Journal of Machine Learning Research 27 (2026) 1-41 Submitted 8/25; Revised 5/26; Published 6/26

Ehrenfeucht-Haussler Rank and Chain of Thought

Pablo Barceló PBARCELO@UC.CL
Institute for Mathematical and Computational Engineering,
Pontifical Catholic University of Chile
& IMFD, Chile
& Centro Nacional de Inteligencia Artificial (CENIA), Chile
Vicuña Mackena 4860, Macul, Santiago, Chile

Alexander Kozachinskiy ALEXANDER.KOZACHINSKYI@CENIA.CL
Centro Nacional de Inteligencia Artificial (CENIA), Chile
Vicuña Mackena 4860, Macul, Santiago, Chile

Tomasz Steifer TSTEIFER@IPPT.PAN.PL
Centre for Credible AI, Warsaw University of Technology
Rektorska 4, Warsaw 00-614, Poland
& Institute of Fundamental Technological Research
Polish Academy of Sciences
Pawińskiego 5B, Warsaw 02-106, Poland

Editor: Benjamin Guedj

Abstract

The notion of *rank* of a Boolean function has been a cornerstone in PAC learning, enabling quasipolynomial-time learning algorithms for polynomial-size decision trees. We present a novel characterization of rank, grounded in the well-known Transformer architecture. We show that the rank of a function f corresponds to the minimum number of *Chain of Thought* (CoT) iterations required by a single-layer Transformer with hard attention to compute f. Based on this characterization, we establish tight bounds on the number of CoT iterations required for specific problems, showing that ℓ-fold function composition necessitates exactly ℓ CoT iterations. Furthermore, we analyze the problem of identifying the position of the k-th occurrence of 1 in a Boolean sequence, proving that it requires k CoT iterations. Finally, we introduce the notion of the multi-head rank that captures multi-head single-layer transformers, and perform the analysis of PAC-learnability of the classes of functions with bounded multi-head rank.

Keywords: transformers, chain of thought, decision-tree rank, pac learning, function composition

1. Introduction

Ehrenfeucht and Haussler (1989) introduced the notion of the *rank* of a Boolean function and showed that, for any constant r, the class of Boolean functions with rank at most r is properly PAC-learnable in polynomial time. As a corollary, they derived their renowned quasipolynomial-time PAC-learning algorithm for polynomial-size decision trees. Pudlák and Impagliazzo (2000) further characterized the rank—not only for Boolean functions but also for Boolean relations—through Prover-Delayer games. Since its introduction, this

concept has played a significant role in proof complexity, see, e.g., (Kullmann, 1999; Esteban and Torán, 2003).

In this paper, we present a new characterization of the notion of rank. Surprisingly, this characterization is grounded in the *Transformer architecture* (Vaswani et al., 2017), which has revolutionized the field of NLP and facilitated the development of LLMs. In essence, we show that the rank of a function f corresponds to the minimum number of *Chain of Thought* (CoT) iterations required by a single-layer Transformer to compute f. The Transformers used in our characterization are based on the *hard attention* mechanism—a theoretical abstraction of the *soft attention* mechanism employed in practice. Hard attention has been widely used in theoretical studies (see, e.g., Hahn (2020); Hao et al. (2022); Barceló et al. (2024); Yang et al. (2024)), due to its amenability to formal analysis, while still effectively capturing the essence of practical models (Clark et al., 2019; Voita et al., 2019).

Transformers are defined as functions mapping finite sequences of *tokens* (elements of a finite set called *vocabulary*) into tokens. In more detail, transformers are built upon *embeddings* and *attention layers*. An embedding maps a token to a vector in a Euclidean space (possibly taking into account the position of the token). The input sequence of tokens is thus transformed into a sequence of vectors. This sequence of vectors is then fed to several attention layers. Each attention layer receives a sequence of vectors and outputs a sequence of vectors of the same length. A layer performs position-wise transformations of vectors via feed-forward neural networks, together with *dot-product* attention carried out by *attention heads*, allowing each vector in the sequence to access other vectors based on their similarity. The final output token is determined by the value of the last vector after all attention layers.

In practice, tokens are relatively short sequences of characters into which prompts to LLMs are split. The goal of an LLM is to produce not just a single token but a sequence of tokens forming a meaningful answer to a prompt. To this end, one usually runs a transformer *auto-regressively*. Namely, given an input sequence of tokens p, one computes the output token of the transformer on p, appends this token to the end of p, and then runs the transformer on the extended sequence, again appending the generated token to the end, and so on.

Motivated by this, we consider the following model for computing functions of the form $f\colon \Sigma^n \to O$ using transformers. An input word $\bar{w} = \sigma_1 \dots \sigma_n$ is given as n tokens, one per letter, together with an additional "end-of-prompt" token, included to equalize the significance of all letters (which we explain at page . The transformer is allowed to auto-regressively generate tokens for several steps, usually referred to as *Chain-of-Thought (CoT) iterations* in the literature on the expressivity of transformers (Merrill and Sabharwal, 2024; Li et al., 2024). The last generated token must be $f(\bar{w})$. For a given function f, we are interested in the minimal number of CoT iterations required to compute f in this model.

In the following, we summarize our results.

- We show that the rank of a function f, denoted by $\mathsf{rk}(f)$, is the minimal number of CoT iterations of a single-layer transformer with one hard-attention head that computes f. We establish our result not only for Boolean functions, generalizing the notion of rank to the non-Boolean case (as far as we know, for the first time).

- In practice, Transformers are equipped with multiple attention heads, which enhance their computational capabilities. We show that the ability of such Transformers to

compute functions can also be characterized using the notion of rank. Specifically, we define the *H-head rank* of a function f, denoted as $\mathsf{rk}^{(H)}(f)$, for $H \geq 1$. We prove that $\mathsf{rk}^{(H)}(f)$ equals the minimum number of CoT iterations required by a single-layer transformer with H hard-attention heads to compute f.

- We then explore methods for obtaining tight bounds on the multi-head rank. We begin by observing that $\mathsf{rk}^{(H)}(f)$ is at most a factor of H smaller than $\mathsf{rk}(f)$. Although computing $\mathsf{rk}(f)$ is typically straightforward, it does not always provide an accurate bound for $\mathsf{rk}^{(H)}(f)$. To address this limitation, we propose a general *communication complexity* lower bound for $\mathsf{rk}^{(H)}(f)$.

- Using this technique, we derive a tight bound on the H-head rank for the *t-fold iterated composition* function. This function was introduced and studied for single-layer soft-attention transformers by Peng et al. (2024). By obtaining lower bounds on this function, they aimed to explain why practical transformer models often hallucinate on prompts that essentially require to perform "function composition" for the answer (for instance, on prompts like "what is the birthday of Frederic Chopin's father?" (Guan et al., 2024) that can be seen as a composition `Birthday(Father(Chopin))`). Formally, the function t-Comp takes as input a sequence of n integers from $\{1, \ldots, n\}$, interpreted as the values of a function $\phi\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$. The output of t-Comp is the value of ϕ, composed with itself t times, evaluated at 1.

  It is easy to see that $\mathsf{rk}(t\text{-}\mathsf{Comp}) \leq t$ for any input length n. A transformer establishing this upper bound works by computing $\phi(1)$ in the first iteration, then $\phi(\phi(1))$ in the second iteration, and so on. We prove that this is optimal even if with more than one attention head. Namely, for any H, we show that $\mathsf{rk}^{(H)}(t\text{-}\mathsf{Comp}) = t$ for all large enough input lengths.

- To illustrate that the communication complexity technique is not universal, we introduce and study the k-thOne function. We formally show that tight bounds, even for the 1-head rank of this function, cannot be obtained via existing communication complexity arguments, and we instead derive them using purely combinatorial arguments involving restrictions of partial inputs. This function takes as input a Boolean sequence of length n and returns the position of the k-th one in it. It is easy to see that $\mathsf{rk}(k\text{-}\mathsf{thOne}) \leq k$ for any input length. In terms of transformers, in the first iteration, we can compute the position of the first one, then the position of the second one in the second iteration, and so on. We prove that for any H and for sufficiently large n, we have $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}) = k$, showing that even by increasing the number of attention heads, we cannot improve upon the trivial solution for sufficiently large input lengths.

- Finally, motivated by the importance of the notion of rank in the theory of PAC learning, we investigate polynomial-time PAC-learnability of classes of functions with bounded multi-head rank. As we have already mentioned, Ehrenfeucht and Haussler established that for any constant r, the class of functions f with $\mathsf{rk}(f) \leq r$ is properly polynomial-time PAC learnable. We extend this result to non-binary alphabets. This implies that for any constant H and r, the class of functions with $\mathsf{rk}^{(H)}(f) \leq r$ is *improperly* polynomial-time PAC learnable as it is a subset of the class of functions with $\mathsf{rk}(f) \leq H \cdot r$.

We complement this observation by showing that the class of functions with *2-head* rank at most 1 is not *properly* polynomial-time PAC-learnable unless NP⊆BPP. This theorem is adjacent to but incomparable with the classical result of Pitt and Valiant (1988) that 2-term DNFs are not properly polynomial-time PAC learnable unless NP⊆BPP (2-term DNFs have 2-head rank at most 1, but not all 2-head rank-1 functions are 2-term DNFs).

Related work. Numerous studies have sought to explore the expressive power of Transformers by treating them as a computational model and investigating what they can compute, see (Pérez et al., 2021; Hao et al., 2022; Yang et al., 2024; Chiang et al., 2023; Merrill and Sabharwal, 2023; Barceló et al., 2024; Merrill and Sabharwal, 2024; Li et al., 2024; Yang and Chiang, 2024; Peng et al., 2024). In particular, several works have investigated how the number of CoT iterations influences the capability of transformers. To start with, Pérez et al. (2021) showed that transformers based on hard attention with an unbounded number of CoT iterations are capable of computing any decidable language (with the parameters of a transformer not depending on the input length). Afterwards, the computational power of transformers with polynomially many CoT iterations was addressed. Merrill and Sabharwal (2024) have shown that in the uniform-regime (when, as in Pérez et al. (2021), parameters do not depend on the input length), such transformers with constant number of layers and softmax attention are capable of computing any polynomial-time language. Similarly, for the non-uniform regime, Li et al. (2024) have shown that such transformers are capable of computing any language recognizable by a polynomial-size family of Boolean circuits.

Our result is the first *exact* characterization of the expressive power of transformers with a given fixed number CoT iterations, although just for a single layer and for hard attention. Recently, Peng et al. (2024) have shown that any single-layer transformer with soft attention requires $\Omega(t)$ CoT iterations to compute t-Comp for $t = \sqrt{n/(dHp)}$, where n is the input length, d is the dimension of vectors, H is the number of attention heads, and p is the number of bits of precision. We point out that our results do not require any assumptions on the dimension and the number of bits of precision.

Organization of the paper. An introduction to decision trees and the notion of rank is found in Section 2, with basic concepts of Transformers being discussed in Section 3. The main results about single-head Transformers are presented in Section 4, with extensions to multi-head Transformers covered in Section 5. Our PAC-learning results can be found in Section 6. Final remarks are given in Section 7.

2. Decision Trees and Rank

Notation We denote $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$.

Decision trees Fix finite sets Σ, O called, respectively, the *input* and the *output* alphabet, and a natural number n called the input length. We are interested in decision trees for functions of the form $f\colon \Sigma^n \to O$. More specifically, we consider decision trees over arbitrary families of *queries*, where a query is a function q whose domain is Σ^n. We write $\mathrm{Im}(q)$ for the image of a query q. If $\mathcal{F}$ is a set of queries, a decision tree over $\mathcal{F}$ is a rooted tree T such that:

- Every non-leaf node v is labeled by some query $q_v \in \mathcal{F}$ and has exactly $|\mathrm{Im}(q_v)|$ outgoing edges, each one of them labeled by a different element from $\mathrm{Im}(q_v)$.

- Every leaf ℓ is labeled by some element $o_\ell \in O$.

Given an input $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$, the output of the decision tree T on $\bar{w}$ is computed by descending from the root to one of the leaves. At each intermediate non-leaf node v, the tree computes the value $q_v(\bar{w}) \in \mathrm{Im}(q_v)$ and descends to the unique child of v that is linked to v through an edge labeled $q_v(\bar{w})$. In this way, we reach some leaf ℓ, where T outputs the element o_ℓ as its result on $\bar{w}$. We denote this output as $T(\bar{w})$.

The function $f : \Sigma^n \to O$ is *computed* by T if $T(\bar{w}) = f(\bar{w})$ for every input $\bar{w} \in \Sigma^n$.

Remark 1. *If we place no restriction on the queries allowed at internal nodes, then decision trees become trivial: every function f can be computed by a decision tree of depth 1, where the query in root is just this function f.*

Example 1. Let $\Sigma = \{0, 1, 2\}$ and consider the function $f : \Sigma^2 \to \{0, 1\}$ defined by

$$f(w_1, w_2) = 1 \quad \text{iff} \quad w_1 = 2 \text{ or } (w_1 \neq 2 \text{ and } w_2 = 1).$$

This function can be computed by a decision tree that requires two queries: (1) a query that asks whether the value of w_1 is 2 and responds YES if this is the case, and NO otherwise; and (2) a query that asks whether the value of w_2 is 1 and responds YES if this is the case, and NO otherwise. At the root, we test whether $w_1 = 2$ using the first query. If the answer is YES, we immediately output 1. Otherwise, we proceed to a second test that checks whether $w_2 = 1$. If the answer is YES, we output 1; if not, we output 0. □

Rank in the Boolean case. Decision trees are often defined for *Boolean* functions, i.e., functions of the form $f\colon \{0,1\}^n \to \{0,1\}$. In our notation, this corresponds to the case $\Sigma = O = \{0, 1\}$. *Boolean decision trees* are decision trees over a family $\{p_1, \ldots, p_n\}$ of queries, where for $i = 1, \ldots, n$ the function $p_i\colon \{0,1\}^n \to \{0,1\}$ is defined as follows on input $(b_1, \ldots, b_n) \in \{0,1\}^n$:

$$p_i(b_1, \ldots, b_n) = b_i.$$

That is, at every node, a Boolean decision tree queries the value of some coordinate of the input.

Ehrenfeucht and Haussler (1989) defined the *rank* of a Boolean decision tree T by inductively defining the rank of its nodes as follows:

- the rank of a leaf is 0, and

- the rank of a non-leaf v, whose two children have ranks r_0, r_1, is $r = \max\{\min\{r_0, r_1\} + 1, \max\{r_0, r_1\}\}$.

The rank of T is then the rank of its root, and the rank of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is the minimum rank of a Boolean decision tree that computes f.

Rank in the non-boolean case and a-queries. We extend the notion of rank to the non-Boolean case through decision trees over *assignment queries*. We start by introducing some terminology. Pairs of the form (i, σ), where $i \in [n]$ and $\sigma \in \Sigma$, are called *assignments*. We denote by $A = [n] \times \Sigma$ the set of assignments. An assignment (i, σ) is *consistent* with an input $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$ if and only if $\sigma_i = \sigma$. By a permutation of a finite set B we mean a bijection $\tau \colon \{1, \ldots, |B|\} \to B$.

An assignment query (*a-query* from now on) is a function of the form $q_\tau : \Sigma^n \to A$, where τ is a permutation of the set of assignments A. For $\bar{w} \in \Sigma^n$, we define $q_\tau(\bar{w})$ as follows: we let $k_{\bar{w}}$ be the minimal element $k \in \{1, \ldots, |A|\}$ such that $\tau(k)$ is consistent with $\bar{w}$, and then set $q_\tau(\bar{w}) = \tau(k_{\bar{w}})$.

It is sometimes convenient to view the computation of an a-query q_τ on an input $\bar{w}$ as follows. Assume that $\tau(j) = (i_j, \sigma_j)$, for each $j = 1, \ldots, |A|$. Imagine that we do not know $\bar{w}$, and we start asking a person who knows $\bar{w}$ questions: "is the i_1-th letter of $\bar{w}$ equal to σ_1?", 'is the i_2-th letter of $\bar{w}$ equal to σ_2?", and so on. We stop once we receive the first YES answer. If this happens at the kth step, we return $q_\tau(\bar{w}) = (i_k, \sigma_k)$.

Example 2. Let $\Sigma = \{0, 1, 2\}$ and $n = 2$, and consider the following permutation τ of the set A of assignments:

$$\tau(1) = (2, 1),\ \tau(2) = (1, 1),\ \tau(3) = (2, 0),\ \tau(4) = (2, 2),\ \tau(5) = (1, 2),\ \tau(6) = (1, 0).$$

Given an input $\bar{w} = (w_1, w_2) \in \Sigma^2$, the a-query q_τ first finds the minimal element k such that $\tau(k)$ is consistent with $\bar{w}$, and then returns $\tau(k)$. For instance, if $\bar{w} = (2, 1)$, then $k = 1$ as $\bar{w}$ is consistent with $(2, 1)$; if $\bar{w} = (0, 0)$, then $k = 3$ since $\bar{w}$ is consistent with $(2, 0)$ but not with $(2, 1)$ and $(1, 1)$. □

We define the rank of an arbitrary function $f : \Sigma^n \to O$ in terms of the class of decision trees over assignment queries that compute f.

Definition 2. *Let $f : \Sigma^n \to O$. We define $\mathsf{rk}(f)$ as the minimal depth of a decision tree over a-queries that computes f.* □

Example 3. Let us go back to the function $f : \{0, 1, 2\}^2 \to \{0, 1\}$ shown in Example 1. We have seen that there is a decision tree of depth 2 that computes f. Thus, the rank of this function is at most 2. □

In the non-Boolean case, one might ask whether, at a given node, the list of tested assignments must include values that have already been ruled out by decisions made higher up in the tree. This is not necessary. Once a value of a variable has been excluded by an ancestor edge, it can be removed from all descendant lists without affecting correctness. Hence, without loss of generality, lists at each node may be assumed to contain only assignments consistent with the decisions made along the path from the root.

Equivalence of notions As we show below, the notion of rank we have just introduced for arbitrary functions aligns, in the case of Boolean functions, with the definition we previously provided for that class of functions.

Proposition 3. *For any Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, its rank, as defined by Ehrenfeucht and Haussler, is equal to $\mathsf{rk}(f)$.*

Proof *(a-query depth $\leq$ rank)* Assume first that $f : \{0,1\}^n \to \{0,1\}$ can be computed by a Boolean decision tree T of rank r. We convert T into a depth-r decision tree $\hat{T}$ over a-queries that also computes f. To do this, we design an inductive strategy based on a-queries such that, for every $t = 0, \ldots, r$ and for every input $\bar{w} \in \{0,1\}^n$, the following holds: after asking t a-queries on input $\bar{w}$, we can compute a node v_t of T of rank at most $r - t$ such that T falls into v_t on $\bar{w}$ after the first t queries. With this knowledge, we can easily build $\hat{T}$: after r a-queries the strategy gets us to a node v_r of rank 0 to which we arrive by evaluating $\bar{w}$ on T. The node v_r has to be a leaf where the value $f(\bar{w})$ is written.

The condition for $t = 0$ is fulfilled with v_0 being the root of T. It remains to explain how, knowing v_t, we can compute v_{t+1} at the cost of a single a-query. We assume that v_t is not a leaf as otherwise we can simply set $v_{t+1} = v_t$. By definition of rank, every non-leaf node has a child of a smaller rank. Consider a mapping ϕ that, to every non-leaf node v of T, assigns a child of v such that the *other* child of v has a smaller rank than v. Call $\phi(v)$ the *elderly* child of v.

Set $u_1 = v_t$ and consider a sequence of $u_1, \ldots, u_d$ of nodes of T where u_ℓ is the elderly child of $u_{\ell-1}$ for $\ell = 2, \ldots, d$ and u_d is a leaf. For each $\ell = 1, \ldots, d$, assume that the node u_ℓ is labeled with the query p_{i_ℓ}, for $i_\ell \in \{1, \ldots, n\}$, i.e., this node asks for the i_ℓ-th value of the input. Further, let $b_\ell \in \{0,1\}$ be the label of the edge from u_ℓ to $u_{\ell+1}$ for $\ell = 1, \ldots, d-1$. Without loss of generality, $i_1, \ldots, i_{d-1}$ are distinct (we may assume that we do not ask the value in the same position twice on the same path, otherwise the number of nodes in the tree can be reduced without increasing its rank).

Define τ to be any permutation of the set of assignments such that $\tau(1) = (i_1, 1 - b_1), \ldots, \tau(d-1) = (i_{d-1}, 1 - b_{d-1})$. We claim that, after getting the value of $q_\tau(\bar{w})$, we are able to find a node v_{t+1} whose rank is smaller than v_t such that T goes through v_{t+1} when processing $\bar{w}$. Namely, $q_\tau(\bar{w}) = \tau(k)$ for the minimal k such that $\tau(k)$ is consistent with $\bar{w}$. If $k \leq d-1$, this means that assignments $(i_1, 1-b_1), \ldots, (i_{k-1}, 1-b_{k-1})$ are inconsistent with $\bar{w}$, while $(i_k, 1 - b_k)$ is consistent. Hence, $\bar{w}$ has values $b_1, \ldots, b_{k-1}$ at positions $i_1, \ldots, i_{k-1}$, respectively, and $1 - b_k$ at position i_k. It means that we descend on T from $v_t = u_1$ to u_k while processing input $\bar{w}$, from where we then move to the non-elderly child of u_k that has a smaller rank than u_k, and, hence, than $u_1 = v_t$. We set v_{t+1} to be the non-elderly child of u_k. Now, if $k \geq d$, then when reading $\bar{w}$ on T we arrive at the leaf u_d, which we set to be v_{t+1} in this case.

(rank $\leq$ a-query depth) We show that a Boolean function $f : \{0,1\}^n \to \{0,1\}$, computable by an r-depth decision tree $\widehat{T}$ over a-queries, has rank at most r. We first convert $\widehat{T}$ into a so-called *YES-NO* decision tree for f. By a YES-NO decision tree we mean a binary rooted tree, where:

- every non-leaf node v is labeled with an assignment (i_v, σ_v), and has one outgoing edge labeled by YES and the other one by NO; and
- every leaf ℓ is labeled with a bit $o_\ell \in \{0,1\}$.

Given an input $\bar{w} = (b_1, \ldots, b_n) \in \{0,1\}^n$, the output of the decision tree T on $\bar{w}$ is computed by descending from the root to one of the leaves. At each intermediate non-leaf node v, the tree compares the i_v-th position of $\bar{w}$ with σ_v. If they coincide, we descend through the YES-labeled edge; if they differ, we descend through the NO-labeled edge. Once we arrive

to a leaf ℓ, we output $o_\ell \in \{0,1\}$. We define the *YES-depth* of a node v of a YES-NO tree as the maximal number of YES-labeled edges on a path from v to a leaf.

Notice that the notion of yes-depth is inherently asymmetric. While a bound on YES-depth limits the number of successful ("yes") tests along any root-to-leaf path, the shape of the tree under yes-branches may be significantly shallower than under no-branches. In particular, a computation may terminate immediately after a successful test, whereas failed tests may lead to further queries.

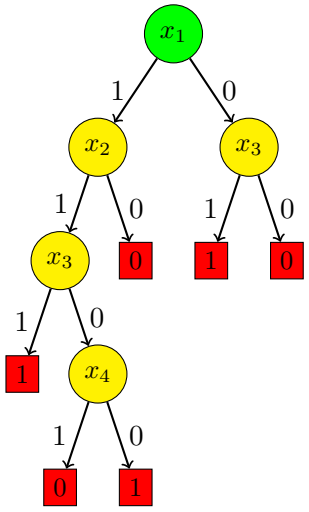
We start by converting our r-depth decision tree $\widehat{T}$ over a-queries into a YES-NO decision tree T for f where the root has YES-depth at most r. In other words, we have to give a way of computing $f(\bar{w})$ by asking questions of the form "is the i-th position of $\bar{w}$ equal to b?", for $i \in \{1,\dots,n\}$ and $b \in \{0,1\}$, and outputting the answer after at most r answers YES. This can be done by noticing that the value of any a-query can be computed in this model after one YES answer. Indeed, a question "is the i-th position of $\bar{w}$ equal to b?" is equivalent to a question "is the assignment (i,b) consistent with $\bar{w}$?" Now, if we want to compute the value $q_\tau(\bar{w})$ for a permutation τ of the set of assignments τ, we start asking questions "is $\tau(1)$ consistent with $\bar{w}$?", "is $\tau(2)$ consistent with $\bar{w}$?", and so on. The first assignment for which we receive a YES is $q_\tau(\bar{w})$.

We now convert the YES-NO tree T into a Boolean decision tree for f that has rank at most r. Namely, for every inner node v that is labeled by an assignment (i_v, b_v), we re-label v by a position i_v, and we label the YES-outgoing edge with b_v, and the NO-outgoing edge with $1 - b_v$. To finalize, we show by induction on the depth of a node that the rank of any node of T is upper bounded by its YES-depth. Any leaf has both YES-depth and rank 0, so the induction base trivially holds. Consider now any node v with two children v_0, v_1 whose ranks r_0, r_1 are upper bounded by the YES-depths of v_0, v_1, respectively. We establish that the rank $r = \max\{\max\{r_0, r_1\}, \min\{r_0, r_1\} + 1\}$ of v is also upper bounded by its YES-depth. The YES depth of v upper bounds the YES depths of both its children, and hence, upper bounds $\max\{r_0, r_1\}$. At the same time, the YES depth of v is at least 1 plus the YES-depth of the child to which the YES-edge points from v, which is at least $1 + \min\{r_0, r_1\}$. ■

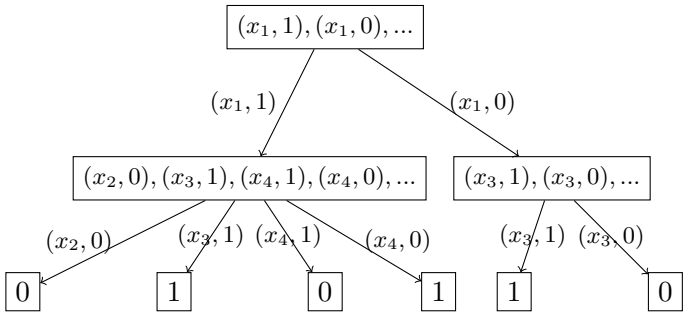
Example 4. In Figure 1, we illustrate Proposition 3 by showing how a Boolean decision tree of rank 2 (according to the Ehrenfeucht-Haussler definition) can be converted into a depth-2 decision tree over a-queries.

At the root of the tree on the right of Figure 1, we have an a-query that starts with assignment $(x_1, 1)$, and then has assignment $(x_1, 0)$ (the ordering of the remaining assignments is irrelevant). The answer to this a-query will reveal the value of variable x_1.

The crux of the construction is contained in the case $x_1 = 1$, when we move to the left child of the root in both trees. The value of the a-query at this point must determine in which leaf we end up in the left tree. Thus, assignments must be put in the order that corresponds to "exits" from the yellow subpath.



(a) A Boolean decision tree of rank 2. Rank 0 = ■, rank 1 = ■, rank 2 = ■.



(b) Equivalent depth-2 decision tree over a-queries.

Figure 1: Transforming a Boolean decision tree into a decision tree over a-queries, illustrating Proposition 3.

An example: Iterated composition We consider the *iterated composition function*. For positive integer numbers t, n, we define:

$$t\text{-}\mathsf{Comp}_n \colon [n]^n \to [n],$$
$$t\text{-}\mathsf{Comp}_n \colon (f(1),\ldots,f(n)) \mapsto \underbrace{f(f(\ldots f}_{t \text{ times}}(1))).$$

A clarification for the second line: an input to t-Comp_n is an n-length word, where every letter is a number from 1 to n. This input is interpreted as a function $f \colon [n] \to [n]$, with $f(1)$ being the first letter of the word, $f(2)$ being the second letter of the word, and so on. Sometimes, we also use the following notation:

$$f^{(\ell)} = \underbrace{f \circ f \circ \ldots \circ f}_{\ell \text{ times}}.$$

In particular, we let $f^{(0)}$ be the identity function.

We claim that the rank of t-Comp_n does not exceed t. Recall that the input is interpreted as a word $(f(1),\ldots,f(n))$, for some $f \colon [n] \to [n]$, and our task is to compute $f^{(t)}(1)$. Consider a decision tree that first tries to guess the value of the first letter, that is, of $f(1)$ by going "is $f(1) = 1$?", "is $f(1) = 2$?", and so on. Once the tree gets it right, receiving the first YES-answer, it already knows $f(1)$, and now it starts guessing the $f(1)$st letter, that is, $f^{(2)}(1) = f(f(1))$. It costs the second YES-answer to get it right. Continuing in this way, the tree will find out $f^{(t)}(1)$ after t YES-answers.

To make the construction more concrete, consider the first step of the rank-t strategy. The list at the root consists of all assignments to the first block of variables that make the

first component $f(1)$ equal to 1, in an arbitrary order. Since exactly one such assignment is consistent with the input, the first successful entry in this list determines the value of $f(1)$. The precise ordering of the remaining entries in the list is irrelevant, as the computation always halts at the first successful test. Subsequent steps proceed analogously for $f(2), \ldots, f(t)$.

By means of a combinatorial argument, it is possible to show that this is the best one can do if n is large enough. The purpose of the following lower bound is to show that the upper bound obtained above is essentially tight: even for this simple and highly structured function, no decision tree of a smaller rank can compute it.

Proposition 4. *For any t and for all $n > 2t$, we have $\mathsf{rk}(t\text{-}\mathsf{Comp}_n) = t$.*

Proof Assume, for a contradiction, that we have a decision tree T of depth $t-1$ over a-queries for t-Comp_n, for some $n > 2t$. We start answering questions for T, descending to one of its leaves, in the following manner. We maintain a set $F \subseteq [n]$ of "forbidden numbers". Initially, $F = \{1\}$. When we receive an a-query with a permutation τ of assignments, we select the first assignment (i, j) such that $j \notin F$ and $f(i)$ is not fixed yet. We fix $f(i) = j$ and continue along the tree as if this were the first consistent assignment. After that, we put j into F. Note that after k values of f have been fixed this way, F consists of precisely $k+1$ distinct elements. Indeed, every a-query we consider adds exactly one new element to F.

Let ℓ denote the leaf of T where we come in this way by answering a-queries. Suppose that $o_\ell \in [n]$ is the value that T outputs in this leaf. We obtain a contradiction by showing that some function $g\colon [n] \to [n]$ with $g^{(t)}(1) \neq o_\ell$ also gets to ℓ.

Observe that, since T is of depth $t-1$, there are $k \leq t-1$ a-queries on the path to ℓ and the same number of values of f have been fixed:

$$f(i_1) = j_1,\ f(i_2) = j_2, \ldots, f(i_k) = j_k. \tag{1}$$

Note that $i_1, \ldots, i_k$ are distinct because we never fix the same value twice. Numbers $j_1, \ldots, j_k$ are distinct too, and they define the evolution of the set F. Initially, $F = \{1\}$ after the first a-query, $F = \{1, j_1\}$ after the second a-query, $F = \{1, j_1, j_2\}$ after the third one, and so on.

Take any $y \in [n] \setminus \{1, i_1, \ldots, i_k, j_1, \ldots, j_k, o_\ell\}$ (it exists because $n > 2t \geq 2(k+1)$). Define a function $g\colon [n] \to [n]$ by

$$\begin{aligned} g(i_1) = j_1, \ldots, g(i_k) = j_k,\\ g(x) = y \text{ for } x \in [n] \setminus \{i_1, \ldots, i_k\}. \end{aligned}$$

We first show that g arrives to ℓ in T. For that, we show that g is consistent with all answers to questions on the path to ℓ. All the assignments corresponding to our answers to a-queries on the path to ℓ are as in (1), and g is consistent with all of them by definition. Next, take an assignment (i, j) and suppose it appears at the m-th a-query along the path to ℓ, ordered before the assignment (i_m, j_m) (which we chose to be the first consistent one). Hence, in our descent along the tree, we ignored this assignment and decided to fix the assignment (i_m, j_m) instead. Hence, we need to observe that g is not consistent with it, that is, that $g(i) \neq j$. Indeed, we could have ignored (i, j) in two cases. Firstly, it could have happened

that $g(i)$ was already fixed to some value different from j. Secondly, we could have ignored it when $g(i)$ was not yet fixed, because j already belonged to the set of forbidden numbers F. But by the definition of g, that means that either $g(i) = y$ or $g(i) = j_s$ for some $s > m$. The first case is not possible since y was chosen to be outside of F, and the second case gives us $g(i) \neq j$.

To finish the proof, we show that $g^{(t)}(1) = y$. Consider a directed graph with vertex set $\{1, \ldots, n\}$, where for every $i \in \{1, \ldots, n\}$ there is a directed edge from i to $g(i)$. The image of the function g consists of $j_1, \ldots, j_k$ and y. In the graph, these are the only nodes with incoming edges. Observe that each of $j_1, \ldots, j_k$ has exactly one incoming edge. Namely, for $s = 1, \ldots, k$, the node j_s has a unique incoming edge from i_s. To compute $g^{(t)}(1)$, we start moving from 1 along the edges for t steps. We will be moving over $j_1, \ldots, j_k$ and y. Note that $g(y) = y$ because $y \notin \{i_1, \ldots, i_k\}$. Hence, it is enough to show that y is reached from 1 in *at most* t steps because then we stay at y forever. Now, if we do not reach y within the first t steps, then we travel over $j_1, \ldots, j_k$ for t steps. Since $k \leq t - 1$, it means that we come into some of $j_1, \ldots, j_k$ two times, but this would mean that one of them has two distinct incoming edges, which is impossible. ■

An example: Position of the k-th one. We define a function $k\text{-}\mathsf{thOne}_n \colon \{0,1\}^n \to [n+1]$ such that:

$$k\text{-}\mathsf{thOne}_n(\sigma_1, \ldots, \sigma_n) = \min\left(\{n+1\} \cup \{i \in [n] : \sigma_1 + \ldots + \sigma_i = k\}\right).$$

In other words, given $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \{0,1\}^n$, the function $k\text{-}\mathsf{thOne}_n$ returns the position of the k-th one in $\bar{w}$ (counting from the left). If there are fewer than k ones in $\bar{w}$, we return $n+1$. We can then show the following by means of a combinatorial argument:

Proposition 5. *For any n, k, we have $\mathsf{rk}(k\text{-}\mathsf{thOne}_n) \leq k$, and for $n \geq 2k-1$, we have $\mathsf{rk}(k\text{-}\mathsf{thOne}_n) = k$.*

Proof We first establish the upper bound on the rank. We start by computing the position of the first one using one a-query. Namely, we ask an a-query, defined by the permutation that is associated with the following ordering of the set of assignments:

$$\tau = (1,1), (2,1), \ldots, (n,1), (1,0), \ldots, (n,0).$$

If there is at least a 1 in the output, this a-query returns an assignment $(i_1, 1)$ with i_1 being the position of the first one. If there are no ones in the input, the a-query returns the assignment $(1,0)$, in which case we can already output $n+1$. Having the position i_1 where the first 1 is found, we compute the position of the second 1 by asking an a-query defined by the following ordering of the set of assignments:

$$\tau_{i_1} = (i_1+1, 1), \ldots, (n,1), (1,0), \ldots, (n,0), (1,1), \ldots, (i_1, 1).$$

If it returns an assignment $(i_2, 1)$ for $i_2 > i_1$, then the number i_2 is the position of the second 1. If it returns an assignment with value 0, then after position i_1 there are no ones, which already allows us to output $n+1$. Continuing similarly, we compute the position of the k-th 1 with k a-queries (or find out that there are fewer than k 1s in the input).

We now establish our lower bound on the rank. Assume for contradiction that for some n, k, d with $n \geq 2k-1$ and $k > d$, there exists a depth-d decision tree T over a-queries that computes k-thOne$_n$. We identify $m \leq d$ positions in $[n]$, along with a specific fixing of Boolean values for these positions, such that all inputs matching these values at those positions arrive at the same leaf ℓ in T.

Consider the permutation of assignments for the a-query asked at the root. Let (i_1, σ_1) be the first assignment in this permutation. We fix the value of the i_1-th position to σ_1, and descend from the root by the (i_1, σ_1)-labeled edge. We end up in some child of the root. Let (i_2, σ_2) be the first assignment in the permutation at this child. We fix the i_2-th position to σ_2 unless it contradicts that first fixation, i.e., unless $i_1 = i_2$ and $\sigma_1 \neq \sigma_2$. In the latter case, we take the second assignment in the permutation as (i_2, σ_2). Proceeding in this way, we come up with $m < d$ numbers $i_1, \ldots, i_m \in [n]$ and m values $\sigma_1, \ldots, \sigma_m \in \{0,1\}$, such that all $\bar{w} \in \{0,1\}^n$ satisfying:

$$\bar{w}_{i_1} = \sigma_1, \ldots, \ \bar{w}_{i_m} = \sigma_m, \tag{2}$$

come to the same leaf ℓ of T.

We obtain our desired contradiction by showing that there are two inputs satisfying (2) with different values of k-thOne$_n$. Indeed, we have $m \leq d \leq k-1$ fixed positions, and hence there are at least $n - (k-1) \geq k$ unfixed positions. If we fix all of the k unfixed positions to 0, we will have fewer than k ones, and the output of k-thOne$_n$ will be $n+1$. In turn, if we fix all of them to 1, we will have at least k ones, and the output of k-thOne$_n$ will be a number from 1 to n. ■

3. Transformers

Attention layers. The principal component of transformers (**?**) are *attention layers*. They are length-preserving transformations of sequences of vectors. We consider layers with the *unique hard-attention mechanism* (Hahn, 2020).

Definition 6. *A d-dimensional H-head UHAL (unique hard-attention layer) is a function $L\colon (\mathbb{R}^d)^* \to (\mathbb{R}^d)^*$, given by*

- *H* **query** *matrices $Q^{(k)} \in \mathbb{R}^{d\times d}$, H* **key** *matrices $K^{(k)} \in \mathbb{R}^{d\times d}$, and H* **value** *matrices $V^{(k)}$ for $k = 1, \ldots, H$,*
- *a matrix $W_O \in \mathbb{R}^{d\times(dH)}$ (parameters of the linear transformation, mixing vectors from different heads).*
- *two matrices $W_1, W_2 \in \mathbb{R}^{d\times d}$ and two vectors $b_1, b_2 \in \mathbb{R}^d$ (parameters of the position-wise feed-forward network at the end of the layer).*

Given an input sequence of vectors $(x_1, \ldots, x_m) \in (\mathbb{R}^d)^m$, the output sequence of vectors $(y_1, \ldots, y_m) = L(x_1, \ldots, x_m) \in (\mathbb{R}^d)^m$ is computed in the following steps:

1. *for each $k = 1, \ldots, H$, and for $i, j = 1, \ldots, m$, compute the* ***attention weight*** *of the j-th position to the i-th position in the k-th head as:*

$$L_{ij}^{(k)} = \langle K^{(k)} x_i, Q^{(k)} x_j \rangle \tag{3}$$

2. *for each head* $k = 1, \ldots, H$ *and position* $j = 1, \ldots, m$*, define:*

$$i_j^{(k)} = \min\{i \in \{1, \ldots, j\} \mid L_{ij}^{(k)} \geq L_{i'j}^{(k)} \ \forall i' \in \{1, \ldots, j\}\}; \tag{4}$$

(this is the position with the maximal attention weight from the j*-th position in the* k*-th head, among positions in* $\{1, \ldots, j\}$*; if there are multiple such positions,* $i_j^{(k)}$ *is set to be the left-most one).*

3. *for each head* $k = 1, \ldots, H$ *and position* $j = 1, \ldots, m$*, define the* ***value of the*** k***-th head at the*** j***-th position*** *as the input vector in the* $i_j^{(k)}$*-th position, multiplied by the* k*-th value matrix:*

$$h_j^{(k)} = V^{(k)} x_{i_j^{(k)}} \in \mathbb{R}^d; \tag{5}$$

4. *for each position* $j = 1, \ldots, m$*, combine values of all the heads in this position via:*

$$h_j = W_O \begin{pmatrix} h_j^{(1)} \\ h_j^{(2)} \\ \vdots \\ h_j^{(H)} \end{pmatrix}; \tag{6}$$

5. *finally, for* $j = 1, \ldots, m$ *define the output in the* j*-th position as the result of applying a feed-forward network, given by matrices* W_1, W_2 *and bias vectors* b_1, b_2*, to* $h_j + x_j$*.*

$$y_j = W_2 \cdot \mathrm{ReLU}\left(W_1(h_j + x_j) + b_1\right) + b_2 \in \mathbb{R}^d. \tag{7}$$

Recall that $\mathrm{ReLU}(x) = \max\{0, x\}$ *for* $x \in \mathbb{R}$*, and if* $x \in \mathbb{R}^d$ *then* $\mathrm{ReLU}(x) \in \mathbb{R}^d$ *is obtained by applying* ReLU *to each one of the components of* x*.*

Let us comment on some choices in this definition. To start off, we assume that *left-most* tie-breaking rule in (3). However, a specific tie-breaking rule will not be important for us as we will make sure that there will be no ties in our constructions (this can always be achieved for UHALs with the use of positional encodings (Hao et al., 2022)).

In (4), we assume the *causal mask*, which means that only positions to the left of j, and j itself, are taken into account in the attention. One could also consider the full attention model, where the maximum of the attention is taken over all m positions for every j; however, this would not affect our results. This is because we consider just one-layer transformers with the output computed in the last token, where there is no difference between causally masked attention and full attention. Sometimes, the strict causal masking (where at position j, the attention to j itself is not taken into the account) is considered in theoretical works, and it can make the difference for the transformer expressivity in case of the absence of the positional encoding (Yang et al., 2024). Nevertheless, all our results hold for the strict causal masking as well.

Transformers. In practice, transformers are defined as functions from sequences of *tokens* (elements of some finite alphabet) into probability distributions of tokens. Given a *prompt* (an input sequence of tokens), a transformer computes a distribution, and then generates a token from it. We consider a deterministic version of transformers where, instead of generation, we simply take as an output the token with the maximal probability.

Definition 7. *An C-layer H-head d-dimensional UHAT (unique hard-attention transformer) over a finite set $\mathcal{V}$ (a "vocabulary" whose elements are called "tokens"), containing a special symbol $\perp$, is a function $U\colon \mathcal{V}^* \to \mathcal{V}$, given by*

- *C H-head d-dimensional UHALs $L_1, \ldots, L_C$;*
- *the input embedding $\mathsf{E}\colon \mathcal{V} \times \mathbb{N} \to \mathbb{R}^d$;*
- *the **output distribution** matrix $W \in \mathbb{R}^{\mathcal{V} \times d}$ (transforming d-dimensional vectors into vectors whose coordinates are indexed by tokens).*

Given a sequence of tokens $t_1 \ldots t_m \in \mathcal{V}^m$, the output token $t = U(t_1 \ldots t_m)$ is computed in the following steps:

1. *we transform the input sequence of tokens into a sequence of vectors via the input embedding:*
$$x_1 = \mathrm{E}(t_1, 1), \ldots, x_m = \mathrm{E}(t_m, m); \tag{8}$$
2. *we apply the sequence of attention layers to this sequence:*
$$(y_1, \ldots, y_m) = L_C \circ \ldots \circ L_1(x_1, \ldots, x_m); \tag{9}$$
3. *we transform the vector in the last position after the last layer into a vector in $\mathbb{R}^{\mathcal{V}}$:*
$$\mu = W y_m \tag{10}$$
4. *finally, we define:*
$$t = U(t_1, \ldots, t_m) := \arg\max_{t \in \mathcal{V}} \mu_t. \tag{11}$$
If there are multiple tokens, attaining the maximum, we set $U(t_1, \ldots, t_m) = \perp$.

In practice, the output probability distribution P over the set of tokens is obtained by applying the so-called *softmax* to the vector μ:

$$P(t) = \frac{\exp\{\mu_t/\tau\}}{\sum_{v \in \mathcal{V}} \exp\{\mu_v/\tau\}}, \qquad t \in \mathcal{V},$$

where $\tau > 0$ is a number called the *temperature*. When $\tau \to 0$, we have that $P(t) \to 1$ for the token $t \in \mathcal{V}$ with maximal value of μ_t, and $P(v) \to 0$ for the rest of the tokens $v \in \mathcal{V}$, effectively giving us the output rule as in (11). The only difference is that when there are multiple tokens, achieving the maximum of μ_t, the distribution P converges to the uniform distribution over these tokens as $\tau \to 0$, while we leave the output of the transformer to be undefined in this case.

Chain-of-thought depth Given a UHAT $U\colon \mathcal{V}^* \to \mathcal{V}$ over a vocabulary $\mathcal{V}$, we define its r-th chain-of-thought iteration as a function $U^{(r)}\colon \mathcal{V}^* \to \mathcal{V}$, computed as follows on an input $p \in \mathcal{V}^*$:

$$\begin{aligned} t_1 &= U(p) \\ t_2 &= U(pt_1) \\ &\vdots \\ t_r &= U(pt_1 \dots t_{r-1}) \\ U^{(r)}(p) &:= t_r. \end{aligned}$$

Definition 8. *Let $f\colon \Sigma^n \to O$ be a function and let $H \in \mathbb{N}$. The H-head chain-of-thought depth of f, denoted by $\mathsf{cotd}^{(H)}(f)$ is the minimial r for which there exists a **one-layer** H-head UHAT U such that*

- *its vocabulary $\mathcal{V}$ is such that $\Sigma \cup O \cup \{\mathsf{EoL}\} \subseteq \mathcal{V}$, where $\mathsf{EoL} \notin \Sigma \cup O$ is a special "end-of-line" symbol;*
- *for any $\bar{w} \in \Sigma^n$, we have $U^{(r)}(\bar{w}\mathsf{EoL}) = f(\bar{w})$.*

The EoL token is added because otherwise the last letter of the input $w = \sigma_1 \dots \sigma_n$ would have been more significant than others; indeed, it would have belonged to the last token where the output of the first CoT iteration is computed.

The CoT depth could have been defined for each number of layers, but in this paper, we require this notion only for single-layer transformers.

4. One-head COT depth vs Tree Rank

In this section, we show that the rank of a function is equivalent to its chain-of-thought depth in the single-head setting.

Theorem 9. *For any function $f\colon \Sigma^n \to O$, we have $\mathsf{rk}(f) = \mathrm{cotd}^{(1)}(f)$.*

As a corollary to Theorem 9 and Proposition 4, we obtain that for suitable n the CoT depth with one head of the iterated composition function $t\text{-}\mathsf{Comp}_n$ is precisely t:

Corollary 10. *For each t and for all $n > 2t$, we have $\mathrm{cotd}^{(1)}(t\text{-}\mathsf{Comp}_n) = t$.*

Also, as a corollary to Theorem 9 and Proposition 5, we obtain that for suitable n the CoT depth with one head of the kth one function $k\text{-}\mathsf{thOne}_n$ is precisely k:

Corollary 11. *For each k, and for every $n \geq 2k-1$, we have $\mathrm{cotd}^{(1)}(k\text{-}\mathsf{thOne}_n) = k$.*

Before proving the main theorem, we provide an intuitive explanation of why the equivalence between rank and CoT depth holds. A single step of a hard-attention transformer operates as follows: given the current sequence of vectors, the attention mechanism assigns a score to each position and selects the position with the maximum score. Thus, one iteration can extract at most one distinguished assignment from a list of candidates — namely, the first (under the induced ordering) among those that are present in the input. This

is precisely the behavior of an a-query in the definition of rank: an a-query consists of an ordered list of assignments, and on input $\bar{x}$ it returns the first assignment in the list that is consistent with $\bar{x}$. In both models, the computation proceeds by a sequence of such "first-success" selections. Consequently, each CoT iteration corresponds to one a-query, and a computation requiring k such sequential selections corresponds exactly to rank k. The equivalence theorem formalizes this structural correspondence. We now prove Theorem 9.

Proof [Theorem 9] We prove both directions separately.

(From transformers to decision trees.) We first show the inequality $\mathsf{rk}(f) \leq \mathrm{cotd}^{(1)}(f)$. Assume that f can be computed by r COT iterations of a 1-layer 1-head UHAT U, for some $r \in \mathbb{N}$, that is,

$$f(\bar{w}) = U^{(r)}(\bar{w}\mathsf{EoL})$$

for all $\bar{w} \in \Sigma^n$. We derive from this $\mathsf{rk}(f) \leq r$.

The proof is based on an observation that the output of a 1-layer 1-head UHAT, for a fixed input length and fixed token in the last position, can be computed in 1 a-query.

Lemma 12. *Let U be a 1-layer 1-head UHAT over the vocabulary $\mathcal{V}$. Then for any $n \in \mathbb{N}$ and $t \in \mathcal{V}$ we have $\mathsf{rk}(U_n^t) \leq 1$, where $U_n^t \colon \mathcal{V}^n \to \mathcal{V}$ is defined by:*

$$U_n^t(t_1 \dots t_n) = U(t_1 \dots t_n t).$$

Proof The output of U is computed from the output of its single attention layer in the last position (the position where on input we have the token t). Thus, the only part of the computation of U on input $t_1 \dots t_n t$ that depends on the values of $t_1, \dots, t_n$ is the computation of the value of the unique attention head in position $n+1$. According to formulas (3–5), to perform this computation, we need to compute the index $j = i_{n+1}^{(1)}$ with the maximal value of the scalar product $\langle Kx_i, Qx_{n+1} \rangle$ over $i \in \{1, \dots, n+1\}$. Afterwards, we need the value of the vector x_j from the input to our attention layer at this position. Here K, Q are the key and the query matrices of U, and

$$x_1 = \mathrm{E}(t_1, 1), \dots, x_n = \mathrm{E}(t_n, n),\ x_{n+1} = \mathrm{E}(t, n+1)$$

are vector embeddings of input tokens. Observe that both x_i and the scalar product $\langle Kx_i, Qx_{n+1} \rangle$ for $i = 1, \dots, n$ are determined by t_i and i, that is, by the assignment we have at the i-th position in $t_1 \dots t_n$. Consider a permutation τ of assignments, defined by the value of this scalar product – the larger this value is, the closer the assignment is to the beginning in τ. If two assignments have the same value of this scalar product, we, in accordance with our left-most tie-breaking rule, prefer one that belongs to a position with the smaller index. It is not important how to order assignments that have the same value of the scalar product and belong to the same position – no two such assignments can appear simultaneously in one input.

As a result, the first assignment in the permutation τ that is present in $t_1 \dots t_n$ will be for a position with the maximal value of $\langle Kx_i, Qx_{n+1} \rangle$ over $i \in \{1, \dots, n\}$ (not taking into account the position $n+1$). It remains to compare this value with the value of $\langle Kx_{n+1}, Qx_{n+1} \rangle$ (observe that x_{n+1} is determined just by t and thus is known) and define x_j as one among x_i and x_{n+1} that gives a larger value (in case of the equality, we prefer x_i). ■

Denote $t_0 = \mathsf{EoL}$, and for a given $\bar{w} \in \Sigma^n$, define the sequence of tokens $t_1, \ldots, t_r$ that U produces on input $\bar{w}\mathsf{EoL} = \bar{w}t_0$:

$$\begin{aligned} t_1 &= U(\bar{w}t_0) \\ t_2 &= U(\bar{w}t_0t_1) \\ &\vdots \\ t_r &= U(\bar{w}t_0t_1 \ldots t_{r-1}) = U^{(r)}(\bar{w}t_0) = f(\bar{w}). \end{aligned}$$

We have to show that one can compute the value of t_r in at most r a-queries to $\bar{w}$. Note that the token $t_0 = \mathsf{EoL}$ is the same for all $\bar{w}$. We will compute the tokens $t_1, \ldots, t_r$ one by one. It thus suffices to show that for all $\ell = 0, \ldots, r-1$, if we have already computed $t_0, \ldots, t_\ell$, it takes just one a-query to $\bar{w}$ to compute

$$t_{\ell+1} = U(\bar{w}t_0 \ldots t_\ell) = U(\sigma_1 \ldots \sigma_n t_0 \ldots t_\ell) = U^{t_\ell}_{n+\ell}(\bar{w}t_0 \ldots t_{\ell-1}).$$

By Lemma 12, this can be done by computing a single a-query to the word $\bar{w}t_0t_1 \ldots t_{\ell-1}$. The last ℓ letters $t_0, \ldots, t_{\ell-1}$ of this word are already known, so in fact it suffices to make 1 a-query to $\bar{w}$.

From decision trees to transformers. We now show the inequality $\mathrm{cotd}^{(1)}(f) \leq \mathsf{rk}(f)$. Assume that T is an r-depth decision tree over a-queries that computes f. We transform it into a one-layer one-head UHAT U whose r-th COT iteration computes f. We assume that T is a complete r-depth $|A|$-ary tree, where $A = \Sigma \times [n]$ is the set of assignments of f. Let $\mathcal{N}$ denote the set of nodes of T.

We give an almost complete description of U, leaving undefined for now parameters W_1, W_2, b_1, b_2 of the position-wise feed-forward network of our attention layer (see (7)), and the output distribution matrix W (see (10)).

- **The vocabulary**. The vocabulary of our UHAT is defined as $\mathcal{V} = \Sigma \cup O \cup \mathcal{N}$ (and we assume that $\mathcal{N}$ is disjoint from $\Sigma \cup O$). The EoL token is defined as the root of the tree T.

- **The dimension.** The dimension of our UHAT will be $d = |A| + |\mathcal{N}|$. Correspondingly, there will be $|A|$ "assignment coordinates", indexed by elements of A, and $|\mathcal{N}|$ "node coordinates", indexed by elements of $\mathcal{N}$. For $i \in A \cup \mathcal{N}$, by $e_i \in \mathbb{R}^d$ we denote the "i-th canonical basis vector" – one that has 1 in the coordinate i and 0 in the rest of the coordinates.

- **The input embedding.** We define $E(t, i)$ for a token $t \in \mathcal{V}$ and a position $i \in \mathbb{N}$ as follows. Assume first that $t = \sigma \in \Sigma$ is a letter of the input alphabet to the function f. If $i > n$, define $E(\sigma, i)$ arbitrarily; this will not be important for the construction. Now, if $i \leq n$, we let $E(\sigma, i)$ be the "one-hot encoding" of (σ, i) in the assignment coordinates:

$$E(\sigma, i)_a = \begin{cases} 1 & a = (\sigma, i) \\ 0 & \text{otherwise,} \end{cases} \qquad a \in A.$$

We also have to define $E(\sigma, i)$ in the node coordinates. Take a node $v \in \mathcal{N}$. If v is a leaf, define $E(\sigma, i)_v = 0$. Now, if v is not a leaf, there is some a-query, asked at v. Let $\tau_v \colon \{1, \ldots, |A|\} \to A$ be the permutation of assignments, defining this a-query. We let $E(\sigma, i)_v$ encode the position of the assignment (σ, i) in the permutation τ_v; the closer (σ, i) is to the beginning in the permutation τ_v, the larger the value $E(\sigma, i)_v$ is:

$$E(\sigma, i)_v = 1 + \frac{1}{\tau_v^{-1}(\sigma, i)}.$$

Next, if $t = v \in \mathcal{N}$ is a node of T, we simply set $E(v, i) = e_v$. Finally, if $t \in O$, we define $E(o, i)$ arbitrarily; it will not be important for the construction.

- **Parameter matrices.** We define the key and the query matrices K, Q to be the identity matrices. The matrix W_O from (6), which, since we have just one attention head, is a square matrix, is also defined as the identity. The value matrix V is defined by the following linear transformation:

$$(Vx)_a = x_a, \qquad a \in A,$$

$$(Vx)_v = 0, \qquad v \in \mathcal{N}$$

That is, it acts as the identity in the assignment coordinates and "nulls" the node coordinates.

Let $\bar{w} = \sigma_1 \ldots \sigma_n$ be an input to f. Let $o = f(\bar{w})$ and let $v_0, v_1, \ldots, v_r$ be the sequence of nodes of the tree T that arises when we compute $T(\bar{w})$ (with $v_0 = \mathsf{EoL}$ being the root, v_1 being a node at depth 1, and so on).

Our goal is to construct U such that on input $\bar{w}v_0 = \bar{w}\mathsf{EoL}$, our UHAT generates the following sequence of tokens:

$$v_1, v_2, \ldots, v_{r-1}, o,$$

with the r-th generated token being the output of the function, as required.

In other words, for any $\ell = 0, \ldots, r-1$, our UHAT U has to satisfy:

$$U(\sigma_1 \ldots \sigma_n v_0 \ldots v_\ell) = \begin{cases} v_{\ell+1} & \ell < r-1 \\ o & \ell = r-1. \end{cases} \tag{12}$$

Fix $\ell \in \{0, 1 \ldots, r-1\}$. The input to U in (12) for this ℓ is $\sigma_1 \ldots \sigma_n v_0 v_1 \ldots v_\ell$. It is transformed into a sequence of vectors as it is given to the unique attention layer L of U:

$$x_1 = E(\sigma_1, 1), \ \ldots, \ x_n = E(\sigma_n, n),$$

$$x_{n+1} = E(v_0, n+1), \ \ldots, \ x_{n+1+\ell} = E(v_\ell, n+1+\ell).$$

Let $y_1 \ldots y_{n+1+\ell} = L(x_1 \ldots x_{n+1+\ell})$ be the output of the attention layer. We, in fact, are just interested in the vector $y_{n+1+\ell}$, the output in the last position (only this output is used to compute U on our current input). Let us go through the computation of $y_{n+1+\ell}$. First, we compute attention weights:

$$L_{i,n+1+\ell} = \langle x_i, x_{n+1+\ell} \rangle$$

(recall that the query and the key matrices are identities). The vector $x_{n+1+\ell} = E(v_\ell, n+1+j) = e_{v_\ell}$, by definition, has 1 in the v_ℓ-th node coordinate and 0 elsewhere. Hence, $\langle x_i, x_{n+1+\ell}\rangle$ is equal to $(x_i)_{v_\ell}$. If $i \geq n+1$, then $x_i = E(v_{i-n-1}, i)$ and all its coordinates are at most 1. Now, if $i \leq n$, then, by definition,

$$(x_i)_{v_\ell} = E(\sigma_i, i)_{v_\ell} = 1 + \frac{1}{\tau_{v_\ell}^{-1}(\sigma_i, i)}.$$

This value is strictly bigger than 1 and is maximized at position $i_\ell \in \{1, 2, \ldots, n\}$ that has the smallest value of $\tau_{v_\ell}^{-1}(\sigma_i, i)$. The assignment $a_\ell = (\sigma_{i_\ell}, i_\ell)$ at this position thus will be the output of the a-query, asked at v_ℓ, on the input $\bar{w} = \sigma_1 \ldots \sigma_n$.

The value of our unique head at position $n+1+\ell$ thus is $h^{(1)}_{n+1+\ell} = V x_{i_\ell}$. The vector $x_{i_\ell} = E(a_\ell)$ has 1 in the a_ℓ-th assignment coordinate, 0 in the other assignment coordinates, and some values in the node coordinates. The matrix V "nulls" its node coordinates, so we get $h^{(1)}_{n+1+j} = e_{a_\ell}$.

Since the matrix W_O is the identity matrix, the "combined" head value (see (6)) becomes $h_{n+1+\ell} = W_O h^{(1)}_{n+1+\ell} = e_{a_\ell}$. Finally, the output of the attention layer at position $n+1+\ell$, according to (7), is computed as:

$$\begin{aligned} y_{n+1+\ell} &= W_2 \cdot \mathrm{ReLU}(W_1 \cdot (h_{n+1+\ell} + x_{n+1+\ell}) + b_1) + b_2 \\ &= W_2 \cdot \mathrm{ReLU}(W_1 \cdot (e_{a_\ell} + e_{v_\ell}) + b_1) + b_2 \end{aligned}$$

Our intermediate goal now is to define W_1, W_2, b_1, b_2 so that $y_{n+1+\ell}$ will become equal to $e_{v_{\ell+1}}$. Note that $v_{\ell+1} = \mathit{child}(v_\ell, a_\ell)$, where $\mathit{child}\colon \mathcal{N} \times A \to \mathcal{N}$ is defined as follows: given a non-leaf node v of T and an assignment a, the node $\mathit{child}(v, a)$ is the child of v, obtained by descending through the a-labeled edge (if v is a leaf, define $\mathit{child}(v, a)$ arbitrarily, for instance, $\mathit{chile}(v, a) = v$).

Lemma 13. *There exists $W_1, W_2 \in \mathbb{R}^{d\times d}$ and $b_1, b_2 \in \mathbb{R}^d$ such that for any non-leaf $v \in \mathcal{N}$ and for any $a \in A$, we have:*

$$e_{\mathit{child}(v,a)} = W_2 \cdot \mathrm{ReLU}(W_1 \cdot (e_a + e_v) + b_1) + b_2$$

Proof We set W_2 to be the identity matrix and $b_2 = 0$. Now our task is to define W_1 and b_1 so that:

$$e_{\mathit{child}(v,a)} = \mathrm{ReLU}(W_1 \cdot (e_a + e_v) + b_1). \tag{13}$$

We set b_1 to be the vector having -1 in each coordinate. As for the matrix W_1, we define it row-wise. Its rows, corresponding to the assignment coordinate or to the root of T, are set to 0. Hence, the output vector in (13) will always have 0 in these coordinates, as required. Now, take any non-root node u' of T and define its row in W_1 as follows. Let v' be its parent node in T, and let a' be the assignment on the edge from v' to u' (so that $u' = \mathit{child}(v', a')$). Set

$$(W_1)_{u'i} = \begin{cases} 1 & i \in \{v', a'\} \\ 0 & \text{otherwise.} \end{cases}$$

The value of the u'-th coordinate in (13) is thus equal to

$$\mathrm{ReLU}(W_{u'a} + W_{u'v} - 1).$$

Numbers $W_{u'a}, W_{u'v}$ take values in $\{0,1\}$. The output of ReLU above is 1 if and only if both of them are equal to 1 (and the output is 0 otherwise). Thus, we have 1 in the output precisely when $a = a'$ and $v = v'$, that is, when $u = child(v, a)$, as required. ■

Choosing W_1, W_2, b_1, b_2, we get that $y_{n+1+\ell} = e_{v_{\ell+1}}$. The final output token of U is then computed through (10–11), namely, we take the maximal coordinate of the vector $\mu_\ell = Wy_{n+1+\ell} = We_{v_{\ell+1}} \in \mathbb{R}^{\mathcal{V}}$, where $W \in \mathbb{R}^{\mathcal{V}\times d}$ is the output distribution matrix yet to be defined. Recall that our vocabulary consists of letters of the input alphabet of f, outputs of f, and nodes of T. Correspondingly, some coordinates of μ_ℓ are indexed by elements of Σ, some by elements of O, and some by elements of $\mathcal{N}$. We will use a similar notation for canonical basis vectors $e_i, i \in \mathcal{V}$ in $\mathbb{R}^{\mathcal{V}}$.

To make sure that (12) holds, it is enough to choose W so that $\mu_\ell = We_{v_{\ell+1}} = e_{v_{\ell+1}}$ if $\ell < r-1$ and $\mu_\ell = We_{v_{\ell+1}} = e_o$ if $\ell = r-1$ (where $o = f(\bar{w})$). We will achieve this if, for every $v \in \mathcal{N}$, we set the v-th column of W to be equal to e_v is v is not a leaf, and to e_o if v is a leaf and o is the output of T in it. ■

5. Multihead Rank

In order to generalize Theorem 9 to multi-head transformers, we define the notion of H-head rank for a function $f : \Sigma^n \to O$. For that, we require a notion of the *product* of two functions with the same domain. Namely, by the product of $f\colon A \to B$ and $g\colon A \to C$, we mean a function $(f \otimes g)\colon A \to B \times C$, defined by:

$$(f \otimes g)(a) = (f(a), g(a)).$$

An *H-degree a-query* is a product of H a-queries.

Definition 14. *The H-head rank of a function $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$, denoted $\mathsf{rk}^{(H)}(f)$, is the minimal depth of a decision tree over H-degree a-queries that computes f.*

A simple generalization of the construction of Theorem 9 allows us to obtain the following result.

Theorem 15. *For any $H \in \mathbb{N}$ and for any function $f\colon \Sigma^n \to O$, we have $\mathsf{rk}^{(H)}(f) = \mathrm{cotd}^{(H)}(f)$.*

Proof

We first show that $\mathsf{rk}^{(H)}(f) \leq \mathrm{cotd}^{(H)}(f)$. The proof for the case $H = 1$ works verbatim for the general case modulo the following generalization of Lemma 12.

Lemma 16. *Let U be a 1-layer H-head UHAT over the vocabulary $\mathcal{V}$. Then for any $n \in \mathbb{N}$ and $t \in \mathcal{V}$ we have $\mathsf{rk}^{(H)}(U_n^t) \leq 1$, where $U_n^t\colon \mathcal{V}^n \to \mathcal{V}$ is defined by:*

$$U_n^t(t_1 \ldots t_n) = U(t_1 \ldots t_n t).$$

The proof of the Lemma requires no new argument – with one a-query per attention head, we can compute all H of their values. The computation of attention heads in a transformer is performed in parallel, so all this requires just one H-degree a-query.

We now establish the inequality $\mathrm{cotd}^{(H)}(f) \le \mathsf{rk}^{(H)}(f)$. We have to convert an r-depth decision tree T over H-degree a-queries for f into a 1-layer H-head UHAT U whose r-th CoT iteration computes f.

As in the construction of Theorem 9, on input $\bar{w}\mathsf{EoL}$, where $\bar{w} \in \Sigma^n$, our UHAT U will produce the following sequence of tokens:

$$v_1, \dots, v_{r-1}, o,$$

where v_i is the depth-i node, visited by T on input $\bar{w}$, and $o = f(\bar{w})$. The vocabulary of U, as before, is the set $\Sigma \cup O \cup \mathcal{N}$, where $\mathcal{N}$ is the set of nodes of T, with EoL token being identified with the root of T.

This time, each non-leaf node v of T has an H-degree a-query q_v in it, defined by H permutations $\tau_v^{(1)}, \dots, \tau_v^{(H)}$ of the set A of assignments. The computation of the tree at the node v on input $\bar{w}$ goes as follows: for $k = 1, \dots, H$, one finds the smallest i such that the assignment $a_v^{(k)} = \tau_v^{(k)}(i)$ is consistent with $\bar{w}$. Then from v we descend by an edge, labeled by the tuple:

$$a_v = (a_v^{(1)}, \dots, a_v^{(H)}).$$

We proceed to the construction of U. The **dimension** will be H times larger than in the construction of Theorem 9, that is, $d = H \cdot (|A| + |\mathcal{N}|)$, where A is the set of assignments. Coordinates will be indexed by elements of $A \times [H] \cup \mathcal{N} \times [H]$. For $c \in A \times [H] \cup \mathcal{N} \times [H]$, we let $e_c \in \mathbb{R}^d$ denote the vector, having 1 in the c-th coordinate and 0 in the remaining coordinates. We will also use the notation $e[c]$ in some cases to improve readability.

The **input embedding** is defined as follows. For $v \in \mathcal{N}$ and any $i \in \mathbb{N}$, define $E(v, i) = e_{v,1}$. Now, for $\sigma \in \Sigma$ and $i \in [n]$, we set:

$$E(\sigma, i)_{a,k} = \begin{cases} 1 & a = (\sigma, i), \\ 0 & \text{otherwise,} \end{cases}, \qquad E(\sigma, i)_{v,k} = \begin{cases} 1 + \frac{1}{(\tau_v^{(k)})^{-1}((\sigma,i))} & v \text{ is a non-leaf,} \\ 0 & \text{otherwise,} \end{cases}$$

for $a \in A, v \in \mathcal{N}, k \in [H]$. For the remaining inputs, the function E is defined arbitrarily (this will not be relevant for the proof).

The **key, query, and value matrices** are defined as follows. All the key matrices $K^{(1)}, \dots, K^{(H)}$ are identity matrices. For $k = 1, \dots, H$, the k-th query matrix is defined such that

$$(Q^{(k)} e_{v,1}) = e_{v,k}$$

for all $v \in \mathcal{N}$. Finally, for $k = 1, \dots, H$, the k-th value matrix $V^{(k)}$ is defined by "nulling" all coordinates except coordinates of the form $c = (a, k)$:

$$(V^{(k)} x)_c = \begin{cases} x_c & c = (a, k) \text{ for some } a \in A, \\ 0 & \text{otherwise.} \end{cases}$$

As before, the matrix W_O, the matrices and bias vectors W_1, W_2, b_1, b_2 of the position-wise feed-forward network, and the output distribution matrix W will be defined later.

Our task is to make sure that for $\ell = 0, \dots, r-1$, we have

$$U(\bar{w} v_0 \dots v_\ell) = \begin{cases} v_{\ell+1} & \ell < r-1, \\ o & \ell = r-1, \end{cases}$$

where $v_0 = \mathsf{EoL}$ is the root of T.

Fix some $\ell = 0, \ldots, r-1$ and an input to U of the form $\bar{w} v_0 \ldots v_\ell$ (its length is $n+\ell+1$). It is easy to verify that with the key, query, and value matrices defined as above, the value of the k-th attention head at the $(n+\ell+1)$-st position will be:

$$h^{(k)}_{n+\ell+1} = e[a^{(k)}_{v_\ell}, k].$$

The matrix W_O is defined by:

$$W_O \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_H \end{pmatrix} = f_1 + \ldots + f_H,$$

so that

$$h_{n+\ell+1} = e[a^{(1)}_{v_\ell}, 1] + \ldots + e[a^{(H)}_{v_\ell}, H].$$

Our goal now is to make sure that the output of the attention layer in the last position satisfies:

$$y_{n+\ell+1} = e[v_{\ell+1}, 1].$$

This is achieved via the following analog of Lemma 13.

Lemma 17. *There exists $W_1, W_2 \in \mathbb{R}^{d \times d}$ and $b_1, b_2 \in \mathbb{R}^d$ such that for any non-leaf $v \in \mathcal{N}$, and for any $a^1, \ldots, a^H \in A$, for*

$$\bar{a} = (a^1, \ldots, a^H), \qquad z = e[v, 1] + e[a^1, 1] + \ldots + e[a^H, H],$$

we have

$$e[\mathit{child}(v, \bar{a}), 1] = W_2 \cdot \mathrm{ReLU}(W_1 z + b_1) + b_2,$$

where $\mathit{child}(v, \bar{a})$ denotes the son of v in T, followed by the $\bar{a}$-labeled edge.

Proof We set W_2 to be the identity matrix and $b_2 = 0$. The vector b_1 has $-H$ in every coordinate. The matrix W_1 is defined row by row. Rows that correspond to coordinates, other than of the form $(\hat{u}, 1)$ for a non-root $\hat{u} \in \mathcal{N}$, are set to 0. The key observation is that for a non-root $\hat{u} \in \mathcal{N}$ one can uniquely define $\hat{v} \in \mathcal{N}$ and $\hat{a}^1, \ldots, \hat{a}^H \in A$ such that

$$\hat{u} = \mathit{child}(\hat{v}, (\hat{a}^1, \ldots, \hat{a}^H)).$$

We set:

$$(W_1)_{(\hat{u},1),c} = \begin{cases} 1 & c \in \{(\hat{v}, 1), (\hat{a}^1, 1), \ldots, (\hat{a}^H, H)\}, \\ 0 & \text{otherwise.} \end{cases}$$

With this, we obtain:

$$\mathrm{ReLU}(W_1 z + b_1)_{(\hat{u},1)} = \mathrm{ReLU}\left(\mathbb{I}\{v = \hat{v}\} + \mathbb{I}\{\hat{a}^1 = a^1\} + \ldots + \mathbb{I}\{\hat{a}^H = a^H\} - H\right).$$

We have $\hat{u} = \mathit{child}(v, (a^1, \ldots, a^H))$ if and only if all the indicators above are equal to 1. As a result, the quantity above will be equal to 1 if and only if $\hat{u} = \mathit{child}(v, (a^1, \ldots, a^H))$.

■

It remains to define the output distribution matrix W so that $We[v_\ell + 1, 1] = e_{v_{\ell+1}}$ for $\ell < r - 1$ and $We[v_\ell + 1, 1] = e_o$ for $\ell = r - 1$. This can be done in exactly the same way as at the end of the proof of Theorem 9. ■

We observe that the H-head rank can be at most H times smaller than the normal rank. Specifically, each H-degree a-query can be computed by performing H individual a-queries sequentially.

Proposition 18. *For $f : \Sigma^n \to O$ and $H \geq 1$, we have $\mathsf{rk}(f) \leq H \cdot \mathsf{rk}^{(H)}(f)$.*

Proposition 18 allows us to reduce, up to a factor of H, lower bounds on $\mathsf{rk}^{(H)}(f)$ to lower bounds on $\mathsf{rk}(f)$. However, this proposition is sometimes unable to provide tight bounds on $\mathsf{rk}^{(H)}(f)$. This occurs, for instance, when $\mathsf{rk}^{(H)}(f)$ is not smaller at all than $\mathsf{rk}(f)$. We present two examples of this phenomenon in this section.

To establish a precise lower bound on the CoT-depth of a function with H heads, it suffices to derive a lower bound on its H-head rank (Theorem 15). However, this task proves to be significantly more challenging than for the single-head rank. Specifically, for the iterated composition function, combinatorial arguments alone, as employed in the proof of Proposition 4, are no longer sufficient. Instead, we must rely on techniques from communication complexity to address the problem. For the k-thOne_n function, we develop a combinatorial argument that is notably more intricate than the one used in the proof of Proposition 5.

5.1 Multi-head CoT-depth of iterated composition

In this section, we show a method for lower-bounding the multihead rank of a function based on communication complexity, see Kushilevitz and Nisan (1996). Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be finite sets and $f\colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a function. Imagine that there are two players, Alice and Bob. Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$. Their goal is to cooperatively compute $f(x, y)$. For that, they can send each other messages that are binary words. They want to minimize the number of messages and their total length in bits.

Formally, a *k-round Alice-first communication protocol* Π is given by:

- k positive integer numbers $\ell_1, \ldots, \ell_k$ (messages lengths);
- a function $M_i\colon \{0,1\}^{\ell_1+\ldots+\ell_{i-1}} \times \mathcal{X} \to \{0,1\}^{\ell_i}$ for every odd $i \in \{1, \ldots, k\}$;
- a function $M_i\colon \{0,1\}^{\ell_1+\ldots+\ell_{i-1}} \times \mathcal{Y} \to \{0,1\}^{\ell_i}$ for every even $i \in \{1, \ldots, k\}$; and
- the output function $\mathit{out}\colon \{0,1\}^{\ell_1+\ldots+\ell_k} \to \mathcal{Z}$.

The *communication complexity* of Π is the sum $\ell_1 + \ldots + \ell_k$.

On input $(x, y) \in \mathcal{X} \times \mathcal{Y}$, the *output* of Π on (x, y) is computed as follows. We inductively define a sequence of binary words $m_1 \in \{0,1\}^{\ell_1}, \ldots, m_k \in \{0,1\}^{\ell_k}$ by setting

$$m_i = M_i(m_1 \ldots m_{i-1}, x) \text{ for odd } i \in \{1, \ldots, k\},$$
$$m_i = M_i(m_1 \ldots m_{i-1}, y) \text{ for even } i \in \{1, \ldots, k\}.$$

Intuitively, $m_1 = M_1(\varepsilon, x)$ is the first message of Alice that she sends to Bob in the protocol on input x. Upon receiving m_1, Bob replies with the second message $m_2 = M_2(m_1, y)$ that depends on his input and the first of Alice's messages. Then Alice sends the third message $m_3 = M_3(m_1 m_2, x)$, and so on. The output of the protocol is defined as $out(m_1 \dots m_k) \in \mathcal{Z}$.

By $C^{k,A}(f)$ we mean the minimal communication complexity of a *k-round Alice-first protocol* that computes f. By reversing the roles of Alice and Bob, we define *k-round Bob-first protocols*, and $C^{k,B}(f)$, the minimal communication complexity of a k-round Bob-first protocol for a function f.

Assume we have a function $f\colon \Sigma_1 \times \ldots \times \Sigma_n \to O$ and a subset $S \subseteq [n]$. Suppose that positions of an input word $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$ are split between Alice and Bob like this: Alice knows letters of $\bar{w}$ at positions $i \in S$, and Bob knows letters of $\bar{w}$ at positions $i \in [n] \setminus S$. Their goal is to find out $f(\bar{w})$. This defines a function:

$$f^S\colon \left(\prod_{i \in S} \Sigma_i\right) \times \left(\prod_{i \in [n] \setminus S} \Sigma_i\right) \to O,$$

where the two inputs correspond to the parts of $\bar{w}$ that Alice and Bob know, respectively, and the output of is $f(\bar{w})$.

Assuming that the H-head rank of f is r, we construct low-communication $(r+1)$-round Alice-first and Bob-first protocols for f^S, for any $S \subseteq [n]$. This gives a method for lower-bounding the multihead rank of f: by showing that either $C^{r+1,A}(f)$ and $C^{r+1,B}$ is large enough, we conclude that the H-head rank of f is larger than r.

Lemma 19. *For every $f\colon \Sigma_1 \times \ldots \times \Sigma_n \to \{0,1\}$, for every $S \subseteq [n]$, and for every $H \geq 1$, we have*

$$C^{r+1,A}(f^S) \leq 2Hr \cdot \lceil \log_2 a \rceil, \qquad C^{r+1,B}(f^S) \leq 2Hr \cdot \lceil \log_2 a \rceil.$$

where $r = \mathsf{rk}^{(H)}(f)$ and a denotes the number of assignments to f.

Proof We first notice that Alice and Bob can compute the value of any H-degree a-query $q_{\tau_1} \otimes \ldots \otimes q_{\tau_H}$ by exchanging messages of length $H \cdot \lceil \log_2 a \rceil$. In fact, for a given input $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$, there are exactly n assignments consistent with $\bar{w}$. A part of them is known to Alice (for positions in S) and the other part to Bob (for positions in $[n] \setminus S$). For each $h = 1, \ldots, H$, Alice and Bob have to calculate the first assignment in the permutation τ_h which is consistent with $\bar{w}$. Alice can see which $\bar{w}$-consistent assignment, known to her, goes first in τ_h, and the same for Bob. Among these two assignments, the one that goes first is the answer to q_{τ_h}. Alice and Bob just have to exchange the indices of these assignments. For both Alice and Bob it is thus enough to send a $H\lceil \log_2 a \rceil$-bit message with indices of H assignments.

We already see that an r-depth decision tree over H-degree a-queries can be simulated by a communication protocol with $2Hr \cdot \lceil \log_2 a \rceil$ bits. We need to explain how to arrange this communication in $r+1$ rounds. For that, Alice and Bob have to alternate the order in which they exchange their messages in a computation of the H-degree a-queries. For example, for the Alice-first protocol, in the computation of the first query, Alice has to send her message first and then Bob. Now, for the second query, *Bob has to send his message*

first and then Alice. In this way, Bob's messages for the first and for the second query merge into a single round of communication. Similarly, for the third query, Alice has to send first, and then Bob, and so on, getting overall $r+1$ rounds. The Bob-first protocol is constructed in an analogous fashion. ■

As a corollary, we obtain the following:

Corollary 20. *For every H and t, for all but finitely many n, we have $\mathsf{rk}^{(H)}(t\text{-}\mathsf{Comp}_n) = t$.*

Proof We reduce from a communication problem called *pointer chasing*. In this problem, Alice is given $f_A\colon \{1,\dots,m\} \to \{1,\dots,m\}$ and Bob is given $f_B\colon \{1,\dots,m\} \to \{1,\dots,m\}$. In the k-step pointer chase, denoted here by PC_k^m, the goal of Alice and Bob is to compute:

$$\underbrace{\dots f_A(f_B(f_A}_{k \text{ times}}(1))\dots)$$

It is easy to see that $C^{k,A}(\mathrm{PC}_k^m) = O(k \log m)$ (Alice starts by sending $m_1 = f_A(1)$, Bob replies by sending $m_2 = f_B(m_1)$, and so on). It is known that this task requires much longer communication for k-round *Bob-first* protocols. Namely, for any constant k, we have $C^{k,B}(\mathrm{PC}_k) = \Omega(m)$, see Duris et al. (1987).

It remains to notice that $\mathrm{PC}_t^{n/2}$ is a special case of the problem $t\text{-}\mathsf{Comp}_n^S$, for $S = \{1,\dots,n/2\}$, where Alice gets $(\phi(1),\dots,\phi(n/2))$ and Bob gets $(\phi(n/2+1),\dots,\phi(n))$, for some function $\phi\colon \{1,\dots,n\} \to \{1,\dots,n\}$, and the task is to compute $\phi^{(k)}(1)$. Namely, we obtain $\mathrm{PC}_t^{n/2}$as a special case when ϕ maps the first half of the inputs into the second half, and the second half into the first half. Note that $t\text{-}\mathsf{Comp}_n$ has $a = n^2$ assignments. Assuming that $\mathsf{rk}^{(H)}(t\text{-}\mathsf{Comp}_n) < t$, by Lemma 19 we obtain:

$$\Omega(n) = C^{t,B}(\mathrm{PC}_t^{n/2}) \le C^{t,B}(t\text{-}\mathsf{Comp}_n^S) \le 2Ht \cdot \lceil \log_2 n^2 \rceil$$

For any fixed H, t this is true only for finitely many n. ■

5.2 Multi-head CoT depth of kth One

In this section, we establish a tight lower bound on the multi-head rank of k-thOne.

Theorem 21. *For any $k, H \in \mathbb{N}$, for all but finitely many $n \in \mathbb{N}$, we have $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}_n) = k$.*

Proof For brevity, inside the proof, we refer to "decision trees over H-degree a-queries" as "H-degree decision trees". We show the statement of the theorem by induction on k. For $k = 1$, it is enough to notice that our function is not constant, meaning that it cannot be computed by a 0-depth H-degree decision tree.

We proceed to the induction step. Assume that the theorem is proven for $k-1$. We establish the theorem for k. We will use a following terminology. By a *partial input* we mean a string $y \in \{0,1,*\}^n$ (with the symbol $*$ informally meaning "undefined"). A string $x \in \{0,1\}^n$ is a *completion* of a partial input $y \in \{0,1,*\}$ if $x_i = y_i$ for every $i \in [n]$ with $y_i \in \{0,1\}$.

The induction step is based on the following proposition.

Proposition 22. *Let $q = q_1 \otimes \ldots \otimes q_H$ be an H-degree a-query over n bits. Then there exists a partial input $y \in \{0, 1, *\}^n$ such that*

a) all completions of y have the same value of q;

b) there is at least one coordinate in y with value 1;

c) between the first coordinate with value 1 in y and the second coordinate with value 1 in y (or, if there are no other coordinates with value 1, between the first coordinate with value 1 and the end of y), there are at least $\Omega(n)$ coordinates with value $*$ (the constant in $\Omega(\cdot)$ might depend on H).

Assume that Proposition 22 is proved. We use it to show that any H-degree decision tree T for k-thOne on n input bits can be converted into an H-degree decision tree T' of smaller depth for $(k-1)$-thOne on $\Omega(n)$ input bits. Indeed, apply Proposition 22 to the H-degree a-query at the root of T. Let $y \in \{0, 1, *\}$ be a partial input, satisfying items a) – c). Mark by red all coordinates in y between the first coordinate with value 1 and the second coordinate with value 1 (if it exists, otherwise, until the end of y) that have value $*$. Let $m = \Omega(n)$ be the number of red coordinates. Given $z \in \{0, 1\}^m$, we can put bits of z into red coordinates of y (keeping the same relative order on the coordinates). Bits of z will be located after the first 1-coordinate of y and before the second 1-coordinate. The position of the k-th 1 in the resulting input y_z will thus determine the position of the $(k-1)$-st one in z. Hence, computing T on y_z (filling remaining undefined coordinates of y_z arbitrarily) will give us the value of $(k-1)$-thOne on z. Observe that it is not necessary to ask the first H-degree a-query of T on y_z because all completions of y, due to item a) of Proposition 22, have the same value on it. In other words, we will get an H-degree decision tree of strictly smaller depth than T, as required.

Thus, modulo Proposition 22, if we had $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}_n) \leq k-1$ for infinitely many n, we would have had $\mathsf{rk}^{(H)}((k-1)\text{-}\mathsf{thOne}_n) \leq k-2$ for infinitely many n, a contradiction with the induction hypothesis. It remains to establish the proposition. We reduce it to the following combinatorial problem.

A penguin problem. Imagine that there are n penguins, standing in a line. There are also H "penguin experts", each having their own ranking of penguins (a linear order on the set of penguins). We can remove some penguins from the line, keeping the order of the remaining penguins. After that, we can give a medal to some penguins, remaining in line. An expert is left happy if the best penguin, according to this expert, among those that remain in the line has a medal.

An observation that we will use later is that if an expert is currently happy, and we further remove some penguins without medals, or give medals to the others, this expert is still happy (nothing happens to the best penguin according to this expert).

Lemma 23. *For any fixed $H \geq 0$, it is possible to make all experts happy and to give at least 1 medal so that the following holds: if we go from left to right, then in the remaining line, starting from the left-most penguin with a medal, there are $\Omega(n)$ penguins until the next penguin with a medal (or until the end of the line if there are no other penguins with a medal). The constant in $\Omega(\cdot)$ might depend on H.*

Let us first deduce Proposition 22 from Lemma 23. Penguins will correspond to n coordinates of the input. We now define "penguin experts". They will correspond to a-queries $q_1, \ldots, q_H$, constituting our H-degree a-query $q = q_1 \otimes \ldots \otimes q_H$. Each q_i defines the following ranking (a linear order) on coordinates (penguins). Recall that q_i is defined by a permutation of the set of assignments. Consider a restriction of this permutation to assignments with value 1:

$$(j^i_1, 1), (j^i_2, 1), \ldots, (j^i_n, 1). \tag{14}$$

This restriction defines a ranking of coordinates: the j^i_1-th coordinate being the best in the ranking, the j^i_2-th coordinate being the second best, and so on. This will be the "ranking of penguins" of the expert q_i.

We remove/give medals to penguins in a manner that satisfies the requirements of Lemma 23. We then use this to construct a partial input $y \in \{0, 1, *\}^n$ as follows. Removing a penguin will mean assigning 0 to the corresponding coordinate in y; giving it a medal will mean assigning 1; and leaving it in the line without a medal will mean assigning $*$.

Observe that the resulting y satisfies the conditions b) and c) of Proposition 22. Indeed, the left-most penguin with a medal corresponds to the first coordinate in y with value 1; $\Omega(n)$ penguins without medal to the right until the other penguin with a medal (or until the end if no more medals are given) correspond to $\Omega(n)$ coordinates with value $*$ until the second value-1 coordinate in y.

We now have to show that all completions of y have the same value of q. It is enough to show this for all queries $q_1, \ldots, q_H$. Consider any of these queries q_i and the corresponding penguin expert. Let j^i_k be the best penguin (coordinate) according to this expert among penguins that are left in the line (that is, among coordinates with values 1 and $*$). This penguin has to be given a medal, meaning that in y the j^i_k-th coordinate has to be assigned 1, while coordinates $j^i_1, \ldots, j^i_{k-1}$ have to be assigned 0 (otherwise j^i_k would not be the best remaining penguin or q_i would not be happy).

Hence, the assignment $(j^i_k, 1)$ will be the first assignment in (14) which is present in y (independently of possible completions of $*$-coordinates). This almost implies that the value of q_i will be the same for all completions of y. However, there is a slight problem as (14) only takes into account assignments to the value 1. It could be that some assignment of the form $(j, 0)$ goes before $(i^i_k, 1)$ in the permutation of assignments, defining q_i, while the j-th coordinate is not fixed in y. In that case, we take the first such assignment in the permutation and additionally fix the j-th coordinate of y to 0; after that, the value of all completions of y on q_i will truly be the same. Overall, we will need to additionally fix at most $H = O(1)$ coordinates of y to 0, which will not violate conditions b)-c).

It thus remains to establish Lemma 23.

Proof [of Lemma 23] The proof is by induction on H. For the induction base, $H = 0$, there are no experts, so we can just give a medal to the first penguin.

We now establish the induction step. Assume that the lemma is proved for $H \geq 0$ experts. We prove it for $H + 1$ experts. Take $\alpha > 0$ satisfying:

$$\frac{1}{4(H+1)} < \alpha < \frac{1}{3(H+1)} \tag{15}$$

such that α is a multiple of $1/n$ (so that αn is an integer). We also assume that n is even, and if not, we initially remove an arbitrary penguin. For an expert E and $i \in \{1, \ldots, n\}$,

let R_i^E denote the ranking number of the i-th penguin in the line (counting from the left) according to E, with $R_i^E = 1$ if this penguin is the best, $R_i^E = 2$ if this penguin is the second best, and so on.

Case 1. Assume that there exists $i \in \{1, \dots, n\}$ and an expert E such that $R_i^E \leq i - \alpha n$ (intuitively, there exists an undervalued penguin, standing too far in the line according to one of the experts). Let P_i denote the penguin in the i-th position. We give a medal to P_i and all penguins to the right of it. Now, among penguins to the left of P_i, we remove those that are better than P_i in the ranking of E. For now, the remaining penguins to the left of P_i are left without a medal.

Observe that E is already happy – all penguins better than P_i are either removed or are given a medal. To make the other H experts happy, and to satisfy the condition about the left-most penguin with a medal, consider the set L of remaining penguins to the left of P_i (they are all currently without a medal). Observe that there are at least αn penguins in L. Indeed, there were initially $i - 1$ penguins to the left of P_i, and we removed at most $R_i^E - 1 \leq i - \alpha n - 1$ of them.

For a moment, forget about the expert E and penguins with medals standing to the right of the penguins in L. Use the induction hypothesis for the remaining H experts and restrictions of their rankings to L. By removing some penguins in L and giving medals to others, we make all these H experts happy while satisfying the condition about the left-most penguin – to the right of it, there are $\Omega(|L|) = \Omega(\alpha n) \geq \Omega(n/4(H+1)) = \Omega(n)$ penguins until the next penguin with a medal in L, or until the end of L if there are no other penguins with a medal.

What happens if we get back P_i and all penguins to the right of it? Since all penguins we add stand to the right of L, the condition about the number of penguins to the right of the left-most one with a medal still holds. Likewise, since we only add penguins with a medal, experts who were happy with restricting to L will still be happy (maybe for some experts, some better penguins will be added, but they will be with medals anyway). Finally, expert E who was happy before we used the induction hypothesis for penguins in L, will still be happy as we just removed some penguins and gave medals to the others.

Case 2. Assume that

$$R_i^E \geq i - \alpha n + 1 \qquad \text{for every } i \in \{1, \dots, n\} \text{ and every expert } E. \tag{16}$$

Let L denote the set of the first $n/2 + \alpha n$ penguins from the left, and R denote the set of the remaining penguins. We will make all experts happy by giving exactly one medal, namely, to a penguin in L, and removing only penguins in L. Then to the right of the unique penguin with a medal, there will be at least $|R| = n/2 - \alpha n = \Omega(n)$ penguins (recall that $\alpha < 1/3$ by (15)), as required.

It remains to explain how to make all experts happy in that way. We first show that there is a penguin in L whose position in the ranking is at most $n/2$ for every expert (meaning that it belongs to the top half of penguins). Indeed, re-writing (16), we get $i \leq R_i^E + \alpha n - 1$ for every $i \in \{1, \dots, n\}$ and every expert E. In other words, if some penguin has ranking r for some of the experts, its position in the line is at most $r + \alpha n - 1$. In particular, penguins with ranking at most $n/2$ for at least one of the experts belong to L. We now go through all $H+1$ experts E, and for each of them, make a mark on the best $n/2$ penguins according

to E (possibly leaving multiple marks on a single penguin). We will mark only penguins in L, and we have to show that at least one of them gets $H+1$ marks. Indeed, if the latter is not the case, the total number of marks is at most $|L| \cdot H$. On the other hand, for each of $H+1$ experts we make precisely $n/2$ marks, meaning that:

$$n/2 \cdot (H+1) \leq |L| \cdot H = (n/2 + \alpha n)H.$$

if $H = 0$, this is already a contradiction because the left-hand side is positive while the right-hand side is zero. In turn, if $H > 0$, this inequality implies $\alpha \geq \frac{1}{2H}$, contradicting (15).

Let P be a penguin from L whose ranking is at most $n/2$ according to all experts. We give P a medal, and for each expert E, remove all penguins better than P according to E. In the end, all experts will be happy, and we will be removing only penguins in L, as we only remove penguins whose rankings do not exceed $n/2$ according to at least one of the experts. ■

■

We observe that our communication complexity tool is not applicable in this case, as for any partition of the input positions between Alice and Bob, there exists a 2-round protocol with logarithmic communication that computes the position of the k-th one: Alice sends the positions of the first k ones in her part of the input, and Bob does the same.

Proposition 24. *For any k, n and $S \subseteq [n]$:*

$$C^{2,A}(k\text{-}\mathsf{thOne}_n^S) = C^{2,B}(k\text{-}\mathsf{thOne}_n^S) = O(k \log n).$$

If we wanted to use Lemma 19 to obtain a lower on $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}_n)$, we would have needed $C^{2,A}(k\text{-}\mathsf{thOne}_n^S)$ or $C^{2,B}(k\text{-}\mathsf{thOne}_n^S)$ to grow super-logarithmically with n for some $S \subseteq [n]$.

6. PAC Learning of Functions of Bounded Multi-Head Rank

We start by recalling the basics of the polynomial-time PAC learning model due to Valiant (1984). It starts with a sequence of "hypothesis classes" $\{H_n \subseteq \{0,1\}^{X_n}\}_{n \in \mathbb{N}}$, where X_n and H_n are sets of size $2^{poly(n)}$ for each n. We assume that elements of X_n and H_n are indexed by $poly(n)$-length strings such that, given encodings of a function $f \in H_n$ and an input $x \in X_n$, it is possible to evaluate $f(x)$ in $poly(n)$-time.

In the PAC-learning model, a "learner" receives n (index of the hypothesis class), the accuracy parameter $\varepsilon \in (0,1)$, and the confidence parameter $\delta \in (0,1)$. There is also a function $h \in H_n$ and a probability distribution μ over the set X_n. The learner does not know h and μ but has "sample access" to them. It means that the learner can sample a pair $(x, h(x))$, where $x \sim \mu$. This sampling can be repeated independently as many times as the learner wants.

We want to achieve that with probability at least $1-\delta$ (over randomness of the samples), the learner outputs a "hypothesis" $\widehat{h}\colon X_n \to \{0,1\}$ such that $\Pr_{x \sim \mu}[h(x) = \widehat{h}(x)] \geq 1 - \varepsilon$.

If there is a learner that for every n, ε, δ, achieves that in time $poly(n, 1/\varepsilon, 1/\delta)$, we say that our sequence of classes is polynomial-time PAC-learnable.

How precisely does the learner output $\widehat{h}$? We might require that $\widehat{h} \in H_n$, then it is enough to output a binary string, indexing this element of H_n. This model is called *proper* PAC learner. In turn, in the *improper* PAC learner, the output hypothesis $\widehat{h}$ might be outside H_n. In this case, different formalizations are possible; for instance, we might ask the learner to output a Boolean circuit, computing $\widehat{h}$. In what follows, we will focus on the proper PAC learning.

It is folklore (see, for instance, the proof of Theorem 1.3 in Kearns and Vazirani (1994)) that proper polynomial-time PAC-learning for a sequence of hypothesis classes $\{H_n\}_{n=1}^{\infty}$ is equivalent, under randomized polynomial-time reductions, to the *consistency problem* for $\{H_n\}_{n=1}^{\infty}$. In the consistency problem, one receives a number n and a "sample"

$$(x_1, y_1) \dots (x_m, y_m) \in (X_n \times \{0,1\})^m.$$

The question is whether there exists a function $h \in H_n$ (and we have to find it if it exists) that is "consistent" with the sample, that is,

$$h(x_1) = y_1, \dots, h(x_m) = y_m.$$

We will consider hypothesis classes consisting of functions of bounded multi-head rank. More precisely, by "functions of H-head rank at most k over the alphabet Σ" we mean a sequence of hypothesis classes $\{H_n\}_{n=1}^{\infty}$, where $H_n = \{f \colon \Sigma^n \to \{0,1\} \mid \mathsf{rk}^{(H)}(f) \leq k\}$. We first generalize a result of Ehrenfeucht and Haussler (1989) who gave a polynomial-time algorithm for the consistency problem for functions of rank at most k, for any fixed k. This is for their version of rank, coinciding with our 1-head rank. It was established only for the binary alphabet (and the notion of the rank was only defined for this case), and we extend this result to an arbitrary alphabet Σ.

Theorem 25. *For any k and Σ, the consistency problem for functions of 1-head rank at most k for a sample $S \subseteq \Sigma^n \times \{0,1\}$ can be solved in*

$$n|S| \cdot O(n|\Sigma|)^{3k+1}$$

time.

Hence, functions of 1-head rank at most k are polynomial-time properly PAC-learnable, for any fixed k. Another interesting corollary is that for any fixed k and H, functions of H-head rank at most k are *improperly* polynomial-time PAC-learnable (simply because by Proposition 18 they belong to a larger properly polynomial-time PAC-learnable class of functions with 1-head rank at most kH). This leaves a question – can we make them *properly* polynomial-time PAC-learnable? Our main result in this section refutes this possibility already for 2-head rank-1 functions, and for the binary alphabet.

Theorem 26. *The decision version*[1] *of the consistency problem for functions of 2-head rank at most 1 over $\Sigma = \{0,1\}$ is NP-complete.*

1. One just needs to answer whether a hypothesis, realizing the sample, exists, no need to find it.

This result implies that functions of 2-head rank at most 1, already in the binary case, are not properly polynomial-time PAC learnable (unless NP is a subset of BPP).

We conjecture that the consistency problem for functions of H-head rank at most r is NP-complete for any $r \geq 1$ and $H \geq 2$. We leave this as an open question

6.1 Proof of Theorem 25

For brevity, within the proof, we call the consistency problem for functions of 1-head rank at most k the "rank-k consistency problem". Likewise, if a sample S is consistent with some function of 1-head rank at most k, we say that S is "rank-k consistent".

By induction on k, we show that the rank-k consistency problem is polynomial-time solvable. For $k = 0$, we notice that functions, having 1-head rank 0, are precisely the all-0 and the all-1 functions. Checking if a given sample is consistent with one of them takes linear time. We now assume that the rank-k consistency problem is polynomial-time solvable. Based on that, we now give a polynomial-time algorithm for the rank-$(k+1)$ consistency problem.

We use the following notation: for a sample S and an assignment a, let S_a denote the subsample of S, consisting of all inputs of S that have a.

Lemma 27. *If S is rank-$(k+1)$ consistent and non-empty, then there exists an assignment a such that S_a is non-empty and rank-k consistent.*

Proof Let T be a depth-$(k+1)$ a-query decision tree that is consistent with S. Let q_τ be its top a-query, and τ be the permutation of assignments, defining it. Take the first assignment a in τ such that S_a is non-empty (such a exists because S is non-empty so inputs from S have some assignments). Note that all inputs from S_a go to the sub-tree of T that lies beneath the a-labeled child of the root. This sub-tree, having depth at most k, establishes that S_a is rank-k consistent. ■

Lemma 27 provides the following algorithm for the rank-$(k+1)$ consistency problem. If S is empty, there is nothing to do. If S is non-empty, we check if there exists an assignment a such that S_a is non-empty and rank-k consistent (using the induction hypothesis to check the latter in polynomial time). If no such a exists, due to Lemma 27, we know that S is not rank-$(k+1)$ consistent.

Now, if such a exists, we take any such a and recursively solve the rank-$(k+1)$ consistency problem on the sample $S' = S \setminus S_a$. If it returns that S' is not rank-$(k+1)$ consistent, we know that S is not as well.

Now, suppose that the recursive procedure returns a depth-$(k+1)$ a-query decision tree T', consistent with S', We turn it into a depth-$(k+1)$ a-query decision tree T, consistent with S. Indeed, notice that inputs from $S' = S \setminus S_a$ do not have the assignment a. In particular, from the root of T', we can delete the sub-tree under the a-labeled child, this will not affect the computation of T' on inputs from S'. Likewise, we can move a to the first position in the permutation, defining the top a-query of T'. Finally, under the a-labeled child, where it is currently empty, we can put the tree T_a, the k-depth a-query decision tree that is consistent with S_a (and that can be found using the algorithm from the induction

hypothesis). This modification ensures that inputs from S_a are also computed correctly by the resulting tree T.

To analyze the time complexity of our algorithm, we introduce the parameter $B = A+n+2$, where A is the number of possible assignments A and n is the length of examples. By construction, $B \geq 2$. Each of our recursive calls is for the value of B, which is at least 1 smaller. Indeed, when we call the recursion for S_a and k (at most $A \leq B$ times), the length n can be decreased because the coordinate of the assignment a is fixed. And when we call the recursion for $S \setminus S_a$ for $k+1$ (just once), at least one assignment is disregarded. We obtain the following recurrence:

$$T(B, k+1) \leq B \cdot T(B-1, k) + T(B-1, k+1) + U,$$

where U is the upper bound on the time besides recursive calls. One can easily show the upper bound

$$T(B, k) \leq U \cdot (B)^{2k}$$

by induction. Indeed, $T(B, 0) \leq U = U \cdot B^0$ and

$$\begin{aligned} T(B, k+1) &\leq U + B \cdot T(B-1, k) + T(B-1, k+1) \leq U\left(1 + B(B-1)^{2k} + (B-1)^{2k+2}\right) \\ &\leq U \cdot (B-1)^{2k}(1 + B + (B-1)^2) = U \cdot (B-1)^{2k}(1 + B + (B-1)^2) \\ &= U \cdot (B-1)^{2k}(B^2 - B + 2) \leq U \cdot (B-1)^{2k} B^2 \leq U \cdot B^{2k+2} \end{aligned}$$

(for the inequality before the last one, recall that $B \geq 2$). It remains to bound U. We need linear time for each of $A \leq n|\Sigma|$ recursive calls, that is, at most $|S|n^2|\Sigma|$. In the end, we need time, linear in the size of the output tree, bounded by $O(n|\Sigma|)^k$. The overall time complexity is as required:

$$n|S| \cdot O(n|\Sigma|)^{3k+1}.$$

6.2 Proof of Theorem 26

Recall how a depth-1 decision tree T over 2-head a-queries works. It is given by a 2-head a-query, defined in turn by two permutations of assignments. These two permutations can also be seen as linear orders on the set of assignments. To compute $T(x)$, we treat $x = x_1 \dots x_n$ as a set of assignments $\{(1, x_1), \dots, (n, x_n)\}$ that are present in x. Then we take a pair of assignments that are maximal in this set w.r.t our linear orders. This pair of elements determines $T(x)$.

Hence, a sample $S = S^+ \cup S^-$ (where S^+, S^- denote the parts of S that are classified positively and negatively, respectively) is head-2 rank-1 consistent if and only if there exist 2 linear orders on the set of assignments such that no $x \in S^+$ and $y \in S^-$, viewed as sets of assignments, have the same maxima in both linear orders.

We define a more general problem that we call the *2-order separability problem*. In this problem, we are given two families $\mathcal{F}$ and $\mathcal{G}$ of non-empty subsets of some set U. The question is whether there exist two linear orders on U such that no two sets $S \in \mathcal{F}$ and $T \in \mathcal{G}$ have the same maxima in both orders.

In our proof, we first establish NP-completeness of the 2-order separability problem, and then reduce it to the consistency problem for 2-head rank-1 functions over the binary alphabet. Inclusions in NP are trivial and are omitted.

Lemma 28. *The 2-order separability problem is NP-complete.*

Proof We reduce from the monotone NAE-3-SAT problem (3-SAT where all variables are without negations and the task is to have at least one value 0 and at least one value 1 in every clause) whose NP-completeness is proved in (Schaefer, 1978). Let ϕ be an instance of monotone NAE-3-SAT. Elements of the set U in the 2-order separability problem will be variables of ϕ, a special "zero" element 0, and some other "fresh elements".

First, for every variable x of ϕ, we put the following sets into families $\mathcal{F}, \mathcal{G}$:

$$\{u, v, w, x, 0\}, \{u, x, 0\}, \{v, x, 0\} \text{ into } \mathcal{F}, \tag{17}$$

$$\{u, w, x, 0\}, \{v, w, x, 0\}, \{u, 0\}, \{v, x\} \text{ into } \mathcal{G}, \tag{18}$$

where u, v, w are fresh elements, different for different variables.

Additionally, for every NAE clause NAE(x, y, z) of ϕ, we put the following sets into our families:

$$\{u, v, w, x, y, z, 0\}, \{u, x, y, z, 0\}, \{v, x, y, z, 0\} \text{ into } \mathcal{F}, \tag{19}$$

$$\{u, w, x, y, z, 0\}, \{v, w, x, y, z, 0\}, \{u, 0\}, \{v, 0\} \text{ into } \mathcal{G}, \tag{20}$$

where again, u, v, w are fresh and unique for each clause (and different from the corresponding fresh elements for variables). Description of the reduction is finished.

Assume first that ϕ is satisfiable. We show that (17–20) is also satisfiable (separable by 2 orders). Take a satisfying assignment to ϕ. If $x = 1$, set it to be larger than 0 in the first order and smaller than 0 in the second order. If $x = 0$, set x to be smaller than 0 in the first order and larger than 0 in the second order. The relative order between the variables is not important.

To satisfy (17–18) for a variable x, we define the relative orders of u, v, w with respect to $x, 0$ as follows:

$$(u, w, 0, x, v) \text{ and } (v, w, x, 0, u)$$

(taking into account that in one order x is smaller than 0, and bigger than 0 in the other). Indeed, then the pairs of maxima in (17) are $(u, v), (u, x), (0, v)$, and in (18) are $(u, w), (w, v), (u, 0), (x, v)$. Notice also that every pair includes either u, v, or w. This distinguishes these sets from all the remaining sets in our construction.

To satisfy (19–20) for a clause NAE(x, y, z), we define the relative orders of u, v, w with respect to $x, y, z, 0$ as follows:

$$(u, w, ord_1(x, y, z, 0), v) \text{ and } (v, w, ord_2(x, y, z, 0), u),$$

where $ord_1(x, y, z, 0), ord_2(x, y, z, 0)$ are relative orderings of $x, y, z, 0$ in the first and the second order, respectively. Since NAE(x, y, z) is satisfied, among x, y, z, there is a variable equal to 1, and there is a variable equal to 0. Hence, in both orders, one of x, y, z is larger than 0. That is, in (19), pairs of maxima are (u, v), $(u, m_2), (m_1, v)$, where $m_1, m_2 \neq 0$ are the maxima of $\{x, y, z, 0\}$ in the first and the second order, respectively. In turn, in (20), pairs of maxima are $(u, w), (w, vs), (u, 0), (0, v)$. Once again, every pair includes either u, v, w, distinguishing these sets from the remaining sets of the construction for other variables and clauses.

Finally, we show that if (17–20) is satisfiable, then ϕ is satisfiable as well. Take a separating pair of orders. We first show that every variable x is bigger than 0 in one of the orders and smaller than 0 in the other order. First, in (17) we have a set with all 5 elements, and in the other, we have a set without u and the set without v. This enforces u to be the maximum in one of the orders and v to be the maximum in the other order (restricted to 5 elements in question). Now, to separate $\{u, x, 0\}$ from $\{u, 0\}$, the element x must be larger than 0 in the order where u is not maximal. Likewise, 0 must be larger than x in the order where v is not maximal, otherwise we do not separate $\{v, x, 0\}$ from $\{v, x\}$.

We set $x = 1$ if and only if x is larger than 0 in the first order. We show that this assignment satisfies ϕ. Take any clause $\mathrm{NAE}(x, y, z)$ of ϕ. It is enough to show that the maximum of $\{0, x, y, z\}$ is not 0 in both orders. Indeed, by definition, if we look at the first order, some variable out of x, y, z is larger than 0 there, meaning that it is set to 1. But they cannot all be set to 1, because then they are all smaller than 0 in the second order.

To show that 0 cannot be the maximum of $\{0, x, y, z\}$ in neither of the orders, we first notice that by the same argument, in (19–20) the maximum of one order is u, and of the other is v (because we have a set with everything versus sets without u and without v). The claim then follows from the fact that we must separate $\{u, x, y, z, 0\}$ from $\{u, 0\}$ and $\{v, x, y, z, 0\}$ from $\{v, 0\}$. ■

We now reduce from the 2-order separability problem to the consistency problem for 2-head rank-1 functions. The reduction is rather technical and we have to start with an idea. Consider some instance of the 2-order separability problem, for example,

$$U = \{u, v, w\}, \qquad \mathcal{F} = \{\{u, v, w\}\}, \qquad \mathcal{G} = \{\{u, v\}, \{u, w\}, \{v, w\}\}. \tag{21}$$

Convert this instance into an instance of the 2-head rank-1 consistency problem. Coordinates of vectors are indexed by elements of U, and sets in families $\mathcal{F}, \mathcal{G}$ are turned into characteristic vectors of these sets. As a result, we get:

$$S^+ = \{111\}, \qquad S^- = \{110, 101, 011\}. \tag{22}$$

If the initial instance is satisfiable, then this new instance is also satisfiable. Indeed, we first identify elements of U with 1-assignments. Two linear orders on U that satisfy the initial instance naturally define two linear orderings on the 1-assignments. We have to extend these ordering to 0-assignments, and we simply make all of them smaller than all 1-assignments in both orders. From the 2 orders, separating $\mathcal{F}$ and $\mathcal{G}$, we will get 2 orders, separating S^+ and S^-. Note that by definition, sets in $\mathcal{F}, \mathcal{G}$ are non-empty, and thus every vector in S^+, S^- will have at least one 1.

This simple reduction, unfortunately, does not work. The problem is that the new instance can be satisfiable while the initial instance is not. Intuitively, this is because 0-assignments are not forced to go after 1-assignments in the 2-head rank-1 consistency problem. For instance, we can separate S^+ from S^- in (22) by making all 0-assignments bigger than all 1-assignments (in fact, just one order is enough). However, the initial families $\mathcal{F}$ and $\mathcal{G}$ in (21) are not separable by 2 linear orders. Indeed, for any pair of orders, the set $\{u, v, w\} \in \mathcal{F}$ will have both maxima, and some set from $\mathcal{G}$ too.

The challenge is to enforce all, or at least *almost all* 0-assignments to be smaller than all 1-assignments. As a warm-up, consider a sample where S^+ consists of vectors with exactly

one 1, and $S^- = \{00\dots0\}$. We claim the following: in any linear order that separates S^+ from S^-, all 1-assignments except, possibly, one are larger than all 0-assignments Indeed, can we make some 0-assignment maximal? No, because every 0-assignment appear in some vector in S^+ and in the unique vector of S^-. Hence, we have to choose some 1-assignment as maximal: this makes the vector that has this 1-assignment distinguished from all the other vectors. Removing it from S^+, we see that still all 0-assignments appear both in S^+ and S^-, meaning that we have to make another 1-assignment to be the next maximal element. The same is true until at least 2 vectors are still in S^+. When we removed all but one vector from S^+, there is now a 0-assignment that does not appear in S^+, namely, in the position with 1 of the unique vector left in S^+. In principle, this 0-assignment can be made larger than the 1-assignment in the same position, but all the other 1-assignments have to be larger than all 0-assignments.

Note, however, that this works for just 1 order, and we will need something similar for 2. Our actual construction, which we are now beginning to present, will be more involved. Namely, if the size of U in the initial 2-order separability instance is n, we will have $3n+3$ coordinates: $3n$ "working" coordinates, indexed by pairs $U \times \{1,2,3\}$, and 3 auxiliary ones called u, v, w. We split the working coordinates into 3 groups of size n. We do the reduction with characteristic vectors, described above, in these 3 groups of coordinates independently. That is, for each group, sets from families are mapped into their characteristic vectors in coordinates of the group, with 0s outside the group. More formally, we map two families $\mathcal{F}, \mathcal{G}$ of non-empty subsets of U into sets S^+, S^- of binary vectors:

$$S^+ = \{\chi^1_A, \chi^2_A, \chi^3_A \mid A \in \mathcal{F}\}, \qquad S^- = \{\chi^1_B, \chi^2_B, \chi^3_B \mid B \in \mathcal{G}\},$$

where for $A \subseteq U, i \in \{1,2,3\}$, we define:

$$(\chi^i_A)_c = \begin{cases} 1 & c = (a,i) \text{ for some } a \in A \\ 0 & \text{otherwise.} \end{cases}$$

For instance, if $U = \{1,2,3,4\}$, a set $A = \{2,3\}$ will be mapped into three 15-bit binary vectors (rows of the table above):

	u	v	w	1st group	2nd group	3rd group
χ^1_A	0	0	0	0110	0000	0000
χ^2_A	0	0	0	0000	0110	0000
χ^3_A	0	0	0	0000	0000	0110

If the initial instance of the 2-separability problem is satisfiable, the constructed instance is satisfiable as well. Namely, we take two linear orderings on U that separate the initial instance, then for each group, use these 2 orders as orders on 1-assignments in the coordinates of this group, and make all 0-assignments smaller than all 1-assignments (comparison of 1-assignments from different groups will not be relevant). Two vectors of the from χ^i_A, χ^i_B for $A \in \mathcal{F}, B \in \mathcal{G}, i \in \{1,2,3\}$ will be separated because the two orders on U that we use separate A from B. In turn, vectors χ^i_A, χ^j_B for $i \neq j$ will be separated because the maxima in them will be for 1-assignments in different groups.

We will now construct a gadget (two sets of binary strings $S^+_{gadget}, S^-_{gadget}$) with the following property: for any 2 linear orders on assignments that separate S^+_{gadget} from S^-_{gadget},

it holds that in both linear orders all but, possibly, one 1-assignment in working coordinates are larger than all 0-assignments in the working coordinates. This will ensure that if $S^+ \cup S^+_{gadget}$ is separable from $S^- \cup S^-_{gadget}$, then $\mathcal{F}$ will be separable from $\mathcal{G}$ in the initial 2-separability problem. Indeed, there will be at most 2 "bad" working coordinates, and in one of the 3 groups, all 1-assignments will be larger than all 0-assignments. Restrictions of our 2 orders to 1-assignments in this group will separate the families in the initial instance of the 2-order separability problem.

We will also make sure that adding the gadget cannot ruin satisfiability – if the initial instance is satisfiable, the instance with the gadget will also be satisfiable.

We now describe our gadget. We assume that auxiliary coordinates go first. To S^+, we add all vectors that have 100 in the auxiliary coordinates and exactly one 1 in the working coordinates, and also the vector 0100...0 (it has 010 in the auxiliary coordinates and all-0s in the working coordinates). Now, to S^- we add all vectors that have 010 in the auxiliary coordinates and exactly one 1 in the working coordinates, and now the vector 1000...0. We also add two more vectors to each S^+ and S^-. Namely, we add 1011...1 and 0111...1 to S^+, and we add 1111...1 and 0011...1 to S^-:

$S^+_{gadget} =$

u	v	w	Working coordinates			
1	0	0	1	0	…	0
1	0	0	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋱	⋮
1	0	0	0	0	…	1
0	1	0	0	0	…	0
1	0	1	1	1	…	1
0	1	1	1	1	…	1

,

$S^-_{gadget} =$

u	v	w	Working coordinates			
0	1	0	1	0	…	0
0	1	0	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋱	⋮
0	1	0	0	0	…	1
1	0	0	0	0	…	0
1	1	1	1	1	…	1
0	0	1	1	1	…	1

In this picture, we call auxiliary coordinates u, v, and w, and we have separated them by a vertical line from the working coordinates. We also separated each part into 3 groups by horizontal lines to increase readability.

Lemma 29. *Any pair of linear orders where*

- *all 1-assignments are larger than all 0-assignments*
- *in one linear order, the maximal element is $(u, 1)$ and the next biggest is $(w, 1)$, but $(v, 1)$ is smaller than all the other 1-assignments; in the other linear order, the maximal element is $(v, 1)$ and the next biggest is $(w, 1)$, and $(u, 1)$ is smaller than all the other 1-assignments.*

separates S^+_{gadget} from S^-_{gadget}.

Proof Let us look first at the last 2 vectors in both S^+_{gadget} and S^-_{gadget} (the third group). They will be separated from each other and from everything else. The all-1 vector has as its two maxima $(u, 1), (v, 1)$, the vector $0011\ldots1$ has both maxima $(w, 1)$, and vectors $1011\ldots1$ and $011\ldots1$ have pair of maxima $(u, 1), (w, 1)$ and $(w, 1), (u, 1)$, correspondingly. All the other vectors have the maximal element in one of the orders, but something smaller than $(w, 1)$ in the other order.

Now, all vectors in the 1st group of S^+_{gadget} have the maxima $(u, 1)$ in one of the orders. This distinguishes them from the 1st group of S^-_{gadget}. The vector of the second group of S^-_{gadget} also has $(u, 1)$, but it does not have other 1-assignments, so both its maxima are $(u, 1)$. This distinguishes this group from the 1st of S^+_{gadget} because vectors there have the second maxima in one of the working coordinates (recall that $(u, 1)$ is the smallest 1-assignment in the order where it is not maximal).

Swapping the roles of u and v, one can show that the 1st group of S^-_{gadget} will be separated from the 1st and the 2nd group of S^+_{gadget}. Now, the second groups of S^+_{gadget} and S^-_{gadget} are separated because in one of them both maxima are $(u, 1)$, and in the other $(v, 1)$. ■

This lemma implies that if the initial instance of the 2-separability problem is satisfiable, this instance with the gadget is also satisfiable, because we have freedom to choose any order on 1-assignments in the working coordinates in both linear orders, and the gadget will be separated from characteristic vectors in working coordinates since in the gadget, at least one maxima will be $(u, 1), (v, 1)$, or $(w, 1)$, which will not be the case for the characteristic vectors.

To ensure that the other direction also holds (if the instance with the gadget is satisfiable, then the initial instance of the 2-separability problem also is), it is enough, as we discussed, to prove the following:

Lemma 30. *For any pairs of orders that separate S^+_{gadget} from S^-_{gadget}, we have, for both orders, that all but one 1-assignment in working coordinates are larger than all 0-assignments in working coordinates.*

Proof We need to establish the following technical claim: for any pair of orders that separate S^+_{gadget} from S^-_{gadget}, the only possible pairs of maximal elements in those orders are the following:

- $(u, 1), (v, 1)$;
- $(u, 0), (v, 0)$;
- $(u, 0), (u, 1)$;
- $(v, 0), (v, 1)$.

Imagine that we have already established this claim. Observe the following: for any of these 4 cases, the 1st group in S^+_{gadget} and the 2nd group in S^-_{gadget} have the maximal element in one of the orders, and the 2nd group in S^+_{gadget} with the 1st group in S^-_{gadget} have the maximal element in the other order. Take now, for example, the 1st group in S^+_{gadget} and the 2nd group in S^-_{gadget}. All vectors have the same values in auxiliary coordinates, and then in the working coordinates, these are vectors with exactly one 1 vs the all-zero vector. Now, since all these vectors have the maximal element of one of the orders, they have to be separated by their maxima in the other order. As we discussed in our motivating example, in that other order, all but one 1-assignments in the working coordinates are larger than all 0-assignments. Thus, we obtained what we want for one of the orders; for the other order, this can be obtained by considering the 2nd group of S^+_{gadget} and the 1st group in S^-_{gadget}.

It remains to establish our technical claim about 4 possible pairs of maximal elements. First, we need to see that no assignment in a coordinate other than u, v can be made maximal in either of the orders. For instance, take the w coordinate. Why cannot we make the assignment $(w, 0)$ maximal in one of the orders? Then vectors having this assignment have to be distinguished by the other order. The problem, however, is that among vectors that have $(w, 0)$, we have all possible assignments in all the other coordinates both in S^+_{gadget} and S^-_{gadget}. Making any of them maximal, we confuse some vector from S^+_{gadget} with some vector from S^-_{gadget}.

We will have a similar problem with making $(w, 1)$ one of the maxima. In restrictions to vectors that have $(w, 1)$, both S^+_{gadget} and S^-_{gadget} have both 0 and 1 in coordinates u and v, and both have only 1 in the working coordinates. This makes these restrictions indistinguishable by the second order.

The same check can be done for the working coordinates – by symmetry, it is enough to take only the first one. Restriction to the 1-assignment in this coordinate has both 0,1 in every other coordinate, in both S^+_{gadget} and S^-_{gadget}. In turn, for the 0-assignment, almost the same is true except for the w-coordinate, where both S^+_{gadget} and S^-_{gadget} have only 0.

Thus, it is already established that both maxima have to belong to u, v-coordinates. "Bad pairs" that we have to refute are:

- $(u, 0)$ and $(v, 1)$;
- $(u, 1)$ and $(v, 0)$;
- $(u, 0)$ and $(u, 0)$ (the same maximal elements in both orders);
- $(u, 1)$ and $(u, 1)$;
- $(v, 0)$ and $(v, 0)$;
- $(v, 1)$ and $(v, 1)$.

One can check that for any of these cases, both S^+_{gadget} and S^-_{gadget} have a vector having both assignments from the pairs. Those 2 vectors would have been indistinguishable. ■

7. Final Remarks

We have established a tight correspondence between the expressive power of single-layer transformers with hard attention and the rank of functions. This correspondence allows rank lower bounds to be interpreted as lower bounds on the number of chain-of-thought iterations required by such transformers, and, conversely, provides an operational reading of rank as a measure of sequential "first-success" extraction. Although we do not use the transformer perspective to obtain new rank bounds, we view the result mainly as a structural link between decision-tree complexity and simplified transformer architectures.

Extending our characterization to more layers or to soft attention is a challenging future direction. In a contemporaneous manuscript, Chen et al. (2025) have proved unconditional lower bounds on the embedding dimension of multilayer decoder-only Transformers with

soft attention that compute iterated function composition. However, their version of the problem differs significantly from the one considered here: they have several functions to compose, and and the answer is to be given in one CoT step.

Another independent paper Bavandpour et al. (2025) gave lower bounds on the number of chain-of-thought iterations in hard attention for some functions—including PARITY. A significant difference is that the current contribution provides a precise quantification for the number of iterations, while their paper deals only with an approximate asymptotic lower bound.

Acknowledgments

Kozachinskiy is supported by ANID Fondecyt Iniciación grant 11250060. Barceló and Kozachinskiy are funded by the Centro Nacional de Inteligencia Artificial CENIA FB210017, Basal ANID. Barceló is also funded by ANID Millennium Science Initiative Program Code ICN17002. Additionally, Steifer was financially supported by the Foundation for Polish Science (FNP) grant 'Centre for Credible AI' No. FENG.02.01-IP.05-0058/24.

References

Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir V. Podolskii. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=gbrHZq07mq`.

Alireza Amiri Bavandpour, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 3274–3306. PMLR, 13–19 Jul 2025. URL `https://proceedings.mlr.press/v267/bavandpour25a.html`.

Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2631–2653. IEEE, 2025. doi: https://doi.org/10.1109/FOCS63196.2025.00136.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/chiang23a.html`.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? An analysis of bert's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@ACL 2019, Florence, Italy, August 1, 2019*, pages 276–286. Association for Computational

Linguistics, 2019. doi: 10.18653/V1/W19-4828. URL https://doi.org/10.18653/v1/W19-4828.

Pavol Duris, Zvi Galil, and Georg Schnitger. Lower Bounds on Communication Complexity. *Information and Computation*, 73(1):1–22, 1987. URL https://doi.org/10.1145/800057.808668.

Andrzej Ehrenfeucht and David Haussler. Learning Decision Trees from Random Examples. *Information and Computation*, 82(3):231–246, 1989. URL https://doi.org/10.1016/0890-5401(89)90001-1.

Juan Luis Esteban and Jacobo Torán. A Combinatorial Characterization of Treelike Resolution Space. *Information Processing Letters*, 87(6):295–300, 2003. URL https://doi.org/10.1016/S0020-0190(03)00345-4.

Xinyan Guan, Yanjiang Liu, Hongyu Lin, Yaojie Lu, Ben He, Xianpei Han, and Le Sun. Mitigating large language model hallucinations via autonomous knowledge graph-based retrofitting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18126–18134, 2024. URL https://doi.org/10.1609/aaai.v38i16.29770.

Michael Hahn. Theoretical Limitations of Self-Attention in Neural Sequence Models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. URL https://doi.org/10.1162/tacl_a_00306.

Yiding Hao, Dana Angluin, and Robert Frank. Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022. URL https://doi.org/10.1162/tacl_a_00490.

Michael J Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994. URL https://doi.org/10.7551/mitpress/3897.001.0001.

Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of cnf's based on short tree-like resolution proofs. *Electron. Colloquium Comput. Complex.*, 41, 1999. URL https://eccc.weizmann.ac.il//eccc-reports/1999/TR99-041.

Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996. URL https://doi.org/10.1017/CBO9780511574948.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *Proceedings of The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3EWTEy9MTM.

William Merrill and Ashish Sabharwal. The Parallelism Tradeoff: Limitations of Log-Precision Transformers. *Trans. Assoc. Comput. Linguistics*, 11:531–545, 2023. URL https://doi.org/10.1162/tacl_a_00562.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *Proceedings of the Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/pdf?id=CDmerQ37Zs.

Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the transformer architecture. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=KidynPuLNW.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing-Complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021. URL https://dl.acm.org/doi/10.5555/3546258.3546333.

Leonard Pitt and Leslie G Valiant. Computational Limitations on Learning from Examples. *Journal of the ACM (JACM)*, 35(4):965–984, 1988. URL https://doi.org/10.1145/48014.63140.

Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT (preliminary version). In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 128–136, 2000. URL https://dl.acm.org/doi/proceedings/10.5555/338219.

Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978. URL https://doi.org/10.1145/800133.804350.

Leslie G Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. URL https://doi.org/10.1145/1968.1972.

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*, volume 37, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/P19-1580.

Andy Yang and David Chiang. Counting like transformers: compiling temporal counting logic into softmax transformers. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FmhPg4UJ9K.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. *Advances in Neural Information Processing Systems*, 37:10202–10235, 2024. URL https://dl.acm.org/doi/proceedings/10.5555/3737916.