives preserve the maximum depth, and X and U add one:

$$\mathsf{td}(X\varphi) = 1 + \mathsf{td}(\varphi), \qquad \mathsf{td}(\varphi U \psi) = 1 + \max\{\mathsf{td}(\varphi), \mathsf{td}(\psi)\}.$$

In the following we provide an example of LTL formula evaluation on a string.

Example 1. *Let $\Sigma = \{p, q, r\}$ and let $w =$ `ppqrr`. Consider the formula*

$$\varphi := p \, U \, (q \vee Xr).$$

Below, we evaluate φ and all its subformulas at each position (w, i) of w.

	td	0	1	2	3	4
i		0	1	2	3	4
w_i		p	p	q	r	r
r	0	0	0	0	1	1
p	0	1	1	0	0	0
q	0	0	0	1	0	0
Xr	1	0	0	1	1	0
$q \vee Xr$	1	0	0	1	1	0
$p \, U \, (q \vee Xr)$	2	1	1	0	0	0

Since $(w, 0) \models \varphi$, we have $w \models \varphi$. We will often view formula evaluations at (w, i) as features of position i. In the table, the feature vector for position i is the column indexed by i. In this case it records the truth values of the displayed subformulas of φ at that position.

3.2 Transformer encoder

We model a transformer encoder as a layered sequence model over a finite alphabet Σ. Given an input string $w = w_0 w_1 \cdots w_{n-1} \in \Sigma^+$, an input embedding function produces an initial sequence of vector valued $\mathbf{v}_i^{(0)} \in \mathbb{R}^d$ token representations,

$$\mathcal{H}^{(0)}(w) = (\mathbf{v}_0^{(0)}, \ldots, \mathbf{v}_{n-1}^{(0)}).$$

A k-layer transformer encoder F then iteratively updates this sequence, producing for each layer $i \in \{1, \ldots, k\}$ a sequence of token representations

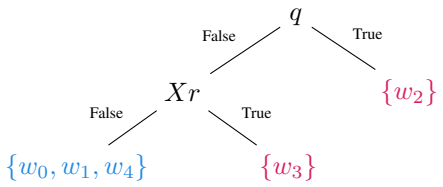
$$\mathcal{H}^{(i)}(w) = (\mathbf{v}_0^{(i)}, \ldots, \mathbf{v}_{n-1}^{(i)})$$

A transformer encoder can be used for string classification using a final classification head that maps $\mathcal{H}^{(k)}(w)$ to a Boolean output.

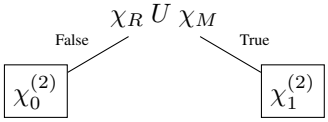
3.3 Transformers and linear temporal logic

Recent work suggests a close connection between transformer expressivity and different variants of LTL. This work has focused on theoretical abstractions of the transformer models with *hard attention scores*[2]. We will briefly revisit some of these results that motivate our methods. A key result has been that transformers can express LTL.

[2]Transformers where, for a fixed i, given the attention scores a_{ij} for each position j, the pooling function f_{pool} is restricted to only the top-scoring vectors $\mathbf{v}_j$: in unique hard attention it selects only one winner $\mathbf{v}_j$ with largest a_{ij}, while in average-hard attention it averages over all $\mathbf{v}_j$ with corresponding maximum a_{ij} with tied scores. In practice f_{pool} is usually weighted sum of $\mathbf{v}_j$, with each $\mathbf{v}_j$ weighted by $\mathsf{softmax}(a_{ij})$



(a) Layer 1



(b) Layer 2

Figure 1: (Left) We have a temporal decision tree for $\psi = q \vee Xr$ on $w =$ ppqrr. Positive outputs ($\{w_2, w_3\}$) are in red, and negative outputs ($\{w_0, w_1, w_4\}$) are in blue. Each leaf represents a formula: the left leaf $\chi_L := \neg q \wedge \neg Xr$, the middle leaf $\chi_M := \neg q \wedge Xr$, and the right leaf $\chi_R := q$. (Right) In an ITDT, we can add a second TDT layer $\mathcal{T}_2$ evaluating $\chi_R \; U \; \chi_M$.

Theorem 1 (Informal. Barcelo et al. [8])**.** *Let Σ be a finite alphabet and ϕ be an LTL formula, then there exists a hard attention transformer F that positively classifies a sequence w iff $w \models \phi$.*

Yang et al. [9] have further strengthened Barcelo et al. [8]'s results by showing that there are families of hard attention transformers that are equally expressive as LTL. A key theoretical insight from the proof of Barcelo et al. [8] and strengthened by Yang et al. [9] is that each encoder layer of a hard attention transformer can only increase the since/until temporal operator depth by one.

Theorem 2 (Informal since/until depth hierarchy, Yang et al. [9])**.** *For the temporal depth measured by nesting of the binary temporal operators U (or Since or both), the languages recognized by k-layer hard-attention transformers form a strict hierarchy by k. Equivalently, adding one encoder layer increases expressivity by at most one unit of temporal-depth, and this bound is tight.*

Li et al. [10] and Yang et al. [28] have reduced the gap between expressivity of hard attention transformers, soft attention transformers, and LTL.

4 Iterated temporal decision trees

We now introduce the formal structure of our distillation model, the *Iterated Temporal Decision Tree* (ITDT). ITDTs consist of a sequence of *temporal decision trees* (TDTs), which are decision trees with LTL formulas as decision predicates. In an ITDT, each subsequent TDT layer allows nesting of an additional temporal modality. Hence, a k-layer ITDT can represent any LTL formula with temporal depth up to k. In the following, we formalize the ITDT model, starting with TDTs.

Definition 1 (Temporal Decision Tree (TDT))**.** *Let Σ be a finite alphabet and let Δ be a set of LTL formulas over Σ. A temporal decision tree over Δ is a binary decision tree whose internal nodes are labeled by formulas from Δ and whose leaves are labeled in $\{0, 1\}$.*

A TDT $\mathcal{T}$ maps each position (w, i) of a word $w \in \Sigma^+$ to a value in $\{0, 1\}$. At each internal node labeled φ, it follows the true branch if $(w, i) \models \varphi$ and the false branch otherwise. The label of the reached leaf is denoted by $\mathcal{T}[w, i]$. Since a decision tree can realize any Boolean function over finitely many Boolean tests, a TDT over Δ can represent any Boolean combination of LTL formulas in Δ. The next example illustrates this construction.

Example 2. *Let $\Delta = \{q, Xr\}$, Figure 1a shows a TDT for the formula $\psi := q \vee Xr$. Let $w =$ ppqrr be a string, then the TDT outputs $\mathcal{T}[w, i] = 1$ at $i \in \{2, 3\}$, and $\mathcal{T}[w, i] = 0$ at $i \in \{0, 1, 4\}$.*

The leaves of a TDT naturally represent LTL formulas. For a leaf ℓ, let $\mathrm{path}(\ell)$ be the root-to-ℓ path, then the leaf formula for leaf ℓ is defined as

$$\chi_\ell := \bigwedge_{u \in \mathrm{path}(\ell)} \mathrm{lit}(u, \ell)$$

where $\mathrm{lit}(u, \ell) = \phi_u$ if the path to ℓ takes the true-branch at node u, and $\mathrm{lit}(u, \ell) = \neg\phi_u$ otherwise. Figure 1a shows leaf formulas for the TDT in Example 2. We now introduce the notion of *temporal expansions*, which formalizes adding one additional temporal modality to a given set of LTL formulas.

Definition 2 (Temporal Expansion)**.** *Given a set of LTL formulas Δ, let $\mathsf{Bool}(\Delta)$ be its Boolean closure under $\neg, \wedge, \vee$. Then the temporal expansion, $t(\Delta)$, of Δ is defined as:*

$$t(\Delta) := \mathsf{Bool}(\Delta) \cup \{ X\phi \mid \phi \in \mathsf{Bool}(\Delta) \} \cup \{ \phi \; U \; \psi \mid \phi, \psi \in \mathsf{Bool}(\Delta) \}.$$

The following example illustrates how temporal expansion provides a new set of deeper LTL formulas.

Example 3. *Let us define $\Delta = \{p, q\} \cup \{\chi_L, \chi_M, \chi_R\}$, where χ_L, χ_M and χ_R are the leaf formulas of the TDT from Example 2 (see Figure 1a), where $\chi_L := \neg q \wedge \neg Xr$, $\chi_M := \neg q \wedge Xr$, and $\chi_R := q$. Then $\mathsf{Bool}(\Delta)$ contains all Boolean combinations of these formulas, for instance $\chi_R \vee \chi_M \equiv q \vee Xr$. The temporal expansion introduces new formulas such as $p\, U(\chi_R \vee \chi_M) \equiv p\, U(q \vee Xr)$.*

TDTs can represent arbitrary Boolean combinations of a given set of LTL formulas. Temporal expansions can create a new set of features with one deeper temporal depth from a given set of LTL formulas. *Iterated Temporal Decision Trees* allow us to stack multiple layers of TDTs to express arbitrary LTL formulas. In the following we formally define ITDTs.

Definition 3 (Iterated Temporal Decision Tree (ITDT))**.** *Fix a base set of LTL formulas Δ_1 over Σ. An ITDT with k layers is a sequence of pairs $\{(\Delta_i, \mathcal{T}_i)\}_{i=1}^{k}$ such that:*

- *for every $i = 1, \ldots, k$, $\mathcal{T}_i$ is a temporal decision tree over Δ_i and*
- *for every $i = 1, \ldots, k-1$, we have that*

$$\Delta_{i+1} := t\Big(\bigcup_{1 \le j \le i} \{\chi_\ell^{(j)} \mid \ell \text{ is a leaf of } \mathcal{T}_j\} \Big),$$

where $\chi_\ell^{(j)}$ is the path formula of leaf ℓ in $\mathcal{T}_j$.

The output of the ITDT on a word w is $\mathcal{T}_k[w, 0]$.

In the following theorem we show that ITDTs with k layers can express any LTL formula with temporal depth up to k.

Theorem 3. *Let Σ be a finite alphabet and let φ be an LTL formula of temporal depth at most $k \ge 1$. Then there exists an ITDT with k temporal decision tree layers $\{\mathcal{T}_i\}_{i=1}^{k}$, with $\Delta_1 = t(\Sigma \cup \{\top\})$, such that for every word $w \in \Sigma^+$, we have that $\mathcal{T}_k[w, 0] = 1 \iff w \models \varphi$.*

Proof sketch. The proof proceeds by induction on temporal depth. We show that, after layer i, the accumulated leaf formulas of $\{\mathcal{T}_j\}_{1 \le j \le i}$ are rich enough for a TDT to represent every subformula of φ of temporal depth at most i.

The base case holds because $\Delta_1 = t(\Sigma \cup \{\top\})$ contains atoms, their Boolean combinations, and their one-step temporal extensions. Hence, $\mathcal{T}_1$ can express the subformulas of φ of temporal depth at most 1. For the induction step, assume that the accumulated leaf formulas up to i-th layer are sufficient to represent all subformulas of φ of temporal depth at most i. By Definition 3, the feature set Δ_{i+1} is obtained by applying $t(\cdot)$ to these accumulated leaf formulas. This expansion makes available the Boolean combinations of depth-i subformulas and their one-step temporal extensions using X and U. Hence $\mathcal{T}_{i+1}$ can express every subformula of φ of temporal depth at most $i + 1$. Iterating this construction for k layers yields a TDT $\mathcal{T}_k$ that can express φ. Therefore, for every $w \in \Sigma^+$, $\mathcal{T}_k[w, 0] = 1 \iff w \models \varphi$. □

We provide the full proof of Theorem 3 in the Appendix A. In the following we provide an example of a two-layer ITDT.

Example 4 (A two-layer ITDT with an Until predicate)**.** *Let $w = \texttt{ppqrr}$ and consider the first-layer temporal decision tree $\mathcal{T}_1$ from Example 2 for $\psi := q \vee Xr$, with decision predicates $\mathcal{D}(\mathcal{T}_1) = \{q, Xr\}$. Its leaves induce the formulas $\chi_L \equiv (\neg q \wedge \neg Xr), \chi_M \equiv (\neg q \wedge Xr)$, and $\chi_R \equiv q$. From these we can obtain a formula in the temporal expansion, e.g. $q\, U\, (\neg q \wedge Xr)$, and build a second-layer TDT $\mathcal{T}_2$ that evaluates it (Figure 1b).*

5 Distilling transformers into iterated temporal decision trees

Our distillation algorithm is inspired by proofs for transformer expressivity by Barcelo et al. [8]. Their proof shows that hard-attention transformers can express any LTL formula φ, by encoding the truth value of sub-formulas with temporal depth up to i in the outputs of the i^{th} layer of a transformer.

Furthermore, Yang et al. [9] show that such a construction is tight for temporal depth measured in terms of since/until operators. These observations guide our layer-wise distillation algorithm.

We present a meta-algorithm for distilling ITDTs from a trained transformer model. We later concretize it by describing algorithmic design choices relevant for practical implementation.

Let $\mathcal{W}$ be a training set of input sequences and let F be a trained transformer encoder with k layers. We want to distill a transformer with k layers into an ITDT with k layers. Let $\mathcal{H}^{(0)}$ be the Boolean encoding of the input string, with one layer of temporal expansion, i.e., each position in the string has a feature vector containing a Boolean entry for $(w, i) \models \xi$, where $\xi \in t(\Sigma \cup \{\top\})$. As explained in Section 3.2, each transformer layer updates these feature vectors. For each $i \in \{1, \ldots, k\}$, let $\mathcal{H}^{(i)}$ denote the token representations of all sequences in $\mathcal{W}$ after the i^{th} transformer layer. Let $\mathcal{H}_F := \left(\mathcal{H}^{(0)}, \mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(k)}\right)$ be the set of all sequence representations.

Algorithm 1 Distillation Procedure for ITDTs

1: **procedure** LEARNITDT($\mathcal{W}, \mathcal{H}_F$)
2: ITDT$\leftarrow \emptyset$
3: $\mathcal{U} \leftarrow \mathcal{H}^{(0)}$
4: **for** $i = 1, \ldots, k$ **do**
5: $\mathcal{T}_i \leftarrow \mathsf{LearnTDT}(\mathcal{W}, \mathcal{U}, \mathcal{H}^{(i)})$
6: $\Gamma_i \leftarrow t(\bigcup_{1 \le j \le i} \{\chi_\ell^{(j)} \mid \ell \text{ is a leaf of } \mathcal{T}_j\})$
7: **for** $\xi \in \Gamma_i$ **do**
8: $\mathcal{U} \leftarrow \mathsf{Append}(\mathcal{U}, \mathsf{FeatureVector}(\mathcal{W}, \xi))$
9: **end for**
10: ITDT$\leftarrow \mathsf{Append}(\text{ITDT}, \mathcal{T}_i)$
11: **end for**
12: **return** ITDT
13: **end procedure**

Algorithm 1 proceeds layer-wise. It maintains a feature pool $\mathcal{U}$ that assigns to each position (w, j) in each training sequence $w \in \mathcal{W}$ a feature vector of logical predicates. Initially, $\mathcal{U} = \mathcal{H}^{(0)}$ contains the Boolean input encoding with one temporal expansion. At iteration i, the subroutine $\mathsf{LearnTDT}(\mathcal{W}, \mathcal{U}, \mathcal{H}^{(i)})$, a decision-tree regressor, learns a TDT $\mathcal{T}_i$ that predicts the layer-i transformer representations $\mathcal{H}^{(i)}$ from the current features $\mathcal{U}$. Thus, layer i of F supervises layer i of the ITDT. After learning $\mathcal{T}_i$, we expand the feature pool with the leaves of all the previous trees up to $\mathcal{T}_i$. Concretely, we collect the path-formulas of all the TDTs until layer i, compute their temporal expansion using the function $t(\cdot)$, and append to $\mathcal{U}$ the feature column $\mathsf{FeatureVector}(\mathcal{W}, \xi)$ by evaluating each formula ξ in the temporal expansion, at each position of each string. Hence, at iteration $i{+}1$, the learner has access to temporally expanded concepts from layers 1 through i.

5.1 Practical algorithm design

Algorithm 1 captures the theoretical core of our distillation procedure. We now discuss different adaptations to the algorithm that enable implementation and improve performance on real-world data.

Learning TDTs. The subroutine $\mathsf{LearnTDT}$ learns a TDT with supervision from vector-valued transformer representation. We therefore treat this step as a multi-output regression problem. In practice, we instantiate $\mathsf{LearnTDT}$ with a standard decision-tree regressor [45]: internal nodes split on features from the current logical pool $\mathcal{U}$, and each leaf stores the mean transformer representation of the training tokens that reach it. The resulting tree's leaf predictions are trained to approximate the transformer layer, while its root-to-leaf paths define the formulas that are temporally expanded in the next iteration. At test time, only the formulas are relevant for prediction and the mean transformer representations are not used anymore.

Restricted temporal expansion. Temporal expansion (as given in Definition 2, used in line 6 in Algorithm 1) is highly intractable in the worst case. For a finite set of formulas Δ, the Boolean closure $\mathsf{Bool}(\Delta)$ can contain $O(2^{2^{|\Delta|}})$ semantically distinct formulas. Moreover, the temporal operators introduce all pairs $\varphi U \psi$ with $\varphi, \psi \in \mathsf{Bool}(\Delta)$, adding a quadratic overhead in $|\mathsf{Bool}(\Delta)|$. We

therefore use a restricted temporal expansion without Boolean closure. Furthermore, in iteration i we temporally expand only the leaves of $\mathcal{T}_i$, denoted by L_i, i.e., $L_i = \{\chi_\ell^{(i)} \mid \ell \text{ is a leaf of } \mathcal{T}_i\}$. Hence, in practice line 6 of Algorithm 1 is implemented as:

$$\Gamma_i \leftarrow L_i \cup \{X\phi \mid \phi \in L_i\} \cup \{\phi U \psi \mid \phi, \psi \in L_i\} \cup \Big(\bigcup_{1 \le j \le i-1} L_j \Big).$$

This restriction does not eliminate Boolean composition: each subsequent TDT can still realize arbitrary Boolean combinations of the formulas available in its input pool. The approximation only removes the exhaustive enumeration of all Boolean combinations before temporal expansion. Instead, the trees learn a task-dependent set of formulas to expose to later layers. In the experiments we allow additional temporal operators, Since, Previous, Globally and Eventually for temporal expansion, in the same way as U and X, respectively. However, in our setting, i.e., over finite strings evaluated at $(w, 0)$, this does not change the expressive power, but it can yield shorter formulas.

The following algorithmic additions are only made for experiments on protein sequence datasets.

Count features. In $\mathsf{FeatureVector}(\mathcal{W}, \xi)$, we also add the number of times a ξ is true in a trace, i.e., $\mathsf{Count}(\xi)$, to each position-wise feature vector of each string. For instance in Example 1, if $\xi = p$, then we add the entry $\mathsf{Count}(\mathsf{p}) = 2$ to all the feature vectors of (w, i). This is useful for proteins, where the label may depend not only on whether a local pattern occurs, but also on how often it occurs. Moreover, this is also consistent with expressivity results of Barcelo et al. [8], who show that transformers can express LTL extended with counting, LTL(C,+).

Avoiding over-compression. Skean et al. [46] show that in layered models like transformers, the final layers can often be over-compressed and less informative than the intermediate or even initial transformer layers. For last layer TDT regression this is especially challenging as decision trees are not able to create partitions fine enough for effective regression to transformer representations. Hence, in the k^{th} iteration of Algorithm 1, we completely skip $\mathsf{LearnTDT}(\cdot)$ and only create an additional set of temporal features so that deeper formulas can still be learned and expressed. Hence, the last layer only augments the feature vector with formulas as follows:

$$\Gamma_k \leftarrow \Gamma_{k-1} \cup \{X\phi \mid \phi \in \Gamma_{k-1}\} \cup \{\phi U \psi \mid \phi, \psi \in \Gamma_{k-1}\} \cup \Big(\bigcup_{1 \le j \le k-2} L_j \Big).$$

Thus, although no tree $\mathcal{T}_k$ is learned in the protein experiments, the final readout still receives formulas with one additional unit of temporal depth.

Final classifier. In the final layer for Boolean classification of the whole string we use a linear threshold with Boolean and Count features. This linear thresholding is also expressible in LTL(C,+). Let $\mathcal{B}$ be the final set of Boolean LTL features available after the last temporal expansion. For a word w, the final classifier has the form

$$\hat{y}(w) = \mathbb{1}\left[\beta_0 + \sum_{\phi \in \mathcal{B}} \beta_\phi \, \mathbb{1}[(w,0) \models \phi] + \sum_{\xi \in \mathcal{B}} \alpha_\xi \, \mathsf{Count}_w(\xi) > 0 \right],$$

Any rational affine threshold of this form is expressible in $\mathrm{LTL}(C, +)$ [8]. We learn such a threshold through ridge regression on Γ_k.

6 Experiments

We organize the experiments around five research questions (RQs) that test the main claims behind ITDT distillation. The method is motivated by hard-attention expressivity results [8, 9], but practical transformer encoders use soft attention [1]. We therefore first test ITDTs in a synthetic setting where the target concept is known to be LTL-expressible. **RQ1.** Can ITDTs recover accurate LTL formulas from soft-attention transformer encoders? We then ask whether the method extends to real-world protein sequence tasks. **RQ2.** Can ITDTs accurately distill transformer encoders

on real-world sequence tasks? Next, we isolate the two main components of the method: layer-wise transformer supervision and temporal feature expansion. **RQ3.** How much does layer-wise transformer supervision improve ITDT learning? **RQ4.** How much does temporal feature expansion improve ITDT learning? Finally, we ask whether the learned model is interpretable, not merely accurate. **RQ5.** Do ITDTs recover compact and interpretable rules?

6.1 Experimental setup

We evaluate ITDTs on an established LTL benchmark and four protein sequence datasets. For the protein tasks, we fine-tune the protein language model ESM-2-T6 (8M parameters) [19]. To run the experiments we used a single H100 80G GPU. We describe both experimental settings below.

LTL benchmarks. Learning LTL specifications from traces is a standard benchmark problem in formal methods. We use an established synthetic benchmark for LTL learning [12–14]. The benchmark contains nine formulas covering three common classes of sequence properties: existence, absence, and universality. We give the formulas and trace encoding in Appendix B. For each formula, we train a transformer with 10^5 parameters from scratch using 700 training traces, 100 validation traces, and 200 test traces. Trace lengths range from 10 to 100, and traces are generated with the sampling suite of Bathie et al. [14]. We distill ITDTs in this case without any counting features or linear thresholds. This keeps ITDT expressivity strictly within LTL.

Protein sequence benchmarks. Protein sequences are naturally represented as strings over the amino-acid alphabet. This makes them a suitable testbed for ITDTs, as temporal formulas can express sequence-level patterns such as the presence, absence, order, or repeated occurrence of amino-acid motifs. We also use the counting features and skip distillation of the last layer[3] as explained in Section 5.1. Specifically, we focus on binary peptide classification tasks and protein language models to test whether ITDTs can be used in practical applications. We have selected four distinct datasets for our research:

CPP924. A widely adopted dataset for cell-penetrating peptide prediction [18, 47, 48] containing 924 peptides sequences labeled as cell-penetrating or non cell-penetrating.

ToxinPred 3.0 [16]. A peptide toxicity prediction dataset. Peptides sequences are labeled as either toxic or non-toxic. The training set contains 8828 samples while the independent test set contains 2208 samples. The dataset is an updated version of the ToxinPred benchmark [49].

ACP740 [17, 50, 51]. A dataset for anticancer peptide (ACP) prediction, comprising 740 peptide sequences labeled as anticancer or non-anticancer.

AMP Scanner V2 [52, 53]. A dataset for antimicrobial peptide (AMP) prediction, comprising 1732 samples in the train-val dataset and 1024 samples in the independent test set.

CPP924, Toxinpred 3.0 and AMP Scanner V2 come with an independent test set, therefore we use the training set for finetuning and distillation and the test set for evaluation, running a total of 5 seeds per dataset. Performance on ACP740 is instead evaluated through a 5 fold cross validation procedure.

Table 1: Accuracy (mean $\pm$ std) on four protein classification datasets. ESM-2-T6 is the fine-tuned teacher model. ITDT is distilled with layer-wise ESM-2 supervision (RQ2). ITDT-Random substitutes transformer representations with random Gaussian vectors (RQ3). ITDT-GT Guidance replaces them with the ground-truth label broadcast to all sequence positions (RQ3). ITDT-No Expansion removes temporal expansion, retaining only aggregation features (RQ4).

Method	CPP924	ToxinPred3.0	ACP740	AMP Scanner V2
ESM2	$0.92_{\pm 0.01}$	$0.88_{\pm 0.01}$	$0.86_{\pm 0.05}$	$0.92_{\pm 0.00}$
ITDT	$\mathbf{0.92}_{\pm 0.02}$	$\mathbf{0.85}_{\pm 0.00}$	$\mathbf{0.84}_{\pm 0.03}$	$\mathbf{0.87}_{\pm 0.01}$
ITDT-Random	$0.90_{\pm 0.00}$	$0.83_{\pm 0.00}$	$0.79_{\pm 0.03}$	$0.86_{\pm 0.00}$
ITDT-GT Guidance	$0.87_{\pm 0.02}$	$\mathbf{0.85}_{\pm 0.00}$	$0.78_{\pm 0.05}$	$0.85_{\pm 0.00}$
ITDT-No Expansion	$0.88_{\pm 0.01}$	$0.81_{\pm 0.00}$	$0.80_{\pm 0.02}$	$0.81_{\pm 0.01}$

[3]We do identical skipping of supervision over last layer in all the ITDTs for ESM-2 ablations as well.

RQ1. ITDTs can recover accurate LTL formulas from soft attention transformers. ITDTs match transformer accuracy on all nine LTL formulas, with near-perfect ground-truth prediction accuracy throughout (Appendix B, Table 3). The learned formulas have temporal depth at most 1 and tree depth between 2 and 4. Hence, the formulas are both accurate and succinct. Hyperparameters used for the model are in Appendix B.6.

RQ2. ITDTs accurately distill transformer encoders on real-world sequence tasks. Table 1 shows that **ITDT** closely tracks ESM-2 performance across all four datasets, with a maximum gap of 5% on AMP Scanner V2. On CPP924 and ACP740 the difference falls within one standard deviation of ESM2, indicating that the distilled temporal formulas capture most of the predictive signal learned by the teacher. We report additional metrics and fidelity against teacher prediction in Appendix B. Hyperparameters used for the models are in Appendix B.6.

RQ3. Transformer guidance helps in learning ITDTs. To isolate the role of transformer supervision, we use two ablations with alternative supervision signals. In *ITDT-Random*, we replace transformer hidden states with random Gaussian vectors. In *ITDT-GT Guidance*, we replace hidden states with the ground-truth label broadcast to all positions. Both are strong baselines: GT Guidance receives label information at every ITDT layer, while *ITDT-Random* resembles *extremely randomized trees* by generating many independent random features, from which ridge regression selects features useful for classification [54].

On LTL benchmarks (Appendix B in Table 3) we observe that both ITDT trained on transformer guidance and ITDT-GT guidance achieve near perfect accuracy, while ITDT trained on random guidance performs well on seven out of nine formulas, but fail catastrophically on two formulas. We make two key inferences from these observations: first, that transformer guidance is at least as good as uniform ground-truth guidance at each ITDT layer, and second, that intermediate transformer guidance is clearly better than random guidance.

On protein benchmarks (see Table 1), ITDTs generally outperform, and otherwise match, the ablations. This confirms that ESM-2-T6 transformer representations provide better supervision than random noise or ground-truth labels alone.

RQ4. Temporal features are necessary for accurate distillation. To determine whether temporal expansion drives distillation quality or whether transformer guidance alone is sufficient, we ablate ITDT-No Expansion, which removes temporal expansion and restricts each layer to boolean and count features of the sequence alphabet. We observe in table 1 that performance drops consistently across all four datasets, confirming that the LTL hypothesis space is a necessary component of accurate distillation and not merely a redundant enrichment of the feature pool.

RQ5. ITDTs can recover biologically relevant discriminative patterns. To test whether final-layer ITDT rules are interpretable beyond their predictive accuracy, we use an LLM-as-expert pipeline inspired by auto-interpretability [55, 56]. We use Lasso instead of Ridge to obtain sparser rules, at the cost of some accuracy degradation (5% in the worst case) on ToxinPred 3.0 and AMP Scanner V2. For each dataset we take 3 runs at random, we then select the five largest positive- and negative-coefficient rules, corresponding to the positive and negative classes, and rank them by absolute coefficient value.[4] We then ask an LLM-as-expert (GPT-5.5) [57] to identify the class suggested by each rule, given only the task description, the rule, and the operator semantics. Figure 2 reports average accuracy across seeds. The evaluator performs above chance on all four tasks: 90.0% on AMP Scanner V2, 76.7% on ToxinPred 3.0, and 70.0% on both ACP740 and CPP924. Thus, ITDT rules appear to capture sequence-level patterns that are broadly consistent with biochemical priors, rather than only predictive artifacts. We provide per-seed results, the LLM prompt, examples of learned rules and quantitative results for Lasso regression in Appendix B.

[4] Lasso sparsity makes coefficient magnitude a useful proxy for rule predictiveness, while reducing the spread of weight across correlated rules.

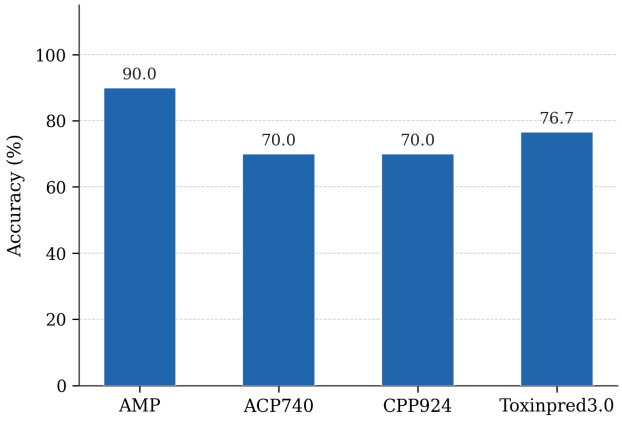


Figure 2: Average accuracy of LLM-as-expert on predicting the correct label for distilled rules.

7 Conclusion and limitations

We presented a method for distilling transformer encoders into accurate and interpretable symbolic models. The key idea is Iterated Temporal Decision Tree (ITDT): a layer-wise sequence of temporal decision trees whose decision splits are linear temporal logic (LTL) formulas. ITDTs can express any LTL formula and our distillation algorithm learns them from transformer hidden representations in a layer-by-layer fashion. Each ITDT layer introduces features with one additional temporal operator, mirroring the expressivity view that each transformer layer can add temporal depth. Empirically, ITDTs achieve near perfect accuracy on LTL learning benchmarks and distill interpretable rules for protein sequence tasks, often with minimal loss in accuracy. Our LLM-as-expert analysis further suggests that the learned formulas are aligned with broad biochemical priors.

Limitations. Even with practical adaptations (Section 5), the ITDT features grow quadratically with formulas being expanded. This makes large-vocabulary domains, e.g., natural language, challenging for the current implementation. However, once the feature set is fixed, LTL evaluation can be made linear in sequence length [58], whereas transformer self-attention is quadratic [1].

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. URL http://arxiv.org/abs/1706.03762.

[2] Leonard Bereska and Stratis Gavves. Mechanistic interpretability for AI safety - a review. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ePUVetPKu6. Survey Certification, Expert Certification.

[3] Etowah Adams, Liam Bai, Minji Lee, Yiyang Yu, and Mohammed AlQuraishi. From mechanistic interpretability to mechanistic biology: Training, evaluating, and interpreting sparse autoencoders on protein language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zdOGBRQEbz.

[4] Kaiyue Wen, Yuchen Li, Bingbin Liu, and Andrej Risteski. Transformers are uninterpretable with myopic methods: a case study with bounded dyck grammars. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=OitmaxSAUu.

[5] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

[6] Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tcsZt9ZNKD.

[7] Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4e82CIDxv.

[8] Pablo Barcelo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gbrHZq07mq.

[9] Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 10202–10235. Curran Associates, Inc., 2024. doi: 10.52202/079017-0327. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/13d7f172259b11b230cc5da8768abc5f-Paper-Conference.pdf.

[10] Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of fixed-precision transformer language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=29LwAgLFpj.

[11] Edmund M Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, London, Cambridge, 1999. ISBN 0-262-03270-8.

[12] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*, FMSP '98, pages 7–15, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919548. doi: 10.1145/298595.298598. URL https://doi.org/10.1145/298595.298598.

[13] Daniel Neider and Ivan Gavran. Learning linear temporal properties. In Nikolaj S. Bjørner and Arie Gurfinkel, editors, *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–10. IEEE, 2018. doi: 10.23919/FMCAD.2018.8603016. URL https://doi.org/10.23919/FMCAD.2018.8603016.

[14] Gabriel Bathie, Nathanaël Fijalkow, Théo Matricon, Baptiste Mouillon, and Pierre Vandenhove. Ltlf learning meets boolean set cover. In Sebastian Junges and Guy Katz, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 480–501, Cham, 2026. Springer Nature Switzerland.

[15] Felix Teufel, José Juan Almagro Armenteros, Alexander Rosenberg Johansen, Magnús Halldór Gíslason, Silas Irby Pihl, Konstantinos D. Tsirigos, Ole Winther, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. SignalP 6.0 predicts all five types of signal peptides using protein language models. *Nature Biotechnology*, 40(7):1023–1025, July 2022. ISSN 1546-1696. doi: 10.1038/s41587-021-01156-3. URL https://www.nature.com/articles/s41587-021-01156-3.

[16] Anand Singh Rathore, Shubham Choudhury, Akanksha Arora, Purva Tijare, and Gajendra P. S. Raghava. ToxinPred 3.0: An improved method for predicting the toxicity of peptides. *Computers in Biology and Medicine*, 179:108926, September 2024. ISSN 1879-0534. doi: 10.1016/j.compbiomed.2024.108926.

[17] Hai-Cheng Yi, Zhu-Hong You, Xi Zhou, Li Cheng, Xiao Li, Tong-Hai Jiang, and Zhan-Heng Chen. ACP-DL: A Deep Learning Long Short-Term Memory Model to Predict Anticancer Peptides Using High-Efficiency Feature Representation. *Molecular Therapy - Nucleic Acids*, 17:1–9, September 2019. ISSN 2162-2531. doi: 10.1016/j.omtn.2019.04.025. URL https://www.sciencedirect.com/science/article/pii/S2162253119300988.

[18] Leyi Wei, Jijun Tang, and Quan Zou. SkipCPP-Pred: an improved and promising sequence-based predictor for predicting cell-penetrating peptides. *BMC Genomics*, 18(7):742, October 2017. ISSN 1471-2164. doi: 10.1186/s12864-017-4128-1. URL https://doi.org/10.1186/s12864-017-4128-1.

[19] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan Dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, March 2023. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.ade2574. URL https://www.science.org/doi/10.1126/science.ade2574.

[20] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguistics*, 8:156–171, 2020.

[21] Shunyu Yao, Binghui Peng, Christos H. Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *ACL/IJCNLP*, pages 3770–3785, 2021.

[22] Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021.

[23] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *ICML*, volume 139, pages 11080–11090, 2021.

[24] Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics*, 10: 800–810, 2022.

[25] David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *ACL*, pages 7654–7664, 2022.

[26] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. In *EMNLP*, pages 7096–7116, 2020.

[27] David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *ICML*, volume 202, pages 5544–5562. PMLR, 2023.

[28] Andy Yang, Lena Strobl, David Chiang, and Dana Angluin. Simulating hard attention using soft attention. *Trans. Assoc. Comput. Linguistics*, 14:147–166, 2026. doi: 10.1162/TACL.A.597. URL https://doi.org/10.1162/tacl.a.597.

[29] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

[30] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[31] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors, *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL https://aclanthology.org/W19-4828/.

[32] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf.

[33] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446/.

[34] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/92650b2e92217715fe312e6fa7b90d82-Paper.pdf.

[35] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf.

[36] Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

[37] Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=JYs1R9IMJr.

[38] Paul Chon and William B. Andreopoulos. Mechanistic interpretability for protein language models: A validation framework. *bioRxiv*, 2026. doi: 10.64898/2026.05.29.727021. URL https://www.biorxiv.org/content/early/2026/06/02/2026.05.29.727021.

[39] Gal Pomerants, Yaniv Nikankin, Anja Reusch, Tomer Tsaban, Ora Schueler-Furman, and Yonatan Belinkov. Induction meets biology: Mechanisms of repeat detection in protein language models. In Tong Zhang, Miroslav Dudik, Martin Jaggi, Alekh Agarwal, Sharon Li, Dale Schuurmans, Jerry Zhu, Felix Berkenkamp, Hanze Dong, and Alberto Bietti, editors, *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 99094–99137. PMLR, 06–11 Jul 2026. URL https://proceedings.mlr.press/v306/pomerants26a.html.

[40] Darin Tsui, Kunal Talreja, Daniel Saeedi, and Amirali Aghazadeh. Protein circuit tracing via cross-layer transcoders. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Ki0eWT1sFS.

[41] Alessio Ragno, Marc Plantevit, and Céline Robardet. On logic-based self-explainable graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[42] Matthew Morris and Ian Horrocks. Sound logical explanations for mean aggregation graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=7TY89cqLfE.

[43] Masahiro Negishi, Thomas Gärtner, and Pascal Welke. WILTing trees: Interpreting the distance between MPNN embeddings. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=lflqQWP1jy.

[44] Alexander Pluska, Pascal Welke, Thomas Gärtner, and Sagar Malhotra. Logical distillation of graph neural networks. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, KR '24, 2024. ISBN 978-1-956792-05-8. doi: 10.24963/kr.2024/86. URL https://doi.org/10.24963/kr.2024/86.

[45] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984. ISBN 0-534-98053-8.

[46] Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=WGXb7UdvTX.

[47] Xin Zhang, Lesong Wei, Xiucai Ye, Kai Zhang, Saisai Teng, Zhongshen Li, Junru Jin, Min Jae Kim, Tetsuya Sakurai, Lizhen Cui, Balachandran Manavalan, and Leyi Wei. SiameseCPP: a sequence-based Siamese network to predict cell-penetrating peptides by contrastive learning. *Briefings in Bioinformatics*, 24(1):bbac545, January 2023. ISSN 1467-5463, 1477-4054. doi: 10.1093/bib/bbac545. URL https://academic.oup.com/bib/article/doi/10.1093/bib/bbac545/6958502.

[48] Kexin Shi, Yuanpeng Xiong, Yu Wang, Yifan Deng, Wenjia Wang, Bingyi Jing, and Xin Gao. PractiCPP: a deep learning approach tailored for extremely imbalanced datasets in cell-penetrating peptide prediction. *Bioinformatics*, 40(2):btae058, February 2024. ISSN 1367-4803, 1367-4811. doi: 10.1093/bioinformatics/btae058. URL https://academic.oup.com/bioinformatics/article/doi/10.1093/bioinformatics/btae058/7596624.

[49] Neelam Sharma, Leimarembi Devi Naorem, Shipra Jain, and Gajendra P S Raghava. ToxinPred2: an improved method for predicting toxicity of proteins. *Briefings in Bioinformatics*, 23(5):bbac174, September 2022. ISSN 1467-5463, 1477-4054. doi: 10.1093/bib/bbac174. URL https://academic.oup.com/bib/article/doi/10.1093/bib/bbac174/6590152.

[50] Xian-gan Chen, Wen Zhang, Xiaofei Yang, Chenhong Li, and Hengling Chen. ACP-DA: Improving the Prediction of Anticancer Peptides Using Data Augmentation. *Frontiers in Genetics*, 12:698477, June 2021. ISSN 1664-8021. doi: 10.3389/fgene.2021.698477. URL https://www.frontiersin.org/articles/10.3389/fgene.2021.698477/full.

[51] Aoyun Geng, Zhenjie Luo, Aohan Li, Zilong Zhang, Quan Zou, Leyi Wei, and Feifei Cui. ACP-CLB: An Anticancer Peptide Prediction Model Based on Multichannel Discriminative Processing and Integration of Large Pretrained Protein Language Models. *Journal of Chemical Information and Modeling*, 65(5):2336–2349, March 2025. ISSN 1549-9596, 1549-960X. doi: 10.1021/acs.jcim.4c02072. URL https://pubs.acs.org/doi/10.1021/acs.jcim.4c02072.

[52] Daniel Veltri, Uday Kamath, and Amarda Shehu. Deep learning improves antimicrobial peptide recognition. *Bioinformatics*, 34(16):2740–2747, August 2018. ISSN 1367-4803, 1367-4811. doi: 10.1093/bioinformatics/bty179. URL https://academic.oup.com/bioinformatics/article/34/16/2740/4953367.

[53] Chenkai Li, Darcy Sutherland, S. Austin Hammond, Chen Yang, Figali Taho, Lauren Bergman, Simon Houston, René L. Warren, Titus Wong, Linda M. N. Hoang, Caroline E. Cameron, Caren C. Helbing, and Inanc Birol. AMPlify: attentive deep learning model for discovery of novel antimicrobial peptides effective against WHO priority pathogens. *BMC Genomics*, 23(1):77, January 2022. ISSN 1471-2164. doi: 10.1186/s12864-022-08310-4. URL https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-022-08310-4.

[54] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Mach. Learn.*, 63(1):3–42, April 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6226-1. URL https://doi.org/10.1007/s10994-006-6226-1.

[55] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[56] Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/paulo25a.html.

[57] OpenAI. GPT-5.5 system card. Technical report, OpenAI, April 2026. URL https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf.

[58] Grigore Roşu and Klaus Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical Report Technical Report 01.15, RIACS, 2001.

A Appendix to theoretical results

Theorem 3. *Let Σ be a finite alphabet and let φ be an LTL formula of temporal depth at most $k \geq 1$. Then there exists an ITDT with k temporal decision tree layers $\{\mathcal{T}_i\}_{i=1}^{k}$, with $\Delta_1 = t(\Sigma \cup \{\top\})$, such that for every word $w \in \Sigma^+$, we have that $\mathcal{T}_k[w, 0] = 1 \iff w \models \varphi$.*

Proof. Fix an LTL formula φ with temporal depth at most k. We construct the ITDT layer by layer.

For a finite set of formulas Γ, let $\mathrm{FullTree}(\Gamma)$ denote the complete decision tree that contains every formula in Γ on each root-to-leaf path. Hence, its leaves encode all truth assignments to Γ. Therefore every Boolean combination of formulas in Γ is equivalent to a Boolean combination of the tree's leaf path formulas.

We prove by induction that, after constructing layer i, every subformula of φ with temporal depth at most i is equivalent to a Boolean combination of the leaf path formulas produced up to layer i.

For $i = 1$, set $\Delta_1 = t(\Sigma \cup \{\top\})$. Every subformula of φ with temporal depth at most 1 is a Boolean combination of formulas in Δ_1: depth-0 formulas are Boolean combinations of atoms, and depth-1 temporal formulas have the form $X\alpha$ or $\alpha U \beta$ with $\alpha, \beta \in \mathsf{Bool}(\Sigma \cup \{\top\})$. Let $P_1 \subseteq \Delta_1$ be the finite set of predicates needed for these subformulas, and set

$$\mathcal{T}_1 := \mathrm{FullTree}(P_1).$$

Then all subformulas of depth at most 1 are Boolean combinations of the leaf path formulas of $\mathcal{T}_1$.

Now suppose layers $1, \ldots, i$ have been constructed, and let Γ_i be the set of all leaf path formulas produced so far. By the induction hypothesis, every subformula of φ of depth at most i is equivalent to a Boolean combination of formulas in Γ_i. Since

$$\Delta_{i+1} = t(\Gamma_i),$$

the set Δ_{i+1} contains $\mathsf{Bool}(\Gamma_i)$ and all formulas

$$X\alpha \qquad \text{and} \qquad \alpha U \beta$$

with $\alpha, \beta \in \mathsf{Bool}(\Gamma_i)$. Hence every subformula of φ of depth at most $i + 1$ is a Boolean combination of formulas in Δ_{i+1}. Let $P_{i+1} \subseteq \Delta_{i+1}$ be the finite set of predicates needed for these subformulas, and set

$$\mathcal{T}_{i+1} := \mathrm{FullTree}(P_{i+1}).$$

Then the leaf path formulas of $\mathcal{T}_{i+1}$ make all subformulas of depth at most $i + 1$ recoverable as Boolean combinations.

After k layers, φ itself is a Boolean combination of the predicates queried by $\mathcal{T}_k$. We label each leaf of $\mathcal{T}_k$ by 1 exactly when the corresponding truth assignment satisfies φ , and by 0 otherwise. Therefore, for every word $w \in \Sigma^+$,

$$\mathcal{T}_k[w, 0] = 1 \quad \iff \quad w \models \varphi.$$

□

B Appendix to experiments

B.1 LTL benchmarks experiments and results

Each trace is encoded as a token string over a state alphabet. For formulas over two atomic propositions, the four possible valuations are mapped to the symbols {A, B, C, D}, corresponding to $\neg p_0 \wedge \neg p_1$, $\neg p_0 \wedge p_1$, $p_0 \wedge \neg p_1$, and $p_0 \wedge p_1$, respectively. For formulas over three propositions, four additional symbols {E, F, G, H} are introduced analogously. Trace lengths range from 10 to 100 tokens.

Table 2: Common LTL patterns used in practice [12, 13]. G is globally operator, F is eventually and U is until

Absence	Existence	Universality
$\varphi_{a1} : G(\neg p_0)$	$\varphi_{e1} : F(p_0)$	$\varphi_{u1} : G(p_0)$
$\varphi_{a2} : F(p_1) \rightarrow (\neg p_0 U p_1)$	$\varphi_{e2} : G(\neg p_0 \vee F(p_0 \wedge F(p_1)))$	$\varphi_{u2} : F(p_1) \rightarrow (p_0 U p_1)$
$\varphi_{a3} : G(p_1 \rightarrow G(\neg p_0))$	$\varphi_{e3} : G(p_0 \wedge (\neg p_1 \rightarrow (\neg p_1 U (p_2 \wedge \neg p_1))))$	$\varphi_{u3} : G(p_1 \rightarrow G(p_0))$

Table 3: Ground-truth prediction accuracy on the LTL learning benchmark across ablations. Results are averaged over three random seeds, error bars report standard deviation across seeds. Bold marks the best non-transformer method for each formula.

Formula	Transformer	ITDT	ITDT- GT Guidance	ITDT- Random	ITDT- No Expansion
Absence					
φ_{a1}	$0.999_{\pm 0.002}$	$\mathbf{0.997}_{\pm 0.003}$	$\mathbf{0.997}_{\pm 0.003}$	$0.995_{\pm 0.009}$	$\mathbf{0.997}_{\pm 0.003}$
φ_{a2}	$0.995_{\pm 0.005}$	$0.997_{\pm 0.006}$	$\mathbf{1.000}_{\pm 0.000}$	$0.697_{\pm 0.103}$	$0.500_{\pm 0.010}$
φ_{a3}	$1.000_{\pm 0.001}$	$0.998_{\pm 0.003}$	$\mathbf{1.000}_{\pm 0.000}$	$0.967_{\pm 0.036}$	$0.998_{\pm 0.003}$
Existence					
φ_{e1}	$1.000_{\pm 0.001}$	$0.998_{\pm 0.003}$	$0.998_{\pm 0.003}$	$0.987_{\pm 0.013}$	$\mathbf{1.000}_{\pm 0.000}$
φ_{e2}	$0.996_{\pm 0.002}$	$\mathbf{1.000}_{\pm 0.000}$	$0.997_{\pm 0.003}$	$0.987_{\pm 0.003}$	$\mathbf{1.000}_{\pm 0.000}$
φ_{e3}	$0.998_{\pm 0.003}$	$\mathbf{0.995}_{\pm 0.005}$	$\mathbf{0.995}_{\pm 0.005}$	$0.988_{\pm 0.008}$	$0.993_{\pm 0.003}$
Universality					
φ_{u1}	$1.000_{\pm 0.000}$	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{1.000}_{\pm 0.000}$
φ_{u2}	$0.995_{\pm 0.004}$	$0.997_{\pm 0.003}$	$\mathbf{1.000}_{\pm 0.000}$	$0.538_{\pm 0.032}$	$0.502_{\pm 0.006}$
φ_{u3}	$0.999_{\pm 0.003}$	$0.997_{\pm 0.006}$	$0.997_{\pm 0.006}$	$0.990_{\pm 0.010}$	$\mathbf{1.000}_{\pm 0.000}$

B.2 Quantitative results with Lasso regression

We show here the results with substituting Ridge regression in ITDT with Lasso regression to promote sparsity and select the rules for the llm-as-expert pipeline. While we experiment with three different settings of regularization weight α, for the LLM-as-expert pipeline we select the rules learned with α=0.005 as it consistently provides a low number of features with the smallest degradation of performance across datasets and seeds. Lasso is less accurate than Ridge on two datasets (ToxinPred 3.0 and AMP Scanner V2), yet it retains at least 90% of the accuracy of the teacher on every dataset. This gap reflects a deliberate trade-off: whereas the Ridge readout is optimized for predictive accuracy, the Lasso readout keeps only a small subset of rules, making the most predictive ones easier to isolate and inspect.

Table 4: Results of ITDT with Lasso regression across protein datasets and regularization strengths α. Values are mean $\pm$ std over seeds. Non-zero: mean number of rules with non-zero coefficients.

Dataset	α	Accuracy	F1 macro	MCC	ROC-AUC	Non-zero
TOXINPRED	0.005	0.81 ± 0.00	0.81 ± 0.00	0.63 ± 0.01	0.89 ± 0.00	71 ± 6
	0.01	0.80 ± 0.01	0.80 ± 0.01	0.61 ± 0.01	0.89 ± 0.00	39 ± 8
	0.05	0.78 ± 0.01	0.78 ± 0.01	0.60 ± 0.01	0.84 ± 0.02	8 ± 2
CPP924	0.005	0.93 ± 0.00	0.93 ± 0.00	0.87 ± 0.00	0.98 ± 0.00	97 ± 3
	0.01	0.93 ± 0.01	0.93 ± 0.01	0.85 ± 0.02	0.97 ± 0.00	58 ± 3
	0.05	0.85 ± 0.01	0.84 ± 0.01	0.69 ± 0.02	0.93 ± 0.01	18 ± 5
ACP740	0.005	0.84 ± 0.03	0.84 ± 0.03	0.68 ± 0.07	0.91 ± 0.03	102 ± 4
	0.01	0.83 ± 0.04	0.83 ± 0.04	0.66 ± 0.07	0.90 ± 0.03	50 ± 7
	0.05	0.76 ± 0.03	0.76 ± 0.03	0.53 ± 0.06	0.84 ± 0.03	11 ± 4
AMP SCANNER V2	0.005	0.84 ± 0.00	0.84 ± 0.00	0.69 ± 0.00	0.93 ± 0.00	120 ± 7
	0.01	0.82 ± 0.01	0.82 ± 0.01	0.65 ± 0.01	0.91 ± 0.00	60 ± 2
	0.05	0.71 ± 0.02	0.70 ± 0.03	0.46 ± 0.02	0.80 ± 0.01	7 ± 2

B.3 Additional Metrics for ITDT

We report here additional useful metrics for ITDT algorithm on protein tasks. We show the results for three models:

- **ESM2** Pretrained Transformer model finetuned on each dataset and utilized as teacher.
- **ITDT-GT** Standard ITDT algorithm, the last classification layer is learned with supervision by ground truth labels of the training set and evaluated both against test set ground truth labels (vs GT) and against ESM2 predictions on the test set (vs Teach).
- **ITDT-Teach** The last classification layer is learned with supervision by ESM2 predictions on the training set and evaluated both against test set ground truth labels (vs GT) and against ESM2 predictions on the test set (vs Teach).

Table 5: CPP924 additional metrics

Method		Accuracy	F1 Macro	ROC-AUC	MCC
ESM2		$0.92_{\pm 0.01}$	$0.92_{\pm 0.02}$	$0.98_{\pm 0.00}$	$0.84_{\pm 0.03}$
ITDT	GT vs GT	$0.92_{\pm 0.02}$	$0.92_{\pm 0.01}$	$0.97_{\pm 0.00}$	$0.84_{\pm 0.02}$
	GT vs Teach	$0.93_{\pm 0.01}$	$0.93_{\pm 0.01}$	$0.97_{\pm 0.00}$	$0.86_{\pm 0.03}$
ITDT-Teach	Teach vs GT	$0.92_{\pm 0.01}$	$0.92_{\pm 0.01}$	$0.97_{\pm 0.01}$	$0.84_{\pm 0.01}$
	Teach vs Teach	$0.93_{\pm 0.01}$	$0.93_{\pm 0.01}$	$0.98_{\pm 0.00}$	$0.87_{\pm 0.02}$

Table 6: ToxinPred3.0 additional metrics

Method		Accuracy	F1 Macro	ROC-AUC	MCC
ESM2		$0.88_{\pm 0.01}$	$0.88_{\pm 0.01}$	$0.94_{\pm 0.00}$	$0.75_{\pm 0.01}$
ITDT	GT vs GT	$0.85_{\pm 0.00}$	$0.85_{\pm 0.00}$	$0.92_{\pm 0.00}$	$0.70_{\pm 0.00}$
	GT vs Teach	$0.89_{\pm 0.00}$	$0.89_{\pm 0.00}$	$0.96_{\pm 0.01}$	$0.77_{\pm 0.01}$
ITDT-Teach	Teach vs GT	$0.85_{\pm 0.01}$	$0.85_{\pm 0.01}$	$0.91_{\pm 0.00}$	$0.70_{\pm 0.02}$
	Teach vs Teach	$0.91_{\pm 0.01}$	$0.91_{\pm 0.01}$	$0.97_{\pm 0.01}$	$0.82_{\pm 0.02}$

Table 7: ACP740 additional metrics

Method		Accuracy	F1 Macro	ROC-AUC	MCC
ESM2		$0.86_{\pm 0.05}$	$0.86_{\pm 0.05}$	$0.91_{\pm 0.05}$	$0.73_{\pm 0.10}$
ITDT	GT vs GT	$0.84_{\pm 0.03}$	$0.84_{\pm 0.02}$	$0.91_{\pm 0.03}$	$0.69_{\pm 0.05}$
	GT vs Teach	$0.86_{\pm 0.03}$	$0.86_{\pm 0.03}$	$0.94_{\pm 0.01}$	$0.72_{\pm 0.05}$
ITDT-Teach	Teach vs GT	$0.84_{\pm 0.03}$	$0.84_{\pm 0.03}$	$0.91_{\pm 0.04}$	$0.68_{\pm 0.07}$
	Teach vs Teach	$0.86_{\pm 0.03}$	$0.86_{\pm 0.03}$	$0.94_{\pm 0.01}$	$0.72_{\pm 0.05}$

Table 8: AMP Scanner V2 additional metrics

Method		Accuracy	F1 Macro	ROC-AUC	MCC
ESM2		$0.92_{\pm 0.00}$	$0.92_{\pm 0.00}$	$0.97_{\pm 0.00}$	$0.85_{\pm 0.01}$
ITDT	GT vs GT	$0.87_{\pm 0.01}$	$0.87_{\pm 0.01}$	$0.94_{\pm 0.00}$	$0.74_{\pm 0.02}$
	GT vs Teach	$0.88_{\pm 0.01}$	$0.88_{\pm 0.01}$	$0.95_{\pm 0.00}$	$0.76_{\pm 0.02}$
ITDT-Teach	Teach vs GT	$0.87_{\pm 0.01}$	$0.87_{\pm 0.01}$	$0.94_{\pm 0.00}$	$0.75_{\pm 0.01}$
	Teach vs Teach	$0.88_{\pm 0.00}$	$0.88_{\pm 0.00}$	$0.95_{\pm 0.00}$	$0.77_{\pm 0.01}$

B.4 Complexity analysis of ITDT learned rules

In order to analyze the complexity of the rules learned by ITDT, we select the 30 most important rules that appear in the decision boundary of the linear classifier learned by ITDT-ESM2 and plot the average number of rules learned at each layer in Figure 3. Note that a rule learned at layer k can have temporal depth $0 \leq i \leq k$, therefore this analysis functions as a proxy for complexity of the rules. We notice that rules originating from the first layer consistently outnumber rules from the other layers, suggesting a pattern of specialization at deeper layers.

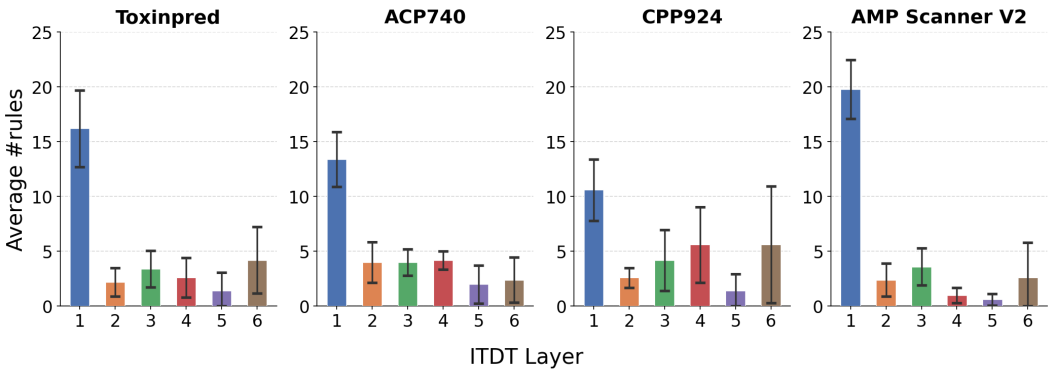


Figure 3: Average number of rules learned at each ITDT layer that appear among the 30 most important features for the final linear classifier, averaged across seeds. A rule learned at layer k can have temporal depth $0 \leq i \leq k$.

B.5 LLM-as-expert results across seeds

Here we provide the results with LLM-as-expert pipeline for each seed. We utilize 3 seeds for CPP924, Toxinpred 3.0 and AMP Scanner V2 and 5-fold cross-validation with ACP740 to maintain consistency with the cross-validation procedure utilized in main experiment and ablations.

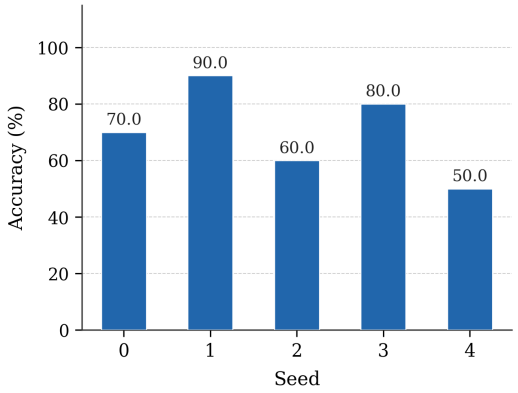


Figure 4: LLM-as-expert results for ACP740

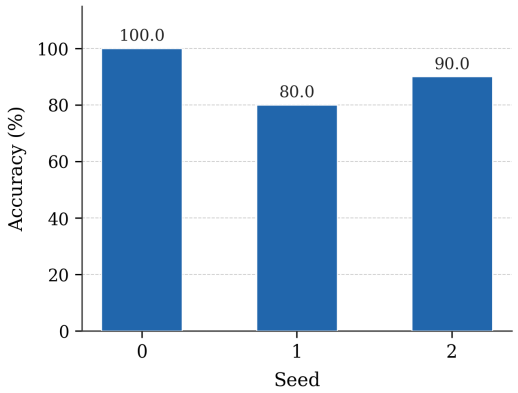


Figure 5: LLM-as-expert results for AMP Scanner V2

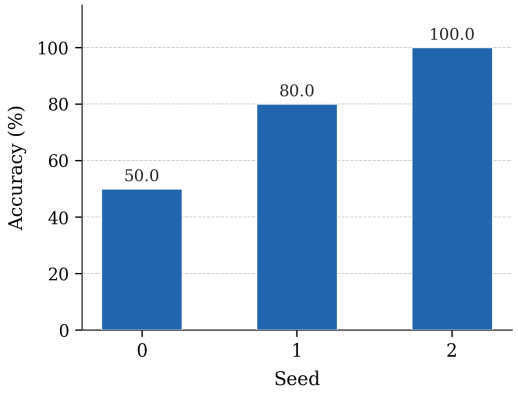


Figure 6: LLM-as-expert results for Toxinpred 3.0

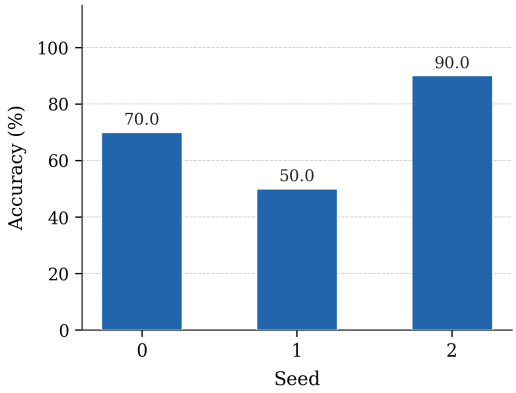


Figure 7: LLM-as-expert results for CPP924

LLM-as-expert accuracy by coefficient rank. To assess whether interpretability extends beyond the most extreme rules, we rank the non-zero Lasso rules of each run by absolute coefficient and report LLM-as-expert accuracy on disjoint rank bins (1–10, 11–20, 21–30), together with the aggregate over the top 30. With 70–120 active rules per dataset, the top 30 cover 25–43% of all active rules. These results come from a separate run of the LLM-as-expert pipeline, in which rules are ranked by absolute coefficient regardless of sign; values for the top-ranked rules therefore differ from Figure 2, due to the non-determinism of the model API.

Table 9: LLM-as-expert accuracy (mean over seeds) on disjoint bins of rules ranked by absolute Lasso coefficient.

Dataset	1–10	11–20	21–30	Top-30
ACP740	0.62	0.74	0.50	0.62
ToxinPred 3.0	0.76	0.60	0.47	0.61
CPP924	0.73	0.87	0.83	0.81
AMP Scanner V2	0.93	0.87	0.72	0.84

On CPP924 and AMP Scanner V2, accuracy remains well above chance across all bins. On ACP740 and ToxinPred 3.0, accuracy is above chance for the top 20 rules but drops to chance level for ranks 21–30, indicating that on these datasets the interpretability captured by the LLM-as-expert is concentrated in the most predictive rules. As accuracy also varies across seeds, we read these results as an above-chance signal on average across seeds rather than a per-seed guarantee.

B.6 Hyperparameters

LTL benchmark. The transformer model was trained with the following hyperparameters. The token embedding dimension was 64, with 8 attention heads and a feed-forward dimension of 256. Layer normalization was applied with an epsilon of 10^{-5}, and GELU was used as the activation function. Learned positional encodings with a maximum length of 105 were employed. The model consists of a single transformer layer. Training was conducted for 50 epochs with a batch size of 64, using the AdamW optimizer with a learning rate of 10^{-4} and a linear decay scheduler with no warmup steps.

Protein tasks. The ESM2 models for all the protein tasks share most training hyperparameters. Batch size of 64, a learning rate of 10^{-5}, a weight decay of 0.01, and the AdamW optimizer. The models differ only in the number of training epochs: the model for Toxinpred 3.0 was fine-tuned for 10 epochs, the model for CPP924 for 30, the model for AMP for 40, and the one for ACP740 for 20.

B.7 Prompt for LLM-as-expert pipeline

Here we provide the prompt used for LLM-as-expert pipeline, this specific instance of the prompt is applied to the CPP924 dataset. The same prompt is used for all other tasks by just changing the details about task description.

Note that:

- **Sum** is Count operator
- **Length** is $\mathsf{Count}(\top)$
- **Max** on finite traces $\mathsf{Max} \geq 0.5$ is Eventually

System Prompt: CPP Rule Interpretation

```
You are an expert computational biologist specialising in peptide
property prediction and interpretable machine learning. You are given
logical rules derived from a Cell Penetrating Peptides (CPP) vs non Cell Penetrating
     Peptides prediction task .

Each rule is expressed in a formal logic notation over amino-acid
features. Your task is to:
  1. Read and interpret the rule carefully.
  2. Decide whether the rule characterises an CPP (positive class)
     or a NONCPP (negative class).
  3. Provide a concise biological justification (max two sentences).

Notation guide (also provided in the user message):
  • Q_Y          -- feature for amino acid Y at a specific position (binary)
  • Q_Y_sum      -- count of amino acid Y across the whole sequence
  • Q_Y_length   -- length of the sequence (only length, not composition)
  • Q_Y_max      -- maximum value of feature Y across positions
  • A U B        -- LTL "Until": one or more A followed by B (regex: A+B)
  • X A          -- LTL "Next":  A holds at the next position (regex: .A)
  • Thresholds on plain Q_Y: < 0.5 means = 0 (absent), >= 0.5 means = 1 (present)
  • _sum / _length thresholds are real-valued counts / lengths
  • The rule is existential (binary): it either holds or it does not

Respond ONLY with a single valid JSON object -- no markdown fences,
no extra text -- with exactly these fields:

  {
    "prediction":  "1" or "0",   // "1" = CPP (positive), "0" = NONCPP (negative)
    "explanation": "<max two sentences>"
  }
```

User Prompt Template

```
{RULE}

Do you think this is a rule that identifies a NONCPP or CPP?
Please answer with a JSON object with the following fields:

  {
    "prediction":  "1" or "0",   // "1" = CPP (positive), "0" = NONCPP (negative)
    "explanation": "<max two sentences>"
  }
```

B.8 Examples of Rules Learned by ITDT with Lasso Regression

We present examples of rules learned on protein tasks by ITDT with Lasso regression, along with the class prediction and biological explanation produced by the LLM-as-expert pipeline (Section 6.1).

All displayed rules were correctly identified by the LLM. While the LLM received the raw rule expressions as output by ITDT, we apply a minor manual simplification here for readability.

ToxinPred 3.0

Rule 1 *Predicted: Toxic*
$F(\texttt{C}\,U\,\texttt{C})$
Explanation: Cysteine-rich motifs — here, at least two consecutive cysteines — are common in disulfide-stabilized toxic peptides such as venom toxins, making this pattern consistent with TOXIC peptides.

Rule 2 *Predicted: Toxic*
$F(\texttt{G}\,U\,\texttt{K})$
Explanation: This pattern encodes a glycine run followed by lysine ($G^{+}K$), a flexible cationic motif. Lysine-containing flexible regions are prevalent in membrane-active peptides, which are more commonly classified as TOXIC.

Rule 3 *Predicted: Non-Toxic*
$F(\texttt{Count(C)} < 4 \wedge ((\neg\texttt{K}) \wedge (\neg\texttt{P}) \wedge (\neg\texttt{G}))\ U\ \texttt{E})$
Explanation: This rule identifies peptides with low cysteine content that contain glutamate (E) but lack lysine, proline, and glycine. The presence of an acidic residue alongside the absence of cationic lysine is consistent with reduced membrane-disruptive activity, pointing toward NON-TOXIC peptides.

AMP Scanner V2

Rule 1 *Predicted: AMP*
$F(\texttt{T}\,U\,\texttt{C})$
Explanation: This rule corresponds to a motif of one or more threonines followed by a cysteine ($T^{+}C$). Cysteine-containing motifs are common in many AMPs, especially disulfide-stabilized peptides, making this pattern indicative of AMP behavior.

Rule 2 *Predicted: Non-AMP*
$F(\texttt{D}\,U\,\texttt{G})$
Explanation: This rule corresponds to a motif of one or more aspartic acid residues followed by glycine ($D^{+}G$). Aspartic acid is negatively charged, whereas AMPs are typically enriched in cationic residues such as Lys and Arg to interact with bacterial membranes, making this pattern more consistent with NON-AMP peptides.

Rule 3 *Predicted: Non-AMP*
$F(\texttt{E})$
Explanation: This rule flags the presence of glutamic acid (E), an acidic negatively charged residue, at a given position. Since AMPs are typically enriched in cationic residues and depleted in acidic ones, this feature is more consistent with NON-AMP peptides.

CPP924

Rule 1 *Predicted: CPP*
$F(\texttt{R}\,U\,\texttt{R})$
Explanation: This rule corresponds to a run of arginine residues, effectively requiring consecutive R positions. Arginine-rich motifs are a hallmark of many CPPs, as guanidinium groups promote membrane interaction and cellular uptake, making this pattern consistent with CPP peptides.

Rule 2 *Predicted: CPP*
$F(\texttt{K}\,U\,\texttt{K})$
Explanation: This rule indicates a stretch of lysine residues, effectively requiring consecutive K-rich positions. Lysine is positively charged, and cationic residues are strongly associated with CPP membrane interaction and uptake.

Rule 3 *Predicted: Non-CPP*
$F((\neg \mathtt{R} \wedge \neg \mathtt{K} \wedge \neg \mathtt{P} \wedge \neg \mathtt{L})\, U\, \mathtt{M})$

Explanation: This rule describes a stretch lacking arginine, lysine, proline, and leucine before a methionine. The absence of cationic residues typically important for membrane interaction makes this pattern more consistent with NON-CPP peptides.

ACP740

Rule 1 *Predicted: ACP*
$F(\mathtt{Count(C)} < 3 \wedge \mathtt{Length} < 25 \wedge \mathtt{L}\, U\, \mathtt{P})$

Explanation: This rule describes a short peptide with low cysteine content and a leucine-to-proline motif ($L^{+}P$). Short hydrophobic motifs involving leucine and proline are compatible with membrane-interacting ACPs, while low cysteine content avoids disulfide-rich non-ACP-like patterns, making this consistent with ACP peptides.

Rule 2 *Predicted: ACP*
$F(\mathtt{X}\, \mathtt{Y})$

Explanation: This rule indicates a tyrosine at the next sequence position, effectively capturing a positional Tyr feature. Aromatic residues such as tyrosine can enhance membrane interactions and hydrophobic anchoring, properties often associated with ACP activity.

Rule 3 *Predicted: ACP*
$F(\mathtt{F}\, U\, \mathtt{L} \wedge \mathtt{Length} < 25 \wedge \mathtt{Count(C)} < 4)$

Explanation: This rule captures a short peptide ($\leq$24 residues) with limited cysteine content and a hydrophobic $F^{+}L$ motif. Such short aromatic-hydrophobic motifs are consistent with amphipathic membrane-disruptive ACP peptides.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the contribution as an algorithm for distilling transformer encoders into ITDT/LTL formulas, and the empirical claims are limited to LTL benchmarks and protein sequence datasets evaluated in the paper. The main claims are matched by the formal model, Algorithm 1, and the experiments in Sections 4–6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a limitations discussion that clarifies the scope of the method and its current implementation choices. In particular, it explains the practical use of restricted temporal expansion and situates the method as an interpretable surrogate evaluated on the studied benchmark settings.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical results are stated with explicit assumptions on the alphabet, LTL formulas, temporal depth, and ITDT construction. The paper provides both a proof sketch in the main text and a complete proof in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the ITDT model, the layer-wise distillation procedure, the practical algorithmic choices, and the final classifier in sufficient detail to understand and reproduce the method. Algorithm 1 summarizes the full distillation pipeline. And then we dedicate a full subsection to practical algorithmic choices.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer:[Yes].

   Justification: We provide the full algorithmic description, implementation choices, dataset sources, ablations, and evaluation protocol needed to reproduce the reported results. We also provide publicly avialbale code for a prototype implementing ITDTs.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup specifies the datasets, models, evaluation protocol, ablations, and number of seeds and folds.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main empirical results report mean performance and standard deviations across multiple seeds or folds. This allows readers to assess both average performance and variability across runs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Section 6

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer:[Yes]

   Justification: The work is consistent with the NeurIPS Code of Ethics. It uses public or standard benchmark datasets, does not involve human-subject experiments.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No].

Justification: The paper does not include a dedicated discussion of both positive and negative societal impacts. The work is primarily methodological: it studies how to distill transformer encoders into symbolic ITDT models and evaluates the method on benchmark LTL and protein sequence classification tasks. Its most direct potential impact is positive, namely making transformer predictions more interpretable and easier to inspect. Possible negative impacts are indirect and would arise mainly from downstream use of the original transformer models or from over-trusting distilled rules as faithful explanations. We discuss methodological limitations, including scalability and the surrogate nature of the explanations, but do not provide a broader societal impact analysis.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk model, scraped dataset, generative model, or other asset requiring special misuse safeguards. The work studies offline distillation of existing transformer encoders into interpretable temporal formulas.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes].

Justification: The paper uses existing public assets, including ESM-2-T6 and protein sequence datasets, and cites their original sources. We state the available licenses and terms of use where they could be identified: ESM is released under the MIT license, ToxinPred3.0 under GPL-3.0. For ACP740, we cite the original ACP-DL source but could not identify an explicit license in the public repository. We do not redistribute these datasets.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A].

Justification: The paper does not release new datasets, code packages, or model checkpoints. The paper introduces a method and evaluates it on existing public datasets and pretrained models. We provide public code for the prototype implemented for this work.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing or human subjects research. Only synthetic or existing publicly available datasets are used.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer:[N/A]

    Justification: This paper does not involve human subjects research, and therefore no IRB approval or equivalent review was required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We use an LLM-as-expert only for a post-hoc interpretability analysis of learned ITDT rules. The LLM is given the task description, a learned rule, and the operator semantics, and is asked to identify the class suggested by the rule. It is not used to train ITDTs, tune the method, generate labels, or produce the main quantitative results. We describe the model, its role, the information provided to it, and report the prompt and per-seed results in the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.