Ehrenfeucht-Haussler Rank and Chain of Thought

Pablo Barceló[1,2,3], Alexander Kozachinskiy[1], Tomasz Steifer[3,4]

[1]National Center for Artificial Intelligence (CENIA Chile)
[2]Millennium Institute for Foundational Research on Data (IMFD Chile)
[3]Institute for Mathematical and Computational Engineering,
Pontifical Catholic University of Chile
[4]Institute of Fundamental Technological Research, Polish Academy of Sciences

Abstract

The notion of *rank* of a Boolean function has been a cornerstone in the theory of 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) steps required by a single-layer transformer decoder with hard attention to compute f. Based on this characterization we establish tight bounds on the number of CoT steps required for specific problems, showing that ℓ-fold function composition necessitates exactly ℓ CoT steps. 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 steps.

1 Introduction

Ehrenfeucht and Haussler [6] 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 [19] 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 [12, 7].

In this paper, we present a new characterization of the notion of rank. Surprisingly, this characterization is grounded in the *Transformer architecture* [20], which has recently 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) steps 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 [8, 10, 2, 23] due to its amenability to formal analysis, while still effectively capturing the essence of practical models [4, 21].

The Transformer architecture is built upon *attention* layers and a *decoder*. An attention layer performs attention on the input sequence, mapping a sequence of input vectors to another sequence of vectors of the same length. Attention layers are used to generate vector representations of sentences in natural language. However, a more common application of Transformers is *sequence generation*, where the input sequence is mapped to an unbounded sequence of output vectors, generated iteratively, one at a time. This task is carried out by the decoder. In the first iteration, the decoder processes the input sequence through the attention layers and outputs the vector in the last position. This output is then appended to the input sequence. During subsequent iterations, the decoder applies its attention layers to the extended sequence, computes the next output, and appends it to the sequence. These are the CoT steps mentioned earlier [16, 14].

Below we summarize our main results:

- We show that the rank of a function f, denoted by $\mathsf{rk}(f)$, is the minimal number of iterations of a single-layer decoder with one hard-attention head that computes f. We establish our result not only for Boolean functions, generalizing the notion of the 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 iterations required by a single-layer decoder 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 $\mathsf{rk}(f)$. While 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*, a function whose complexity has been previously studied for single-layer decoders with soft attention [17]. 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 decoder, 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 we increase the number of attention heads. Namely, for any H, we show that $\mathsf{rk}^{(H)}(t\text{-}\mathsf{Comp}) = t$ for all large enough input lengths.

- Finally, we study the k-thOne function. This function takes as input a Boolean sequence of length n, and it 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 decoders, in the first iteration we can compute the position of the first one, then of the second one in the second iteration, and so on. We prove that for any H and for large enough n, we have $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}) = k$, showing that even increasing the number of attention heads we cannot improve upon the trivial solution for large enough input lengths. Interestingly, this result cannot be obtained via the communication complexity techniques used for iterated composition. Instead, our proof relies on a purely combinatorial argument.

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 [9, 18, 11, 1, 3, 15, 2, 16, 14, 22, 17]. In particular, several works have investigated how the capability of decoders depends on the number of iterations. To start with, Pèrez, Barceló, and Marinkovic [18] showed that decoders based on hard attention with an unbounded number of iterations are capable of computing any decidable language (with the parameters of the decoder not depending on the input length). Afterwards, the computation power of decoders with polynomially many iterations was addressed. Merrill and Sabharwal [16] have shown that in the uniform-regime (when, as in [18], parameters do not depend on the input length), such decoders with constant number of layers and softmax attention are capable of computing any polynomial-time language. Similarly, for the non-uniform regime, Liu, Liu, Zhou, and Ma [14] have shown that such decoders 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 decoders with a given fixed number of iterations, although just for a single layer and for hard attention. Recently, Peng, Narayanan, and Papadimitriou [17] have shown that any single-layer decoder with soft attention requires $\Omega(t)$ 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 instead 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.

2 Decision Trees and Rank

We use a notation $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{N}$.

Consider $n + 1$ finite sets $\Sigma_1, \ldots, \Sigma_n, O$, for $n > 0$. We are interested in decision trees that compute functions:

$$f \colon \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_n \to O.$$

To do this, we consider decision trees over arbitrary families of *queries*, where a query is a function q whose domain is $\Sigma_1 \times \ldots \times \Sigma_n$. We write $\mathrm{Im}(q)$ for the image of 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 $|\mathrm{Im}(q_v)|$ out-going 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 $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \times \Sigma_n$, the output of 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 $q(\bar{w})$. 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 $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$ is *computed* by T, if $T(\bar{w}) = f(\bar{w})$ for every input $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$.

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 $\Sigma_1 = \ldots = \Sigma_n = O = \{0,1\}$. *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 [6] 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 $\sigma \in \Sigma_i$, are called *assignments*. We denote by

$$A = \{1\} \times \Sigma_1 \cup \cdots \cup \{n\} \times \Sigma_n$$

the set of assignments. An assignment (i, σ) is *consistent* with an input $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \Sigma_n$ 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 $q_\tau : \Sigma_1 \times \ldots \times \Sigma_n \to A$, where τ is a permutation of the set of assignments A. For $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$, we let $k_{\bar{w}}$ be the minimal element $k \in \{1, \ldots, |A|\}$ such that $\tau(k)$ is consistent with $\bar{w}$. We then define $q_\tau(\bar{w}) = \tau(k_{\bar{w}})$.

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 $q_\tau(\bar{w}) = (i_k, \sigma_k)$.

We define the rank of an arbitrary function $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$ in terms of the class of decision trees over assignment queries that compute f.

Definition 1. *Let $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$. We define $\mathsf{rk}(f)$ as the minimal depth of a decision tree over a-queries that computes f.* □

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 1. *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. (rank $\implies$ a-query depth) 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 or 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 or 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 smaller rank. Consider a mapping ϕ that, to every non-leaf node v of T, it assigns a child of v such that the *other* child of v has 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 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.

(a-query depth $\implies$ rank) 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 out-going 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 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.

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 $i \in \{1, \ldots, n\}$ 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 $1 + \min\{r_0, r_1\}$. □

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 out 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.

By means of a combinatorial argument, it is possible to show that this is the best one can do if n is large enough.

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

Proof. Assume for 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 leafs, 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 was 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 to 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 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 3. *For any n, k, we have $\mathsf{rk}(k\text{-}\mathsf{thOne}_n) \leq k$, and for $n \geq k^2 + k$, 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 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 in a similar way, 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 $n \geq k^2+(k-1)$ 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 fixation 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. In fact, we have $m \leq d \leq k-1$ fixed positions. These positions split the remaining positions into at at most k consecutive intervals. Since $n \geq k^2+k$, one of these intervals I has length at least $k+1$. To the left of this interval, we have s positions fixed to 1, with $0 \leq s \leq k-1$. We fix the first $k-1-s$ positions of I to 1, hence before the kth position of I there are exactly $k-1$ ones. Both the k-th and the $(k+1)$-st positions of I thus can be the value of k-thOne_n. □

3 Attention Layers and Decoders

Attention layer. We consider layers with *unique hard attention*, and possibly multiple attention heads, where the output of the layer is computed in the last token. By unique hard attention we refer to the mechanism in which each position attends to the element with the highest attention score (breaking ties arbitrarily).

Formally, a *unique hard-attention layer* (or, simply, attention layer) with H heads and embedding dimension d is a function $L\colon (\mathbb{R}^d)^* \to \mathbb{R}^d$, which is defined by

- H *query* matrices $Q^{(h)} \in \mathbb{R}^{d\times d}$ and H *key* matrices $K^{(h)} \in \mathbb{R}^{d\times d}$, for $h = 1,\ldots,H$,
- two matrices $W_1, W_2 \in \mathbb{R}^{d\times d}$, and
- a matrix $W_O \in \mathbb{R}^{d\times(dH)}$.

Consider an input sequence of vectors $(x_1, \ldots, x_m) \in (\mathbb{R}^d)^m$. The output of L on $(x_1, \ldots, x_m)$ is computed as follows. For every $h = 1, \ldots, H$, we compute the *value of the h-th head* on $(x_1, \ldots, x_m)$, which is a vector from $\mathbb{R}^d$ denoted by $\mathsf{head}_h \in \mathbb{R}^d$. Namely, we start by computing "attention scores"

$$a_{i,m}^{(h)} = \langle K^{(h)} x_i, Q^{(h)} x_m \rangle, \tag{3}$$

defining, for every $i = 1, \ldots, m$, the *attention* from the last token to the i-th token with respect to the h-th head. The vector $K^{(h)} x_i$ is called the *key* of the i-th token, and the vector $Q^{(h)} x_m$ is called the *query* of the mth token.

For every $h = 1, \ldots, H$, we let $i_h \in \{1, \ldots, m\}$ to be the index maximizing (3). If there are multiple indices achieving the maximum, we let i_h be the leftmost one. We then set $\text{head}_h = u_{i_h}$, for $h = 1, \ldots, H$, and define:

$$\text{multihead} = W_O \cdot \begin{pmatrix} \mathsf{head}_1 \\ \vdots \\ \mathsf{head}_H \end{pmatrix} \in \mathbb{R}^d \tag{4}$$

Finally, we define:

$$L(x_1, \ldots, x_m) = W_2 \cdot \text{ReLU}\left(W_1(\text{multihead} + x_m)\right) \in \mathbb{R}^d.$$

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

Decoders. A *decoder*, defined by the d-dimensional attention layer L, is a function that takes on input a sequence of vectors $(x_1, \ldots, x_m) \in (\mathbb{R}^d)^m$ and in the output produces an infinite sequence of vectors $\{y_t \in \mathbb{R}^d\}_{t=1}^{\infty}$, defined by:

$$\begin{aligned} y_1 &= L(x_1, \ldots, x_m), \\ y_t &= L(x_1, \ldots, x_m, y_1, \ldots, y_{t-1}), \qquad t \geq 2. \end{aligned}$$

That is, the decoder works in iterations: first, it computes the output of L, adds it to the end of the input sequence, computes the output of L on the new sequence, adds this output to the end, and so on. We refer to y_t as the output of the decoder after t iterations (sometimes these iterations are called "chain of thought steps").

Computation of functions by decoders. Fix n and $n+1$ finite sets $\Sigma_1, \ldots, \Sigma_n, O$. We want to define how a decoder computes functions of the form:

$$f \colon \Sigma_1 \times \ldots \times \Sigma_n \to O.$$

Inputs to f are interpreted as words with n letters, with the i-th letter coming from the alphabet Σ_i, for $i = 1, \ldots, n$ (alphabets are possibly different at different positions). We put this word as an input to a decoder using $n+1$ tokens, one per letter plus a special token at the end for the "end of line" symbol. Input tokens can use arbitrary encodings of letters by d-dimensional vectors, potentially different at different positions of the input word, utilizing in this form a *positional* information. We then run the decoder on the resulting input for some number t of iterations. The output of f is computed by applying an output function to the decoder's output y_t from the final iteration.

Definition 2 (Computation of functions by decoders)**.** *Let n be a natural number and $\Sigma_1, \ldots, \Sigma_n, O$ be $n+1$ finite sets. A function $f \colon \Sigma_1 \times \ldots \times \Sigma_n \to O$ can be* computed by t iterations of a decoder with H heads*, if there exist:*

- *$d \in \mathbb{N}$ and an attention layer L of embedding dimension d with H heads,*

- *a* positional encoding p, *i.e. a function* $p\colon \Sigma_1 \times \{1\} \cup \ldots \cup \Sigma_n \times \{n\} \cup \{\mathsf{EoL}\} \to \mathbb{R}^d$, *where* EoL *denotes a special "end-of-line" symbol, and*
- *an* output function $\alpha : \mathbb{R}^d \to O$,

such that for any $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \times \Sigma_n$, *the value* $f(\bar{w})$ *is determined by the following procedure:*

1. *Define a sequence* $(x_1, \ldots, x_n, y_0)$ *of* d*-dimensional vectors by:*
$$x_1 = p(\sigma_1, 1),\ \ldots, x_n = p(\sigma_n, n),\ y_0 = p(\mathsf{EoL}).$$
2. *Place* $(x_1, \ldots, x_n, y_0)$ *as an input to the the decoder defined by* L*, and let* y_t *for* $t \geq 1$ *denote the output of this decoder after* t *iterations.*
3. *Set* $f(\bar{w}) = \alpha(y_t)$. □

Next, we define the following important notion.

Definition 3 (Decoder depth of a function)**.** *The* decoder depth with H heads *of* $f\colon \Sigma_1 \times \ldots \times \Sigma_n \to O$, *denoted* $\mathsf{dd}^{(H)}(f)$, *is the minimum* $t \geq 0$ *such that* f *can be computed by* t *iterations of a decoder with* H *heads.* □

As an illustration of these definitions, we provide a simple 1-head single-layer decoder, computing the t-Comp function in t iterations.

Proposition 4. *For any positive integers* t, n, *the function* t-Comp_n *can be computed by* t *iterations of a decoder layer with one head and embedding dimension six. Hence,* $\mathsf{dd}^{(1)}(t\text{-}\mathsf{Comp}_n) \leq t$.

Proof. We use the following positional encoding:

$$x_i = p(f(i), i) \mapsto \begin{pmatrix} 0 \\ \cos i \\ \sin i \\ \cos f(i) \\ \sin f(i) \\ f(i) \end{pmatrix}, \qquad y_0 = p(\mathsf{EoL}) \mapsto \begin{pmatrix} 0 \\ 0 \\ 0 \\ \cos 1 \\ \sin 1 \\ 1 \end{pmatrix},$$

where $i \in \{1, \ldots, n\}$. Let us fix the notation:

$$y_\ell = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \cos f^{(\ell)}(1) \\ \sin f^{(\ell)}(1) \\ f^{(\ell)}(1) \end{pmatrix} \tag{5}$$

for $\ell \geq 0$. Our goal is to devise a decoder layer whose output after the ℓ-th iteration is y_ℓ. Then, after t iterations, the sixth coordinate of y_t will be the output of the function.

We first observe that $p(\mathsf{EoL}) = y_0$. We need an attention layer L satisfying the following property:

$$L(x_1, \ldots, x_n, y_0, \ldots, y_\ell) = y_{\ell+1},$$

for every $\ell \geq 0$. We set $Q, K \in \mathbb{R}^{6\times 6}$ such that:

$$Q \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{pmatrix} = \begin{pmatrix} a_2 \\ a_3 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad K \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{pmatrix} = \begin{pmatrix} a_4 \\ a_5 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

We obtain the following attention "scores":

$$\langle Qx_i, Ky_\ell \rangle = \cos i \cdot \cos f^{(\ell)}(1) + \sin i \cdot \sin f^{(\ell)}(1), \qquad \langle Qy_j, Ky_\ell \rangle = 0,$$

for $i = 1, \ldots, n$ and $j = 0, \ldots, \ell$. The maximum of these expressions is 1, attained at $i = f^{(\ell)}(1)$. Thus, the value of the (unique) head will be:

$$\mathsf{head} = x_{f^{(\ell)}(1)} = \begin{pmatrix} 0 \\ \cos f^{(\ell)}(1) \\ \sin f^{(\ell)}(1) \\ \cos f^{(\ell+1)}(1) \\ \sin f^{(\ell+1)}(1) \\ f^{(\ell+1)}(1) \end{pmatrix}.$$

In (4), we consider the matrix $W_O \in \mathbb{R}^{6\times 6}$ that moves the 4th, 5th and 6th coordinate to the 1st, 2nd and 3rd coordinate, respectively, and writes 0 to the 4th, 5th and 6th coordinates, yielding:

$$\text{multihead} = W_O \cdot \mathsf{head} = \begin{pmatrix} \cos f^{(\ell+1)}(1) \\ \sin f^{(\ell+1)}(1) \\ f^{(\ell+1)}(1) \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

It remains to define the matrices $W_1, W_2 \in \mathbb{R}^6$ that define:

$$W_2 \cdot \mathrm{ReLU}\left(W_1(\text{multihead} + y_\ell)\right) = y_{\ell+1}.$$

Observe that:

$$\text{multihead} + y_\ell = \begin{pmatrix} \cos f^{(\ell+1)}(1) \\ \sin f^{(\ell+1)}(1) \\ f^{(\ell+1)}(1) \\ \cos f^{(\ell)}(1) \\ \sin f^{(\ell)}(1) \\ f^{(\ell)}(1) \end{pmatrix}, \qquad y_{\ell+1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \cos f^{(\ell+1)}(1) \\ \sin f^{(\ell+1)}(1) \\ f^{(\ell+1)}(1) \end{pmatrix}.$$

If we did not have the ReLU layer in middle of (3), we could set W_1 as the identity matrix and W_2 as the linear transformation that moves the 1st, 2nd and the 3rd coordinate to the 4th, 5th and 6th, respectively, and places 0 in the first 3 coordinates. However, if we do just this, ReLU can zero the 1st and the 2nd coordinates as they can be negative. To avoid this, we modify W_1 to add the third coordinate to the 1st and 2nd and W_2 to do the inverse linear transformation before redistributing coordinates as described above. □

4 One-Head Decoder Depth vs Tree Rank

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

Theorem 1. *For any function $f\colon \Sigma_1 \times \ldots \Sigma_n \to O$, we have $\mathsf{rk}(f) = \mathsf{dd}^{(1)}(f)$.*

As a corollary to Theorem 1 and Proposition 2, we obtain that for suitable n the decoder depth with one head of the iterated composition function t-Comp_n is precisely t:

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

Also, as a corollary to Theorem 1 and Proposition 3, we obtain that for suitable n the decoder depth with one head of the kth one function k-thOne_n is precisely k:

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

We now prove our main theorem.

Proof of Theorem 1. We first show the inequality $\mathsf{rk}(f) \leq \mathsf{dd}^{(1)}(f)$. Assume that f can be computed by a decoder with one head in r iterations, for some $r \in \mathbb{N}$. We deduce that $\mathsf{rk}(f) \leq r$. For that, we show that at the cost of t a-queries one can compute the outputs of the decoder in the first t iterations on a given input. Hence, in r a-queries, we can compute the rth output of the decoder, which uniquely determines the value of f, implying that $\mathsf{rk}(f) \leq r$.

Consider any input $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \times \Sigma_n$. Define then:

$$x_1 = p(1, \sigma_1),\ \ldots, x_n = p(n, \sigma_n),\ y_0 = p(\mathsf{EoL}) \in \mathbb{R}^d,$$

where d is the dimension of our decoder and p is its positional encoding function. Let $\{y_t \in \mathbb{R}^d\}_{t=1}^{\infty}$ be the sequence of the outputs of our decoder on input $(x_1, \ldots, x_n, y_0)$. Assume that we have already computed $y_1, \ldots, y_t$ for some $t \geq 0$ (if $t = 0$, we just know $y_0 = p(\mathsf{EoL})$). We explain how to compute y_{t+1} using one a-query. By definition,

$$y_{t+1} = L(x_1, \ldots, x_n, y_0, y_1, \ldots, y_t),$$

where L is the attention layer defining our decoder. It is enough to compute $s \in \{x_1, \ldots, x_n, y_0, y_1, \ldots, y_t\}$ with the maximal value of $\langle Ks, Qy_t \rangle$ for the key and query matrices $K, Q \in \mathbb{R}^{d \times d}$ of our attention layer. If there are multiple vectors $s \in \{x_1, \ldots, x_n, y_0, \ldots, y_t\}$ with the maximal value of this scalar product, we need to compute the leftmost one among them. Since we already have computed $y_0, y_1, \ldots, y_t$, it suffices to find this maximal s over $\{x_1, \ldots, x_n\} = \{p(1, \sigma_1), \ldots, p(n, \sigma_n)\}$.

Consider the following linear order of the set A of assignments. Given two different assignments $a = (i, \sigma), a' = (i', \sigma')$, we say that a is larger than a' if either $\langle Kp(a), Qy_t \rangle > \langle Kp(a'), Qy_t \rangle$ or $\langle Kp(a), Qy_t \rangle = \langle Kp(a'), Qy_t \rangle$ and $i < i'$. We arbitrarily order assignments with $\langle Kp(a), Qy_t \rangle = \langle Kp(a'), Qy_t \rangle$ and $i = i'$. Our task is to find the maximal assignment from $\{p(1, \sigma_1), \ldots, p(n, \sigma_n)\}$ in this order. For that, we ask the a-query q_τ for a permutation τ, where the first assignment is the maximal in our linear order, the second one is the second maximal, and so on.

We now show the inequality $\mathsf{dd}^{(1)}(f) \leq \mathsf{rk}(f)$. Assume that T is an r-depth decision tree over a-queries that computes f. We transform into a decoder with one head that computes f in r iterations. We assume that T is a complete r-depth $|A|$-ary tree, where A is the set of assignments.

The embedding dimension of our decoder will be:

$$\begin{aligned} d = {} & 1 + |A| + \ldots + |A|^{r-1} \\ & + 1 + |A| + \ldots + |A|^{r} \\ & + |A| \\ & + 1. \end{aligned}$$

The coordinates will be split into 4 groups:

- the first $1 + |A| + \ldots + |A|^{r-1}$ coordinates are called *positional coordinates* and are indexed by non-leaf nodes of T;
- the second $1 + |A| + \ldots + |A|^{r}$ coordinates are called *output coordinates* and are indexed by nodes of T;
- the third $|A|$ coordinates are called *assignment coordinates* and are indexed by assignments;
- the last coordinate will be called *special*.

Our goal is to construct a decoder that "simulates" T in the following sense. On input $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$, for any $t \geq 0$, we want the t-the output of the decoder, denoted by $y_t \in \mathbb{R}^d$, to be the one-hot encoding of the node where T comes on $\bar{w}$ at depth t. More specifically, this one-hot encoding will take place in output coordinates, the remaining coordinates of y_t will all be 0.

To achieve this, we start with defining $y_0 = p(\mathsf{EoL}) \in \mathbb{R}^d$ as follows. In the restriction to the output coordinates it is the one-hot encoding of the root of T; all the other coordinates of y_0 are 0. Next, we define the positional encoding $p(a) \in \mathbb{R}^d$ for an assignment $a = (i, \sigma) \in A$. In the restriction to the assignment coordinates, it is the one-hot encoding of a. Now, for each non-leaf node v of T and its corresponding positional coordinate $p(a)_v$, we set $p(a)_v = 1/\tau_v^{-1}(a)$, where $\tau_v \colon \{1, \ldots, |A|\} \to A$ is the permutation defining the a-query asked at v. We let the special coordinate of $p(a)$ to be 1. Finally, all output coordinates of $p(a)$ are set to 0.

Having our positional encoding defined, we move to the construction of the attention layer and define the query matrix $Q \in \mathbb{R}^{d \times d}$ by the following linear transformation $\alpha \mapsto Q\alpha$ for $\alpha \in \mathbb{R}^d$: for every non-leaf node v of T, the v-th positional coordinate of $Q\alpha$ is equal the v-th output coordinate of α; the remaining coordinates of $Q\alpha$ are 0. The key matrix $K \in \mathbb{R}^{d \times d}$ is set to be the identity matrix.

Assume that, as an input, for some $t < r$, we give to a layer the following sequence of vectors:

$$x_1, \ldots, x_n,\ y_0, y_1, \ldots, y_t \in \mathbb{R}^d,$$

where $x_i = p(i, \sigma_i)$ for $i = 1, \ldots, n$ and for some $\bar{w} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \times \Sigma_n$, $y_0 = p(\mathsf{EoL})$, and for every $i = 1, \ldots, t$, the vector y_i is the one-hot encoding, inside the output coordinates, of some depth-i node v_i of T, and has 0 in the remaining coordinates. Let $q = q_{v_t}$ be the a-query asked at v_t, and let $\tau = \tau_{v_t}$ be the corresponding permutation of the set of assignments (the node v_t is a non-leaf node because $t < r$). We claim that the attention on this input will be maximized for the position with the assignment which is the output of q on $\bar{w}$.

Indeed, the vector y_t has the unique 1 at the v_t-th output coordinate, with the remaining coordinates of v_t being 0. The matrix Q moves this 1 into the v_t-th positional coordinate, and the rest of the coordinates of Qy_t are 0. Thus, for any $\alpha \in \mathbb{R}^d$, the product $\langle K\alpha, Qy_t \rangle$ equals the value of α in the v_t-th positional coordinate. If $\alpha = p(i, \sigma_i)$ for $i \in [n]$, this value is $1/\tau^{-1}(i, \sigma_i)$. The maximum of this value is attained for $(i, \sigma_i) \in \{(1, \sigma_1), \ldots, (n, \sigma_n)\}$ with the minimal value of $\tau^{-1}(i, \sigma_i)$, i.e, for $(i, \sigma_i) = q(\bar{w})$. Now, for $\alpha \in \{y_0, y_1, \ldots, y_t\}$, the value of the v_t-th positional coordinate, as well as any other positional coordinate, is 0. Hence, the output of the head will be the vector $p(q(\bar{w}))$.

The output of the layer is now computed as:

$$y_{t+1} = W_2 \cdot \mathrm{ReLU}\left(W_1 \cdot \beta\right), \tag{6}$$

$$\beta = p(q(\bar{w})) + y_t. \tag{7}$$

We need to choose $W_1, W_2 \in \mathbb{R}^{d \times d}$ such that the resulting y_{t+1} will encode the node v_{t+1} where the tree goes from v_t by following the $q(\bar{w})$-labeled edge. More specifically, we want y_{t+1} to be the one-hot encoding of v_{t+1} in the output coordinates, and we want all the other coordinates of y_{t+1} to be 0. We will set W_2 to be the identity matrix. To define W_1, we fix the following notation. For a non-root node v of T, let $\mathit{parent}(v)$ denote the parent node of v, and let $\mathit{label}(v) \in A$ denote the label of the edge from $\mathit{parent}(v)$ to v. We define W_1 by the following linear transformation $\alpha \mapsto W_1\alpha, \alpha \in \mathbb{R}^d$: for every non-root node v of T, we define the v-th output coordinate of $W\alpha$ as

$$\text{the } \mathit{parent}(v)\text{-th output coordinate of } \alpha \tag{8}$$

$$+ \ \text{the } \mathit{label}(v)\text{-th assignment coordinate of } \alpha \tag{9}$$

$$- \ \text{the special coordinate of } \alpha. \tag{10}$$

We set all the other coordinates of $W_1\alpha$ to 0.

We have to show now that $\mathrm{ReLU}(W_1 \cdot \beta)$, with β as in (6–7) has 1 in the v_{t+1}-th output coordinate and 0 in all the other coordinates. Indeed, $W_1 \cdot \beta$ has 0 in any coordinate which is not an output coordinate for

a non-root node of T. Now, consider any non-root node v of T. It is enough to show that the v-th output coordinate of $W_1 \cdot \beta$ is 1 for $v = v_t$ and is 0 or -1 for $v \neq v_t$ (applying ReLU to 0 and -1, we get 0).

To calculate the v-th output coordinate of $W_1\beta$, as stated in (8–10), we calculate the $parent(v)$-th output coordinate of β, the $label(v)$-th assignment coordinate of β, and the special coordinate of β. Recall that positional encodings of assignments have 0 in the output coordinates. Hence, the sum $\beta = p(q(\bar{w})) + y_t$, in the restriction to the output coordinates, is the one-hot encoding of v_t. In other words, the $parent(v)$-th output coordinate of β is the indicator $\mathbb{I}\{parent(v) = v_t\}$. Likewise, since y_t has only 0 in the non-output coordinates, the sum $\beta = p(q(\bar{w})) + y_t$, in the restriction to the assignment coordinates, is the one-hot encoding of the assignment $q(\bar{w})$. Again, this means that the $label(v)$-th assignment coordinate of β is equal to the indicator $\mathbb{I}\{label(v) = q(\bar{w})\}$. Finally, the special coordinates of $p(q(\bar{w}))$ and y_t are 1 and 0, respectively, meaning that the special coordinate of β is 1. Plugging these equalities into (8–10) for $\alpha = \beta$, we obtain that the v-th output coordinate of $W_1\beta$ equals:

$$\mathbb{I}\{parent(v) = v_t\} + \mathbb{I}\{label(v) = q(\bar{w})\} - 1.$$

This expression takes values in $\{-1, 0, 1\}$ and it is equal to 1 if and only if both indicators are 1. It remains to note that v_{t+1} is the only node whose parent is v_t and such that the label of the edge from v_t to this node is $q(\bar{w})$.

The r-th output of the decoder, y_r, in restriction to the output coordinates, will be the one-hot encoding of the leaf to which we come while computing T on input $\bar{w}$. Since this leaf uniquely determines $f(\bar{w})$, we are done. □

5 Multihead Rank

In order to generalize Theorem 1 to decoders with many heads, we define the notion of H-head rank for a function $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$. For that we require a notion of the *product* of two functions with the same domain. Namely, by the product of $g\colon A \to B$ and $h\colon A \to C$, 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 4. *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 1 allows us to obtain the following result.

Theorem 2. *For any $H \in \mathbb{N}$ and for any function $f\colon \Sigma_1 \times \ldots \times \Sigma_n \to O$, we have $\mathsf{rk}^{(H)}(f) = \mathsf{dd}^{(H)}(f)$.*

Proof. We first show that $\mathsf{rk}^{(H)}(f) \leq \mathsf{dd}^{(H)}(f)$. The proof for the case $H = 1$ works almost verbatim for the general case. As we have shown in the proof of Theorem 1, for a given decoder with 1 head, knowing the first t outputs on an input $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$, we can compute the value of the head (which would give us the $(t+1)$-st output), asking one a-query about $\bar{w}$. For H-head decoders, we simply compose H a-queries for each of H heads into a single H-degree a-query.

We now establish the inequality $\mathsf{dd}^{(H)}(f) \leq \mathsf{rk}^{(H)}(f)$. Assume that T is an r-depth decision tree over H-degree a-queries, computing f. In the construction of Theorem 1, we need to multiply the number of positional and assignment coordinates by H. Positional coordinates are now indexed by pairs (v, i), where v is a non-leaf node of T and $i \in [H]$ (with i referring to one of the H a-queries, asked at v). Likewise, assignment coordinates are now indexed by pairs (a, i), where a is an assignment and $i \in [H]$.

The positional encoding of the assignment a is modified as follows. As before, output coordinates of $p(a)$ are 0 and the special coordinate of $p(a)$ is 1. Next, $p(a)$ has 1 in the assignment coordinate, indexed by $(a, 1)$, and 0 in the remaining assignment coordinates. Finally, for a non-leaf node v of T and $i \in [H]$, and for the corresponding positional coordinate $p(a)_{v,i}$, we set $p(a)_{v,i} = 1/\tau_{v,i}^{-1}(a)$, where $\tau_{v,i}$ is the permutation

for the i-th a-query, asked at the node v. With this, we have that the closer a to being the first position in the permutation $\tau_{v,i}$, the higher the value of $p(a)_{v,i}$ is.

As before, our goal is to maintain that y_t, the t-th output of the decoder on input $\bar{w} \in \Sigma_1 \times \ldots \times \Sigma_n$, encodes the node v_t, which is the t-depth node of T where this tree comes on input $\bar{w}$. More specifically, we want y_t to have 1 in the v_t-th output coordinate and 0 in all the other coordinates. We achieve this for $y_0 = p(\mathsf{EoL})$ by defining $p(\mathsf{EoL})$ to have 1 in the output coordinate, corresponding to the root of T, and 0 in all the other coordinates.

Assume that this invariant is maintained for v_t. Our positional encoding is defined in such a way that the i-th attention head, for the right choice of key and query matrices, will return $p(a_i)$, where a_i is the output of the i-th a-query at v_t on input $\bar{w}$. For that, we just need to define $Q^{(i)} \in \mathbb{R}^{d \times d}$, the query matrix of the i-th head, as the matrix of a linear transformation that moves the v-th output coordinate to the (v,i)-th positional coordinate (and the key matrix $K^{(i)}$ is set to be the identity matrix). Then the scalar product $\langle K^{(i)}\beta, Q^{(i)}y_t\rangle$ will be equal to the (v_t,i)-th positional coordinate of β. For $\beta = p(a)$, this coordinate is inversely proportional to the position of a in the permutation $\tau_{v_t,i}$, and for $\beta = y_\ell$, $\ell = 0,\ldots,t$, the value of this coordinate is 0. Hence, it will be maximized at a_i, and the output of the i-th head will be $\mathsf{head}_i = p(a_i)$.

Next, our goal now is to define a matrix $W_O \in \mathbb{R}^{d \times dH}$ in (4) such that the vector

$$\mathsf{multihead} = W_O \cdot \begin{pmatrix} \mathsf{head}_1 \\ \vdots \\ \mathsf{head}_H \end{pmatrix} \in \mathbb{R}^d$$

will be the one-hot encoding of a_1 in the first $|A|$ assignment coordinates, the one-hot encoding of a_2 in the second $|A|$ assignment coordinates, and so on. Notice that $\mathsf{head}_1,\ldots,\mathsf{head}_H$, in the restriction to the first $|A|$ assignment coordinates, are one-hot encodings of $a_1,\ldots,a_H$, respectively. Thus, it remains to define W_O to be the matrix of a linear transformation that copies the first $|A|$ assignment coordinates of head_1 to the first $|A|$ assignment coordinates of $\mathsf{multihead}$, the first $|A|$ assignment coordinates of head_2 to the second $|A|$ assignment coordinates of $\mathsf{multihead}$, and so on.

As a result, we can make the sum $y_t + \mathsf{multihead}$ to be a Boolean vector with exactly $H+1$ ones that one-hot encodes v_t (in the output coordinates), and also, for $i \in [H]$, one-hot encodes a_i in the i-th $|A|$ assignment coordinates. We now achieve that the next output of the decoder:

$$y_{t+1} = W_2 \cdot \mathrm{ReLU}\left(W_1(\mathrm{multihead} + y_t)\right)$$

one-hot encodes (in the output coordinates) the node v_{t+1} , which is a child of v_t followed by the $(a_1,\ldots,a_t)$-labeled edge. We set W_2 to be the identity matrix. As for W_1, we use the same trick as in the end of the proof of Theorem 1. Namely, we notice that for each potential value of v_{t+1}, there is precisely one possible value of v_t and $a_1,\ldots,a_H$. This allows us to express any output coordinate of $W_1(\mathrm{multihead} + y_t)$ as a logical conjuction of $H+1$ coordinates of $\mathsf{multihead} + y_t$. The conjunction of $H+1$ bits $b_0, b_1, \ldots, b_H$ can be written as $\mathrm{ReLU}(b_0 + b_1 + \ldots + b_H - (H-1))$, giving us an expression for the matrix W_1 (where we use the special coordinate of the input to express $H-1$) □

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 5. *For $f : \Sigma_1 \times \ldots \times \Sigma_n \to O$ and $H \geq 1$, we have $\mathsf{rk}(f) \leq H \cdot \mathsf{rk}^{(H)}(f)$.*

Proposition 5 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 precise lower bound on the decoder depth of a function with H heads, it suffices to derive a lower bound on its H-head rank (Theorem 2). 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 2, 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 3.

5.1 Multihead decoder depth of iterated composition

In this section, we show a method for lower bounding the multihead rank of a function based on communication complexity [13]. 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 $out\colon \{0,1\}^{\ell_1+\ldots+\ell_k} \to \mathcal{Z}$.

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 $out(m_1 \ldots m_k) \in \mathcal{Z}$.

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 letter 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 knows, 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 1. *For every $f\colon \Sigma_1 \times \ldots \times \Sigma_n \to \{0,1\}$, for every $S \subseteq [n]$, and for every $H \geq 1$, denoting $r = \mathsf{rk}^{(H)}(f)$ and $|A|$ the number of assignments for f, we have:*

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

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 the 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 3. *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 $f_A \colon \{1, \ldots, m\} \to \{1, \ldots, m\}$ and Bob is given $f_B \colon \{1, \ldots, m\} \to \{1, \ldots, m\}$. In the k-step pointer chase, denoted here by PC_k^m, the goal of Alice and Bob is to compute:

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

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 $C^{k,B}(\mathrm{PC}_k) = \Omega(m)$ [5].

It remains to notice that $\mathrm{PC}_t^{n/2}$ is a special case of the problem $t\text{-}\mathsf{Comp}_n^S$, for $S = \{1, \ldots, n/2\}$, where Alice gets $(\phi(1), \ldots, \phi(n/2))$ and Bob gets $(\phi(n/2+1), \ldots, \phi(n))$, for some function $\phi \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, 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. Assuming that $\mathsf{rk}^{(H)}(t\text{-}\mathsf{Comp}_n) < t$, by Lemma 1 we obtain:

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

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

5.2 Multihead decoder depth of the k-th one

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

Theorem 3. *For any $k, H \in \mathbb{N}$, for all but finitely many $n \in \mathbb{N}$, we have $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}_n) = k$.*

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 6. *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 1 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 $S \subseteq [n]$. Instead, we use a self-reducibility technique by means of partial fixations.

Proof of Theorem 3. 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 derive the theorem for k. More specifically, for any H we construct a function $f\colon \mathbb{N} \to \mathbb{N}$ with $\lim_{n\to\infty} f(n) = +\infty$ such that the following holds: for any n, any H-degree decision tree T for $k\text{-}\mathsf{thOne}_n$ can be transformed into an H-degree decision tree of smaller depth for $(k-1)\text{-}\mathsf{thOne}_{f(n)}$.

Let us first finish the induction step assuming the above statement is proven. We have to show the existence of n_0 such that for all $n \geq n_0$, we have $\mathsf{rk}^{(H)}(k\text{-}\mathsf{thOne}_n) \geq k$. By the induction hypothesis, there exists n_1 such that for all $n \geq n_1$, we have $\mathsf{rk}^{(H)}((k-1)\text{-}\mathsf{thOne}_n) \geq k-1$. It is enough to take n_0 such that $f(n) \geq n_1$ for all $n \geq n_0$. Such n_0 exists because $\lim_{n\to\infty} f(n) = +\infty$. Indeed, assume for contradiction the existence of a $(k-1)$-depth H-degree decision tree, computing $k\text{-}\mathsf{thOne}_n$ for some $n \geq n_0$. We deduce the existence of a $(k-2)$-depth H-degree decision tree, computing $(k-1)\text{-}\mathsf{thOne}_{f(n)}$. This gives a contradiction since $f(n) \geq n_1$.

We now show the existence of a function f with the above properties. Take an H-degree decision tree T, computing $k\text{-}\mathsf{thOne}_n$. Let d be the depth of T and let $q = q_{\tau_1} \otimes \ldots \otimes q_{\tau_H}$ be the H-degree a-query at the root of T.

By a *partial input* we mean a string $y \in \{0,1,*\}^n$. A string $x \in \{0,1\}^n$ is a complementation of a partial input $y \in \{0,1,*\}$ if $x_i = y_i$ for every $i \in [n]$ with $y_i \in \{0,1\}$. Our task is to find a partial input y with the following properties:

- (a) all complementations of y have the same value of Q;
- (b) between the first fixed 1 and the second fixed 1, there are at least $f(n)$ unfixed positions in y.

Such y yields a $(d-1)$-depth H-degree decision tree for $(k-1)\text{-}\mathsf{thOne}_{f(n)}$ as follows. Consider any $z \in \{0,1\}^{f(n)}$. Fill the unfixed positions in y before the first fixed 1 by 0s, and unfixed positions after the second fixed 1 by 1s. To the unfixed positions between the first and the second 1, put bits of z. Let $x \in \{0,1\}^n$ be the resulting vector. Observe that $k\text{-}\mathsf{thOne}_n(x)$ determines $(k-1)\text{-}\mathsf{thOne}_{f(n)}(z)$. Observe also that any a-query about x can be computed, asking one a-query about z. It thus remains to compute the value of T on x. We can skip the first query of T because all complementations of y, one of which is x, have the same value of q. This gives us a depth-$(d-1)$ H-degree decision tree for $(k-1)\text{-}\mathsf{thOne}_{f(n)}$.

We now give a partial input y with the above properties. We gradually construct y by fixing more and more positions and making sure that all complementations of y have the same values on all a-queries $q_{\tau_1}, \ldots, q_{\tau_H}$, constituting q. We proceed in at most H iterations. After ℓ iterations, we make sure that the following two conditions are met: (a) at least ℓ a-queries out of $q_{\tau_1}, \ldots, q_{\tau_H}$ are already "fixed", meaning that any two complementations of the current y have the same values on any of these ℓ a-queries; (b) before the first fixed 1, there are $n_\ell \geq n^{2^{-\ell}}$ unfixed positions in y.

For $\ell = 0$, these conditions are trivially met. Assume that we have performed $\ell \geq 0$ iterations. For notational simplicity, assume that a-queries that are still not fixed are $q_{\tau_1}, \ldots, q_{\tau_{H-\ell}}$. We now need the following definition.

Definition 5. *Let B be a finite set and $\gamma, \tau\colon \{1, \ldots, |B|\} \to B$ be two its permutations. We say that τ is **close** to γ if $\gamma^{-1}(\tau(j)) \leq j + \sqrt{|B|}$ for every $j \in \{1, \ldots, |B|\}$ (the j-th element of the permutation τ has position at most $j + \sqrt{|B|}$ in the permutation γ, for every j). Otherwise, we say that τ is **far** from γ.*

Set $m = n_\ell$, let $i_1 < i_2 < \ldots < i_m$ be unfixed positions before the first fixed 1 in y, and let $B = \{(i_1, 1), \ldots, (i_m, 1)\}$ be the set of assignments to value 1 at these positions. We consider a permutation γ of B where assignments go in the increasing order of the indices of their positions:

$$\gamma = (i_1, 1) \ldots (i_m, 1).$$

We "compare" γ with restrictions of $\tau_1, \ldots, \tau_{H-\ell}$ to B (recall that τ is the permutation corresponding to a-query q_τ). If one of these restrictions is far from γ, we do one more iteration. If all these restrictions are close to γ, we finish the construction of y.

In more detail, assume first that one of the restrictions is far from γ, say, τ_1. Then for some $j \in \{1, \ldots, m\}$, the j-th element of this restriction has position at least $j + \sqrt{m} + 1$ in γ. In other words, for some $r \geq j + \sqrt{m} + 1$, there are most $j - 1$ assignments in B that precede $(i_r, 1)$ in τ_1. We now try to fix τ_1 while maintaining at least $\sqrt{m} \geq n^{2^{-\ell-1}}$ unfixed positions before the first fixed 1 in y, proceeding after that to the next iteration.

We go through all the assignments of τ_1, starting with $\tau_1(1)$, then $\tau_1(2)$, and so on. For each assignment (i, σ) under consideration, the value at i-th position of y is either already fixed or we fix it according to the rule defined in the next paragraph. If the assignment is fixed to $\neg\sigma$, we go to the next assignment of τ_1. If it is fixed to σ, we stop – τ_1 is now fixed.

Our fixation rule for unfixed positions in y is as follows: given (i, σ), we fix the i-th position to σ unless $(i, \sigma) \in B \setminus \{(i_r, 1)\}$. As a result, the position of the first fixed 1 moves to i_r, or it stays the same (because among the positions $i_1, \ldots, i_m$, we are only allowed to fix i_r to 1). In any case, observe that we do not go any further in τ_1 than the assignment $(i_r, 1)$. Hence, in our process, we can only fix the positions of the assignments in B that precede $(i_r, 1)$ in τ_1 (there are at most $j - 1$ of them) and one final position more. In particular, among positions $i_1, \ldots, i_{r-1}$, at least $r - 1 - j \geq \sqrt{m}$ positions will remain unfixed, and all of them will precede the first fixed 1 in y, even if it moves to i_r.

Finally, we stop repeating iterations once all restrictions of $\tau_1, \ldots, \tau_{H-\ell}$ are close to γ. The construction of y is finished with the use of the following combinatorial lemma.

Lemma 2. *For any fixed h and for all sufficiently large m, the following holds. Let B be a finite set of size m, and let $\gamma, \tau_1, \ldots, \tau_h$ be its permutations such that $\tau_1, \ldots, \tau_h$ are all close to γ. Then there exists $r \in [1, m/2 + \sqrt{m}]$ such that $\gamma(r)$ has the position at most $m/2$ in all permutations $\tau_1, \ldots, \tau_h$.*

Proof. For every $s = 1, \ldots, h$, let us put a mark on the elements $\tau_s(1), \ldots, \tau_s(m/2)$. Since $\tau_1, \ldots, \tau_h$ are all close to γ, we mark only the first $m/2 + \sqrt{m}$ elements in the permutation γ. We need to show that one of these elements gets h marks. Indeed, overall we put $hm/2$ marks. Since $hm/2 > (h-1)(m/2 + \sqrt{m})$ for large enough m, we conclude that for these m it is impossible that all elements get at most $h - 1$ marks. □

Using the lemma, we take $r \in [1, \ldots, m/2 + \sqrt{m}]$ such that for every $s = 1, \ldots, h = H - \ell$, the position of $(i_r, 1)$ in τ_s is at most $m/2$. We now fix all permutations $\tau_1, \ldots, \tau_h$ exactly in the same way as before. Namely, for each τ_s, we consider assignments $\tau_s(1), \tau_s(2), \tau_s(3), \ldots$ one by one, and we fix a corresponding position in a way that fixes τ_s unless it would involve the assignment from $B \setminus \{(i_r, 1)\}$. Thus, when considering τ_s, among positions $i_1, \ldots, i_m$, we fix only positions for the assignments of B that precede $(i_r, 1)$ in τ_s, and possibly one final position more. The assignment $(i_r, 1)$ has the position at most $m/2$ in τ_s, which means that all B-assignments that precede it in τ_s have the positions at most $m/2 + \sqrt{m}$ in γ since τ_s is close to γ. In other words, when fixing τ_s, we fix at most 1 position among $i_{r+1}, \ldots, i_m$.

Doing so for all permutations $\tau_1, \ldots, \tau_h$, we obtain that among $i_1, \ldots, i_m$, only position i_r can be fixed to 1, and among $i_{r+1}, \ldots, i_m$, at most $h = O(1)$ are fixed to 0. As a result, all a-queries are fixed, and between the first fixed 1 (which is now at i_r) and the second fixed one there are at least $m - r - O(1) \geq m/3 \geq (1/3)n^{2^{-H}} = f(n)$ unfixed positions, as required. □

References

[1] Angluin, D., Chiang, D., and Yang, A. Masked hard-attention transformers and boolean RASP recognize exactly the star-free languages. *CoRR abs/2310.13897* (2023).

[2] Barceló, P., Kozachinskiy, A., Lin, A. W., and Podolskii, V. V. 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* (2024), OpenReview.net.

[3] Chiang, D., Cholak, P., and Pillay, A. Tighter bounds on the expressivity of transformer encoders. In *ICML* (2023), vol. 202, pp. 5544–5562.

[4] Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. 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* (2019), T. Linzen, G. Chrupala, Y. Belinkov, and D. Hupkes, Eds., Association for Computational Linguistics, pp. 276–286.

[5] Duris, P., Galil, Z., and Schnitger, G. Lower bounds on communication complexity. *Information and Computation 73*, 1 (1987), 1–22.

[6] Ehrenfeucht, A., and Haussler, D. Learning decision trees from random examples. *Information and Computation 82*, 3 (1989), 231–246.

[7] Esteban, J. L., and Torán, J. A combinatorial characterization of treelike resolution space. *Information Processing Letters 87*, 6 (2003), 295–300.

[8] Hahn, M. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics 8* (2020), 156–171.

[9] Hahn, M. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguistics 8* (2020), 156–171.

[10] Hao, Y., Angluin, D., and Frank, R. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics 10* (2022), 800–810.

[11] Hao, Y., Angluin, D., and Frank, R. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics 10* (2022), 800–810.

[12] Kullmann, O. Investigating a general hierarchy of polynomially decidable classes of cnf's based on short tree-like resolution proofs. Citeseer.

[13] Kushilevitz, E., and Nisan, N. *Communication Complexity*. Cambridge University Press, 1996.

[14] Liu, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. In *ICLR* (2024).

[15] Merrill, W., and Sabharwal, A. The parallelism tradeoff: Limitations of log-precision transformers. *Trans. Assoc. Comput. Linguistics 11* (2023), 531–545.

[16] Merrill, W., and Sabharwal, A. The expressive power of transformers with chain of thought. In *ICLR* (2024).

[17] Peng, B., Narayanan, S., and Papadimitriou, C. H. On limitations of the transformer architecture. *CoRR abs/2402.08164* (2024).

[18] Pérez, J., Barceló, P., and Marinkovic, J. Attention is turing-complete. *J. Mach. Learn. Res. 22* (2021), 75:1–75:35.

[19] Pudlák, P., and Impagliazzo, R. A lower bound for dll algorithms for k-sat (preliminary version). In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* (2000), pp. 128–136.

[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS* (2017), pp. 5998–6008.

[21] Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. 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* (2019), Association for Computational Linguistics.

[22] Yang, A., and Chiang, D. Counting like transformers: Compiling temporal counting logic into softmax transformers. *CoRR abs/2404.04393* (2024).

[23] Yang, A., Chiang, D., and Angluin, D. Masked hard-attention transformers recognize exactly the star-free languages. In *NeurIPS* (2024).