What Formal Languages Can Transformers Express? A Survey

Lena Strobl
Umeå University, Sweden
lena.strobl@umu.se

William Merrill
New York University, USA
willm@nyu.edu

Gail Weiss
EPFL, Switzerland
gail.weiss@epfl.ch

David Chiang
University of Notre Dame, USA
dchiang@nd.edu

Dana Angluin
Yale University, USA
dana.angluin@yale.edu

Abstract

As transformers have gained prominence in natural language processing, some researchers have investigated theoretically what problems they can and cannot solve, by treating problems as *formal languages*. Exploring such questions can help clarify the power of transformers relative to other models of computation, their fundamental capabilities and limits, and the impact of architectural choices. Work in this subarea has made considerable progress in recent years. Here, we undertake a comprehensive survey of this work, documenting the diverse assumptions that underlie different results and providing a unified framework for harmonizing seemingly contradictory findings.

1 Introduction

Transformers (Vaswani et al., 2017) have gained prominence in natural language processing (NLP), both in direct applications like machine translation and in pretrained models like BERT (Devlin et al., 2019) and GPT (Radford et al., 2018; Brown et al., 2020; OpenAI, 2023). Consequently, some researchers have sought to investigate their theoretical properties. Such studies can broadly be divided into studies of *expressivity* and *trainability*. While trainability is very important and the focus of much study (e.g., Bhattamishra et al., 2023; Allen-Zhu and Li, 2023), here we focus on expressivity, which is a prerequisite for trainability.

Studies of expressivity could be further divided into those from the perspectives of approximation theory and of formal language theory. The former (e.g., Yun et al., 2020; Sanford et al., 2023), investigates transformers as approximators of various classes of *functions*, along the lines of the universal approximation theorem for feedforward neural networks (Hornik et al., 1989; Cybenko, 1989). The latter, which is the subject of this survey, investigates transformers as recognizers or generators of *formal languages*—that is, the inputs or outputs are treated as sequences of discrete symbols from a finite alphabet, and crucially as sequences of unbounded length.

The core research question in this subarea is: *How can we characterize the expressivity of transformers in relation to various formal models, such as automata, Boolean circuits, or formal logic?* Applications of this subarea, which are not addressed by the papers surveyed here but could be by future work, would hopefully answer questions like:

- What new transformer variants are suggested by formal models?
- Do failure cases anticipated from formal models occur in practice?
- What insights into the complexity of human language are offered by a characterization of transformer expressivity?

This paper provides a comprehensive survey of research in this subarea. Compared to the surveys of Ackerman and Cybenko (2020) and Merrill (2021, 2023), which cover convolutional neural networks (CNNs), RNNs, and transformers, this is a narrower, but deeper, survey on transformers only.

Interpreting theoretical transformer results is complex due to diverse assumptions. Many variants of transformers exist in practice, and even more have been proposed in theory. This diversity

Transactions of the Association for Computational Linguistics, vol. 12, pp. 543–561, 2024. https://doi.org/10.1162/tacl_a_00663
Action Editor: Mark-Jan Nederhof. Submission batch: 11/2023; Revision batch: 1/2024; Published 4/2024.

Lower bound	Source	PE	Attention	Notes
$\ni$ Majority	Pérez et al. 2019	none	average-hard	
$\ni$ Shuffle-Dyck-k	Bhattamishra et al. 2020a	none	softmax, future mask	
$\supseteq$ SSCMs	Bhattamishra et al. 2020a	none	softmax, future mask	
$\ni$ Dyck-k	Yao et al. 2021	i/n, i/n^3, n	softmax & leftmost-hard	
$\supseteq$ P	Pérez et al. 2021	i, $1/i$, $1/i^2$	average-hard	poly(n) steps
$\ni$ Parity	Chiang and Cholak 2022	i/n, $(-1)^i$	softmax	
$\supseteq$ FOC[MOD; +]	Chiang et al. 2023	sinusoidal	softmax	
$\supseteq$ FO[Mon]	Barceló et al. 2024	arbitrary	leftmost-hard	
$\supseteq$ LTL+C[Mon]	Barceló et al. 2024	arbitrary	average-hard	
Upper bound	**Source**	**Precision**	**Attention**	**Notes**
$\not\ni$ Parity, Dyck-1	Hahn 2020	$\mathbb{R}$	leftmost-hard	
$\not\ni$ Parity, Dyck-2	Hahn 2020	$\mathbb{R}$	softmax, future mask	$\varepsilon_\mathcal{N} > 0$, vanishing KL
$\subseteq$ AC^0	Hao et al. 2022	$\mathbb{Q}$	leftmost-hard	
$\subseteq$ TC^0	Merrill et al. 2022	$\mathbb{F}$	average-hard	
$\subseteq$ FOC[MOD; +]	Chiang et al. 2023	$O(1)$	softmax	
$\subseteq$ L-uniform TC^0	Merrill and Sabharwal, 2023a	$O(\log n)$	softmax	
$\subseteq$ FOM[BIT]	Merrill and Sabharwal, 2023b	$O(\log n)$	softmax	
$\subseteq$ L-uniform TC^0	Strobl 2023	$\mathbb{F}$	average-hard	
Equivalent	**Source**	**PE**	**Attention**	**Notes**
= RE	Pérez et al. 2021	i, $1/i$, $1/i^2$	average-hard	unbounded steps
= FO	Angluin et al. 2023	none	rightmost-hard, strict future mask	
= FO[MOD]	Angluin et al. 2023	sinusoidal	rightmost-hard, strict future mask	
= FO[Mon]	Angluin et al. 2023	arbitrary	rightmost-hard, strict future mask	
= P	Merrill and Sabharwal, 2024	none	average-hard, future mask	poly(n) steps

Table 1: Surveyed claims and their assumptions. Please see the main text for full details of assumptions.

leads to varied, even seemingly contradictory, results. We set up a unified framework for talking about transformer variants (§4), and discuss how some of these variants compare to one another in expressivity.

We then provide background on various formal models that transformers have been compared with (§5). Then, in §6, we systematically survey current results in this literature, documenting their assumptions and claims in terms of the definitions of Sections 4 and 5.

2 Overview

Table 1 summarizes the results surveyed here. One way to classify them is into *lower bounds* (what transformers *can* do) and *upper bounds* (what transformers *can't* do).

Much work on lower bounds has looked at *automata* like finite automata, counter machines, and Turing machines, all of which had been successfully related to RNNs before (Siegelmann and Sontag, 1995; Merrill, 2020). This wide diversity of machines is due to different variants of transformers, especially whether a transformer decoder is allowed to take a number of intermediate steps before outputting a decision (§4.3.4), which dramatically increases its power (§6.1).

By contrast, investigation of upper bounds has mainly focused on *circuit complexity* (§5.2), which had been successfully related to feedforward networks before (Parberry, 1994; Siu et al., 1995; Beiu and Taylor, 1996; Šíma and Orponen, 2003). This line of research began with restricted models of transformer encoders and progressed to increasingly realistic variants and tighter bounds. One way to restrict transformers is by discretizing the attention mechanism (§4.2.1); another is to limit the precision of number representations (§4.4).

More recent work has turned to *formal logic* (§5.3) as a way of characterizing the expressive power of transformers. The finer control afforded by logics opens the possibility for them to be used as upper bounds, lower bounds, or both.

3 Preliminaries

Sets We denote by $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ and $\mathbb{N} = \mathbb{N}_0 \setminus \{0\}$ the set of natural numbers with and without 0, respectively. We write $[n] = \{0, 1, 2, \ldots, n-1\}$ for any $n \in \mathbb{N}$. We write Σ for a finite

alphabet, which, in NLP applications, is the set of words or subwords known to the model.

Vectors We use d, d', etc., for dimensionalities of vector spaces, lowercase bold letters ($\mathbf{x}, \mathbf{y}, \ldots$) for vectors, and uppercase bold letters ($\mathbf{X}, \mathbf{Y}, \ldots$) for matrices. For any vector $\mathbf{x} \in \mathbb{R}^d$, we number its elements starting from 0. For $i \in [d]$, we write $\mathbf{x}_i$ or $[\mathbf{x}]_i$ (not x_i) for the i-th component of $\mathbf{x}$.

Sequences For any set A, we write A^* for the set of all finite sequences over A. We write the length of a sequence $s \in A^*$ as $|s|$ and number its elements starting from 0; thus, $s = s_0 s_1 \cdots s_{|s|-1}$. We use the variable w for a string in Σ^* and n for the length of w. For sequences in $\mathbb{R}^*$, we use lowercase bold letters ($\mathbf{x}, \mathbf{y}, \ldots$), and for sequences in $(\mathbb{R}^d)^*$, we use the variable X.

A function $f\colon A^* \to B^*$ is *length-preserving* if $|f(w)| = |w|$ for all $w \in A^*$. For every function $g\colon A \to B$, we denote its extension to sequences by g as well. That is, $g\colon A^* \to B^*$ is defined as follows: for all $s \in A^*$ and $i \in [|s|]$, $g(s)_i = g(s_i)$.

Neural Networks An *affine transformation* is a function $L\colon \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ parameterized by weights $\mathbf{W}_L \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and bias $\mathbf{b}_L \in \mathbb{R}^{d_{\text{out}}}$ such that for every $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, $L(\mathbf{x}) = \mathbf{W}_L \mathbf{x} + \mathbf{b}_L$. We say that L is *linear* if $\mathbf{b}_L = \mathbf{0}$.

The activation functions we use are the *rectified linear unit* (ReLU) $\mathcal{R}(x) = \max(x, 0)$ and the logistic *sigmoid* function $\sigma(x) = 1/(1 + e^{-x})$.

The *softmax* function $\mathcal{S}\colon \mathbb{R}^* \to \mathbb{R}^*$ converts any sequence of reals into a probability distribution:

$$\mathcal{S}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{i \in [|\mathbf{x}|]} e^{\mathbf{x}_i}} \qquad \forall i \in [|\mathbf{x}|].$$

4 Transformers

In this section, we define transformers and relevant variants, and how transformers are used to describe formal languages. For additional background on transformers (not in relation to formal languages), Huang et al. (2022) give a lucid commentary on the original paper, Phuong and Hutter (2022) give formal definitions and pseudocode, and Lin et al. (2022) survey many variants of transformers.

Transformers are composed of an input layer (§4.1), one or more hidden layers (§4.2), and an output layer (§4.3). The inputs and outputs of the layers are sequences of vectors, which we treat as members of $(\mathbb{R}^d)^*$.[1]

4.1 Input Layer

Strings are initially mapped to sequences of vectors using a length-preserving function $e\colon \Sigma^* \to (\mathbb{R}^d)^*$, which is the sum of a *word embedding* $\text{WE}\colon \Sigma \to \mathbb{R}^d$ and a *position(al) embedding* or *encoding* $\text{PE}_n\colon [n] \to \mathbb{R}^d$ for $n \in \mathbb{N}$:

$$e(w_0 \cdots w_{n-1})_i = \text{WE}(w_i) + \text{PE}_n(i).$$

In theoretical constructions, the word embedding can be any computable function.

The original transformer paper (Vaswani et al., 2017) introduced the following position embedding:

$$[\text{PE}_n(i)]_j = \begin{cases} 10000^{-j/d} \sin i & \text{if } j \text{ even} \\ 10000^{-(j-1)/d} \cos i & \text{if } j \text{ odd.} \end{cases}$$

Theoretical papers have explored other position embeddings, including i itself (Pérez et al., 2021), i/n (Yao et al., 2021; Chiang and Cholak, 2022), and $1/i$ or $1/i^2$ (Pérez et al., 2021).

4.2 Hidden Layers

A *transformer layer* is a length-preserving function $\mathcal{L}\colon (\mathbb{R}^d)^* \to (\mathbb{R}^d)^*$. There are two variants. The *post-norm* variant (Vaswani et al., 2017) is

$$\begin{aligned} X' &= \mathcal{N}_1(X + \mathcal{A}(X)) \\ \mathcal{L}(X) &= \mathcal{N}_2(X' + \mathcal{F}(X')) \end{aligned} \tag{1}$$

and the *pre-norm* variant (Wang et al., 2019) is

$$\begin{aligned} X' &= X + \mathcal{A}(\mathcal{N}_1(X)) \\ \mathcal{L}(X) &= X' + \mathcal{F}(\mathcal{N}_2(X')) \end{aligned} \tag{2}$$

where

- $\mathcal{A}$ is a multi-head self-attention with d input/output dimensions, H heads, and d_{kv} key/value dimensions per head
- $\mathcal{F}$ is a feed-forward network (§4.2.2) with d input/output dimensions and d_{ff} hidden dimensions

[1] This differs from the original paper (Vaswani et al., 2017), which treats them as matrices in $\mathbb{R}^{n \times d}$. Our notation aligns better with notation for formal languages and emphasizes the variability of the sequence length.

- $\mathcal{N}_1$ and $\mathcal{N}_2$ are layernorms with d dimensions.

We define each of these components below.

4.2.1 Attention

Attention was initially developed to facilitate retrieval of previously processed data from a variable-length history (Bahdanau et al., 2015). Transformers use a simple variant of attention known as *scaled dot-product attention*.

Scaled Dot-product Attention with d input/output dimensions and d_{kv} key/value dimensions is a function $\mathrm{A}\colon \mathbb{R}^d \times (\mathbb{R}^d)^* \to \mathbb{R}^d$ parameterized by linear transformations

$$W_{\mathrm{A}}^{\mathrm{Q}}, W_{\mathrm{A}}^{\mathrm{K}}, W_{\mathrm{A}}^{\mathrm{V}}\colon \mathbb{R}^d \to \mathbb{R}^{d_{\text{kv}}} \quad W_{\mathrm{A}}^{\mathrm{O}}\colon \mathbb{R}^{d_{\text{kv}}} \to \mathbb{R}^d$$

and defined for every $\mathbf{z} \in \mathbb{R}^d$, $X \in (\mathbb{R}^d)^*$ (with $|X| = n$), and $j \in [n]$ as

$$\mathbf{s}(\mathbf{z}, X)_j = \frac{W_{\mathrm{A}}^{\mathrm{Q}}(\mathbf{z}) \cdot W_{\mathrm{A}}^{\mathrm{K}}(X_j)}{\sqrt{d_{\text{kv}}}} \tag{3}$$

$$\alpha(\mathbf{z}, X) = \mathcal{S}(\mathbf{s}(\mathbf{z}, X)) \tag{4}$$

$$\mathrm{A}(\mathbf{z}, X) = W_{\mathrm{A}}^{\mathrm{O}}\Big(\sum_{j \in [n]} \alpha(\mathbf{z}, X)_j \, W_{\mathrm{A}}^{\mathrm{V}}(X_j)\Big).$$

Typically, A is extended to a function $\mathrm{A}\colon (\mathbb{R}^d)^* \times (\mathbb{R}^d)^* \to (\mathbb{R}^d)^*$ that is length-preserving in its *first* argument. In *cross*-attention, $\mathbf{z}$ is computed by the decoder while X is computed by the encoder. In *self*-attention, the two arguments are identical:

$$\mathrm{SA}\colon (\mathbb{R}^d)^* \to (\mathbb{R}^d)^*$$
$$\mathrm{SA}(X) = \mathrm{A}(X, X).$$

Attention Masking In *future masked* (also known as *causally* masked) self attention, a term $m(i, j)$ is added to Eq. (3) to force every position to attend only to preceding positions:

$$m(i, j) = \begin{cases} 0 & \text{if } j \le i \\ -\infty & \text{otherwise.} \end{cases}$$

Some papers use *strict* future masking, that is, $m(i, j) = 0$ iff $j < i$, and occasionally *past* masking ($j \ge i$) and strict past masking ($j > i$).

Multi-head Attention with d_{kv} key/value dimensions per head is the sum of H attentions with d_{kv} key/value dimensions:

$$\mathcal{A}(\mathbf{z}, X) = \sum_{h \in [H]} \mathrm{A}_h(\mathbf{z}, X).$$

Multi-head self attention is defined analogously. This is equivalent to the original formulation, which concatenated the outputs of the heads and passed the result through a shared, larger, $W_{\mathrm{A}}^{\mathrm{O}}$.

Hard Attention Some theoretical analyses simplify attention by replacing the softmax with variants that focus attention only on the position(s) with the maximum value, breaking ties in various ways. For any $\mathbf{s} \in \mathbb{R}^*$, let $M(\mathbf{s}) = \{i \in [|\mathbf{s}|] \mid \forall j \in [|\mathbf{s}|], \mathbf{s}_j \le \mathbf{s}_i\}$ be the set of indices of the maximal elements of $\mathbf{s}$. In *leftmost-argmax*, the leftmost maximal element is used:

$$[\mathcal{S}_{\mathrm{h}}(\mathbf{s})]_i = \mathbb{I}[i = \min M(\mathbf{s})]$$

whereas in *average-argmax* the maximal elements share weight equally:

$$[\mathcal{S}_{\mathrm{a}}(\mathbf{s})]_i = \frac{\mathbb{I}[i \in M(\mathbf{s})]}{|M(\mathbf{s})|}.$$

If softmax is thought of as a Boltzmann distribution, then average-argmax is its low-temperature limit.

By substituting $\mathcal{S}_{\mathrm{h}}$ or $\mathcal{S}_{\mathrm{a}}$ for $\mathcal{S}$ in Eq. (4), we get *leftmost-hard* and *average-hard* attention, respectively. Leftmost-hard attention was previously called *hard* attention by Hahn (2020) and *unique hard* attention by Hao et al. (2022). One may also consider *rightmost-hard* attention, in which the rightmost maximal element is used. Average-hard attention was also called *hard* attention by Pérez et al. (2021) and *saturated* attention by Merrill et al. (2022), and has been argued to be a realistic approximation to how trained transformers behave in practice (Merrill et al., 2021).

4.2.2 Feed-forward Networks

Although feed-forward networks can take many forms, in the context of transformers, we use the following definition. A *feed-forward network* (FFN) with d input/output dimensions and d_{ff} hidden dimensions is a function $\mathcal{F}\colon \mathbb{R}^d \to \mathbb{R}^d$

parameterized by two affine transformations, $L^1_{\mathcal{F}} : \mathbb{R}^d \to \mathbb{R}^{d_{\text{ff}}}$ and $L^2_{\mathcal{F}} : \mathbb{R}^{d_{\text{ff}}} \to \mathbb{R}^d$, such that

$$\mathcal{F}(\mathbf{x}) = L^2_{\mathcal{F}}(\mathcal{R}(L^1_{\mathcal{F}}(\mathbf{x})))$$

where $\mathcal{R}$ is applied component-wise.

4.2.3 Layer Normalization

A d-dimensional *layer normalization* (Ba et al., 2016), or *layernorm* for short, is a function $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}^d$ parameterized by vectors $\gamma_{\mathcal{N}}, \beta_{\mathcal{N}} \in \mathbb{R}^d$ and scalar $\varepsilon_{\mathcal{N}} \geq 0$:

$$\mathcal{N}(\mathbf{x}) = \gamma_{\mathcal{N}} \odot \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\text{var}(\mathbf{x}) + \varepsilon_{\mathcal{N}}}} + \beta_{\mathcal{N}}$$

where $\odot$ is component-wise multiplication and

$$\bar{\mathbf{x}} = \frac{1}{d} \sum_{i \in [d]} \mathbf{x}_i \qquad \text{var}(\mathbf{x}) = \frac{1}{d} \sum_{i \in [d]} (\mathbf{x}_i - \bar{\mathbf{x}})^2.$$

The original definition of layernorm (Ba et al., 2016) sets $\varepsilon_{\mathcal{N}} = 0$, but, for numerical stability, all implementations we are aware of set $\varepsilon_{\mathcal{N}} > 0$. Observe that $\mathcal{N}$ is Lipschitz-continuous iff $\varepsilon_{\mathcal{N}} > 0$.

Some transformer analyses omit $\mathcal{N}$ for simplicity (Pérez et al., 2021), while others set $\varepsilon_{\mathcal{N}}$ to achieve various effects (Hahn, 2020; Chiang and Cholak, 2022).

4.3 Networks and Output Layers

We now define a complete transformer network.

4.3.1 Transformer Encoders

A *transformer encoder* is a length-preserving function $\mathcal{T} : \Sigma^* \to (\mathbb{R}^d)^*$ parameterized by the weights of an input layer e and D transformer layers $\mathcal{L}_1, \ldots, \mathcal{L}_D$. A *post-norm* transformer encoder is:

$$\mathcal{T}(w) = \mathcal{L}_D \circ \cdots \circ \mathcal{L}_2 \circ \mathcal{L}_1 \circ e(w)$$

where each $\mathcal{L}_l$ is a post-norm layer (1) and $\circ$ is function composition. A *pre-norm* transformer encoder is additionally parameterized by the weights of a final layernorm $\mathcal{N}$ and is defined as:

$$\mathcal{T}(w) = \mathcal{N} \circ \mathcal{L}_D \circ \cdots \circ \mathcal{L}_2 \circ \mathcal{L}_1 \circ e(w)$$

where each $\mathcal{L}_l$ is a pre-norm layer (2).

The encoder's output is a sequence of vectors in $(\mathbb{R}^d)^*$. To use it as a language recognizer, we add an output layer that converts $\mathcal{T}(w)$ to a probability

$$\hat{p} = \sigma(\mathbf{w} \cdot [\mathcal{T}(w)]_i + b)$$

where $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, and i is a distinguished position. The encoder accepts iff $\hat{p} \geq \frac{1}{2}$.

Chiang and Cholak (2022) also consider a requirement that an encoder accepts/rejects strings with bounded cross-entropy. That is, we say that an encoder recognizes a language L with cross-entropy at most η iff for all strings w, if $w \in L$ then $-\log \hat{p} \leq \eta$, and if $w \notin L$ then $-\log(1 - \hat{p}) \leq \eta$.

We are aware of two choices for the distinguished position i. Most papers use the last position ($i = n - 1$), but some (Chiang and Cholak, 2022; Chiang et al., 2023), inspired by binary classifiers based on BERT (Devlin et al., 2019), prepend a special symbol `CLS` at position 0 and use $i = 0$. While this is a minor difference, it should be noted that the guarantee of exactly one occurrence of `CLS` in the input can be useful in some constructions.

4.3.2 Transformer Decoders

A *transformer decoder* is a transformer encoder $\mathcal{T}$ with future masking in its attention, typically used to generate rather than recognize strings. The input is the prefix of previously-generated symbols, $w_{<t} = w_0 \cdots w_{t-1}$, and the output is a probability distribution $\hat{p}(w_t \mid w_{<t})$ over the next symbol,

$$\hat{p}(\cdot \mid w_{<t}) = \mathcal{S}(\mathbf{W}\,[\mathcal{T}(w_{<t})]_{t-1} + \mathbf{b})$$

where $\mathbf{W} \in \mathbb{R}^{|\Sigma| \times d}$ and $\mathbf{b} \in \mathbb{R}^{|\Sigma|}$. We assume $w_0 = \texttt{BOS}$ and every string ends with `EOS`, where `BOS` and `EOS` are special symbols that do not occur anywhere else. To sample a string, we first sample w_1 from $\hat{p}(w_1 \mid \texttt{BOS})$, then, for each time step $t > 1$, sample w_t from $\hat{p}(w_t \mid w_{<t})$. The process stops when $w_t = \texttt{EOS}$. Because each sampled output symbol becomes part of the input at the next time step, this kind of model is called *autoregressive*.

While a decoder can be used to recognize strings similarly to an encoder, it can also be used to generate the entire string; at least two definitions have been given for this.

First, Hahn (2020) considers a weighted language as a distribution over strings $p(w)$. For any length t, the KL divergence (relative entropy) of the model $\hat{p}(w)$ from the true distribution $p(w)$, for predicting w_t conditioned on all previous words, is

$$\Delta_t[\hat{p} \parallel p] = \sum_{w_{<t}} \sum_{w_t} p(w_{<t} w_t) \log \frac{p(w_t \mid w_{<t})}{\hat{p}(w_t \mid w_{<t})}.$$

As Hahn's results are negative, he does not spell out a positive criterion, but he seems to implicitly require that this divergence vanish at infinity:

$$\lim_{t \to \infty} \Delta_t[\hat{p} \parallel p] = 0. \tag{5}$$

Second, let us say that a transformer decoder *ε-generates* L iff

$$L = \{w \mid \forall t \in [|w|], \hat{p}(w_t \mid w_{<t}) \geq \varepsilon\}.$$

Then Yao et al. (2021), following Hewitt et al. (2020), say that a transformer decoder T generates a language L iff there exists an $\varepsilon > 0$ such that T ε-generates L. (This means that a transformer decoder may generate more than one language, depending on the ε chosen.) They also show that any ε-generator can be converted into a recognizer.

While not focusing on transformers, Lin et al. (2021) demonstrate limitations of autoregressive models for generation; for example, that there is a language $L \in \mathsf{P}$ that cannot be ε-generated in polynomial time for any $\varepsilon > 0$ if $\mathsf{P} \neq \mathsf{NP}$.

4.3.3 Transformer Encoder–Decoders

A *transformer encoder–decoder* combines a transformer encoder and decoder, adding to each layer of the decoder an additional attention sublayer, known as *cross attention*, which attends to the output of the encoder. In the literature surveyed here, only the construction of Pérez et al. (2021) and related constructions (Bhattamishra et al., 2020b; Wei et al., 2022a) employ an encoder–decoder.

4.3.4 Intermediate Steps

When a transformer decoder or encoder–decoder is run as a language recognizer, it allows for the possibility of inserting a number of *intermediate* time steps between the end of the input string and the decision. The encoder–decoder models above do this, as do some decoder-only models (Feng et al., 2023; Merrill and Sabharwal, 2024). As we will see (§6.1), intermediate steps vastly increase the model's power, which has also been observed in practice in the form of a ''scratchpad'' (Nye et al., 2021) or ''chain of thought'' (Wei et al., 2022b).

4.4 Uniformity and Precision

Although meaningful theoretical claims can be made about transformers for fixed-length strings (e.g., Yun et al., 2020), it is crucial when examining transformers as language recognizers to allow for unbounded string length. Fixing a maximum length makes all languages finite, collapsing many language classes into one.

It might be objected that considering unbounded lengths is too abstract, because in practice one can always fix a maximum length. But this maximum length, driven by practical needs, is growing steadily: for example, GPT-4 Turbo uses 128,000 tokens of context. At the same time, some theoretical findings surveyed here seem to have practical consequences for modest string lengths. For example, we will see that there are reasons to think that in theory, transformers cannot recognize Parity; in practice, they fail to learn Parity for strings with lengths in $[2, 50]$ (Bhattamishra et al., 2020a).

Some theoretical studies of transformers do allow them to depend on the input length n. To borrow a term from circuit complexity (§5.2), they allow certain kinds of *non-uniformity*. As we have seen, some position embeddings (§4.1) depend on n. We discuss some other instances below.

Numeric Precision Transformers operate, in principle, on real numbers. While hard attention transformers could be defined using only rational numbers, even rational numbers can represent an arbitrary amount of information. With RNNs, the use of real or rational numbers has led to results that make them appear more powerful in theory than in practice (Siegelmann and Sontag, 1994, 1995; Weiss et al., 2018).

Consequently, many studies use limited-precision numbers. Some studies limit number representations to have $O(1)$ bits, as floating-point numbers do in practice (Chiang et al., 2023). But Merrill and Sabharwal (2023b) argue that in $O(1)$ precision, attention cannot attend

uniformly to a string of sufficient length n, as the attention weights (α) would all round down to zero. So $O(\log n)$ bits of precision is a common choice (Yao et al., 2021; Merrill and Sabharwal, 2023a,b). Other choices are possible as well: Merrill and Sabharwal (2023a) use the set $\mathbb{F} = \{a/2^b \mid a \in \mathbb{Z}, b \in \mathbb{N}\}$.

Restricting intermediate activations to limited precision introduces many decisions about when and how rounding should take place, which can potentially affect expressivity. For example, when summing n numbers, one could round after each addition or only at the end of the summation. Better formalizing these decisions and their impact on expressivity is an area for future research.

Parameters A few constructions allow the parameters themselves to depend on n, which we consider to be a stronger dependence, because if these transformers were to be learned from data, different transformers would have to be learned for different maximum lengths. Finally, a few papers construct transformers in which d, and therefore the number of parameters, depends on n, which we consider to be stronger still.

4.5 Summary

In summary, transformers can vary in at least the following ways, any of which could *a priori* impact theoretical claims:

- Architecture: encoder-only, decoder-only, or encoder–decoder
- For encoders: definition of recognition
- For decoders and encoder–decoders: definition of generation and how many intermediate steps
- Position embedding (PE)
- Attention pattern: leftmost-hard, rightmost-hard, average-hard, or softmax
- Attention masking: none, future, or past
- Layernorm: inclusion or omission, value of $\varepsilon_{\mathcal{N}}$
- Residual connections: pre-norm or post-norm
- Precision: infinite, $O(\log n)$, $O(1)$
- Uniformity: whether parameter values or number of parameters depend on n.

5 Languages and Language Classes

Next, we present various formal models that transformers are compared to in the literature surveyed.

5.1 Automata and Classes L, NL, P

We assume familiarity with finite automata and Turing machines; for definitions, please see the textbook by Sipser (2013). Counter machines are automata with integer-valued registers (Fischer et al., 1968); they have been studied extensively in connection with LSTM RNNs (Weiss et al., 2018; Suzgun et al., 2019; Merrill, 2019, 2020).

The language classes L (languages decidable in $O(\log n)$ space) and P (languages decidable in polynomial time) are defined using deterministic Turing machines (with a read-only input tape and a read/write work tape). The class NL (languages decidable in nondeterministic $O(\log n)$ space) uses nondeterministic Turing machines. The class $\mathsf{DLOGTIME}$ (languages decidable in $O(\log n)$ time) uses random-access Turing machines (Barrington et al., 1990). It is known that

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}$$

but none of these inclusions are known to be strict.

5.2 Circuits and Classes AC^0, ACC^0, TC^0, NC^1

Circuits are a model of parallel computation particularly relevant to transformers. For more details, please see the textbook by Arora and Barak (2009).

Circuits operate on binary values. If we choose a fixed-length encoding of the symbols of Σ as strings of $b = \lceil \log_2 |\Sigma| \rceil$ bits, then a circuit can simulate input alphabet Σ by encoding the value of the i-th input symbol into positions ib to $ib + (b - 1)$. For the rest of this section, we assume $\Sigma = \{0, 1\}$.

Circuits A *circuit* C with input length n is a directed acyclic graph with n *input* vertices $s_1, \ldots, s_n$ and zero or more *gate* vertices, each labeled with a *type* NOT, AND, or OR. Input vertices have fan-in (in-degree) zero, NOT gates have fan-in one, and the fan-in of AND and OR gates can be either two or unbounded. One

(input or gate) vertex t is designated the *output* of the circuit.

Given an input string $w \in \{0,1\}^n$, each input vertex s_i is assigned the value w_i, and each gate vertex is assigned the value computed by applying the logical function corresponding to its type to the values assigned to its in-neighbors. The circuit computes the Boolean function $C\colon \{0,1\}^n \to \{0,1\}$, mapping each input string to the value assigned to t. The *depth* of C, denoted $D(C)$, is the length of the longest directed path from any s_i to t. The *size* of C, denoted $|C|$, is the number of vertices in C.

Circuit Families A *circuit family* is a sequence $C = \{C_n\}_{n\in\mathbb{N}}$ such that for each n, C_n is a circuit with input length n. We treat C as a function on $\{0,1\}^*$ as follows: for every $w \in \{0,1\}^*$, $C(w) = C_{|w|}(w)$. Then C defines the language $L(C) = \{w \in \{0,1\}^* \mid C(w) = 1\}$, and we say that C recognizes $L(C)$. The *depth* and *size* of C are the functions $n \mapsto D(C_n)$ and $n \mapsto |C_n|$.

Uniformity As defined, a circuit family contains a different circuit for each length n, with no constraint on the relationship between the circuits. For example, let L be any *unary* language: $L \subseteq \{1\}^*$. For $n \in \mathbb{N}$, if $1^n \notin L$, define C_n to be a circuit for the constant 0 function (an OR gate with fan-in 0), and if $1^n \in L$, define C_n to be a circuit for the AND of all the inputs. Thus, every unary language, even an undecidable one, is recognized by a circuit family of size $O(n)$ and depth $O(1)$.

A uniformity restriction on a circuit family $\{C_n\}_{n\in\mathbb{N}}$ requires that the task of constructing a description of the circuit C_n given input n be computable within some specified resource bound as a function of n, potentially making it comparable with classes defined by bounds on Turing machine time or space. Two such uniformity bounds are used in the work here: L and $\mathsf{DLOGTIME}$. Because these bounds are very restrictive, a special representation of the circuit C_n is used, namely, the ability to answer queries of the type of a gate and whether the output of one gate is an input to another gate.

We assume that the vertices of the circuit C_n are numbered from 0 to $|C_n|-1$. The *direct connection language* of a family of circuits C is the set of all tuples $\langle f, i, j, 1^n\rangle$ such that in C_n, vertex i has type f and there is an edge from vertex i to vertex j (Barrington et al., 1990). Given a computable function bounding the size of C and access to a membership oracle for the direct connection language, for any n it is straightforward to write out the list of vertices, edges, and types in C_n.

Then a circuit family C is L-*uniform* (resp., $\mathsf{DLOGTIME}$-*uniform*) if there is a Turing machine that runs in logarithmic space (resp., deterministic logarithmic time) to decide membership in the direct connection language of C.

Circuit Complexity Classes Circuit complexity classes classify circuit families and the languages they recognize based on uniformity, depth, size, fan-in bound, and the allowed gates. Since transformers have constant depth, circuit classes with constant depth are of particular interest; the classes that are used in the work we survey are:

- AC^0 contains those languages that can be recognized by families of circuits with unbounded fan-in, constant depth, and polynomial size.
- ACC^0 is like AC^0, but also has gates that output 1 iff the inputs sum to 0 modulo some constant.
- TC^0 is like AC^0, but also allows MAJORITY gates, which have unbounded fan-in and output 1 iff at least half of their inputs are 1.
- NC^1 is like AC^0, but with fan-in at most 2 and depth in $O(\log n)$.

The known relationships between these classes are:

$$\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$$

in the $\mathsf{DLOGTIME}$-uniform, L-uniform, and nonuniform settings; moreover, L-uniform $\mathsf{NC}^1 \subseteq \mathsf{L}$.

5.3 Logic

A formal language can also be defined as a set of finite strings that satisfy a closed formula of a logic. For more details, refer to Thomas (1997) or Straubing (1994).

In the *first-order logic of strings*, or FO, the formulas are the smallest set containing:

- Variables x, y, and so on.
- Atomic formulas $Q_a(x)$, $x = y$, $x < y$, where $a \in \Sigma$ is a symbol and x, y are variables.

- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\neg\phi_1$, where ϕ_1 and ϕ_2 are formulas.
- $\forall x.\phi$, $\exists x.\phi$, where x is a variable and ϕ is a formula.

Under the intended interpretation, variables stand for positions of a finite string w, and $Q_a(x)$ is true iff $w_x = a$. For example, if $\Sigma = \{a, b\}$, $\forall x.\forall y.Q_a(x) \wedge Q_b(y) \rightarrow x < y$ defines the regular language a^*b^*. The language defined by a closed formula ϕ consists of those strings that satisfy ϕ.

The languages definable in FO are exactly the *star-free* languages (McNaughton and Papert, 1971). Other variants add more quantifiers:

- FOC adds counting quantifiers $\exists^{=x} y.\phi$, which hold iff there are exactly x values of y that make ϕ true (Barrington et al., 1990).
- FOM adds majority quantifiers $\mathsf{M}x.\phi$, which hold iff at least half of the values of x make ϕ true (Barrington et al., 1990).

We are also interested in various sets of predicates:

- Modular predicates $\mathsf{MOD}^r_m(x)$, which hold iff $x \equiv r \pmod m$ (Barrington et al., 1992).
- BIT(x, y), which holds iff the y-th bit of x is 1.
- Mon, the set of all predicates on one position, possibly depending on n.[2]
- ARB, the set of all predicates on one or more positions.

A logic extended with predicates is conventionally written with the predicates in square brackets; for example, we write FO[BIT] for first-order logic with the BIT predicate.

In *linear temporal logic* or LTL (Kamp, 1968), every formula implicitly depends on a single time (or position). There are atomic formulas Q_a for every $a \in \Sigma$, the connectives $\wedge$, $\vee$, and $\neg$, as well as operators **since** and **until**. The formula α **since** β is true iff α was true at some past time i and β was true from i to now (exclusive). LTL is equivalent to FO (Kamp, 1968).

[2]Although Barrington et al. (2005) define Mon to be the collection of all monadic predicates without dependence on n, Barceló et al. (2024) do allow them to depend on n.

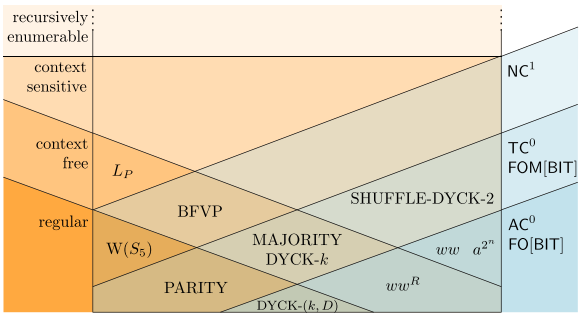

Figure 1: Relationship of some languages and language classes discussed in this paper (right) to the Chomsky hierarchy (left), assuming that $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$ and $\mathsf{L} \subsetneq \mathsf{NL}$. Circuit classes are DLOGTIME-uniform.

5.4 Relationships

Figure 1, which depicts the relationships between the language classes defined above, shows that the classes defined by circuits/logics cut across the (perhaps more familiar) Chomsky hierarchy. In this figure and in this section, all circuit classes are understood to be DLOGTIME-uniform unless specified otherwise.

5.4.1 Beyond AC^0

The classic examples of languages not in AC^0 are Parity and Majority. The language Parity $\subseteq \{0,1\}^*$ contains all bit strings containing an odd number of 1's, and Majority $\subseteq \{0,1\}^*$ consists of all bit strings in which more than half of the bits are 1's. Other problems in TC^0 but not AC^0 include sorting, integer multiplication (Chandra et al., 1984), and integer division (Hesse, 2001).

Dyck Languages The language Dyck-k for $k > 0$ is the language of strings over k pairs of parentheses that are correctly balanced and nested. If we write the i-th parenthesis pair as $(_i$ $)_i$ for each $i \in [k]$, then Dyck-k is generated by the context-free grammar $\{S \rightarrow (_i S)_i S \mid i \in [k]\} \cup \{S \rightarrow \varepsilon\}$. These languages are of interest because any context-free language can be obtained by applying a string homomorphism to the intersection of a Dyck language with a regular language (Chomsky and Schützenberger, 1963).

Some papers surveyed here consider variations on Dyck languages. The language Dyck-(k, D) for $D > 0$ is the subset of Dyck-k consisting of strings with maximum nesting depth D; it is a star-free regular language (and therefore in AC^0).

The language Shuffle-Dyck-k is the set of strings over k pairs of parentheses in which, for

each parenthesis pair, erasing the other types of parentheses leaves a correctly balanced and nested string. For example, [(()]) is in SHUFFLE-DYCK-2. If $k > 1$, SHUFFLE-DYCK-k is not context free.

5.4.2 Beyond TC^0

As we will see (§6.3.2), some transformer variants lie within TC^0. What problems lie beyond?

The Word Problem for Permutation Groups A permutation of $[k]$ is a bijection $\pi\colon [k] \to [k]$, and S_k is the set of all permutations of $[k]$. Treating S_k as an alphabet and compositions of permutations as strings, we can define the language $\mathrm{W}(S_k)$ of compositions of permutations of $[k]$ that equal the identity permutation. For example, in S_3, the permutation (120) maps $0 \mapsto 1$, $1 \mapsto 2$, and $2 \mapsto 0$, so that $\mathrm{W}(S_3)$ contains (120) $\circ$ (120) $\circ$ (120) but not (120) $\circ$ (120). These languages are easy for finite automata to recognize, but difficult with only fixed computation depth. Indeed, $\mathrm{W}(S_5)$ is complete for NC^1 under AC^0 reductions (Barrington, 1989), so it is not in TC^0, assuming that $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$ (as is widely believed). This makes it an example of a regular language that transformer encoders probably cannot recognize.

The languages $\mathrm{W}(S_k)$ have some relevance to natural language: they resemble expressions like *the child of the enemy of Ann* where the interpretation of *the child of* is (roughly) a permutation of possible referents (Paperno, 2022), and problems that have been used to benchmark transformers' state-tracking abilities (Kim and Schuster, 2023).

Other Languages that are widely believed to be not in TC^0 include:

- The language of closed Boolean formulas that are true (BFVP) is context-free but complete for NC^1 under $\mathsf{DLOGTIME}$ reductions (Buss, 1987), so it is outside TC^0 if $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$.
- Undirected graph connectivity is L-complete under L-uniform NC^1 reductions (Cook and McKenzie, 1987; Reingold, 2008), so it is outside L-uniform NC^1 (and therefore outside TC^0) if L-uniform $\mathsf{NC}^1 \subsetneq \mathsf{L}$.
- There is a context-free language L_P that is NL-complete under L reductions (Sudborough, 1975), so it is outside L (and therefore outside NC^1 and TC^0) if $\mathsf{L} \subsetneq \mathsf{NL}$.
- Solving systems of linear equalities and universal context-free grammar recognition are P-complete under L reductions (Jones and Laaser, 1976; Greenlaw et al., 1995), so they are outside TC^0 if $\mathsf{L} \subsetneq \mathsf{P}$.
- Matrix permanent is known to be outside of TC^0 (Allender, 1999).

5.4.3 Circuits and Logics

$\mathsf{DLOGTIME}$-uniform AC^0 and TC^0 are equivalent to $\mathsf{FO}[\mathsf{BIT}]$ and $\mathsf{FOM}[\mathsf{BIT}]$, respectively. There are many such equivalences between circuit classes and logics. As a rule of thumb, adding unbounded fan-in gates to a circuit family correlates with adding quantifiers to the corresponding logic, and increasing the degree of nonuniformity of a circuit family correlates with adding numerical predicates to the corresponding logic (Barrington and Immerman, 1994). For example, making AC^0 and TC^0 completely nonuniform corresponds to adding arbitrary numerical predicates (ARB) to FO and FOM, respectively (Immerman, 1997; Barrington et al., 1990).

As we will see below, circuits and logics have their advantages and disadvantages for capturing the expressivity of transformers. An advantage of the circuit approach is that they have a more transparent resemblance to transformers. Transformers are computations with bounded depth, so it's not hard to see that they should be computable by circuit families with bounded depth (AC^0 or TC^0). On the other hand, an advantage of the logical approach is that if we seek an exact characterization of transformers, it can be easier in a logic to add or remove quantifiers or predicates, to limit quantifier depth or number of variables, to partition terms into different sorts, and so on, than to make adjustments to a circuit family.

6 Current Results

While this area of research still has many unresolved questions, the emerging picture has three levels of expressivity. At the upper end are decoders or encoder–decoders with intermediate steps; these are equivalent to Turing machines (§6.1). At the lower end are encoders with leftmost-hard or rightmost-hard attention; these can recognize only languages in AC^0 (§6.2). In the middle are encoders with average-hard or softmax attention,

which are the least well-understood but appear to lie between AC^0 and TC^0 (§6.3).

In this section, ''transformer'' refers to a transformer encoder unless otherwise indicated.

6.1 Decoders with Intermediate Steps

Pérez et al. (2021) consider transformer encoder–decoders with several modifications:

- The PE includes components i, $1/i$, and $1/i^2$.
- In self attention, Eq. (3) takes the negative absolute value of the dot-product, and Eq. (4) uses average-hard attention.
- The FFNs use sigmoids instead of ReLUs.

As described above (§4.3.3), the decoder is allowed to run for arbitrarily many time steps until an acceptance criterion is met. Under these assumptions, transformer encoder–decoders can recognize any recursively enumerable language.[3] This result uses arbitrary precision, but as a corollary, it shows that a $T(n)$-time-bounded Turing machine can be simulated in a transformer using $O(\log T(n))$ precision and $O(T(n))$ intermediate steps.

Bhattamishra et al. (2020b) provide a simpler proof of Pérez et al.'s result by reducing to an RNN and appealing to the construction of Siegelmann and Sontag (1995). They do this for two sets of assumptions. First,

- The PE includes only i.
- The self attention sublayers are as above.
- The FFNs use saturated linear activation functions: $\sigma(x) = \max(0, \min(1, x))$.

Second, they show the same with no PE and standard dot-product attention with future masking.

Wei et al. (2022a) define a notion of *statistically meaningful* (SM) approximation and show that transformer encoder–decoders SM-approximate Turing machines. Both the decoder and Turing machine are limited to N time steps; additionally,

- The PE can be an arbitrary computable function on $[N]$.
- Attention is average-hard.
- The FFNs have three ReLU layers.

Feng et al. (2023) observe that the problems of evaluating arithmetic expressions or solving linear equations over $\mathbb{Z}_p$ are NC^1-hard under $\mathsf{DLOGTIME}$ reductions, so (if $\mathsf{TC}^0 \subsetneq \mathsf{NC}^1$) they cannot be solved by $O(\log n)$-precision transformer decoders without intermediate steps.[4] Similarly, the universal recognition problem for CFGs is P-complete, so (if $\mathsf{L} \subsetneq \mathsf{P}$) it cannot be solved by $O(\log n)$-precision transformer decoders without intermediate steps.

However, these problems can be solved by a transformer decoder using (a polynomial number of) intermediate steps. The decoder has GELU activations (Hendrycks and Gimpel, 2016) and PE including i and (for linear equation solving) $m^2 \sin \frac{2i\pi}{m}$ and $m^2 \cos \frac{2i\pi}{m}$ where m is the number of variables. More generally, they define a class of dynamic-programming algorithms that these transformers can solve using intermediate steps. All these decoders have parameters that depend on n.

Merrill and Sabharwal (2024) show that a transformer decoder with $O(\log(n + T(n)))$ precision and $O(T(n))$ intermediate steps can simulate a Turing machine for $T(n)$ steps, and in particular, decoders with a polynomial number of intermediate steps recognize *exactly* the languages in P. The proof is similar to that of Pérez et al. (2021), but uses a standard definition of transformers without PEs, relying only on the mild assumption that the input string begins with `BOS`.

6.2 Leftmost-hard/Rightmost-hard Attention

Hahn (2020) shows that leftmost-hard attention transformers cannot recognize Parity or Dyck-1, using a variant of Furst et al.'s random restriction method for proving that Parity is outside of AC^0.

Hao et al. (2022) show more generally that any language recognized by a transformer with leftmost-hard attention is in AC^0. The proof gives a normal form for transformers with leftmost-hard attention and uses it to construct an AC^0 circuit family. It uses the fact that only $O(\log n)$ bits of information are needed per position.

[3] Pérez et al. (2021) define both Turing machines and encoder–decoders to halt only when accepting. The construction could easily be modified to capture decidable languages.

[4] This uses the result of Merrill and Sabharwal (2023b), which would have to be adapted to transformer decoders, but this should be straightforward.

Barceló et al. (2024) give a lower bound on leftmost-hard-attention transformers with arbitrary PEs depending on a single position i and length n, including i, $\frac{1}{i+1}$, $(-1)^i$, $\cos \frac{\pi(1-2^{-i})}{10}$, and $\sin \frac{\pi(1-2^{-i})}{10}$. They show that these transformers can recognize any language definable in FO[Mon]. Their proof converts a FO[Mon] formula to LTL (§5.3), which is simulated in a transformer.

Angluin et al. (2023) exactly characterize rightmost-hard-attention transformers with strict future masking. Without PEs, these transformers recognize exactly the class of star-free languages, that is, languages definable in FO. With periodic PEs, they are exactly equivalent to FO[MOD], and with arbitrary PEs, they are exactly equivalent to FO[Mon]. Strict masking is important, as nonstrict masking is less expressive. They give two proofs of the star-free to transformer direction, one which goes through LTL (§5.3) and one which uses Krohn-Rhodes theory. These proofs use a Boolean-valued version of RASP (Weiss et al., 2021) as an intermediate representation.

6.3 Average-hard and Softmax Attention

Theoretical results on average-hard and softmax attention transformers have not yet clearly separated the two, so we treat them together. Both kinds of attention enable counting, which can be used to solve problems like Majority that are outside AC^0. But these transformers are no more powerful than DLOGTIME-uniform TC^0, implying that they likely cannot solve problems complete for NC^1, L, and other classes believed to be above TC^0 (§5.4).

6.3.1 Lower Bounds: Particular Languages

The languages Majority, Dyck-k, and Parity are all not in AC^0, so are interesting test cases.

Pérez et al. (2019) prove that a transformer encoder–decoder with a trivial decoder and without any PE recognizes Majority; Merrill et al. (2022) prove the same for transformer encoders.

Bhattamishra et al. (2020a) prove that Shuffle-Dyck-k (which equals Dyck-1 when $k = 1$) is recognizable by a soft-attention transformer with future masking, no PE, no layernorm, and no residual connections. Yao et al. (2021) show that a transformer decoder can generate Dyck-k using $O(\log n)$ precision, softmax and leftmost-hard attention, future masking, and a PE including i/n, i/n^3, and n. They also give constructions for Dyck-(k, D).

Chiang and Cholak (2022) show that transformers whose PE includes i/n and $(-1)^i = \cos i\pi$ can recognize Parity.

On the other hand, Hahn (2020) shows that softmax attention transformers cannot generate Parity or Dyck-2 under the following two conditions:

1. all position-wise functions are Lipschitz-continuous, and
2. generation is defined using the KL divergence criterion in Eq. (5).

The apparent contradiction is resolved by considering the different assumptions underlying each result. Chiang and Cholak (2022) address this by giving two constructions corresponding to Hahn's two conditions. The first has Lipschitz-continuous position-wise functions, but has high cross-entropy (§4.3.1); as a generator, it would not meet criterion (5). The second construction uses layernorm with $\varepsilon_{\mathcal{N}} = 0$, which is not Lipschitz-continuous, but it has arbitrarily low cross-entropy.

A number of authors have tested empirically whether transformers can learn the above languages. Ebrahimi et al. (2020) find that they are competitive with LSTMs at learning Dyck-2 and Dyck-4, and that prepending a BOS symbol helps.

Bhattamishra et al. (2020a) train transformers with future masking and no PE on Dyck-1 and Shuffle-Dyck-k, finding near-perfect learning and length generalization. For the languages Dyck-$(1, D)$ with learned or sinusoidal PEs, they find that the models do not generalize well for $D > 1$. Yao et al. (2021) then investigate Dyck-(k, D) for several values of k and D and several PEs. They report strong generalization only when using i/n for the PE, and posit that this is the key. It is hard, however, to directly compare the two results: Bhattamishra et al. (2020a) require correct prediction of the possible next symbols at each string prefix, while Yao et al. (2021) average over predictions of right brackets.

Delétang et al. (2023) study experimentally how well transformers (and other networks) learn tasks at various levels of the Chomsky hierarchy, including generalization to longer strings. They find that transformers learn Majority, but not Parity.

6.3.2 Upper Bounds: TC^0

Merrill et al. (2022) prove an upper bound analogous to that of Hao et al. (2022), but for average-hard-attention transformers. They show that an average-hard-attention transformer with activations in $\mathbb{F}$ can be simulated in TC^0. Strobl (2023) tightens this bound to L-uniform TC^0.

Furthermore, Merrill and Sabharwal (2023a) show that softmax attention, $O(\log n)$-precision transformers are in L-uniform TC^0, and then tighten this bound to DLOGTIME-uniform TC^0 (Merrill and Sabharwal, 2023b). The proof constructs subroutines to answer queries about the types of nodes and connectivity of pairs of nodes in the computation graph of a transformer, and shows that these queries can be translated to queries for a TC^0 circuit family with $O(\log n)$ time overhead.

An upper bound of DLOGTIME-uniform TC^0 immediately implies an upper bound of FOM[BIT] (Merrill and Sabharwal, 2023b). Chiang et al. (2023) prove a tighter upper bound using a logic called FOC[MOD; +], but on transformers with $O(1)$ precision. This result is discussed below.

6.3.3 Other Lower Bounds

In addition to explicit constructions for particular languages mentioned above, various lower bounds have been proven, which are quite diverse.

Counter Machines Bhattamishra et al. (2020a), following Merrill et al. (2020), define a subclass of counter machines called *simplified and stateless k-counter machines* (SSCMs). These can update each counter based on the current input symbol, but have no state and cannot read the counters until the end of the string. They show that any SSCM can be converted to an equivalent transformer with future masking and no residual connections.

Finite Automata Liu et al. (2023) study the ability of transformers with future masked attention to simulate deterministic finite automata (DFAs), in the sense of computing not only the same acceptance decision but also the same state sequence. Although a transformer with depth N can simulate a DFA for N timesteps, Liu et al. show how to construct lower-depth *shortcuts* for subclasses roughly corresponding to classes of regular languages in Figure 1. Though the parameters of these constructions depend on N, in the context of this survey, a noteworthy finding is that any regular language in ACC^0 can be recognized up to length N by a transformer whose FFNs use sine activations and whose *number* of parameters is independent of N.

First-order Logic Chiang et al. (2023) obtain both an upper and a lower bound by defining a logic FOC[MOD; +], which is first-order logic with counting quantifiers, using two sorts for positions and counts (Immerman, 1999, p. 185–187), where positions have the MOD predicate (but not $<$ or $=$), and counts have $<$, $+$, and $=$, capturing the fact that transformers can add and compare activations, but not positions. They show that this logic is intermediate in expressivity between $O(1)$-precision and infinite-precision transformers. The lower-bound proof uses a normal form that eliminates quantifiers over counts and makes quantifiers over positions have depth 1; a perhaps surprising consequence is that $O(1)$-precision transformers are no more powerful than 2-layer uniform-attention transformers.

Temporal Logic Barceló et al. (2024) show that average-hard-attention transformers with arbitrary PEs depending on a single position i and length n, including i, $\frac{1}{i+1}$, $(-1)^i$, $\cos \frac{\pi(1-2^{-i})}{10}$, and $\sin \frac{\pi(1-2^{-i})}{10}$, can recognize any language definable in LTL with counting operators, Presburger arithmetic on counts, and predicates in Mon.

Programming Languages Weiss et al. (2021) introduce the RASP (Restricted Access Sequence Processing) language as an abstraction of transformers, discussing how its components relate to the transformer architecture. However, they do not prove any relationship. Lindner et al. (2023) present Tracr, a compiler from RASP programs to transformers. To do so, they impose some restrictions: a maximum input length, given at compile time; a mandatory `BOS` token; and the removal of *selector composition*, a RASP operation with no clear parallel in transformers. They rewrite several programs from Weiss et al. (2021) without this operation. In the other direction, Friedman et al. (2023) define a restricted class of transformers that can be learned and decompiled into RASP. Finally, Angluin et al. (2023) use a version of RASP restricted to Boolean values, and Zhou et al. (2023) use a restricted version of RASP to explore length generalization.

7 Conclusions

Out of the large body of research surveyed above, we highlight several conclusions:

1. Transformer decoders can use intermediate steps to simulate Turing machines; with unbounded steps, they are Turing-complete.

2. Regarding the expressivity of transformer encoders, circuit complexity and logic are especially promising frameworks.

3. Leftmost-hard-attention transformer encoders are in AC^0 and cannot solve some intuitively easy problems, like PARITY and MAJORITY.

4. Softmax and average-hard attention give transformer encoders the ability to count. Still, they lie within TC^0 and likely cannot solve problems like evaluating closed Boolean formulas.

Some open questions that we think should be priorities for future research are:

5. Some variants (PEs, average-hard vs. softmax attention, pre-norm vs. post-norm, the presence of BOS/EOS/CLS) appear to be instrumental in proofs reviewed here; can their effect on expressivity be clarified?

6. Can the expressivity of softmax-attention transformers be characterized more tightly or even exactly in terms of some logic?

7. Given the current practical importance of decoder-only transformers and chain-of-thought, what further insights can circuits or logic provide into transformer decoders?

We hope this paper can serve as a valuable resource for researchers pursuing these and other questions.

Acknowledgments

We would like to thank Frank Drewes, Jon Rawski, Ashish Sabharwal, and the anonymous reviewers as well as the TACL action editor for their valuable comments on earlier versions of this paper.

References

Joshua Ackerman and George Cybenko. 2020. A survey of neural networks and formal languages. *arXiv preprint arXiv:2006.01338*.

Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Physics of language models: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*.

Eric Allender. 1999. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999(7). https://doi.org/10.4086/cjtcs.1999.007

Dana Angluin, David Chiang, and Andy Yang. 2023. Masked hard-attention transformers and Boolean RASP recognize exactly the star-free languages. *arXiv preprint arXiv:2310.13897*.

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. In *NIPS 2016 Deep Learning Symposium*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the Third International Conference on Learning Representations (ICLR)*.

Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. 2024. Logical languages accepted by transformer encoders with hard attention. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

David A. Barrington. 1989. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164. https://doi.org/10.1016/0022-0000(89)90037-8

David A. Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. 1992. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44(3):478–499. https://doi.org/10.1016/0022-0000(92)90014-A

David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. 2005. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *Journal of Computer and System Sciences*, 70(2):101–127. https://doi.org/10.1016/j.jcss.2004.07.004

David A. Mix Barrington, Neil Immerman, and Howard Straubing. 1990. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306. https://doi.org/10.1016/0022-0000(90)90022-D

David Mix Barrington and Neil Immerman. 1994. Time, hardware, and uniformity. In *Proceedings of the IEEE 9th Annual Conference on Structure in Complexity Theory*, pages 176–185. https://doi.org/10.1109/SCT.1994.315806

Valeriu Beiu and John G. Taylor. 1996. On the circuit complexity of sigmoid feedforward neural networks. *Neural Networks*, 9(7):1155–1171. https://doi.org/10.1016/0893-6080(96)00130-X, PubMed: 12662590

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020a. 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. https://doi.org/10.18653/v1/2020.emnlp-main.576

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. 2020b. On the computational power of Transformers and its implications in sequence modeling. In *Proceedings of the 24th Conference on Computational Natural Language Learning (CoNLL)*, pages 455–475. https://doi.org/10.18653/v1/2020.conll-1.37

Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. 2023. Simplicity bias in Transformers and their ability to learn sparse Boolean functions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5767–5791. https://doi.org/10.18653/v1/2023.acl-long.317

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pages 1877–1901.

Samuel R. Buss. 1987. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 123–131. https://doi.org/10.1145/28395.28409

Ashok K. Chandra, Larry Stockmeyer, and Uzi Vishkin. 1984. Constant depth reducibility. *SIAM Journal of Computing*, 13(2):423–439. https://doi.org/10.1137/0213028

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7654–7664. https://doi.org/10.18653/v1/2022.acl-long.527

David Chiang, Peter Cholak, and Anand Pillay. 2023. Tighter bounds on the expressivity of transformer encoders. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562.

N. Chomsky and M. P. Schützenberger. 1963. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161, Elsevier. https://doi.org/10.1016/S0049-237X(08)72023-8

Stephen A. Cook and Pierre McKenzie. 1987. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(3):385–394. https://doi.org/10.1016/0196-6774(87)90018-6

G. Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314. https://doi.org/10.1007/BF02551274

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2023. Neural networks and the Chomsky hierarchy. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*.

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 (NAACL HLT)*, pages 4171–4186. https://doi.org/10.18653/v1/N19-1423

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306. https://doi.org/10.18653/v1/2020.findings-emnlp.384

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind Chain of Thought: A theoretical perspective. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. 1968. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283. https://doi.org/10.1007/BF01694011

Dan Friedman, Alexander Wettig, and Danqi Chen. 2023. Learning Transformer programs. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Merrick Furst, James B. Saxe, and Michael Sipser. 1984. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27. https://doi.org/10.1007/BF01744431

Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. 1995. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press. Preliminary version of Appendix A available as Technical Report TR91-11, University of Alberta, Department of Computing Science. https://doi.org/10.1093/oso/9780195085914.001.0001

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. *Transactions of the Association for Computational Linguistics*, 10:800–810. https://doi.org/10.1162/tacl_a_00490

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

William Hesse. 2001. Division is in uniform TC^0. In *Automata, Languages and Programming (ICALP)*, pages 104–114. Springer. https://doi.org/10.1007/3-540-48224-5_9

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. https://doi.org/10.18653/v1/2020.emnlp-main.156

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Austin Huang, Suraj Subramanian, Jonathan Sum, Khalid Almubarak, and Stella Biderman. 2022. The annotated Transformer. Based on original version by Sasha Rush.

Neil Immerman. 1997. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778. https://doi.org/10.1137/0216051

Neil Immerman. 1999. *Descriptive Complexity*. Springer. https://doi.org/10.1007/978-1-4612-0539-5

Neil D. Jones and William T. Laaser. 1976. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105–117. https://doi.org/10.1016/0304-3975(76)90068-2

Johan Anthony Willem Kamp. 1968. *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, University of California, Los Angeles.

Najoung Kim and Sebastian Schuster. 2023. Entity tracking in language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3835–3855. https://doi.org/10.18653/v1/2023.acl-long.213

Chu-Cheng Lin, Aaron Jaech, Xin Li, Matthew R. Gormley, and Jason Eisner. 2021. Limitations of autoregressive models and their alternatives. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, pages 5147–5173. https://doi.org/10.18653/v1/2021.naacl-main.405

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. 2022. A survey of transformers. *AI Open*, 3:111–132. https://doi.org/10.1016/j.aiopen.2022.10.001

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. 2023. Tracr: Compiled transformers as a laboratory for interpretability. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 37876–37899.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*.

Robert McNaughton and Seymour A. Papert. 1971. *Counter-Free Automata*. MIT Press.

William Merrill. 2019. Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13. https://doi.org/10.18653/v1/W19-3901

William Merrill. 2020. On the linguistic capacity of real-time counter automata. *arXiv preprint arXiv:2004.06866*.

William Merrill. 2021. Formal language theory meets modern NLP. *arXiv preprint arXiv:2102.10094*.

William Merrill. 2023. Formal languages and the NLP black box. In *Developments in Language Theory*, pages 1–8. https://doi.org/10.1007/978-3-031-33264-7_1

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 (EMNLP)*, pages 1766–1781. https://doi.org/10.18653/v1/2021.emnlp-main.133

William Merrill and Ashish Sabharwal. 2023a. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545. https://doi.org/10.1162/tacl_a_00562

William Merrill and Ashish Sabharwal. 2023b. A logic for expressing log-precision transformers. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

William Merrill and Ashish Sabharwal. 2024. The expressive power of transformers with chain of thought. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

William Merrill, Ashish Sabharwal, and Noah A. Smith. 2022. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856. https://doi.org/10.1162/tacl_a_00493

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. A formal hierarchy of RNN architectures. In *Proceedings of the 58th Annual Meeting of*

the Association for Computational Linguistics (ACL), pages 443–459. https://doi.org/10.18653/v1/2020.acl-main.43

Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2022. Show your work: Scratchpads for intermediate computation with language models. In *Proceedings of the Workshop on Deep Learning for Code (DLAC)*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Denis Paperno. 2022. On learning interpreted languages with recurrent models. *Computational Linguistics*, 48(2):471–482. https://doi.org/10.1162/coli_a_00431

Ian Parberry. 1994. *Circuit Complexity and Neural Networks*. MIT Press. https://doi.org/10.7551/mitpress/1836.001.0001

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. 2021. Attention is Turing-complete. *Journal of Machine Learning Research*, 22:75:1–75:35.

Mary Phuong and Marcus Hutter. 2022. Formal algorithms for transformers. *arXiv preprint arXiv:2207.09238*.

Jorge Pérez, Javier Marinković, and Pablo Barceló. 2019. On the Turing completeness of modern neural network architectures. In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Omer Reingold. 2008. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24. https://doi.org/10.1145/1391289.1391291

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. 2023. Representational strengths and limitations of transformers. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Hava T. Siegelmann and Eduardo D. Sontag. 1994. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360. https://doi.org/10.1016/0304-3975(94)90178-3

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1). https://doi.org/10.1006/jcss.1995.1013

Jiří Šíma and Pekka Orponen. 2003. General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, 15(12):2727–2778. https://doi.org/10.1162/089976603322518731, PubMed: 14629867

Michael Sipser. 2013. *Introduction to the Theory of Computation*, 3rd edition. Cengage Learning.

Kai-Yeung Siu, Vwani Roychowdhury, and Thomas Kailath. 1995. *Discrete Neural Computation*. Prentice Hall.

Howard Straubing. 1994. *Finite Automata, Formal Logic, and Circuit Complexity*. Springer. https://doi.org/10.1007/978-1-4612-0289-9

Lena Strobl. 2023. Average-hard attention transformers are constant-depth uniform threshold circuits. *arXiv preprint arXiv:2308.03212*.

I. H. Sudborough. 1975. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76. https://doi.org/10.1016/S0022-0000(75)80014-6

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54. https://doi.org/10.18653/v1/W19-3905

Wolfgang Thomas. 1997. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 389–455. Springer. https://doi.org/10.1007/978-3-642-59126-6_7

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez,

Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep Transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*. https://doi.org/10.18653/v1/P19-1176

Colin Wei, Yining Chen, and Tengyu Ma. 2022a. Statistically meaningful approximation: A case study on approximating Turing machines with transformers. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*.

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 (ACL)*, pages 740–745. https://doi.org/10.18653/v1/P18-2117

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like Transformers. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 11080–11090.

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 (ACL-IJCNLP)*, pages 3770–3785. https://doi.org/10.18653/v1/2021.acl-long.292

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. 2020. Are Transformers universal approximators of sequence-to-sequence functions? In *8th International Conference on Learning Representations (ICLR)*.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. 2024. What algorithms can Transformers learn? A study in length generalization. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.