Saturated Transformers are Constant-Depth Threshold Circuits

William Merrill[*†] **Ashish Sabharwal**[*] **Noah A. Smith**[*‡]
[*]Allen Institute for AI, USA [†]New York University, USA [‡]University of Washington, USA
willm@nyu.edu {ashishs, noah}@allenai.org

Abstract

Transformers have become a standard neural network architecture for many NLP problems, motivating theoretical analysis of their power in terms of formal languages. Recent work has shown that transformers with *hard* attention are quite limited in power (Hahn, 2020), as they can be simulated by constant-depth AND/OR circuits (Hao et al., 2022). However, hard attention is a strong assumption, which may complicate the relevance of these results in practice. In this work, we analyze the circuit complexity of transformers with *saturated* attention: a generalization of hard attention that more closely captures the attention patterns learnable in practical transformers. We first show that saturated transformers transcend the known limitations of hard-attention transformers. We then prove saturated transformers with floating-point values can be simulated by constant-depth threshold circuits, giving the class TC^0 as an upper bound on the formal languages they recognize.

1 Introduction

Opening the ''black box'' (Alishahi et al., 2020) of the representations within neural networks is an important step towards building systems with robust and interpretable behavior. In NLP, one part of this question is analyzing the languages that networks can model, and the mechanisms they use to represent linguistic structure and dependencies.

One path toward this goal is via formal analysis of specific network architectures (Merrill, 2021); for example, recurrent neural networks (RNNs). Due to their autoregressive formulation, formal linguistic analysis of RNNs has often characterized their power by relating them to automata-theoretic classes of formal languages (Weiss et al., 2018; Peng et al., 2018; Merrill, 2019, inter alia). Recently, however, RNNs have largely been overtaken in NLP by a new class of models: transformers (Vaswani et al., 2017). Transformers are not autoregressive, and therefore less naturally resemble automata, posing challenges to characterizing their linguistic capacity or inductive biases in the same terms as RNNs. Instead, some recent work has related them to circuit complexity classes, a direction that we continue to pursue in this paper. Drawing on classical circuit lower bound results, Hao et al. (2022) and Hahn (2020) derive theoretical limitations of transformers with *hard* attention, meaning the attention distributions focus all probability mass on one index. Together, their results show that AC^0—the class of languages recognizable by constant-depth circuit families—upper bounds the formal languages hard-attention transformers can recognize.

However, hard attention is a strong assumption, making it unclear how these results transfer to practical transformers. For example, Bhattamishra et al. (2020) showed how transformers can solve synthetic counting tasks by using uniform attention patterns, which hard attention does not allow. Motivated by this potential disconnect between theory and practice, we aim to extend circuit-based analysis to transformers with saturated attention: a generalization of hard attention that has been argued to approximate attention patterns acquired through gradient descent (Merrill et al., 2021). Broadly speaking, saturated attention goes beyond hard attention in that it can ''tie'' across a subset of positions, rather than selecting just *one* position. The tied positions are then aggregated by averaging. Qualitatively, saturated attention heads can ''count'': a capability observed in transformers in practice (Bhattamishra et al., 2020). Further, Merrill et al. (2021) show that transformer training dynamics lead attention heads in several pretrained transformers to approximate saturated attention. In summary, saturated attention strictly generalizes hard attention and should more closely reflect the attention patterns acquired in practical transformers.

Our main contributions are twofold. First, we show that saturated transformers can recognize languages outside AC^0. Then, as depicted in Table 1, we prove that transformers with floating

Transactions of the Association for Computational Linguistics, vol. 10, pp. 843–856, 2022. https://doi.org/10.1162/tacl_a_00493
Action Editor: Mark Johnson. Submission batch: 9/2021; Revision batch: 2/2022; Published 8/2022.

	Float ($\mathbb{F}$)	Rational ($\mathbb{Q}$)
Hard (η)	$\subseteq \mathsf{AC}^0$	$\subseteq \mathsf{AC}^0$
Saturated (ζ)	$\subseteq \mathsf{TC}^0$	$\subseteq \mathsf{ALL}$

Table 1: Summary of combined results from Hao et al. (2022) and this paper. Each cell $\alpha, \mathbb{D}$ characterizes the languages recognizable by transformers with attention function α and datatype $\mathbb{D}$ (floats $\mathbb{F}$ or rationals $\mathbb{Q}$). TC^0 and TC^0 are circuit complexity classes, and $\mathsf{AC}^0 \subset \mathsf{TC}^0$. ALL is the set of all formal languages over alphabet $\{0, 1\}$. See §4 for formal definitions. Out of these results, we view saturated attention with floats as the best model of practical transformers.

point activations and saturated attention can only recognize formal languages in the circuit complexity class TC^0, constituting an upper bound for a more realistic model of transformers than past results with hard attention.

2 Roadmap

In §3, we formally define our model of the transformer, including defining saturated attention in contrast to hard attention. §4 introduces circuits in theoretical computer science and relevant complexity measures and classes for them.

In §5, we first briefly analyze saturated transformers with rational values where the embedding, scoring, and activation functions are allowed to be any size-preserving function. We find such transformers to be universally powerful. We also observe that when the positional embeddings are computed in time linear in the sequence length, saturated rational-valued transformers are exactly as powerful as the complexity class of their activation functions, because the full input sequence can be pooled to a single position, and an activation function can be used as an oracle over the full input sequence. However, this setup relies on the use of unrealistic embedding functions. To move to a more realistic model of computation, we then focus on saturated transformers whose values are restricted to be *floats*, which have a coarser granularity and, thus, cannot encode the full input sequence into a single position.

Building on results of Pérez et al. (2019), we demonstrate in §6 that saturated transformers with *float* activations transcend the theoretical limitations of hard-attention transformers. In particular, we will show that they can recognize the majority language, which lies outside AC^0. We experimentally validate that transformers can learn to recognize the majority language. Taken together, these results suggest that the very weak characterization of hard-attention transformers does not hold in practice for saturated or soft attention.

In §7, we show that, on input sequences of length n, the size of each state vector in a transformer over floats is $\mathrm{O}(\log n)$ bits, similar to saturated LSTMs (cf. Merrill, 2019). Thus, the full transformer state at any given layer has size $\mathrm{O}(n \log n)$, although each feedforward block can only locally access a small, $\mathrm{O}(\log n)$ ''piece''. Thus, while hierarchical representations can be implemented in a transformer (e.g., to process arbitrary-depth Dyck languages or reverse strings as in Weiss et al. [2021]), our result implies that they must be *distributed* in some way across n state vectors, rather than represented compactly within a single vector.

Finally, in §8, we use the bounded size of transformer representations to upper bound the formal languages that can be recognized by saturated transformers with floating-point values. In particular, we show that such transformers can be simulated by constant-depth threshold circuits, and thus only recognize languages in TC^0. Informally, this suggests that moving from hard attention to saturated attention can be thought of as extending the implicit class of circuit gates available in the network to include threshold gates.

Our results make progress in the analysis of transformers by deriving upper bounds for a more realistic model of transformers than has previously been analyzed. RoBERTa, T5, and other pretrained transformers have been shown to be approximately saturated (Merrill et al., 2021), so our results imply that TC^0 may be a meaningful upper bound on the computation expressible within such networks. Our analysis also motivates future work further refining the circuit characterization of saturated transformers, as well as comparing transformers with soft and saturated attention.

3 Definitions and Notation

We will often use w to refer to a string over any generic alphabet Σ, that is, $w \in \Sigma^*$. Semantically, w corresponds to the string a transformer receives as input. In contrast, we use x and other symbols to refer to binary strings in $\{0, 1\}^*$. These binary strings will represent intermediate values within

the transformer computation, rather than the raw input to the transformer.

3.1 Datatypes

Under our model, all values in the transformer are binary strings. In order to compute self attention and other operations over binary strings, we need to define datatypes describing the semantics of these binary strings as numbers. We will describe a semantics for binary strings as integers, as often comes up in circuit complexity. We then extend this to rational numbers and floats, which are necessary for representing the division operations that occur in attention heads within transformers.

Unsigned Integers We can interpret binary strings $x \in \{0,1\}^*$ as unsigned integers in the standard way, namely, the numerical value of $x \in \{0,1\}^n$ is

$$[\![x]\!]_{\mathbb{Z}} = \sum_{i=1}^{n-1} 2^{i-1} x_i.$$

We allow standard integer operations like $+_{\mathbb{Z}}$, $\cdot_{\mathbb{Z}}$, $<_{\mathbb{Z}}$. For example, $101 +_{\mathbb{Z}} 1 = 110$.

Rationals To interpret $r \in \{0,1\}^*$ as a rational number, we first view it as a sign bit s along with a tuple of two unsigned integer substrings $\langle p, q \rangle$.[1] The numerical value represented by r is

$$[\![r]\!]_{\mathbb{Q}} = (2s-1)[\![p]\!]_{\mathbb{Z}} / [\![q]\!]_{\mathbb{Z}}.$$

Let $\text{red}(p,q)$ return $\langle s, t \rangle$ where $s = p/\gcd(p,q)$ and $t = q/\gcd(p,q)$. Then, we can define arithmetic operations over two rationals $r = \langle p, q \rangle$ and $r' = \langle p', q' \rangle$ in the standard way:

$$r +_{\mathbb{Q}} r' = \text{red}(p \cdot_{\mathbb{Z}} q' + p' \cdot_{\mathbb{Z}} q,\ q \cdot_{\mathbb{Z}} q')$$
$$r \cdot_{\mathbb{Q}} r' = \text{red}(p \cdot_{\mathbb{Z}} p',\ q \cdot_{\mathbb{Z}} q').$$

Floats We define floats $\mathbb{F}$ as the subset of the rationals where the denominator is constrained to be a power of 2.[2] Multiplication and addition are defined as for $\mathbb{Q}$, and are guaranteed to produce another float. Notably, division for floats is implemented by multiplying by an approximate multiplicative inverse, so it may be that $(x /_{\mathbb{F}} y) \cdot_{\mathbb{Q}} y \neq x$. See Appendix A for a more formal discussion.

In §5, we will study transformers over rational values. From §6 onwards, we will then take the values in transformers to be floats unless otherwise stated. Going forward, we will generally omit datatype subscripts from operations where they are clear from context. We will sometimes write $\mathbb{D}$ as a set in function signatures, for example, $f : \mathbb{D}^k \to \mathbb{D}^k$. In this usage, it refers to the set $\{0,1\}^*$, but it is often more intuitive to write the datatype shorthand (rather than $\{0,1\}^*$) to hint at the intended semantics of the functional arguments.

Size of Binary Strings Under our model, integers, rationals, and floats are all abstractions built out of binary strings. For any $x \in \{0,1\}^*$ (which can be interpreted semantically as an integer, float, or rational), we define its size $|x|$ as the total length of x measured in bits. We imagine a tuple $\langle p, q \rangle$ is encoded by padding p, q to the same length with leading 0's, and interleaving bits from each sequence. This means the size of a rational is $2\max(|p|,|q|) + 1$. For example, the integer 2 takes 2 bits to specify, while the float $\frac{1}{2}$ takes 5 bits (1 for the sign, 2 for the numerator, 2 for the denominator).

Size Preservation We say that a function $f : \{0,1\}^* \to \{0,1\}^*$ is size-preserving iff there exist constants c, n such that for all inputs x with $n \leq |x|$, $|f(x)| \leq c \cdot |x|$. Let $\mathcal{P}$ be the set of size-preserving functions. While size-preserving functions are defined here over binary strings, they can be equivalently applied over integers, rationals, and floats, since these datatypes, as we have defined them, are just binary strings.

3.2 Transformers

We define the following general transformer model, which can be parameterized to use different types of attention patterns and whose internal functions (e.g., feedforward blocks) can be computed by different function classes.

Definition 1 (Transformer)**.** A *transformer* is a tuple $\langle \Sigma, \mathbb{D}, \alpha, L, H, \phi, \{s_{\ell,h}\}_{\ell,h=1}^{L,H}, \{f_\ell\}_{\ell=1}^{L} \rangle$ where

1. Σ is a finite input alphabet, that is, the set of token types in a formal language.
2. $\mathbb{D}$ is a scalar datatype, that is, a semantics for interpreting binary strings as numbers. We will generally consider $\mathbb{D} = \mathbb{F}$.

[1] Under the hood, we imagine the pair $\langle p, q \rangle$ is encoded by padding p and q to the same length with 0's and interweaving bits from each.

[2] More generally, the denominator may be taken to have a prime factorization of bounded length, although we work with the power of 2 definition, which is both simpler and closely resembles conventional floating point datatypes.

3. α is an attention function that maps a vector of attention scores in $\mathbb{D}^n$ (for any n) to a normalized probability distribution, also in $\mathbb{D}^n$. In this paper we take α to be either hard (η) or saturated (ζ) attention; see §3.3.
4. $L \in \mathbb{N}$ is the number of layers.
5. $H \in \mathbb{N}$ is the number of heads.
6. $\phi : \Sigma \times \mathbb{N} \to \mathbb{D}^m$ is a position-aware embedding function that maps a token and position to a vector, where m is a multiple of H.
7. For each ℓ, h, the function $s_{\ell,h} : \mathbb{D}^m \times \mathbb{D}^m \to \mathbb{D}$ assigns attention scores to pairs of values.
8. For each ℓ, the function $f : \mathbb{D}^m \times \mathbb{D}^m \to \mathbb{D}^m$, maps a previous layer value and attention head output to a new value vector.

On an input string $w \in \Sigma^n$, a transformer computes L layers of output sequences $v_{\ell,1}, \cdots, v_{\ell,n}$ (for $\ell \leq L$), where each $v_{\ell,i} \in \mathbb{D}^m$. In the 0th layer, each token w_i and its position i are embedded into a value $v_{0,i}$. Subsequent layers aggregate information from the previous value sequence v_ℓ using a *multi-head attention mechanism*, and output a new value sequence $v_{\ell+1}$. More formally, these layers are structured as follows:

1. **Embedding Layer:** $v_{0,i} = \phi(w_i, i)$.
2. **Attention Head:** Each of the H attention heads in layer ℓ maps the full previous sequence into a new value via $s_{\ell,h}$ and then applies the attention function α:

$$a_{\ell,h,i,j} = s_{\ell,h}(v_{\ell,i}, v_{\ell,j})$$
$$b_{\ell+1,h,i} = \sum_{j=1}^{n} \alpha(a_{\ell,h,i,:})_j \cdot v_{\ell,j}.$$

   Crucially, the semantics for addition and multiplication here (as well as in the computation of α) come from the datatype $\mathbb{D}$.
3. **Activation Block:**[3]

$$v_{\ell+1,i} = f_{\ell+1}(v_{\ell,i}, b_{\ell,:,i}).$$

[3]Let $V_{\ell,h}$ be a head's value matrix in the standard transformer parameterization. Then f_ℓ is computed by first multiplying each $b_{\ell,h,i}$ by $V_{\ell,h}$, aggregating the multiple attention heads, and applying the feedforward subnetwork.

3.3 Attention Functions

An attention function α maps a vector of scores $a \in \mathbb{D}^n$ to a probability distribution over $1, \cdots, n$. Specifically, we consider two attention functions: *hard* attention $\eta(a)$ and *soft* attention $\zeta(a)$.

Hard attention collapses the attention scores to a one-hot distribution with all mass concentrated at one index. Let $\mathcal{M}(a) = \{i \mid a_i = \max_j a_j\}$.

Definition 2 (Hard attention)**.** Define hard attention $\eta(a)$ as

$$\eta(a)_j = \begin{cases} 1 & \text{if } j = \min_{m \in \mathcal{M}(a)} m \\ 0 & \text{otherwise.} \end{cases}$$

In contrast, saturated attention spreads probability mass evenly across ''tied'' scores.

Definition 3 (Strong saturated attention; Merrill et al. 2021)**.** Define saturated attention $\zeta(a)$ as

$$\zeta(a)_j = \frac{1}{|\mathcal{M}(a)|} \cdot \begin{cases} 1 & \text{if } j \in \mathcal{M}(a) \\ 0 & \text{otherwise.} \end{cases}$$

Merrill (2019) shows how this form of attention can be derived by taking a large-norm limit of the network weights; a derivation can be found there. Saturated attention reduces to hard attention when $|\mathcal{M}(a)| = 1$, and attends uniformly when $|\mathcal{M}(a)| = n$. Both hard and uniform attention can be implemented with numerical stability, motivating *weak* saturated (or, ''uniform'') attention:

Definition 4 (Weak saturated attention)**.** Each head implements either hard attention (Definition 2) *or* the uniform pattern $\upsilon(a)_j = \frac{1}{n}$.

In general, we will use ''saturated attention'' to refer to strong saturated attention and provide upper bounds for this setting. On the other hand, our lower bounds only use weak saturated attention, thereby showing that even weak saturated attention is more powerful than hard attention.

3.4 Language Recognition

Finally, we define language recognition for transformers.

Definition 5 (Language recognition)**.** Write $v_{\ell,i}(w)$ for the value of $v_{\ell,i}$ on input string w. A transformer recognizes a language $\mathcal{L} \subseteq \Sigma^*$ if

there exists a $\mathbb{D}$-valued affine transformation W, b such that, for all $w \in \Sigma^*$,

$$W \cdot v_{L,1}(w) + b > 0 \iff w \in \mathcal{L}.$$

This says that the decision problem of recognizing $\mathcal{L}$ must be linearly separable using the first value in the last layer of the transformer. In practice, the first token in a transformer is often set to `CLS`, and its output can be passed to a classifier during finetuning (Devlin et al., 2019). This inspires Definition 5. There are other potential ways to define language recognition and generation for transformers (Hewitt et al., 2020; Yao et al., 2021), but they do not lead to meaningful differences for our purposes.

Finally, we define $\mathsf{AHAT}(\mathbb{D})$ as the set of languages recognizable by some saturated transformer over $\mathbb{D}$, where the internal functions can be any size-preserving function.[4]

Definition 6. Let $\mathsf{AHAT}(\mathbb{D})$ be the set of languages $\mathcal{L}$ such that there exists a transformer $\langle \Sigma, \mathbb{D}, \zeta, L, H, \phi, s_{\ell,h}, f_\ell \rangle$ that recognizes $\mathcal{L}$ where each $\phi, s_{\ell,h}, f_\ell \in \mathcal{P}$.[5]

We note that size preservation is a weak condition to assume about the internal functions in practical transformers: Because any linear-time-computable function is size-preserving, it is strictly weaker than assuming that the internal functions can be computed in linear time. To further justify this condition, we explicitly show in Appendix B that the component functions within transformers are size-preserving.

4 Circuit Complexity

Circuit complexity is a branch of computational complexity theory that studies circuit families as a model of computation.[6] Intuitively, circuits are useful for formally studying the types of computational problems that can be efficiently solved with parallelism, as the depth of a circuit corresponds to the runtime of a program on an idealized, fully parallel computer. We review background on circuits, circuit families, and relevant complexity measures and classes.

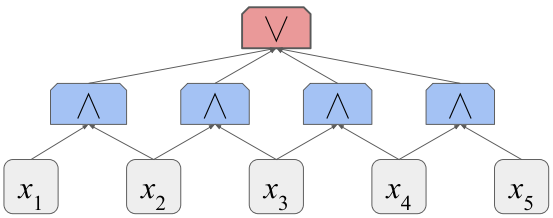

Figure 1: A circuit that takes a string $x \in \{0,1\}^5$ and returns whether it contains the bigram 11.

Circuits For a fixed n, a *circuit* is a computation graph, where leaves correspond to input bits x_i and their negations $\neg x_i$, and the internal nodes are logic gates (typically $\wedge$ and $\vee$), with one labeled as the output node. The gates can conventionally be taken to have either binary or unbounded fan-in. The circuit computes a function $f : \{0,1\}^n \to \{0,1\}$ by substituting the input values into the leaf nodes, propagating the computation through the graph, and returning the value of the output node. Figure 1 shows an example circuit that takes inputs of length 5, and returns whether they contain the bigram 11.

Circuit Families A *circuit family* is an ordered set of circuits $\{C_n\}_{n \in \mathbb{N}}$ where each circuit is identified with a particular input size n. We say a circuit family recognizes a formal language $\mathcal{L} \subseteq \{0,1\}^*$ iff, for all $w \in \mathcal{L}$,[7]

$$C_{|w|}(w) = 1 \iff w \in \mathcal{L}.$$

Circuit Complexity Two important notions of complexity for a circuit are its size and depth. The size of a circuit is the number of gates. The depth is the longest path from an input node to the output node. For a circuit family, both quantities can be expressed as functions of the input size n. A *circuit complexity class* is a set of formal languages that can be recognized by circuit families of a certain size, depth, and set of gates. In particular, we will discuss the classes AC^0 and TC^0.

Definition 7. AC^0 is the set of languages $\mathcal{L} \subseteq \{0,1\}^*$ such that there exists a circuit family recognizing $\mathcal{L}$ with unbounded arity $\{\wedge, \vee\}$ gates, $\text{poly}(n)$ size, and $\mathrm{O}(1)$ depth.

[4] The name AHAT standards for ''averaging hard attention transformer'', and is taken from Hao et al. (2022).

[5] To apply size preservation to the embedding function ϕ, we consider the size of a token to be $\log(|\Sigma|)$.

[6] For more reference material on circuit complexity, we refer the reader to chapters 6 and 14 of Arora and Barak (2009) or chapters 1 and 2 of the *Handbook of Theoretical Computer Science*, Volume A (van Emde Boas, 1991; Johnson, 1991).

[7] Similarly, for any alphabet Σ and $\mathcal{L} \subseteq \Sigma^*$, we interpret w_i as a one-hot vector over Σ and define the family to recognize $\mathcal{L}$ iff, for all $w \in \mathcal{L}$, $C_{|w| \cdot |\Sigma|}(w) = 1 \iff w \in \mathcal{L}$.

Intuitively, AC^0 represents the class of problems that are highly parallelizable when the computational primitives are standard logic gates. In contrast, TC^0 will also represent highly parallelizable computation, but when the gates are expanded to include *threshold gates*.

For a bitstring $x \in \{0,1\}^*$, define the threshold gate $\theta_{\geq k}(x)$ to return 1 iff $\geq k$ bits in x are 1, and equivalently for $\leq k$. For example, $\theta_{\geq 3}(110011) = 1$.

Definition 8. TC^0 is the set of languages $\mathcal{L} \subseteq \{0,1\}^*$ such that there exists a circuit family recognizing $\mathcal{L}$ with unbounded arity $\{\wedge, \vee, \theta\}$ gates, $\text{poly}(n)$ size, and $\mathrm{O}(1)$ depth.

It is known that $\mathsf{AC}^0 \subset \mathsf{TC}^0 \subseteq \mathsf{NC}^1$, where NC^1 denotes the languages recognizable by $\mathrm{O}(\log n)$-depth circuits with bounded gate arity. Whether or not the latter containment between TC^0 and NC^1 is strict is an open question. Whereas parity and other basic regular languages are outside AC^0 (Furst et al., 1981), TC^0 properly contains parity, although it is unknown whether it contains *all* the regular languages. Between AC^0 and TC^0 lies the class ACC^0 (Yao, 1990).

Uniformity The circuit classes defined above (and which we will use in this paper) are *non-uniform*, meaning circuits for different input sizes are not constrained to have any relation to each other. Non-uniform circuit families can recognize some uncomputable languages, such as the language of strings 1^k such that Turing machine k does not halt on the null input (cf. Arora and Barak, 2009). In contrast, the *uniform* variants of circuit families are constrained such that a log-space Turing machine must output a string encoding of circuit C_n on the input string 1^n, forcing any language the circuit family can recognize to be computable. For these uniform classes (which we write with a *u* prefix), it is known that

$$u\mathsf{TC}^0 \subseteq u\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{P},$$

where L and P denote the conventional complexity classes of log-space and polynomial-time decision problems. Thus, it is unknown whether $u\mathsf{TC}^0$ is restricted compared to general polynomial-time computation, but if we accept the common conjecture that one (if not all) of the above containments are strict, then $u\mathsf{TC}^0$ forms a restricted family of problems compared to P, which, intuitively, are more parallelizable than other problems in P.

5 Aren't Transformers Universal?

We now begin our analysis of saturated transformers. Hao et al. (2022) and Hahn (2020) were able to give upper bounds on the power of hard attention without imposing any constraints on the embedding, scoring, and activation functions. The same will not be the case with saturated attention: any bounds on transformers will require leveraging some properties constraining their internal functions. One property we use will be size preservation. We will first show, though, that size preservation is not enough on its own: Deriving a nontrivial upper bound will depend on subtle assumptions about the transformer's datatype.

With rational values and size-preserving internal functions, we will show saturated transformers can recognize *any* formal language, namely, the class $\mathsf{ALL} = \{\mathcal{L} \mid \mathcal{L} \subseteq \{0,1\}^*\}$. Our construction resembles the universal approximation construction of Yun et al. (2020), which relies on the ability of the transformer to uniquely encode the full input string into a single value vector. After the full sequence is encoded locally into a single vector, the activation block can be used as a black box to recognize any language.

Theorem 1. $\mathsf{AHAT}(\mathbb{Q}) = \mathsf{ALL}$.

Proof. We construct a 1-layer rational-valued transformer with a single head to recognize every string w in any formal language $\mathcal{L} \in \mathsf{ALL}$. We will omit ℓ, h subscripts. Let p_i denote the ith prime number. The embedding layer encodes each input token according to

$$\phi(w_i, i) = \begin{cases} 1/p_i & \text{if } w_i = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Since $p_i \sim i \log i$ for large i by the prime number theorem (cf. Goldstein, 1973), the number of bits needed to represent the denominator of $\phi(w_i, i)$ is

$$\leq c \log(i \log i) \leq c \log(i^2) = 2c \log i.$$

Since i had size $\log i$, this implies ϕ is size-preserving.

Now, we define a single uniform attention head that sums across all i, outputting $\sum_{w_i=1} \frac{1}{p_i}$. The denominator q of this sum is the product $\prod_{i=1} p_i$. Observe that $w_i = 1$ iff p_i divides q. Thus, we

can define a function f that extracts the input sequence w from q by checking whether, for each i, p_i divides q. We let g be a function recognizing L, and set $f = g \circ f$. The output of the transformer will now compute whether $w \in \mathcal{L}$, since f outputs an encoding of the original input sequence w, and g decides whether $w \in \mathcal{L}$. Note that any function solving a decision problem is size-preserving, hence $f \in \mathcal{P}$. □

Theorem 1 says that our transformer architecture parameterized with a rational datatype can recognize any formal language. But a construction of this form feels unrealistic for two reasons. First, it requires the embedding layer to implement an unconventional prime encoding scheme in the embedding layer. Second, we are using the activation layer as a black box to recognize any language—even uncomputable ones! On the other hand, the feedforward subnetworks used in practice in transformers cannot even implement all computable functions when the weights are fixed independent of the sequence length n. We can get around both these issues by instead restricting the datatype to floats, which is the direction we will pursue in the remaining sections.[8]

5.1 Resource-Bounded Transformers

In Appendix C, we develop an alternate perspective on the universality of transformers, showing that, if the embedding function is allowed to be computed in time linear in the sequence length, then the transformer's complexity is equivalent to its activation functions' complexity.

Theorem 2 (Informal)**.** *If ϕ can be any function computable in time linear in n, and the scoring and activation functions can be computed in $T(m)$ time on inputs of size m with $T(m) \geqslant m$, then languages recognizable by the transformer are* $\mathsf{TIME}(T(m))$.

Appendix C contains a formal statement and proof. For example, allowing polynomial-time functions inside the transformer implies that the transformer will recognize exactly the complexity class P. A major unrealism about this setup is the assumption that ϕ can be an arbitrary function computable in time linear in n, motivating our main results in a more constrained setting in §8.

5.2 Discussion

We are not stating the results in this section as evidence that practical transformers are capable of universal or arbitrary polynomial computation. Rather, the unnaturalness of these constructions (specifically, the prime numbers based position encoding) motivates us to slightly constrain our model of the transformer in a realistic way: We will switch the datatype from rationals to floats, because even using only simple uniform attention, a model with rationals and unconstrained internal functions is universal. We will soon see that this realistic constraint prevents universal simulation, and in fact bounds the capacity of the saturated transformer within TC^0.

6 Beyond Hard Attention, with Floats

We now move to the setting of saturated transformers over floats. Hao et al. (2022) identified that hard-attention transformers can only recognize languages within AC^0. In contrast, saturated transformers over floats can recognize the ''majority'' language MAJ, which is known to lie outside AC^0 (Furst et al., 1981). Pérez et al. (2019, Prop. 3.3) show how MAJ can be recognized by transformers. In Theorem 3, we offer a simpler construction that leverages only a single uniform attention head, as opposed to the model of transformers they were considering. Thus, this construction is achievable with saturated attention.

Theorem 3. $\mathsf{AHAT}(\mathbb{F}) \not\subseteq \mathsf{AC}^0$.

Proof. Let $\#_\sigma(w) \in \mathbb{N}$ denote the number of σ tokens in string $w \in \{0,1\}^*$. Let $\#(w)$ denote a count vector where each element corresponds to some $\sigma \in \{0,1\}$. We define MAJ as follows:

$$\text{MAJ} = \left\{w \in \{0,1\}^+ \mid \#_1(w) > \#_0(w)\right\}.$$

We will construct a 1-layer transformer with a single head to recognize MAJ, omitting ℓ, h subscripts from s, f, x, b. Figure 2 gives the same construction in RASP (Weiss et al., 2021).

Let $x_i = \phi(w_i, i)$ be a 1-hot encoding of w_i. For all i, j, set $s(x_i, x_j) = 1$, resulting in a single head attending everywhere:

$$b_i = \frac{\#(w)}{n}.$$

[8] It may also be possible to derive tighter bounds for rational-valued transformers by imposing stronger constraints on the internal functions. However, with floats, we will see that size preservation is sufficient to derive a tighter characterization of transformers' power. We leave this alternate direction to future work.

```
frac0 = aggregate(
    select_all,
    indicator(tokens == 0));
frac1 = aggregate(
    select_all,
    indicator(tokens == 1));
maj = frac1 > frac0;
```

Figure 2: A program recognizing MAJ in RASP, a programming language designed to abstract away details of transformer computation (Weiss et al., 2021). `frac{0,1}` measure the fraction of inputs that are 0 or 1. Then `maj` checks whether `frac1 > frac0`.

Finally, set $f(b_i)$ to return whether $\#_1(w)/n > \#_0(w)/n$, which, for $n > 0$, is true iff $w \in$ MAJ. □

Notably, the construction in Theorem 3 is not just possible within our generalized transformer framework, but can also be implemented by the standard parameterization of ϕ, s, and f in real transformers (Vaswani et al., 2017). The uniform attention pattern can be implemented by setting all query and key attention parameters to 0. Then, we can use the affine transformation that aggregates the head outputs to compute the tuple:

$$\left\langle \frac{\#_1(w) - \#_0(w)}{n}, 0 \right\rangle.$$

This tuple is then passed through layer normalization (Ba et al., 2016), resulting in a new tuple $\langle t_1, t_2 \rangle$. Crucially, $t_1 > t_2$ iff the same applies to the quantities in the original tuple. Thus, a linear classifier can decide whether $t_1 > t_2$ to successfully recognize the language, as per Definition 5.

6.1 Empirical Validation

In Figure 3, we show empirically that a 1-layer transformer can learn and generalize MAJ. This supports our argument that the theoretical limitations of hard-attention transformers do not apply to practical transformers. We train with three different types of positional encoding: none, meaning no positional information; learned, where each position gets a trainable embedding vector, and the sinusoidal scheme of Vaswani et al. (2017). The model with no positional embeddings generalizes the best, followed by the learned embeddings. It appears that while MAJ is in the capacity of the transformer, the standard sinusoidal positional embedding scheme provides the wrong inductive bias for learning it. This recalls the finding of Yao et al.

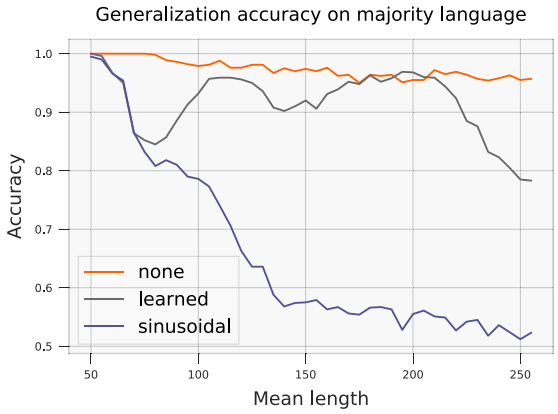

Figure 3: In practice, transformers can learn the majority language (which lies outside AC^0). We train 1-layer transformers on majority, where each line represents a different positional encoding scheme. Training string length was binomial with $n = 100$. Trained models were then evaluated on generalization sets with n ranging from 100 to 500. Mean length (x axis) is $n/2$.

(2021) that the choice of positional encodings seems to greatly impact the transformer's ability to generalize formal language tasks to longer sequences.

7 Size of Transformer Values

The theoretical limits on hard-attention transformers were derived by Hao et al. (2022) by bounding the size in bits of the representation $v_{\ell,i}$ at each layer ℓ and position i. Specifically, they show that the value $v_{\ell,i}$ is representable in $\mathrm{O}(\log n)$ bits on input sequences of length n. Thus, each value can only contain limited information about the input sequence, intuitively explaining their upper bound on hard-attention transformers. Inspired by their analysis, this section will show that, in a saturated transformer, each $v_{\ell,i}$ also has a size of $\mathrm{O}(\log n)$ bits. Later in §8, we will use this property to show that saturated attention transformers are limited in the formal languages they can recognize.

7.1 Size of Float Sums

How many bits does it take to represent the value of an attention head within a saturated transformer? As a naive bound, the output of a saturated attention head is specified by a float for each of n values attended over from the last layer, which would take at least linearly many bits in n. However, this upper bound on its size is not tight. Instead, we will show that all head and activation

values can be represented in $\mathrm{O}(\log n)$ bits. Our analysis will rely heavily on the following lemma:

Lemma 1. *Let $v_1, \cdots, v_n$ be a sequence of floats, each with size at most z. Then there exists c such that $\sum_{i=1}^{n} v_i$ has size at most $4cz + 2\log n + 1$.*

Proof. Let p_i, q_i denote the numerator and denominator of the floating point v_i, respectively. Similarly, let p_s, q_s be the numerator and denominator of the float s. By assumption, there exists c such that each p_i, q_i both have size $\leq cz$ for large enough n. We let $p_{\max} = \max_i p_i$ and analogously for $q_{\max}$. Because all q_i's are powers of 2, the numerator p_s is

$$\sum_{i=1}^{n} p_i \cdot \frac{q_{\max}}{q_i} \leq n p_{\max} q_{\max},$$

which, represented as an integer, has size:

$$\leq \log n + cz + cz = 2cz + \log n.$$

On the other hand, the denominator $q_s = q_{\max}$, which has size $\leq z$. Therefore, the float representing the sum has size

$$\begin{aligned} &\leq 1 + 2\max(2cz + \log n, z) \\ &= 4cz + 2\log n + 1, \end{aligned}$$

which completes the proof. □

In particular, we will use Lemma 1 to show that, when each of a sequence of n values has size $\mathrm{O}(\log n)$, the sum will also have size $\mathrm{O}(\log n)$.

7.2 Size of Transformer Values

We will now leverage Lemma 1 to show that the values are of bounded size in any transformer over floats with an elementwise-size-preserving attention function.

Definition 9. A function $\alpha : \mathbb{D}^n \to \mathbb{D}^n$ is elementwise-size-preserving if, for $1 \leq i \leq n$, the function $x_i \mapsto \alpha(x)_i$ is size-preserving (where $x \in \mathbb{D}^n$).

Note that saturated attention satisfies this definition. We are ready to prove a theorem bounding the size of the representations in transformers with elementwise-size-preserving attention.

Theorem 4. *For any transformer over $\mathbb{F}$ with ϕ, $s_{\ell,h}, f_\ell \in \mathcal{P}$ and α elementwise-size-preserving, for all $\ell \leq L$ and $i \leq n$, $v_{\ell,i}$ has size $\mathrm{O}(\log n)$.*

Proof. By induction over ℓ. The proof follows the definition of transformer computation in §3.2.

Base Case $w_i \in \Sigma$ has size $\mathrm{O}(1)$, and $i \in [n]$ has size $\mathrm{O}(\log n)$. Since $\phi \in \mathcal{P}$, $v_{0,i} = \phi(w_i, i)$ has size $\mathrm{O}(\log n)$ for all i.

Inductive Case Assume $v_{\ell,i}$ has size $\mathrm{O}(\log n)$. Since $s_{\ell+1,h} \in \mathcal{P}$, $a_{\ell+1,h,i,j} = s_{\ell+1,h}(v_{\ell,i}, v_{\ell,j})$ has size $\mathrm{O}(\log n)$ for all i, j. Since α is elementwise-size-preserving, we can conclude that $\alpha(a_{\ell+1,h,i,:})_j$ also has size $\mathrm{O}(\log n)$ for all h, i, j. Multiplying two floats is size-preserving (cf. Appendix B), so $\alpha(a_{\ell+1,h,i,:})_j \cdot v_{\ell,j}$ has size $\mathrm{O}(\log n)$ for all h, i, j. We then apply Lemma 1 to conclude that $b_{\ell+1,h,i}$ has size $\mathrm{O}(\log n)$, where, recall,

$$b_{\ell+1,h,i} = \sum_{j=1}^{n} \alpha(a_{\ell,h,i,:})_j \cdot v_{\ell,j}.$$

Finally, computing $v_{\ell+1,i} = f_{\ell+1}(v_{\ell,i}, b_{\ell,:,i})$, we conclude that $v_{\ell+1,i}$ has size $\mathrm{O}(\log n)$ for all i by size preservation. □

Corollary 4.1. *For any saturated transformer over $\mathbb{F}$ with size-preserving internal functions, for all $\ell \leq L$ and $i \leq n$, $v_{\ell,i}$ has size $\mathrm{O}(\log n)$.*

Corollary 4.1 follows because saturated attention is elementwise-size-preserving. Softmax attention, on the other hand, is not guaranteed to fulfill this property, because it requires computing the exponential function. This technical challenge prevents generalizing our technique to soft attention.

7.3 Discussion

Similar to hard-attention transformers (Hao et al., 2022), the size of each vector representation in a saturated transformer over floats is $\mathrm{O}(\log n)$. This is enough memory for individual vectors to ''count'', a behavior that has been observed in both LSTMs (Weiss et al., 2018) and transformers (Bhattamishra et al., 2020). On the other hand, $\mathrm{O}(\log n)$ space is not enough memory for individual vectors (for example, the `CLS` output) to encode arbitrarily large combinatorial objects like trees. However, transformers are *not* limited to computing in an ''online'' fashion where tokens are consumed sequentially, meaning that their effective state is n values of size $\mathrm{O}(\log n)$. Notably, trees with n leaves can be encoded in a distributed fashion across n values of size $\mathrm{O}(\log n)$. One

construction for this is, at index i, to store w_i and i, along with a pointer j to the parent. Since i, j can both be represented in $\log n$ bits, each vector uses only $\mathrm{O}(\log n)$ bits.

Additionally, the $\mathrm{O}(\log n)$ space bound has implications from the perspective of circuit complexity. While saturated attention cannot be simulated in AC^0, we will show in §8 that saturated transformers over $\mathbb{F}$ *can* be simulated by TC^0 circuits.

8 Threshold Circuit Simulation

We have proved that each value vector in a saturated transformer over floats has $\mathrm{O}(\log n)$ size. Now, we show how this implies saturated transformers can be simulated by TC^0 circuits. Our results heavily leverage the following lemmas:

Lemma 2 (Hao et al. 2022)**.** *Any function $f : \{0,1\}^c \to \{0,1\}^d$ can be computed by a Boolean circuit of depth 3 and size at most $(2^c + c + 1)d$.*

So that our results are self-contained, we reproduce a proof of this lemma in Appendix D. Applying Lemma 2 to a size-preserving function with at most $c \log n$ input bits immediately yields:

Corollary 2.1. *Any size-preserving function with at most $c \log n$ input bits can be computed by a Boolean circuit of depth 3 and polynomial size.*

In other words, such functions can be computed with AC^0 circuits. In addition, we will show that the sum of n floats of size at most $c \log n$ can be computed by TC^0 circuits.

Lemma 3. *Let $v_1, \cdots, v_n$ be a sequence of floats, each with size at most $c \log n$ for some c. Then the sum $\sum_{i=1}^{n} v_i$ is computable by a threshold circuit of constant depth and polynomial size.*

Proof. Let the integers p_i, q_i be the numerator and denominator of v_i. We first compute q_{max}, the maximum q_i, using an AC^0 circuit that compares all pairs q_i, q_j, and returns the first q_i such that $q_j \geq q_j$ for all j. We then use the fact that multiplication and right shift (q_i is a power of 2) are in TC^0, in order to compute

$$r_i = p_i \frac{q_{\text{max}}}{q_i}$$

in parallel for all i. Note that q_{max} and q_i are both powers of 2, so the division will be exact. Next, we leverage the fact that the sum of n integers of size $\mathrm{O}(\log n)$ is in TC^0 (Kayal, 2015), in order to compute the numerator of the sum $p' = \sum_i r_i$. We select the denominator as $q' = q_{\text{max}}$. Finally, we can add an AC^0 circuit that ''reduces'' the fraction by removing shared trailing zeros from p', q', which is possible by Corollary 2.1. Thus, we have constructed a TC^0 circuit to compute the sum of n floats with size $\mathrm{O}(\log n)$. □

We now construct a TC^0 circuit that simulates a saturated transformer over floats.

Theorem 5. $\mathsf{AHAT}(\mathbb{F}) \subseteq \mathsf{TC}^0$.

Proof. For each n, we construct a TC^0 circuit that simulates a saturated transformer on inputs of size n. We construct the circuit modularly, with one subcircuit for the attention mechanism, and another for the feedforward subnetwork.

Attention Head Fix a single head in some layer. We will construct a TC^0 subcircuit that simulates the attention mechanism at position i. The head attends over vectors $v_1, \cdots, v_n$. For all j, v_j has size $\mathrm{O}(\log n)$ by Theorem 4. In parallel for each j, we compute the scores $a_{i,j} = s(v_i, v_j)$ with an AC^0 circuit by Corollary 2.1. We then compute $a_{i,\text{max}} \triangleq \max_j a_{i,j}$ with an AC^0 circuit by comparing all v_j pairwise, and selecting the first v_k such that $v_k \geq v_j$ for all j. We then compute ''masked'' values $u_{i,j}$ for each j via an AC^0 circuit by Lemma 2:

$$u_{i,j} \triangleq \begin{cases} v_j & \text{if } a_{i,j} \geq a_{i,\text{max}} \\ 0 & \text{otherwise.} \end{cases}$$

We then compute the sum $s_i \triangleq \sum_{j=1}^{n} u_{i,j}$ by Lemma 3. By Lemma 1, s_i has size $\mathrm{O}(\log n)$. Now, we similarly define

$$z_{i,j} \triangleq \begin{cases} 1 & \text{if } a_{i,j} \geq a_{i,\text{max}} \\ 0 & \text{otherwise.} \end{cases}$$

Using an analogous sum construction with $z_{i,j}$ instead of $u_{i,j}$, we can use a TC^0 circuit to compute $|\mathcal{M}(a)|$: the number of j such that $a_{i,j} \geq a_{i,\text{max}}$. Finally, since dividing floats is in TC^0 (cf. §9), we can compute the head output as $s_i / |\mathcal{M}(a)|$, which has size $\mathrm{O}(\log n)$ by size preservation of division.

Feedforward As input, f receives v_i as well as H head outputs, all of which have size $\mathrm{O}(\log n)$. As the total size of the input is $\mathrm{O}(\log n)$, we can use Corollary 2.1 to compute the output of f with

an AC^0 circuit. The size of the output is $\mathrm{O}(\log n)$ by size preservation of f. The same idea holds for ϕ as well as the linear classification head.

We have simulated each transformer component with a TC^0 subcircuit, completing the proof. □

8.1 Discussion

Recall that, over rationals, we found that size-preserving saturated transformers could recognize any language. In contrast, we have now shown that using floating-point representations places such transformers within TC^0. In this paper, we have only considered non-uniform AC^0 and TC^0, as opposed to the uniform variants of these classes, which are more closely connected to familiar formal language classes like the regular and context-free languages (cf. Cojocaru, 2016; Mahajan, 2007). As transformers satisfy some intuitive notion of uniformity, an open question is whether saturated transformers also fall into uniform TC^0.

9 Conclusion

Compared with hard attention, saturated attention adds theoretical power to transformers. We showed that saturated attention lets transformers recognize languages outside AC^0, which is the upper bound with hard attention. Further, while saturated transformers with rational values and size-preserving internal functions can recognize any language, we characterize the limits of size-preserving saturated transformers with *floats*. Specifically, saturated transformers with float values fall in TC^0, a more powerful circuit class than AC^0. Thus, going from hard to saturated attention can be understood as augmenting the model with threshold gates. This illustrates one way that the circuit complexity paradigm characterizes the power of transformers. Going forward, there are many interesting open questions that circuit analysis can answer, such as comparing the power of saturated and soft attention, and refining existing upper bounds for transformers in terms of uniform circuit families.

Acknowledgments

Thanks to Yiding Hao, Dana Angluin, and Robert Frank for sharing an early draft of their work. We also appreciate helpful feedback from Dana Angluin, Matt Gardner, Yoav Goldberg, Michael Hahn, Kyle Richardson, and Roy Schwartz.

Appendix A Float Division

Let $/$ be truncated division between integers. We divide a float by an integer p by defining an approximate multiplicative inverse p^{-1}. The numerator is $2^{|p|}/p$ and the denominator is $2^{|p|}$. For division by a float p, q, we simply apply the integer approach and then multiply by q. This yields numerator $2^{|p|}/p \cdot q$ and denominator $2^{|p|}$.

The fact that float division is defined in terms of integer multiplication and division implies that it is size-preserving and can be simulated in TC^0, which we use in §8.

Appendix B Justifying Size Preservation

We justify that feedforward neural networks are size-preserving over floats. Feedforward neural networks are made up of a fixed (with respect to n) number of addition, multiplication, division, ReLU, and square root (for layer norm) operations. Therefore, it suffices to show that these operations are all in $\mathcal{S}(\mathbb{F})$.

For addition, the numerator is

$$\leq p_1 q_2 + p_2 q_1 \leq 2 p_{\max} q_{\max},$$

which has size $\leq \log 2 + |p_{\max}| + |q_{\max}| \leq 2(|p_{\max}| + |q_{\max}|)$ for large enough input size.

For multiplication, the numerator is just $p_1 \cdot p_2$, which has size $\leq 2|p_{\max}|$. Let the denominators be $q_1 = 2^{k_1}$ and $q_2 = 2^{k_2}$. Then the denominator is $2^{k_1+k_2}$, which has size $\leq 2|q_{\max}|$.

Division can be analyzed in terms of the approximate multiplicative inverse (§9).[9] Its numerator has size $\leq |p| + 1 + |q| \leq 2(|p| + |q|)$ for large enough input size. The denominator has size $\leq |p| + 1 \leq 2|p|$ for large enough input size.

Size preservation is trivially satisfied for ReLU, which cannot expand the size of the input.

To make layer norm work, we just need to analyze square root, which we will define in a truncated fashion over integers. The square root of a rational, then, simply takes the square root of p and q. We have that $|\sqrt{p}| \leq |p|$ and analogously for q.

Appendix C Resource-Bounded Transformers

Size preservation is one way to characterize the constraints on transformers' internal functions; a

[9] The exact multiplicative inverse $\langle p, q\rangle \mapsto \langle q, p\rangle$ over unconstrained rationals is also size-preserving. Thus, neural networks are size-preserving over both floats and rationals.

slightly different perspective is to fix ϕ and analyze how the language recognition abilities of the transformer change depending on the computational resources allotted to each $s_{\ell,h}$ and f_ℓ. In this section, we derive an alternate universality theorem in terms of time complexity classes. We will show that as long as ϕ is powerful enough, such transformers have equivalent time complexity to their activation functions.

Recall that a transformer is a tuple $\langle \Sigma, \mathbb{D}, \alpha, L, H, \phi, s_{\ell,h}, f_\ell \rangle$. In contrast to $\mathsf{AHAT}(\mathbb{D})$ (cf. Definition 6), we will now work with a different class of transformer languages $\mathsf{AHAT}(\mathbb{D}, T(m))$ We will allow the embedding functions to be linear in the sequence length, and explore the effect of varying the complexity of the other internal functions. Let $\mathsf{FTIME}(T(m))$ be the set of functions computable by a Turing machine in $T(m)$ time.[10]

Definition 10. Let $\mathsf{AHAT}(\mathbb{D}, T(n))$ be the class of languages $\mathcal{L} \subseteq \Sigma^*$ such that there exists a transformer $\langle \Sigma, \mathbb{D}, \alpha, L, H, \phi, s_{\ell,h}, f_\ell \rangle$ that recognizes $\mathcal{L}$, where ϕ runs in time linear in the sequence length n, and $s_{\ell,h}, f_\ell \in \mathsf{FTIME}(T(m))$.

For any $T(m) \geqslant m$, we will show transformers $\mathsf{AHAT}(\mathbb{D}, T(m))$ have the complexity of their activation functions. Formally:

Theorem 2 (Formal version of Theorem 2)**.** *For $\mathbb{D} \in \{\mathbb{F}, \mathbb{Q}\}$ and $T(m) \geqslant m$,*

$$\mathsf{AHAT}(\mathbb{D}, T(m)) \subseteq \mathsf{TIME}(T(m)).$$

Proof. First, observe that $\mathsf{AHAT}(\mathbb{D}, T(m)) \subseteq \mathsf{TIME}(T(m))$, since the embedding function and saturated attention can be computed in time linear in the input sequence length, and the other internal functions can be computed in $\mathsf{FTIME}(T(m))$ by construction.

We now show $\mathsf{TIME}(m) \subseteq \mathsf{AHAT}(\mathbb{D}, T(m))$. We adopt a 1-layer transformer construction, and thus omit ℓ, h subscripts. We define three components of the embedding function $\phi : \Sigma \times \mathbb{N} \to \mathbb{D}^3$:

$$\phi(w_i, i)_1 = \begin{cases} 2^{i-1} & \text{if } w_i = 1 \\ 0 & \text{otherwise} \end{cases}$$
$$\phi(w_i, i)_2 = i$$
$$\phi(w_i, i)_3 = 2^{|i|}.$$

[10] We write FTIME(m) instead of the conventional FTIME(n) to avoid confusion with the sequence length n.

Each of these components is computable in time linear in n. Define three heads $b_{1,i}, b_{2,i}, b_{3,i}$. Without loss of generality, consider $b_{h,i}$ to act on $\phi(w_i, i)_h$ alone, rather than the full embeding vector. $b_{1,i}$ is defined as a uniform head, while $b_{2,i}$ and $b_{3,i}$ are computed with $s_h(v_i, v_j) = v_j$. Thus,

$$b_{1,i} = \frac{1}{n} \sum_{w_j = 1} 2^{j-1}$$
$$b_{2,i} = n$$
$$b_{3,i} = 2^{|n|}.$$

Finally, we discuss how to set f to compute whether $w \in \mathcal{L}$. Let p be the function that extracts the numerator of a float or rational number, which is computable in $\mathrm{O}(m)$ time on float of size m. Within f, we compute $u = p(b_{1,i})$. At this point, we proceed in two cases depending on the datatype $\mathbb{D}$:

1. **Rationals:** If $\mathbb{D} = \mathbb{Q}$, then u is the binary string w. Any $\mathcal{L} \in \mathsf{TIME}(T(m))$ has an indicator function $\delta \in \mathsf{FTIME}(T(m))$, which we now apply to recognize whether $w \in \mathcal{L}$.
2. **Floats:** If $\mathbb{D} = \mathbb{F}$, then $u = 2^{|n|}/n \cdot w$ as in §9. Therefore, in linear time, we compute

$$\frac{b_{2,i}}{b_{3,i}} \cdot u = \frac{n}{2^{|n|}} \cdot \frac{2^{|n|} w}{n} = w,$$

and feed w through δ as in the $\mathbb{D} = \mathbb{Q}$ case.

So, $\mathsf{TIME}(T(m)) \subseteq \mathsf{AHAT}(\mathbb{D}, T(m))$. □

Appendix D Proof from Hao et al. (2022)

The proof for Lemma 2 largely follows the proof of a core lemma of Hao et al. (2022). We reproduce a slightly adapted version of their proof here, because their manuscript is not yet publicly available, and we wish for our paper to be self-contained.

Lemma 2. Any function $f : \{0,1\}^c \to \{0,1\}^d$ can be computed by a Boolean circuit of depth 3 and size at most $d(2^c + c + 1)$.

Proof. The idea of the proof is to define d subcircuits of size at most $2^c + c + 1$ that compute the d output bits of f in parallel. We will build a circuit that computes each output bit of f according to its representation in disjunctive normal form (DNF).

We define a first layer of the circuit that computes the negation of each input, which takes c gates. The second layer then computes the value of each DNF term by computing a conjunction ($\wedge$ gate) over the corresponding literals or negated literals. Note that a formula of c variables has at most 2^c DNF terms. Finally, the third layer of the circuit computes a disjunction ($\vee$ gate) over the values of all terms, yielding the output of f, and adding a single gate. In summary, we have shown how to compute each output bit with a circuit of size at most $2^c + c + 1$, which implies the full function f can be computed by a circuit of size at most $d(2^c + c + 1)$. □

References

Afra Alishahi, Yonatan Belinkov, Grzegorz Chrupała, Dieuwke Hupkes, Yuval Pinter, and Hassan Sajjad, editors. 2020. *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Online.

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press. https://doi.org/10.1017/CBO9780511804090

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv*, abs/1607.06450.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.576

Liliana Cojocaru. 2016. *Advanced Studies on the Complexity of Formal Languages*. Ph.D. thesis, University of Tampere.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter van Emde Boas. 1991. *Machine Models and Simulations*, chapter 1. MIT Press, Cambridge, MA, USA. https://doi.org/10.1016/B978-0-444-88071-0.50006-0

Merrick Furst, James B. Saxe, and Michael Sipser. 1981. Parity, circuits, and the polynomial-time hierarchy. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 260–270, USA. IEEE Computer Society. https://doi.org/10.1109/SFCS.1981.35

Larry J. Goldstein. 1973. A history of the prime number theorem. *The American Mathematical Monthly*, 80(6):599–615. https://doi.org/10.1080/00029890.1973.11993338

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171. https://doi.org/10.1162/tacl_a_00306

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. 2020. RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.156

David S. Johnson. 1991. *A Catalog of Complexity Classes*, chapter 2. MIT Press, Cambridge, MA, USA. https://doi.org/10.1016/B978-0-444-88071-0.50007-2

Neeraj Kayal. 2015. Lecture notes for topics in complexity theory.

Meena Mahajan. 2007. Polynomial size log depth circuits: Between NC^1 and AC^1. *Bulletin of the EATCS*, 91:42–56.

William Merrill. 2019. Sequential neural networks as automata. In *Proceedings of the Workshop on*

Deep Learning and Formal Languages: Building Bridges, pages 1–13, Florence. Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-3901

William Merrill. 2021. Formal language theory meets modern NLP. *ArXiv*, abs/2102.10094.

William Merrill, Vivek Ramanujan, Yoav Goldberg, Roy Schwartz, and Noah A. Smith. 2021. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1766–1781, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.133

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Rational recurrences. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1214, Brussels, Belgium. Association for Computational Linguistics. https://doi.org/10.18653/v1/D18-1152

Jorge Pérez, Javier Marinkovic, and Pablo Barceló. 2019. On the Turing completeness of modern neural network architectures. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-2117

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like transformers. *ArXiv*, abs/2106.06981.

Andrew C.-C. Yao. 1990. On ACC and threshold circuits. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 619–627 vol.2.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3770–3785, Online. Association for Computational Linguistics.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2020. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*.