A theory of the computational power and limitations of language modeling architectures

by

William C. Merrill

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Center for Data Science

New York University

August, 2025

Dr. Tal Linzen

Dr. Ashish Sabharwal

Dedication

To my grandmothers, Audrey Merrill and Mary Anne Ziezielewicz Forshaw O'Keefe, for finding me Old Norse books and years of making pierogis together.

Acknowledgments

Ósviðr maðr	You should lie down to sleep
vakir um allar nætr	and not think about tomorrow;
ok hyggr at hvívetna.	you'll take care of it then.
Þá er móðr.	If you worry at night, you get nothing done,
Er at morni kømr,	and you're in worse shape for the day.
allt er víl sem var.	

Words attributed to Odin by an anonymous Norse skald, *Hávamál* 23 [Cra19]

First I would like to thank my advisor, Tal Linzen, and close mentor, Ashish Sabharwal, for their roles in my development as a researcher. An offhand conversation with Ashish in 2021 led to an amazing long-term collaboration on the research program in this thesis. I would also like to thank my other committee members He He, Matus Telgarsky, and Joan Bruna. It goes without saying that any remaining errors are my own. I also appreciate the influence and advice of Mehryar Mohri, Julia Kempe, and Sam Bowman, among other faculty across data science, computer science, linguistics, and philosophy at NYU. In addition, I thank Kathryn Angeles, Joshua Gilmore, and others for their administrative efforts that enabled me to focus primarily on research throughout my PhD. And of course, thanks to the National Science Foundation, Two Sigma, and the Allen Institute for AI (Ai2) for funding the research presented here.

I was lucky to have great friends, collaborators, and interlocutors at NYU. For interesting

research collaborations, I thank Alex Warstadt, Nikos Tsilivis, Jacob Pfau, Jackson Petty, Michael Hu, and Ofir Press. I thank Norihito Naka and Selim Jerad for trusting me as their research mentor. It was great to share a lively office with Michael Hu and Wentao Wang, and I appreciate Ofir Press's tendency to burst into it uninvited but (usually) welcome. More broadly, I appreciate the entire CAP lab for interesting collaborations, deeply interdisciplinary discussions, and some random but memorable social outings. I appreciated long walks and conversations with Sophie Hao, Noah Amsel, Noami Saphra, and Vishakh Padmakumar, as well as squash with Jackson Petty and table tennis with Anthony Chen. I also thank Noah for driving me to Aspen, where we survived a bear encounter in a Mexican restaurant. I thank Alberto Bietti for research discussions and starting the transformer theory seminar at the Flatiron Institute, which was a great way to spend Friday afternoons. Thanks to Ryan Teehan and Francheska Jimenez for hosting CDS cocktail happy hours. There are many others at NYU whom I thank for fond memories, including: Nick Lourie, Lindia Tjuatja, Angelica Chen, Betty Hou, and Vlad Sobal.

Before and during my PhD, I was lucky to have a separate community of colleagues and friends in Seattle. I appreciate the investment and support of my mentors at Ai2: Noah A. Smith, Yoav Goldberg, Roy Schwartz, and Ashish Sabharwal. Thanks as well to Dirk Groeneveld and others on the OLMo team for welcoming a theorist into their midst. I appreciated hiking, climbing (rocks and otherwise), and drinking boba with Vivek Ramanujan, Alexis Ross, Apoorv Khandelwal, Amita Kamath, Sonia Murty, Ana Marasovic, Pete Walsh, Kyle Richardson, Andi Peng, Yanai Elazar, Zhaofeng Wu, Tom Sherborne, Ananya Jha, Hamish Ivison, Nishant Subramani, among many other people. Lastly, I thank my former roommates Brian, Angus, Eugine, Parthib, Joe, Gray, Varun, Caleb, and Corbin for their company, in particular on daily walks to the crew docks at the University of Washington, where, if you keep quiet, you can glimpse beavers gliding through Lake Washington under the backdrop of the Cascades.

Many people at other institutions were also influential in my PhD research. Thanks to Cyril Allauzen for mentoring me at Google NYC. Thanks to the organizers of DLT in Umeå, ICGI in Ra-

bat, and the Transformers as a Computational Model Workshop in Berkeley, and the Limitations of Large Language Models Workshop in Bielefeld for inviting me to speak and attend. Thanks as well to Leonie Weissweiler for showing me around when I was in Bielefeld. I also appreciate Shawn Tan, Jacob Andreas & Alexis Ross, Ellie Pavlick & Apoorv Khandelwal, and Hinrich Schütze & Ali Modarressi, and others for hosting me to give talks. I thank John Hewitt for nights of conversation and advice over drinks. I also thank Gail Weiss, Lena Strobl, Andy Yang, David Chiang, and other organizers and members of the languages and neural networks (FLaNN) Discord, which has grown into a vibrant community of passionate people doing interesting research.

My gratitude extends further to my undergrad years at Yale University, where Dana Angluin and Bob Frank were impeccable mentors and role models. Thanks also to Dragomir Radev, Roberta Frank, Valerie Hansen, Hadas Kotek, Claire Bowern, and Josh Phillips for going above and beyond as teachers. Going back to high school, I appreciate Erin Schmitz and Lutz Holzinger for fostering my curiosity for scientific research and Alec Marantz and Phoebe Gaston at NYU linguistics for taking me on as a research intern in high school.

I appreciate the community I have in New York City from various chapters of my life and their presence throughout my PhD. This often manifested as time spent walking or hanging out at the Hideout Chai Bar, with characters including Brian, Gray, Alexandros, Varun, Stefan, Chris, Erica, Josh Y, Noah, Zach, Josh C, Andreas, Michelle, Mubarak, Grant, and others. I am particularly thankful for friends from high school whose presence in my life has only grown richer with time since the days where we built video games together, including David, Jack, Jasen, Julian, Lucas, Toby, Duke, and Gabe. Finally, I thank my family for their support and love. This extends especially to my parents, Mary Beth Forshaw and Thomas Merrill. Thank you for encouraging me to forge own path and being proud of me for it, well before ChatGPT was a household name. To all my friends and family, know that you all have played a role in shaping me and this dissertation.

Abstract

Scaling up language models (LMs) has driven recent progress in NLP and AI. In many ways, it appears that making LMs larger unlocks more complex abilities, but to what extent is progress due to scaling bottlenecked by the fundamental computational limitations of LM architectures? This thesis develops a theory of the computational power of LM architectures, providing a principled framework for addressing this question. First, I consider the computational power of the transformer architecture used by the vast majority of LMs, finding that it can only express problems in the complexity class uniform TC^0. Thus, under standard complexity conjectures, transformers cannot express many simple computational problems including state tracking, evaluating compositional formulas, and graph connectivity. I then consider how chain-of-thought reasoning extends the expressive power of transformer LMs, ameliorating this fundamental weakness, and the potential for LM architectures with expressive power beyond that of transformers. Finally, I discuss the broader implications of these theoretical results for LMs capabilities in practice. A major conceptual takeaway is that there is a fundamental tradeoff between parallelism and expressive power for LM architectures: the parallelism so essential for scaling up transformer LMs also precludes them from expressing many simple computational problems. Overall, the theory developed in this thesis lets us more precisely understand the computational model of LMs, reason about LMs' fundamental limitations, and also provide a strong foundation upon which to develop novel language modeling architectures and inference methods.

Contents

List of Figures

1 Introduction

Since 2018, much of the progress in AI has been intimately tied to large language models (LMs). Large LMs are huge neural neural networks, typically built with the transformer architecture [Vas+17], that are trained to sequentially predict large volumes of text one piece at a time. LMs, originally niche models with a narrow set of uses [Bra+07], have been been given a new life as general-purpose systems that can solve many tasks out of the box or with some minor adaptation. Simply make the LMs bigger and train them on more data, the story goes [Kap+20; Wei+22b], and they will become even more capable, improving in performance on some capabilities and acquiring other ones that smaller LMs did not have at all.

But scaling cannot be the full story: the representational power of the models matters as well, beyond the sheer amount of data and compute going into their training. Past eras of NLP had their own large LMs [Bra+07], but the fact that these were n-gram models fundamentally constrained their capabilities for language processing. In contrast, what is the ceiling on the kinds of problems that can be solved by scaling today's LMs? And what role does the LM architecture (e.g., the transformer) play in controlling the kinds of tasks an LM can solve and the algorithms it can implement? This dissertation takes a step to addressing these questions by developing a theory of the computational expressive power (or *expressivity*) of transformers and related LM architectures. That is, it examines the kinds of computation that LMs can represent, and, perhaps more interestingly, the kinds of computation that they *cannot*. In particular, Chapter 3 will show using circuit complexity theory that the transformer, the dominant architecture for

language modeling, is fairly constrained in the kind of computation it can express: formally, it can only solve problems in the complexity class TC^0, representing highly parallelizable computation. This means that, under standard complexity conjectures, transformers cannot express many computationally simple—but inherently sequential—problems, such as state tracking, evaluating compositional formulas, and graph connectivity—no matter how large or well trained they are.

Going beyond simple transformer LMs, we will also show in Chapter 4 how this theory can help us understand the value of approaches like chain-of-thought (CoT) reasoning and motivate the design of new LM architectures. As we will see, CoT extends the theoretical expressive power of transformer LMs beyond TC^0, allowing them to represent inherently sequential computation. Furthermore, we will discuss how these theoretical results have guided the development of LM architectures that can express some inherently sequential problems that transformers cannot (under standard conjectures). Thus, theoretical understanding does not just better elucidate current LMs, but it can lead to more powerful, efficient, and expressive LMs in the future.

Towards this end, we conclude in Chapter 5 with a discussion of some high-level takeaways from the theory of transformer expressivity. A central one is a notion of a parallelism tradeoff: the computational limitations of transformers and related architectures in some sense come from the fact that these architectures are designed to be scaled up. Scaling up requires (or at least greatly incentivizes) parallelism, and this parallelism is fundamentally at odds with the expressive power for *inherently sequential* computation. We also speculatively discuss the implications of these theoretical results for LMs' ability to process the syntax and semantics of natural language, model the physical world, perform in-context learning, and reason in a general-purpose way.

Overall, this dissertation develops an expressivity theory for transformers and related architectures, explores how these insights can guide the development of new architectures, and considers the implications of the theory for a variety of capabilities of interest for LMs. Hopefully, these contributions may play a small part in recasting the alchemy of deep learning to a principled science built on solid theoretical foundations.

2 Preliminaries

We will use the notation $\mathbf{M} : \mathbb{Q}^n \to \mathbb{Q}^m$ to denote a linear operator (matrix) of shape $n \times m$, which can apply to column vectors. Without loss of generality, these linear transformations can be generalized to affine transformations, which we surpress for notational clarity.

2.1 The transformer architecture

A transformer is a neural network parameterizing a function $\Sigma^* \to \Sigma$. I now define the high-level structure of the transformer in terms of its core components.

Definition 2.1 ([MS23a; MS24b]). *A p-precision decoder-only transformer with h heads, d layers, model dimension m (divisible by h), and feedforward width w is parameterized by*

1. *An embedding matrix $\mathbf{E} : \mathbb{Q}^{|\Sigma|} \to \mathbb{Q}^m$ and position embedding $\pi : \mathbb{N} \to \mathbb{Q}$ that are analytic functions whose rational polynomial power series converge [RT92], capturing standard choices.*
2. *For $1 \le \ell \le d$ and $1 \le k \le h$, query, key, and value projections $\mathbf{Q}_k^\ell, \mathbf{K}_k^\ell, \mathbf{V}_k^\ell : \mathbb{Q}^m \to \mathbb{Q}^{m/h}$.*
3. *For $1 \le \ell \le d$, an output projection matrix $\mathbf{O}^\ell : \mathbb{Q}^m \to \mathbb{Q}^m$, initial feedforward projection $\mathbf{W}_1^\ell : \mathbb{Q}^m \to \mathbb{Q}^w$, and final feedforward projection $\mathbf{W}_2^\ell : \mathbb{Q}^w \to \mathbb{Q}^m$.*
4. *An output projection matrix $\mathbf{U} : \mathbb{Q}^m \to \mathbb{Q}^{|\Sigma|}$.*

Given this parameterization of a transformer, we can define the function it computes, following the standard definition of the transformer architecture [Vas+17].

Definition 2.2 ([MS23a; MS24b]). *A transformer $\mathcal{T}$ computes a function $\Sigma^* \to \Sigma$. Given $w \in \Sigma^n$,*

1. *Embedding layer: For $1 \le i \le n$, $\mathbf{h}_i^0 = \mathbf{E}[w_i] + \pi(i)$, where $\mathbf{E}[w_i]$ is the embedding of w_i.*

2. *Self-attention sublayer: At each layer ℓ, we compute h attention heads. We first compute the sublayer input $\hat{\mathbf{h}}_i^{\ell-1} = \mathsf{layer\_norm}(\mathbf{h}_i^{\ell-1})$. For each head $1 \le k \le h$, we compute queries $\mathbf{q}_{i,k}^\ell = \mathbf{Q}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$, keys $\mathbf{k}_{i,k}^\ell = \mathbf{K}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$, and values $\mathbf{v}_{i,k}^\ell = \mathbf{V}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$. The output of head k is*

$$\mathbf{a}_{i,k}^\ell = \sum_{j=1}^{m} \frac{\exp(1/\tau \cdot (\mathbf{q}_{i,k}^\ell)^\top \mathbf{k}_{j,k}^\ell)}{Z_{i,k}^\ell} \cdot \mathbf{v}_{j,k}^\ell, \text{ where } Z_{i,k}^\ell = \sum_{j=1}^{m} \exp(1/\tau \cdot (\mathbf{q}_{i,k}^\ell)^\top \mathbf{k}_{j,k}^\ell)$$

   and $\tau = \sqrt{d/h}$. Further, let $m = i$ for causally masked attention (the default), $m = i - 1$ for strictly causally masked attention, and $m = n$ for unmasked attention. The self-attention sublayer then concatenates the output of h heads and then applies a linear projection to compute

$$\mathbf{g}_i^\ell = \mathbf{h}_i^{\ell-1} + \langle \mathbf{a}_{i,1}^\ell, \ldots, \mathbf{a}_{i,h}^\ell \rangle \cdot \mathbf{O}^\ell.$$

3. *Feedforward sublayer: The feedforward sublayer computes the following tokenwise transformation: $\mathbf{h}_i^\ell = \mathbf{g}_i^\ell + \mathbf{W}_2^\ell (\mathbf{W}_1^\ell \mathbf{g}_i^\ell)_+$.*

4. *Unembedding layer: The transformer outputs the token associated with $\arg\max \mathbf{h}_n^d \mathbf{U}$.*

In this way, a transformer $\mathcal{T}$ implements a function $\Sigma^* \to \Sigma$. We will write $\mathcal{T}(w)$ to represent the application of transformer $\mathcal{T}$ to sequence w. To view transformers as language recognizers, we can simply identify a token in Σ as a special "accept" symbol a and say that the transformer recognizes a language L if, for any string w, $w \in L \Leftrightarrow \mathcal{T}(w) = a$.

Transformer variants. Certain idealizations are sometimes made to the transformer when implementing constructions for algorithmic problems. First, the transformer can be modified to have averaging hard attention (also called saturated attention; [Mer+21a; MSS22]), meaning

that attention weight is only assigned to positions at which the key-query match is maximized. Formally, we can define averaging hard attention by replacing the attention heads with

$$\mathbf{a}_{i,k}^{\ell} = \lim_{\tau \to 0} \sum_{j=1}^{m} \frac{\exp(1/\tau \cdot (\mathbf{q}_{i,k}^{\ell})^{\top} \mathbf{k}_{j,k}^{\ell})}{Z_{i,k}^{\ell}} \cdot \mathbf{v}_{j,k}^{\ell}.$$

This corresponds to the "low-temperature limit" of the attention head, causing it to attend uniformly over all the tokens that maximize the attention score [Mer+21a]. We refer to a model with attention heads of this form as an averaging hard attention transformer (AHAT).

The second change to the transformer that is sometimes made is allowed masked pre-norm. This means we redefine $\hat{\mathbf{h}}_i^{\ell-1} = \text{layer\_norm}(\mathbf{W}\mathbf{h}_i^{\ell-1})$ for some matrix $\mathbf{W}$. Many of our constructions with transformers will use these modifications, though our upper bounds on the expressive power of transformer (Theorem 3.1) apply with or without them.

Numerical datatype. A transformer processes strings by embedding them into numerical quantities in some datatype $\mathbb{D}$ that permits addition (+), multiplication ($\cdot$), and division (/), as well as some elementwise functions like exp. In much of the published work that culminated in this thesis, $\mathbb{D}$ has typically been log-precision floats, i.e., floating-point numbers with $O(\log n)$ bits of precision, where n is the input sequence length. On the other hand, in this thesis, I will simplify the argument by instead letting $\mathbb{Q}$ be arbitrary rationals [Chiang, p.c.; Chi24]. This elegantly circumvents low-level issues with floating-point arithmetic such as the non-associativity of addition and multiplication that should bear much on the expressivity of transformers in practice. Abstracting away these details lets us present a cleaner proof in a simpler formal model.

However, one data type issue must still be clarified to make this rational data type fully well-defined. In principle, the nonlinearities $\sqrt{\cdot}$ and exp used in the transformer computation graph return irrational outputs, though, in practice, of course, they are implemented with rational approximations on hardware. To capture this in our model, we will assume that these functions are

approximated by their Taylor expansion with at most a polynomial number of terms, providing a close rational approximation to the true irrational output. This approximation is likely more faithful than the lower-precision approximations used in practice, but our arguments would go through if the precision of the transformer is constrained as well.

2.2 Circuit complexity

Circuits are a model of computation introduced in complexity theory introduced to formalize parallel computation. We will briefly review some basic notions from circuit complexity here; for more background, consult a reference [Str+24; AB09].

Circuits. Circuits are directed computation graphs, where each node can represent either an input variable (if it is a leaf) or a function that should be applied to some previously computed values (if it has children). Typically, the values computed by a circuit are considered to be of boolean type, and the internal nodes represent logic functions. We can view a boolean circuit of this form with a single output node as defining a function $\{0,1\}^n \to \{0,1\}$ for some fixed n. Circuits can be naturally generalized to take tokens in a finite vocabulary Σ as input by imaging these tokens are encoded in binary. Thus, they can be taken to define functions $\Sigma^n \to \{0,1\}$.

Circuit families. While fixed-input-size circuits are an interesting object of study in their own right, it is often useful in complexity theory to analyze problems or languages where the input size is dynamic. To generalize circuits to accept variable-sized inputs, we introduce the notion of a *circuit family*: a family of circuits $C = \{C_n\}_{n=0}^{\infty}$. For any input $w \in \Sigma^*$, we process with $C_{|w|}$, where, in this context, $|w|$ represents the size of w encoded in binary. Thus, circuits families can be taken to define functions $\Sigma^* \to \{0,1\}$. In other words, they define formal languages.

Circuit complexity measures. The depth of a circuit is the length of the longest path from an input node to an output node. The size is the number of wires in the circuit. For circuit families, we can express the depth and width as a function of n. For example, "constant depth" means there is some c such that $\text{depth}(C_n) = O(1)$, and "polynomial size" means $\text{size}(C_n) = O(n^c)$ for some c. There are several standard classes of formal languages defined via bounds on circuit depth and width in this way:

1. NC^k is the class of bounded fan-in circuit families of constant depth and polynomial size. Bounded fan-in means each gate has at most two children, and the gate types allowed are AND, OR, and NOT.

2. AC^k is the class of *unbounded* fan-in circuit families of constant depth and polynomial size. It is the same as NC^k except gates can have unbounded fan-in.

3. TC^k is the class of unbounded fan-in *threshold* circuit families of constant depth and polynomial size. It is the same as AC^k but with MAJ gates that take a majority vote of a sequence of bits.

These circuit classes form the following clean relation with each other, for any k:

$$\mathsf{NC}^k \subseteq \mathsf{AC}^k \subseteq \mathsf{TC}^k \subseteq \mathsf{NC}^{k+1}.$$

We define NC as the union over all NC^k, which also naturally contains AC^k and TC^k for any k. The circuit classes that will be most relevant for our analysis of transformer expressivity are AC^0, TC^0, NC^1, and NC.

Uniformity. As defined, circuit families are a non-uniform computational model, meaning the computation has no finite description like we would have for a finite-state machine, Turing machine, or Python program. Non-uniformity is degenerate from the perspective of computability

theory, as it allows circuit families to solve some undecidable problems. For example, we can recognize the language $\{1^n \mid \text{Turing machine } n \text{ halts}\}$ by hard-coding each C_n to output whether machine n halts. To make circuit families more well-behaved, we will therefore variants of circuit families with uniformity conditions: essentially, for these families, there is a constraint that it must be possible to construct C_n from the input 1^n in some complexity class X. There are different versions of uniformity depending on the choice of X, but standard choices are log-space (L) uniformity and the stronger first-order (FO) uniformity[1] [MIS90]. The complexity classes AC^k, TC^k, NC^k all have uniform variants, which have the nice property that these classes form subsets of P (polynomial time). We will generally specify the notion of uniformity that we mean unless it is clear from context.

Hardness, Completeness, and Reductions. As in other parts of complexity theory, a useful way to reason about problems that are *likely* outside of some circuit class is in terms of hardness for a higher class. A X-hard problem is a problem L such that any any other problem in X can be "mapped" to L by some simple reduction, and an X-complete problem is both X-hard and in X. If the reductions are in the simpler class (say, TC^0), then it follows that hard problems for the higher class cannot be solved in the smaller class unless the larger and smaller classes collapse. This is how we can conclude, for example, that NC^1-hard problems are not in TC^0 unless $\mathsf{TC}^0 = \mathsf{NC}^1$, as long as the reductions come from a class in TC^0 (e.g., AC^0).

Computation graphs. We will use the term *computation graph* (or computation graph family) to refer to a generalized circuits whose nodes can represent potentially non-boolean functions in some set $\mathfrak{F}$ (note that these gates can be taken themselves to be specified by circuit families). The standard circuit complexity notions of size, depth, and uniformity apply analogously to computation graph families.

[1] For AC^0 and TC^0, FO uniformity is equivalent to DLOGTIME uniformity.

2.3 Logic

Logics over strings are another computational model closely related to uniform circuit families [Imm98]. In particular, first-order logic can be defined over strings as follows:

Definition 2.3 ([MS23a]). *First-order logic over strings (*FO*) defines a language by mapping every string to a boolean value. The logic contains two types: an index, which represents a pointer to a position in the string, and a formula, which evaluates to true or false.*

Indices in FO *are integers denoting positions in the input string:*

1. *The constant* 1*, representing the first token's position.*
2. *The constant n, representing the last token's position.*
3. *Symbols (e.g., i, j, k) representing variables ranging over positions* 1 *to n.*
4. *Any index built by applying addition or subtraction to other indices.*

Formulas in FO *are constructed as follows:*

1. *Let Σ be a finite alphabet. For each $\sigma \in \Sigma$ and any index i, $Q_\sigma(i)$ is a formula that is true if the i-th input token is σ.*
2. *For any indices i, j, the formula $\mathsf{bit}(i, j)$ returns the j-th bit of the binary expansion of i.*
3. *For two indices i, j, $i = j$, $i \leq j$, and $i \geq j$ are formulas with their conventional semantics.*
4. *For two formulas ϕ, ψ, $\phi \wedge \psi$ and $\phi \vee \psi$ are formulas with their conventional semantics.*
5. *For any formula ϕ (which may refer to i or any over variable), the following are formulas:*
   (a) *$\exists i[\phi]$ means some value of i in $[1, n]$ makes ϕ true.*
   (b) *$\forall i[\phi]$ means all values of i in $[1, n]$ make ϕ true.*

A formula ϕ with no free variables is called a sentence *and returns a value in $\{0,1\}$ for each input string. The language* defined *by ϕ is the set of strings mapped to 1.*

The languages definable in FO are exactly DLOGTIME-uniform AC^0 [MIS90]. Consider the following examples:

Example 2.4 (Bigram matching)**.** Strings containing the bigram ab: $\exists i\,[Q_{\mathsf{a}}(i) \wedge Q_{\mathsf{b}}(i+1)]$.

Example 2.5 (Skip-bigram matching)**.** Strings containing the long-distance pattern a . . . b (cf. "induction heads" [Ols+22]): $\exists i\,[Q_{\mathsf{b}}(i) \wedge \exists j\,[j \leq i \wedge Q_{\mathsf{a}}(j)]]$.

We next define first-order logic with majority quantifiers, which will prove useful for analyzing soft-attention transformers:

Definition 2.6 ([MS23a])**.** *First-order logic with majority, or* $\mathsf{FO}[\mathsf{M}]$*, is defined as* FO *augmented with a* majority quantifier $\mathsf{M}i[\phi]$*, which checks whether $\geq n/2$ values of i in $[1,n]$ make ϕ true.*

The languages definable in $\mathsf{FO}[\mathsf{M}]$ are exactly DLOGTIME-uniform TC^0. The simplest example of a language definable in $\mathsf{FO}[\mathsf{M}]$ but not FO is majority:

Example 2.7 (Majority)**.** Strings with more b's than a's: $\mathsf{M}i\,[Q_{\mathsf{b}}(i)]$.

Beyond this, $\mathsf{FO}[\mathsf{M}]$ is capable of defining more complex languages. $\mathsf{FO}[\mathsf{M}]$ can define threshold or counting quantifiers (e.g., $\exists^{\leq k} i[\phi]$) that check whether ϕ is true at least k times [MS23a]. This can be extended to allow a "count operator" that returns a numerical value, e.g., $\#i[\phi]$. This can be used to succintly give a definition for 1-Dyck:

Example 2.8 (1-Dyck, [MS23a; Hu+25])**.** 1-Dyck is the well-balanced parentheses language. First we define a $\mathsf{depth}(i)$ macro as follows:

$$\mathsf{depth}(i) \equiv \#j \leq i[Q_{(}(i)] - \#j \leq i[Q_{)}(i)].$$

We can define 1-Dyck as

$$\forall i[\mathsf{depth}(i) \leq 0] \wedge \mathsf{depth}(n) = 0.$$

$\mathsf{FO}[\mathsf{M}^2]$ [MIS90] is a variant of $\mathsf{FO}[\mathsf{M}]$ with equivalent expressive power. In $\mathsf{FO}[\mathsf{M}^2]$, the bit predicate is removed, but the majority predicate is replaced with a generalized "paired majority" that quantifies jointly over all pairs of positions i, j in the input string.

C-RASP [YC24; Hua+25b] is programming language or logic that is strictly weaker than $\mathsf{FO}[\mathsf{M}]$. C-RASP can be viewed as a restriction of $\mathsf{FO}[\mathsf{M}]$ without bit where predicates are restricted to be unary. In particular, when a quantifier introduces a new variable, the formula within it cannot depend on any other variable besides the new one [YC24]. The construction for 1-Dyck in Theorem 2.8 can be expressed in C-RASP, but C-RASP cannot express k-Dyck for $k > 1$, whereas $\mathsf{FO}[\mathsf{M}]$ can [Hua+25b; Hu+25].

2.4 Turing machines

We define multitape Turing machines in the standard way [MS24b; HMU01].

Definition 2.9. *A multitape Turing machine is a tuple* $\langle \Sigma, \Gamma, k, b, Q, q_0, \delta, F \rangle$ *where:*

1. Σ *is a finite input vocabulary*
2. Γ *is a finite tape vocabulary with* $\Sigma \subseteq \Gamma$
3. k *is the number of work tapes*
4. b *is a blank symbol such that* $b \in \Gamma$ *and* $b \notin \Sigma$
5. Q *is a finite set of states containing initial state* q_0
6. δ *is a transition function* $(Q \setminus F) \times \Gamma^{k+2} \to Q \times \Gamma^{k+1} \times \{\pm 1\}^{k+2}$
7. $F \subseteq Q$ *is a set of halting states*

A Turing machine takes as input a string $\sigma \in \Sigma^*$. A *configuration* of a Turing machine is a finite state q along with the contents of an *input* tape c^0, k work tapes $c^1, \ldots, c^k$, and an output tape c^{k+1}. Finally, for each tape τ, a configuration specifies a head position h^τ. We start with the initial state q_0 and the input tape c_0^0 containing σ starting at position 0 with infinite b's on each side, and $h_0^0 = 0$. All other tapes start containing all b's and with their head at 0. At each time step i, if $q_i \notin F$, we recurrently update the configuration by first computing:

$$\langle q_{i+1}, \gamma_i^1, \ldots, \gamma_i^{k+1}, d_i^0, \ldots, d_i^{k+1} \rangle = \delta(q_i, c_i^0[h_i^0], \ldots, c_i^{k+1}[h_i^{k+1}]).$$

We then update tape τ by setting $c_{i+1}^\tau[h_i^j] = \gamma_i^j$ and keeping all other tape cells the same. We update the head position on tape τ according to $h_{i+1}^\tau = h_i^\tau + d_i^\tau$. On the other hand, if $q_i \in F$, the Turing machine halts and *outputs* the string of tokens on the output tape from the current head position on the left up to (but not including) the first b on the right. A Turing machine can also be viewed as a language recognizer simply by saying that reaching a halting state constitutes accepting (there are other equivalent ways to define language acceptance as well).

Turing machine complexity classes. Let $\mathsf{TIME}[T]$ be the class of languages recognizable using at most $O(T(n))$ steps on a multitape Turing machine. Similarly, let $\widetilde{\mathsf{TIME}}[T]$ be the same class but with $\tilde{O}(T(n))$ time (i.e., potentially with polylogarithmic overhead). Let $\mathsf{SPACE}[S]$ be the class of languages recognizable using at most $O(S)$ space, i.e., where at most $O(S)$ tape cells are non-empty at any point in the computation.

3 Expressivity theory for transformers

Transformers and related neural architectures are the substrate in which modern language models are built. While there is certainly more to language models than their architecture (crucially, the data they are trained on and the learning algorithm), analysis of the expressive power of the architecture on its own has proven a useful way to pin down the computational power and limitations of language models.

This chapter synthesizes several years of work [MSS22; MS23a; MS23b] establishing upper bounds on the expressive power of soft-attention transformers into a minimal, general form. Chronologically, Merrill, Sabharwal, and Smith [MSS22] analyzed AHATs and established non-uniform TC^0 as an upper bound on their expressive power. Improving upon this work, Merrill and Sabharwal [MS23b] generalized the analysis to soft-attention transformers with log precision and showed a tighter upper bound of L-uniform TC^0, which is substantially more interesting for reasoning about the limitations of transformers. Finally, Merrill and Sabharwal [MS23a] tightened the upper bound for log-precision transformers to FO-uniform TC^0 (the result presented here is close to this, but more general).

After first proving a general version of the result that transformers are in TC^0 (Theorem 3.1), I discuss how this result reveals computational problems that transformers likely cannot express (Section 3.4) and consider its implications for interpreting the computation inside transformer

LMs (Section 3.5). Later chapters will build on the theoretical foundation in this chapter by considering how different approaches and architectures extend LM expressivity relative to transformers (Chapter 4) and discussing how the theoretical limitations represented by Theorem 3.1 might, in practice, impact transformers' ability to process language and solve reasoning problems.

3.1 Why universality results are deceiving

Undergraduate deep learning courses often allude to deep learning architectures as universal function approximators, due to foundational results in the theory of neural networks. For example, Cybenko [Cyb89] showed that one-layer neural networks with sigmoidal activations can arbitrarily approximate continuous functions over compact domains, a result that has been refined in follow-up work [JTX20]. Moving from compact domains to sequence models, Siegelmann and Sontag [SS95] showed that basic recurrent neural networks (RNNs)—the predecessors to transformers—can be Turing-complete, assuming unbounded computation time and precision. In light of these results, why should we care about understanding the expressive power of neural architectures like the transformer, if even simpler architectures are already "universal" in some sense?

The answer lies with the fact that these results rely on assumptions that are unrealistic for sequence modeling in order to establish universality. For example, Cybenko's universal approximation result applies only to continuous functions over compact domains, not sequences [Cyb89].[1] Moreover, the hidden size required by the network could be "astronomical" [Cyb89]. As already mentioned, Siegelmann and Sontag [SS95] require the precision of the RNN to grow *unboundedly* to simulate a Turing machine; in contrast, when LM inference is carried out over long sequences, the precision remains bounded and small. It turns out that unrealistic assumptions of this form

[1] When embedding sequences into a compact vector space, some sequence vectors must get arbitrarily close as the sequence length increases: for any δ, we can find two sequences with distance at most δ. Thus, if these sequences are labeled in $\{0, 1\}$, this function need not be continuous.

are required to establish computational universality: without such assumptions, deep learning architectures cannot express arbitrary discrete functions over sequences. As our results will later show, as long as their size and precision remain polynomial in the input sequence length, transformers have clear computational limitations. Moreover, by drawing connections to circuit complexity, we will be able to develop a finegrained theory of the kinds of computation transformers are capable of and the kinds of computational problems they (likely) cannot solve.

3.2 Prior work: hard-attention transformers

In contrast to universality results, another line of work on the expressive power of neural networks studied transformers with *unique hard attention*, which we refer to as *hard-attention transformers* for short. Hard attention means that each attention head in the transformer (Section 2.1) can only focus on one (i.e., the maximal) position.[2] It was shown that hard-attention transformers are quite limited in power: they cannot express the simple problems of computing the parity or majority vote of a bit string [Hah20], and, more generally, can only recognize languages in the circuit complexity class AC^0 [HAF22]. These results have since been refined to yield exact characterizations of the languages recognizable by different variants of hard-attention transformers in terms of string logics related to linear temporal logic [YCA24; Jer+25]. Overall, these results reveal that, in contrast to universal approximation claims (cf. Section 3.1), hard-attention transformers are not computationally universal, and, in fact, cannot express many very simple computational problems.

However, these strong limitations for hard-attention transformers do not translate to more realistic transformers with soft attention. While hard-attention transformers cannot compute majority, a soft-attention transformer can compute majority with a single uniform attention head [PMB19, Proposition 3.3]. The head's value vector checks whether the input token is a 1 or 0,

[2]There are some subtleties for breaking ties if there are multiple maxima, but under various reasonable tie-breaking schemes, the general characterization within AC^0 applies [HAF22].

which means that it outputs the proportion of 1's in the input, which can be thresholded to solve majority.[3] This shows that transformers that go beyond hard attention will have express power beyond AC^0, motivating a the general theoretical treatment of soft-attention transformers developed in this thesis.

3.3 Soft-attention transformers and threshold circuits

A main contribution of this thesis is a theoretical characterization of the expressive power of soft-attention transformers in terms of the circuit complexity class TC^0. As we will see in Section 3.4, this result has many implications for the computational power and limitations of transformers. We will first state and prove the main theoretical result for transformers before returing to a discussion of its implications. The result presented here applies to very general model of transformers (cf. Definition 2.2): it can use softmax or averaging hard attention, multiple layers (though the number must be fixed w.r.t. n), and precision at most polynomial in n.

Theorem 3.1 ([MSS22; MS23b; MS23a])**.** *Let T be a transformer. Then the function computed by T can also be computed by an* FO*-uniform* TC^0 *circuit family.*

Proof overview. The high-level idea is to show that each component of the transformer computation graph (embedding layer, self-attention sublayer, and feedforward sublayer, and unembedding matrix) can be computed in uniform TC^0. Then, we can use the fact that uniform TC^0 is closed under composition [MS23a, Corollary 3.1] to conclude the full transformer is in uniform TC^0. □

We break the details of the proof into the following lemmas. The first step is to justify that, if all components of the transformer are in TC^0, then their composition will also be in TC^0. This is

[3] More than fixed precision (e.g., $p = c \log n$ in Definition 2.1) is crucial here because, otherwise, uniform attention weights are not representable. In fact, the expressive power of soft-attention transformers with fixed precision reduces to that of hard-attention transformers [LC25] because fixed-precision soft attention can only attend to a bounded number of positions [MS23a, Proposition 1].

straightforward for non-uniform [MSS22] or L-uniform [MS23b] TC^0 but requires the following technical lemma for stronger FO-uniform TC^0 [MS23a], a proof of which we omit here.

Lemma 3.2 (Composition, [MS23a], Corollary 3.2)**.** *Let $\mathcal{G}$ be an* FO*-uniform, polynomial-size, constant-depth computation graph family over a finite set of node types $\mathfrak{F}$, where each $\mathcal{F} \in \mathfrak{F}$ is specified by an* FO*-uniform* TC^0 *family. W.l.o.g., for each $\mathcal{F}$, let* $\mathsf{size}_{\mathcal{F}}(n)$ *be a power of* 2 *computable in $O(\log n)$ time. Then $\mathcal{G}$ can be simulated by an* FO*-uniform* TC^0 *family C.*

In line with this lemma, we can observe that the transformer is a uniform constant-depth computation graph family $\mathcal{G}$ where $\mathfrak{F}$ contains operation for the standard modules of the embedding layer, layer-normalization component, self-attention sublayer, feedforward sublayer, and output projection [MS23a, Lemma 1]. We can thus complete the proof by showing that each of these components is in FO-uniform TC^0. First, we justify that the embedding layer can be computed in FO-uniform TC^0:

Lemma 3.3 (Embedding)**.** *The transformer embedding layer can be computed in* FO*-uniform* TC^0 *as a function of some input string $w_1 \dots w_n \in \Sigma^*$.*

Proof. We justify that (each bit of) the embedding $\mathbf{E}[w_i] + \pi(i)$ for w_i can be defined in $\mathsf{FO}[\mathsf{M}]$, which is equivalent in expressive power to FO-uniform TC^0 [MIS90]. To define the jth bit of the token embedding in $\mathsf{FO}[\mathsf{M}]$, we construct a clause for each $\sigma \in \Sigma$ of the form $\neg Q_\sigma(i) \wedge \mathbf{E}[\sigma]_j$. Thus, this returns $\mathbf{E}[w_i]_j$. To compute the jth bit of the position embedding $\pi(i)_j$, we use the fact that $\pi(i)$ can be approximated to polynomial precision in FO-uniform TC^0 since it is an analytic function with a convergent rational power series [RT92]. Thus, we can compute it over numbers with precision at most $\mathrm{poly}(n)$ in TC^0. Finally, we can implement addition in FO (and thus also FO-uniform TC^0), so we conclude that the embedding of w_i can be computed in TC^0. □

We next analyze the layer-normalization component, which is used at the beginning of each sublayer in a pre-norm transformer (Definition 2.2). We will show that this module can be computed in uniform TC^0.

Lemma 3.4 (Layer normalization)**.** *Layer normalization can be computed in* FO*-uniform* TC^0 *as a function of the previous hidden state* $\mathbf{h}_i$*.*

Proof. Layer normalization involves the arithmetic operations of addition, multiplication, division, and square root. Binary (i.e., between two numbers) addition and multiplication are trivial to implement in uniform TC^0. Exact division was shown to be in uniform TC^0 in a landmark result [Hes01], and square root can be approximated to polynomial precision in uniform TC^0 [RT92; Chi24]. Thus, we can conclude that layer-normalization over polynomial-precision rationals can be computed in uniform TC^0. □

Next, we analyze the softmax-attention sublayer, which is the crucial component of the transformer architecture in the sense that it is the only part that mixes information across tokens. Importantly, the way that the attention mechanism mixes information from tokens can be simulated in parallel in TC^0, a fact that is not true for other architectures like RNNs.

Lemma 3.5 (Self-attention)**.** *Any multihead self-attention sublayer can be computed by an* FO*-uniform* TC^0 *circuit family at a function of the normalized hidden states* $\hat{\mathbf{h}}_1, \ldots, \hat{\mathbf{h}}_i$*.*

Proof. We will first simulate a single head $\mathbf{a}_{i,k}$ (cf. Definition 2.2, omitting layer superscripts):

$$\mathbf{a}_{i,k} = \sum_{j=1}^{m} \frac{\exp(1/\tau \cdot \mathbf{q}_{i,k}^\top \mathbf{k}_{j,k})}{Z_{i,k}} \cdot \mathbf{v}_{j,k}.$$

First, it is straightforward that we can compute each query-key inner product in TC^0, and we can also divide by τ [Hes01]. We can then approximate exp to polynomial precision in uniform TC^0 [RT92; Chi24]. So, for each i, j we can compute $s_{i,j,k} = \exp(1/\tau \cdot \mathbf{q}_{i,k}^\top \mathbf{k}_{j,k})$ over polynomial-precision rationals in uniform TC^0. Next, we can each $Z_{i,k}$ as a sum of these terms (w.r.t. j) because iterated addition is in FO-uniform TC^0 [Imm98]. For each i, j, we then compute $\frac{s_{i,j,k}}{Z_{i,k}} \cdot \mathbf{v}_{j,k}$ in uniform TC^0, again using the fact that we can compute division [Hes01]. We then use iterated addition to compute a sum over j over these terms, yielding $\mathbf{a}_{i,k}$. Finally, having computed each attention

head, we can easily compute the output projection matrix and residual connection in uniform TC^0. We thus conclude that the whole self-attention sublayer can be simulated in FO-uniform TC^0. □

Finally, it is straightforward that the feedforward sublayer can also be computed in TC^0, since they just involve binary addition, binary multiplication, and nonlinearities:

Lemma 3.6 (Feedforward)**.** *Any feedforward sublayer or output projection matrix can be computed by an* FO*-uniform* TC^0 *circuit family at a function of the normalized hidden state* $\hat{\mathbf{h}}_i$.

In fact, with ReLU nonlinearities, the stronger result that feedforward subnetworks can be simulated in FO-uniform AC^0 holds, though TC^0 may be required to simulate more complex activation functions. Either way, we can put together all these results to conclude by Theorem 3.2 that the entire transformer $\mathcal{T}$ can be simulated in TC^0.

Theorem 3.1 has many interesting implications. First, we will consider how, under standard complexity conjectures, it implies that transformers cannot express many inherently sequential problems (Section 3.4). Then, we will discuss the implication that the logic $\mathsf{FO}[\mathsf{M}]$ can be viewed as a "programming language" provably capable of representing any behavior that a transformer can express (Section 3.5), which may be of interest for interpretability research.

3.4 A tour of problems transformers likely cannot express

Combining Theorem 3.1 with different results in complexity theory allows us to identify many computational problems that, under standard complexity conjectures, fixed-depth transformers cannot express, for large enough input lengths. I will now highlight some canonical examples of such problems that are related to language understanding and reasoning, summarized in Figure 3.1. This list is not exhaustive: work on transformer expressivity may provide more discussion of other problems [MS23b; Str+24]. There are also many more problems related to specific classes

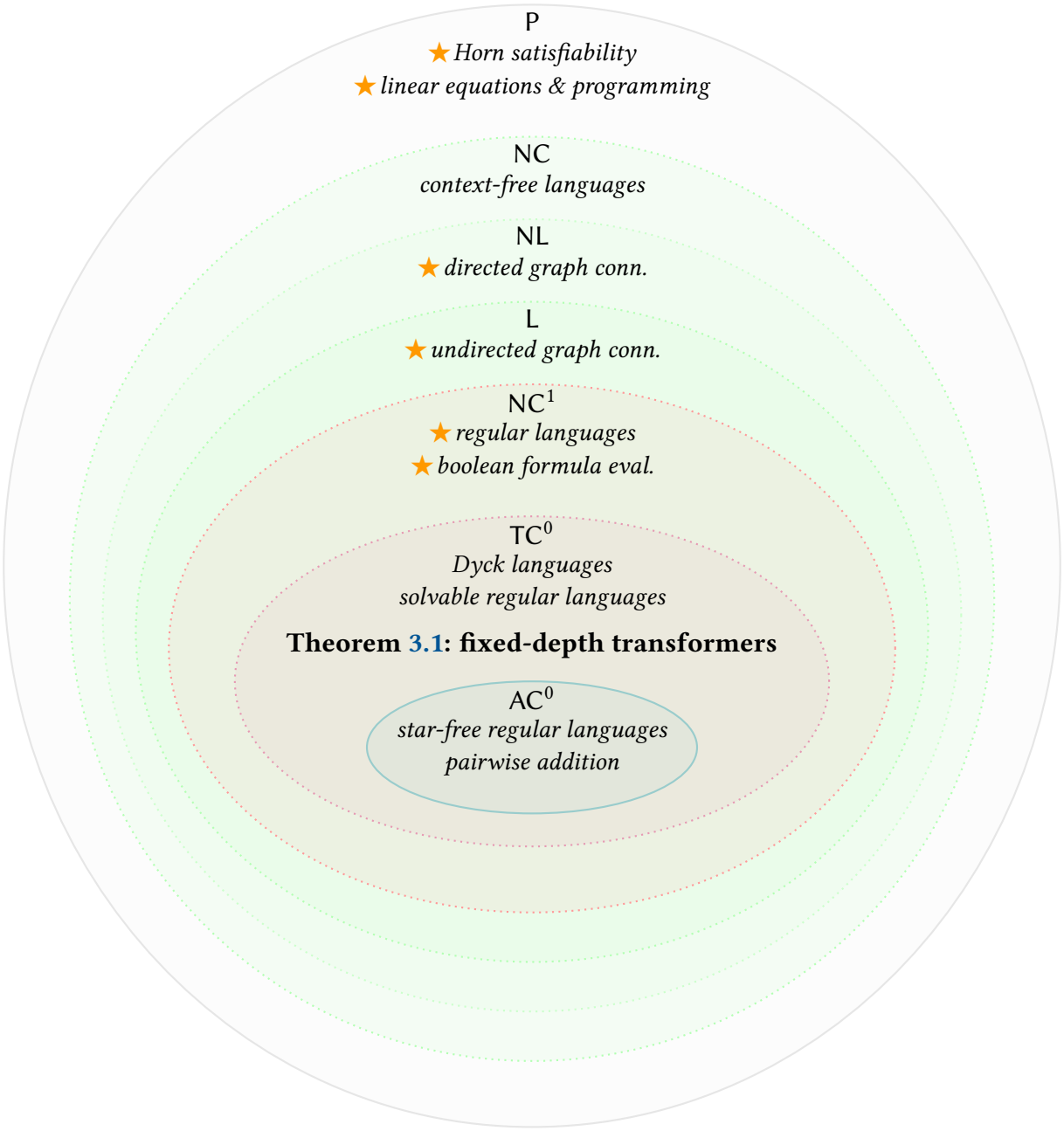

Figure 3.1: Hierarchy of circuit complexity classes within P, including TC^0. Dotted borders represent conjectured separations between classes, and ★ indicates problems complete under many-one reductions. By Theorem 3.1, complete problems in classes above TC^0 cannot be expressed by transformers unless their class collapses to TC^0, which allows us to identify problems transformers likely cannot express.

in the broader complexity literature, and it is often simple to demonstrate the inexpressibility of a new, specific problem to a more general problem by constructing a reduction [cf. MPS24, Section 3.2]. Later on in Chapter 5, I will more speculatively discuss the implications of these formal hardness claims for transformers' ability to implement aspects of language understanding and reasoning that are not as well defined as these model problems.

3.4.1 Regular languages, finite monoids, and state tracking

Regular languages are a fundamental class of formal languages that have been studied from a variety of perspectives and with different motivations. Originally motivated as a computational model of biological neural networks [MP88], the regular languages are also a model of some of the simple syntactic patterns in natural language [Cho56], and they can also be naturally used to formalize AI concepts like state tracking and world modeling [Liu+23b; MPS24; Vaf+24]. This last perspective on regular languages, in particular, motivates analyzing the degree to which LM architectures can express regular language recognition.

Fortunately, the complexity of recognizing regular language with circuits has been extensively studied [KMR67; IL95], so we can use Theorem 3.1 to understand the capabilities of transformers to recognize different kinds of regular languages. In general, the circuit complexity of regular language recognition can be related to the algebraic structure of its transition monoid, which controls the degree to which the automaton the regular language represents can be parallelized. If the transition monoid satisfies an algebraic condition known as *solvability* (which can be thought of as generalizing commutativity), recognition is in TC^0. On the other hand, if the transition monoid is not solvable, recognition is NC^1-complete, and thus by Theorem 3.1 such languages cannot be recognized by fixed-depth transformers. One canonical example of a non-solvable regular language is the word problem for the permutation group S_5, which captures the task of permuting permutation over 5 elements. Since this language is non-solvable, Theorem 3.1 implies this language, which is closely related to state tracking [KS23; MPS24]. This suggests

a fundamental limitation of fixed-depth transformers for state tracking [Liu+23b; MPS24]: see Section 5.4 for further discussion.

3.4.2 Recognizing context-free languages

Context-free languages, which generalize regular languages, have been traditionally argued to be a good formal model of the hierarchical structure in natural languages [Cho56]—moreover, they are explicitly used to capture such hierarchical structure in programming languages and other areas. Already from our regular languages result, we knew that transformers could not recognize context-free languages unless $\mathsf{TC}^0 = \mathsf{NC}^1$, and that log-depth should be required to recognize some context-free languages. But context-free recognition is likely harder than this: from a circuit complexity perspective, the tightest upper bound for context-free recognition is $\mathsf{SAC}^1 \subseteq \mathsf{AC}^1$ [Ruz81; Ven91]. This means context-free free recognition could be harder than regular language recognition for transformers, with regular language recognition already being inexpressible by transformers of context depth (assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$).

On the other hand, while not all context-free languages can be recognized by transformers, there are some interesting subclasses for which recognition is in TC^0 and expressible by fixed-depth transformers. Dyck languages, often taken as a canonical example of hierarchical structure due to the Chomsky-Schützenberger Theorem [CS59] are such an example: k-Dyck falls into TC^0 for any k (in fact, it is TC^0-complete), and can moreover be expressed by some, slightly idealized model of transformers with constant depth [WGY21; Yao+21]. Thus, while Dyck languages are often taken as canonical context-free languages, they are not necessarily the hardest for circuits or transformers—indeed, there are canonical context-free languages that are formally harder [Gre73], which intuitively are distinguished by being more ambiguous. In Section 5.3.1, we will return to the potential implications of this circuit complexity viewpoint for the ability of transformers to represent natural-language syntax.

3.4.3 Evaluating compositional formulas and circuits

A natural compositional problem studied in circuit complexity is formula evaluation: given a hierarchically structured boolean formula as input, the goal is to return the truth value of the formula. This problem is NC^1-complete [Bus87], and thus, by Theorem 3.1, cannot be expressed by constant-depth transformers unless $\mathsf{TC}^0 = \mathsf{NC}^1$. It is an open question whether log-depth transformers can evaluate boolean formulas. This result generalizes to formulas over any finite domains, and also implies formulas over infinite domains cannot be expressed—however, formulas over infinite domains may be even harder than NC^1-complete.

A simple extension of formula evaluation is circuit evaluation: here, the input is a boolean string x of length n along with a circuit C that takes n inputs. The goal is to return $C(x)$, i.e., the circuit C evaluated on x. While superficially similar to boolean formula evaluation, circuit evaluation is actually significantly harder: it is P-complete, and thus, by Theorem 3.1, it cannot be expressed by constant-depth transformers (unless $\mathsf{TC}^0 = \mathsf{P}$) or even polylogarithmic depth transformers (unless $\mathsf{NC} = \mathsf{P}$). Thus suggests that circuit evaluation is not feasible for transformers without chain of thought in practice. However, if the depth of the transformer is allowed to grow *linearly* with the circuit depth (in the worst case, polynomial in n), then [MS23b] show transformers can solve circuit evaluation. In other words, transformers of fixed depth can solve circuit evaluation if the circuit depth is guaranteed to be fixed as well.

Beyond evaluating formulas and circuits, an interesting problem related to code reasoning is model checking: given some code and a property about it expressed in logic, verify whether that property holds. In future work, it would be interesting to formally explore the difficulty of different types of model checking for transformers.

3.4.4 Reasoning over graphs

Many types of reasoning, including pathfinding and logical inference, can be formalized as connectivity questions over graphs. To capture the structure of these problems, complexity theorists have defined the graph connectivity (or reachability) problem as follows: a graph as well as two nodes s, t are given as input, and the goal is to determine whether there is a path in the graph from s to t. For undirected graphs, graph connectivity is L-complete under NC^1 reductions [CM87; Rei08; Str+24], and, for directed graphs, it is NL-complete under FO reductions [Imm98, Theorem 3.16]. Thus, by Theorem 3.1, constant-depth transformers cannot solve general graph connectivity questions unless $\mathsf{TC}^0 = \mathsf{NL}$, although an interesting question remains about what graph-theoretic properties might make this problem more tractable.

3.4.5 Other problems

Even without Theorem 3.1, NP-complete problems cannot be expressed by transformers unless $\mathsf{P} = \mathsf{NP}$, as transformers can be run in polynomial time. This means, for instance, that satisfiability of arbitrary boolean formulas cannot be determined by a transformer. Even the restricted variant of Horn satisfiability, which is efficient to solve with a Turing machine, is P-complete, which means it cannot be expressed by transformer LMs unless $\mathsf{TC}^0 = \mathsf{P}$. Many other problems across domains are known to be P-complete, including linear programming, universal context-free recognition (given grammar G and string w, does $G \longrightarrow w$?), and various questions related to reward computation and reinforcement learning [GHR91]. In general, unless $\mathsf{TC}^0 = \mathsf{P}$, Theorem 3.1 shows none of these inherently sequential problems can be solved by transformers without CoT, while, with enough CoT steps, we will see that they become expressible (Theorem 4.2).

3.5 A logical view on transformer expressivity

The main result in Theorem 3.1 characterizes the computational power of transformers in terms of circuits. Through the research program of descriptive complexity theory [Imm98], close connections have been revealed between circuit complexity and logic, which we can leverage to characterize transformers in terms of logic. In particular, we will be able to relate transformers to a variant of first-order logic over strings augmented with majority quantifiers, which we call $\mathsf{FO}[\mathsf{M}]$ (cf. Section 2.3). Classically, the set of languages definable in $\mathsf{FO}[\mathsf{M}]$ is exactly FO-uniform TC^0 [MIS90]. It thus follows from Theorem 3.1 that any transformer can be translated to a logical "program" in $\mathsf{FO}[\mathsf{M}]$ [MS23a]:

Corollary 3.7. *For any transformer $\mathcal{T} : \Sigma^* \to \Sigma$ and token $\sigma \in \Sigma$, there exists a sentence in $\mathsf{FO}[\mathsf{M}]$ that defines the language $\{\mathcal{T}(w) = \sigma : w \in \Sigma^*\}$.*

In this sense, $\mathsf{FO}[\mathsf{M}]$ can be understood as a "programming language" provably capable of expressing the behavior of any transformer, whether it uses unique hard attention, averaging hard attention, or softmax attention [MS23a]. As mentioned in Section 3.2, FO (without majority) is capable for expressing hard-attention transformers. In the other direction, it is also possible to develop logics that can be compiled *into* transformers. Specifically, Yang and Chiang [YC24] show that any language expressible in C-RASP, a restriction of $\mathsf{FO}[\mathsf{M}]$, can be expressed by soft-attention transformers. C-RASP restricts $\mathsf{FO}[\mathsf{M}]$ so that predicates can only be unary (cf. Section 2.3). Thus, the binary predicates $\mathrm{bit}(i, j)$ and $i < j$ are removed. Moreover, quantifiers are restricted to have either the form $\exists i < n[\phi(i)]$ or $\exists j < i[\phi(j)]$—crucially, ϕ is a unary predicate in either case. Intuitively, restricting all predicates to be unary ensures that the transformer has enough memory to store all variable configurations, which ensures that the transformer can successfully resolve a C-RASP formula. Understanding and potentially closing the gap between $\mathsf{FO}[\mathsf{M}]$ and C-RASP is an interesting open question for future research: can we potentially arrive

at a single logical formalism that exactly characterizes the expressive power of transformers? Can different logical formalisms capture finegrained differences in transformer expressivity like the relative power of softmax and averaging hard attention [Yan+24a]?

In addition to providing a different, potentially more interpretable perspective on transformer expressivity, viewing transformers in terms of logic also appears to be a promising way to understand the interplay betweeen expressivity and learnability. It has been hypothesized transformers learn to length-generalize on a language if and only if it is definable in C-RASP [Hua+25b]. Related to this, [Hu+25] "pre-pretrain" transformer LMs on various formal languages definable in C-RASP and $\mathsf{FO}[M]$ before pretraining on natural language, finding that several formal languages confer a useful inductive bias for learning more efficiently from natural language data. They hypothesize that effective pre-pretraining languages should both be representable by transformers and encode hierarchical structure similar to the syntax of natural language. These works provide case studies in how the logical view of transformer expressivity can be a useful bridge from the expressivity viewpoint to reasoning about the inductive biases and generalization behavior of transformers.

4 Extending language models' expressivity

In Chapter 3, we saw how transformers are limited to expressing on highly parallelizable computation: formally, languages in the class TC^0 (Theorem 3.1). It is important to understand the conditions in which Theorem 3.1 applies: it says that, given a fixed-depth transformer, if it reads some input and must immediately produce an output, it can only solve problems in TC^0. This is certainly one way that transformers are used, but, sometimes, transformers can be used in other, more complicated ways in practice. For example, one can allow the transformer to generate a scratchpad or chain of thought before its final answer, or potentially increase the depth of the model with the length of the problem input. Another, more extreme, option, is to use a different deep learning architecture in an LLM than a transformer. Here, we analyze how all these generalizations change transformers' expressive power, with an eyes towards guiding the design of LM architectures and algorithms with greater expressivity than today's transformers.

4.1 Transformers with chain of thought

As we have seen in Chapter 4, transformers are constrained to performing highly parallelizable computation. One minimal way to extend them, then, is to make them more *sequential* by allowing them to iteratively generate and read tokens before eventually producing a final answer.

This approach of extending transformers with autoregressive generation is often called chain of thought (CoT; [Wei+22a]) or scratchpad [Nye+22], though we will refer to it as CoT for consistency. Originally, chain of thought involved training LLMs to emulate reasoning traces before producing an answer, and, then, at inference time, using the models to generate reasoning traces. Later reasoning models like GPT o1 and DeepSeek R1 leverage a CoT similarly at inference time, except that the model is trained in a different way, relying on reinforcement learning rather than explicit next-token supervision. In either case, the central role that chain-of-thought plays for improving LLM reasoning performance motivates analyzing its impact on the computational power of models. Moreover, at the level of intuition, CoT adds to transformers exactly what TC^0 lacks: whereas TC^0 computation has constant depth (Theorem 3.1), the computation graph of a transformer with CoT *deepens* for longer sequences because the output from each token flows into the input of the next token. Indeed, we show that this deepening of the computation graph allows transformers with CoT to solve inherently sequential problems conjectured to be outside TC^0.

4.1.1 Defining CoT

Abstracting away from the impact of a CoT on learning (we will return to this later), we will define the computational model of a transformer (or other LLM) augmented with a CoT. Given any generative model $f : \Sigma^* \to \Sigma$, we can map an input $x \in \Sigma^*$ to output $y \in \Sigma$ using $t(n)$ steps of chain of thought according to the recurrent process:

$$z_0 = x, \qquad z_{i+1} = z_i \cdot f(z_i), \qquad y = f(z_{t(|x|)}).$$

Definition 4.1. *We let* $\mathsf{CoT}[T]$ *be the class of languages recognizable by an AHAT with masked pre-norm (Definition 2.2) using* $O(t(n))$ *CoT steps, assuming a beginning-of-sequence token.*

Under this definition, the amount of CoT is tied to the length of the input, such that longer inputs can allow more CoT steps. Some natural instantiations for $t(n)$ could be logarithmic ($\log n$),

linear (n), or polynomial (n^c)—transformers without CoT can be recovered by setting $t(n) = 0$. We will be able to characterize and compare the power of CoT in each of these cases, relating CoT steps as a resource to standard TIME and SPACE complexity classes (cf. Section 2.4).

4.1.2 CoT extends transformer expressivity

Our theoretical study of CoT [MS24b] reveals that it expands the computational power of transformers to contain problems in higher complexity classes likely above TC^0. From a more finegrained perspective, we understand CoT steps as a computational resource akin to time or space for Turing machines. Existing work analyzing encoder-decoder transformers showed that slightly idealized transformers could use decoder steps to simulate a Turing machine [PMB19]. Related to this, we show that a transformer can use t CoT steps to simulate t steps of runtime on a Turing machine:

Theorem 4.2 ([MS24b], Theorem 2)**.** $\mathsf{TIME}[t] \subseteq \mathsf{CoT}[t]$.

Proof sketch. We construct a transformer decoder that uses a single decoding step to simulate each Turing machine step using $O(\log t)$ precision. The main difficulty is representing a Turing machine tape in a sequence of transformer state vectors so that the transformer can always correctly reconstruct the value on the tape at the current head position. The key idea is to write the previous Turing machine "diff" δ_{i-1} (i.e., the output of the transition function) at each step i. In the first layer, the transformer will reconstruct h_i^τ, the current head position on tape τ, via a uniform-attention counter [BAG20]. Then, we can use the head position to retrieve the last diff written at this head position using an induction head-like construction [Ols+22], where we attend to j such that $h_i^\tau = h_{j-1}^\tau$ and retrieve δ_j. Implementing this retrieval, similar to pointer matching, relies heavily on the layer-norm hash gadget [MS24b, Section 3.1]. Overall, by combining uniform attention with hard attention, this construction allows us to simulate a Turing machine for t steps in a distributed way using t steps of CoT. □

Theorem 4.2 establishes that CoT steps allow transformers to implement sequential computation that a Turing machine can express with enough CoT steps. Moreover, transformers with linear or more CoT can recognize regular languages, which is NC^1-complete:

Corollary 4.3 (cf. [MS24b], Theorem 1)**.** $\mathsf{CoT}[n]$ *can recognize any regular language, which is* NC^1*-complete under* FO *reductions [IL95].*

Assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, regular language recognition is outside TC^0, and thus linear CoT extends the expressive power of transformers. With more than linear CoT, more complex problems become solvable:

Corollary 4.4. $\mathsf{CoT}[n^2]$ *contains directed graph connectivity, which is* NL*-complete under* FO *reductions [Imm98].*

Classically, graph connectivity can be solved in linear time via search [Wig92]. However, implementing this linear-time algorithm on a multitape Turing machine (i.e., without random access) potentially requires quadratic overhead in the runtime [MS24b], which is how we arrive at the quadratic time characterization for graph connectivity. Thus, we obtain the following general takeaway:

Corollary 4.5 (Informal)**.** *If some problem has a standard (i.e., random-access) algorithm that runs in time at most* $O(t)$*, a transformer can implement this algorithm with at most* $O(t^2)$ *CoT steps.*

4.1.3 Limitations of transformers with CoT

We have seen that CoT extends the expressive power of transformers outside of TC^0, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$. But how far does this go? Can we give an upper bound for the class of languages recognizable by transformers with t steps of CoT? One natural upper bound comes from the fact that transformers can be run on a multitape Turing machine in quadratic time (with a polylogarithmic factor of overhead for floating point arithmetic operations). This yields the following relationship:

Theorem 4.6 ([MS24b], Theorem 3)**.** $\mathsf{CoT}[t] \subseteq \widetilde{\mathsf{TIME}}[t^2 + n^2]$.

While simple, Theorem 4.6 is useful for establishing limits on the expressive power of transformers with linear or more CoT steps, showing that t CoT steps are worth at most roughly t^2 TM steps. If we allow polynomial CoT steps, this is just a polynomial overhead in the number of Turing machine steps, so, combined with Theorem 4.2, we obtain:

Corollary 4.7. *Polynomial CoT (i.e.,* $\bigcup_{c=1}^{\infty} \mathsf{CoT}[n^c]$*) is exactly* P.

To get a more refined understanding of the limitations of transformers with CoT, we also consider another, more involved way to characterize an upper bound class for $\mathsf{CoT}[t]$ in terms of Turing machine *space*:

Theorem 4.8 ([MS24b], Theorem 4)**.** $\mathsf{CoT}[t] \subseteq \mathsf{SPACE}[t + \log n]$.

Proof. Since log-precision transformers can be simulated in uniform TC^0 [MS23b], they can be simulated in L, i.e., with at most $c \log n$ space overhead on inputs of size n. To compute $t(n)$ intermediate decoding steps of a transformer, we store a buffer of at most $t(n)$ generated tokens, which has size $O(t(n))$. To compute the next token, we call the transformer with an input of size $O(n + t(n))$ using at most $c \log(n + t(n))$ space overhead. We then clear the memory used and append the finite token generated to the input buffer. It follows from this algorithm that

$$\begin{aligned} \mathsf{CoT}[t] &\subseteq \mathsf{SPACE}[t + c \log(n + t)] \\ &\subseteq \mathsf{SPACE}[t + \log n]. \end{aligned}$$

□

Theorem 4.8 yields several useful results beyond Theorem 4.6. First, following from the equivalence of context-sensitive languages with linear space [Kur64], we obtain:

Corollary 4.9. $\mathsf{CoT}[n]$ *is contained within the context-sensitive languages.*

Thus, linear CoT likely extends transformers' expressive power outside TC^0 (Theorem 4.3), but at most allows transformers to recognize context-sensitive languages.

Beyond this, Theorem 4.8 allows us to understand the expressivity of transformers with *sub-linear* CoT:

Corollary 4.10. $\mathsf{CoT}[\log n] \subseteq \mathsf{L}$.

In fact, this result can be improved upon to show that $\mathsf{CoT}[\log n] \subseteq \mathsf{TC}^0$ [Li+24b; MS24a]. Thus, while linear CoT extends expressivity outside TC^0 (assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$), logarithmic CoT does not. This suggests that close to linear CoT may be required to get an expressivity benefit from CoT, although it is an open question whether some intermediate function between $\log n$ and n may yield expressivity beyond TC^0.

4.1.4 Discussion: implications of CoT expressivity results

In contrast to older work augmenting recurrent networks with differentiable data structures to make them Turing-complete [GWD14], transformers can simulate Turing machines simply with the addition of CoT. Putting all of our results related transformers and Turing machines together yields the following:

$$\mathsf{TIME}[t] \subseteq \mathsf{CoT}[t] \subseteq \begin{cases} \widetilde{\mathsf{TIME}}[t^2 + n^2] \\ \mathsf{SPACE}[t + \log n]. \end{cases}$$

This is a fairly general characterization: t CoT steps roughly buys between t and t^2 steps on a multitape Turing machine. We can leverage this to understand the power of transformers with different CoT budgets:

1. With logarithmic CoT, transformers can recognize at most L (Theorem 4.10). In fact, it turns out they cannot recognize languages outside TC^0, suggesting they obey similar limitations

to transformers with no CoT.

2. With linear CoT, transformers can recognize NC^1-complete languages (Theorem 4.3). Thus, these transformers can likely solve problems beyond TC^0 and are more expressive than transformers with no CoT. At most, linear-CoT transformers can recognize the context-sensitive languages (Theorem 4.9).

3. With polynomial CoT, transformers converge exactly to polynomial-time Turing machines: i.e., they recognize exactly P (Theorem 4.7).

An interesting open question is the extent to which this characterization can be tightened. There is probably not much opportunity to tighten the time upper bound (Theorem 4.6), as attention is fundamentally quadratic. On the other hand, there is some opportunity to tighten either the time lower bound (Theorem 4.2) or the space upper bound (Theorem 4.8), and there is an interesting connection between these results. For $t(n) \geq \Omega(\log n)$, Merrill and Sabharwal [MS24b] remark:[1]

> *Given our* $\mathsf{TIME}[t]$ *lower bound, the tightest possible space upper bound without making fundamental complexity advances would be* $\mathsf{SPACE}[t/\log t]$ *[HPV77]. Conversely, our lower bound can only be tightened to* $\mathsf{TIME}[t \log t]$.

Excitingly, fundamental complexity advances have been made since Merrill and Sabharwal [MS24b] was published, for the case where $t \geq \Omega(n)$. Williams [Wil25] shows that $\mathsf{TIME}[t] \subseteq \mathsf{SPACE}[\sqrt{t \log t}]$, refining the result of Hopcroft, Paul, and Valiant [HPV77]. Thus means that Theorem 4.8 could in principle be improved to $\mathsf{CoT}[t] \subseteq \mathsf{SPACE}[\sqrt{t \log t}]$, or that Theorem 4.2 could in principle be improved to $\mathsf{TIME}[t^2/\log t] \subseteq \mathsf{CoT}[t]$. While a refined space upper bound is possible in principle, we believe it would not follow straightforwardly from Williams [Wil25]'s approach, which relies heavily on the locality of Turing machine computation. Since each CoT

[1] Quoted mathematical notation and citation formatting been made consistent with that used here.

step on a transformer is not local in the sense that it can aggregate global information from all previous tokens, it is not clear how this technique could be extended to transformers with CoT. For this reason, it may be more likely that t CoT steps is more powerful than t Turing machine steps, and perhaps it will be possible to prove this.

4.2 Padded and looped transformers

CoT is one way to dynamically allocate computation in a transformer at inference time, and it takes a particularly sequential nature. However, there are other ways to allow dynamic computation in a transformer, and it is interesting to consider how they compare to chain of thought in terms of changing the models expressivity. Two of the most natural ways are **padding**, or adding blank tokens (rather than generated tokens) onto the end of the model input buffer, and **looping**, or repeating a block of layers a variable number of times. As we will see, these approaches map closely onto the notions of width and depth in circuit complexity, but in a uniform way, i.e., the width and depth can be expanded *without* adding any new parameters. We will analyze theoretically how these uniform width and depth knobs allow transformers to solve more problems than they (likely) could otherwise.

4.2.1 Padding tokens

Various works [Goy+24; PMB24] have considered how extending a transformers' input buffer with some blank tokens might improve its computational power. Padding tokens added to transformers can be defined as a restricted form of CoT where the tokens on the CoT must be some special "blank" token rather the token previously generated by the LM. Unlike standard CoT, padding tokens do not add any sequential dependencies to the transformer's computation graph, meaning inference remains fully parallelizable.

Just like we can talk about the number of CoT tokens as a resource, we can talk about the

number of padding tokens. Transformers with padding tokens remain in TC^0 as long as the number of padding tokens is polynomial in the input length [MS23b; MS23a; Chi24]. However, this does not mean that padding tokens are useless: they do appear to extend the expressivity of transformers within TC^0. In fact, with polynomial padding tokens, the following theorem shows that transformers can recognize all of TC^0:

Theorem 4.11 ([MS25]). *An* $\mathsf{FO}[\mathsf{M}^2]$ *formula with* k *distinct variables and nesting depth* d *can be computed by AHATs with masked pre-norm using* n^k *padding and (fixed) depth* d.

Mix Barrington, Immerman, and Straubing [MIS90] show that FO-uniform TC^0 is definable by FO formulas with M^2 quantifiers: notably, bit is not necessary in the definition, unlike with standard $\mathsf{FO}[\mathsf{M}]$. Thus, it follows from Theorem 4.11 that transformers can recognize all of TC^0:

Corollary 4.12. *Masked pre-norm AHATs with polynomial padding recognize exactly* FO*-uniform* TC^0.

These results show that adding padding is sufficient to close the (likely) expressivity gap between unpadded transformers and TC^0. Intuitively, padding expands the width of a circuit, or, in logical terms, allows the transformer to quantify over assignments for a larger number of variables. With an unbounded polynomial number of padding tokens, this expansion of the width is sufficient to recognize any language in TC^0, and, with $O(n^k)$ padding tokens for bounded k, it enables simulating formulas with up to k variables.

4.2.2 Looped transformers

Padding is a way to grow the circuit width of a transformer uniformly, i.e., in a way that does not require adding new parameters for larger input lengths. Increasing the width polynomially does seem to add expressive power, but it cannot add expressivity beyond TC^0. To gain expressivity beyond TC^0, we must instead aim to increase the *depth* of the transformer. One

natural way increase depth uniformly is to loop a certain block of layers a variable number of times (depending on input sequence length n). This idea goes under several names (e.g., universal transformers), but we will refer to this model as a *looped transformer*. We will show that minimal looping can add expressivity to transformers, allowing them to solve problems outside TC^0. By mixing looping and padding, we can expand the expressivity even further, all while maintaining a transformer model that is relatively parallelizable.

We define looped transformers with depth $d(n)$ in the natural way. A looped transformer has a fixed list of layers divided into three blocks: a "setup" block that runs first, a "looped block" that is repeated $d(n)$ times, and a "tear-down" block of depth that runs after the final looped block. Note that a $d(n)$-depth looped transformer is a special case of a $O(d(n))$-depth non-uniform transformer (whose parameters can depend on n) with a specific uniform structure. With $d(n) = O(1)$, a looped transformer is equivalent to a general fixed-depth transformer. We are especially interested in *log-depth* looped transformers, which allow only a minimal increase in depth, meaning they are still relatively parallelizable in practice. Our first result shows that log-depth looped transformers (without any padding tokens) can recognize any regular language:

Theorem 4.13 ([MS24a], Theorem 4.1)**.** *Let L be a regular language over Σ and $\$ \notin \Sigma$. Then there exists a log-depth looped AHAT with masked pre-norm that, on any string $w\$$, recognizes whether $w \in L$ when unrolled to $\lceil \log_2 |w| \rceil$ depth.*

Proof sketch. The full proof [MS24a] follows Liu et al. [Liu+23b] in implementing the log-depth transition monoid construction for regular language recognition. However, we implement this in a uniform looped way (with a fixed set of parameters for all n), whereas Liu et al. [Liu+23b] require non-uniform parameters that can depend on n. The core technical innovation for showing the uniform result is to show that integer division and remainder can be implemented by a uniform fixed-depth transformer. This is useful as a subroutine for implementing the routing in the transition monoid construction. □

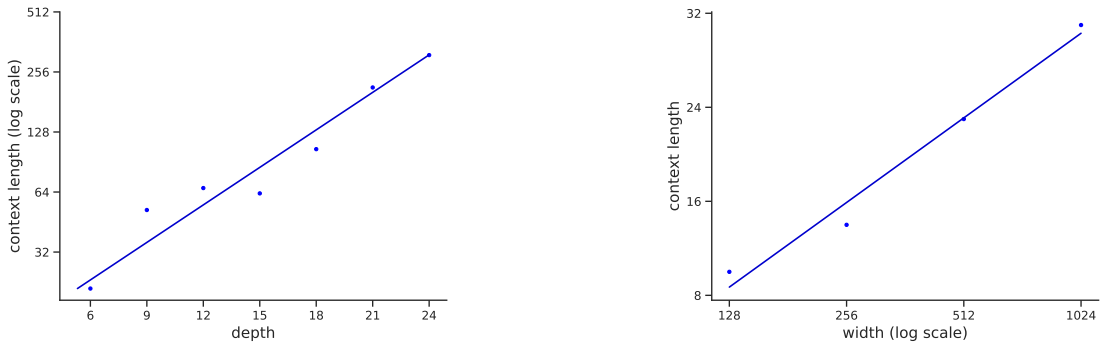

Figure 4.1: Figure taken from Merrill and Sabharwal [MS24a]. Strong linear fits imply theory/experiment match for modeling the impact of **depth** (left, $d = 4.8 \log_2 n - 15.8$ with $r^2 = 0.93$) and **width** (right, $n = 7.2 \log_2 w - 41.7$ with $r^2 = 0.98$) on effective context length for the A_5 state tracking task, a canonical hard regular language recognition problem. As predicted by Theorem 4.13, to recognize strings of length n, depth only needs to increase minimally $\propto \log n$ while width must increase drastically as $\exp(\Theta(n))$.

Because recognizing some regular languages is NC^1-complete, it follows that log-depth looped transformers are more powerful than fixed-depth transformers under standard complexity conjectures:

Corollary 4.14. *If* $\mathsf{TC}^0 \neq \mathsf{NC}^1$*, log-depth looped AHATs are strictly more powerful than fixed-depth AHATs.*

Empirical results suggest that, in line with Theorem 4.13, log depth is necessary and sufficient to recognize regular languages with transformers. Empirically, we train transformers of varying depths and widths on the A_5 word problem, an NC^1-complete regular language. For each transformer, we then measure n, the maximum length to which the transformer can solve regular language recognition on a test set. As shown in Figure 4.1, there is a strong linear fit between $\log n$ and the transformer's depth, in accordance with Theorem 4.13. On the other hand, if we vary width instead of depth, there is a strong linear fit between n and log width. This suggests that while minimal, logarithmic increase in depth is sufficient to recognize regular languages over long sequences, it is intractable to increase width to enable regular language recognition, in accordance with the predictions of our expressivity analysis.

These results show log depth can expand the power of transformers for regular language

recognition, which is closely related to state tracking (cf. Section 5.4), but what are the limits of how it can help for other computational capabilities? To gain a general understanding, we study the expressivity of transformers with both $\log^k n$ looping and $\text{poly}(n)$ padding. Like CoT, these are both ways to uniformly extend the computational resources of the transformer at inference time (varying depth and width, respectively), but, in contrast to CoT, both of these resources preserve the relative parallelism of the transformer computation, at least in theory. Thus, the computational power of this model of transformers represents the best hope for what transformers could achieve without some explicit sequential extension like CoT.

We can characterize the computational power of transformers with looping and padding in terms of first-order logic with majority quantifiers (cf. Sections 2.3 and 3.5):

Theorem 4.15 ([MS25])**.** *For $k \geq 1$, $\log^k$-depth looped, $\text{poly}(n)$-padded AHATs with masked pre-norm recognize exactly* L*-uniform* TC^k.

Theorem 4.15 suggests that log-depth looped transformers with padding recognize exactly TC^1. Moreover, considering arbitrary polylogarithmic-depth looping, Theorem 4.15 shows we can represent exactly the class NC. This represents the fundamental limit for what computation is parallelizable: it contains many somewhat parallelizable problems thought to be outside TC^0 (e.g., recognizing regular and context-free languages), but, unless $\mathsf{NC} = \mathsf{P}$, it does not contain any P-complete problems (cf. Section 3.4.5). Thus, we see that looping and padding are sufficient to extend the computational power of transformers substantially beyond TC^0 while preserving parallelism, though they do not increase the power to the same degree as polynomial CoT, which converges to P.

4.2.3 Discussion: uniform knobs for transformer depth and width

In summary, the results in this section show that padding tokens and looped layers can be understood as uniform compute knobs for expanding the computational power of transformers for

sequential reasoning problems. Growing the depth of a transformer by a factor of only $\Theta(\log n)$, it is possible to extend the expressive power of transformers outside of TC^0, which is not the case with $\Theta(\log n)$ CoT steps. With polynomial padding and polylogarithmic looping, we can extend the power of transformers all the way up to NC. Whereas CoT would reach further to P, this comes at the loss of parallelism, whereas padding and looping preserve parallelism to extent. This suggests that there is massive theoretical potential for dynamically extending the computational power of transformers via looping and padding, but a question remains about whether this can be made into a practical method. For example, it could be the case that it is hard for transformers to learn how to effectively use padding tokens or looped layers: perhaps additional supervision provided during finetuning makes CoT easier to use. Better understanding these practical details is an interesting open question for future work.

4.3 Parallelizable recurrent architectures

Another recent question in the development of LM architectures is whether it is possible to move away from the transformer back towards a sequence modeling architecture based on recurrence. Before transformers, recurrent neural networks (RNNs), based on the idea of processing strings token by token, were the dominant neural approach for processing language. This kind of recurrent processing is arguably more natural for sequential data compared to attention, but the landmark transformer paper established that attention-only architectures could outperform recurrent networks will achieving greater parallelism, which is important for scaling them up on large datasets [Vas+17]. But attention is not without conceptual drawbacks: inference on long sequences requires more memory than with RNNs, and Theorem 3.1 reveals that attention fundamentally cannot express certain kinds of sequential dependencies. Recently, advances in recurrent sequence modeling have made recurrent architectures more parallelizable, begging the question of whether modern recurrent architectures might alleviate some of the drawbacks of

the transformer. One promising way to help understand the tradeoffs between transformers and recurrent LMs is to compare their theoretical expressivity: for example, are there ways in which attention is "not all you need", but where recurrence gives sequential architecture more expressive power on transformers? Expressivity of analysis of linear RNNs has helped evaluate and guide the development of new architectures in this area. Indeed, the theory reveals that there is the potential for parallelizable recurrent architectures that can express sequential computation beyond transformers.

4.3.1 Linear RNNs

The lack of parallelism in traditional and gated RNNs [Elm90; HS97; Cho+14] was a major barrier towards training them at larger scales and sequence lengths: indeed, even before the transformer took off, there was significant research aiming to constrain RNNs to make them more parallelizable [Bra+17; Lei+18]. However, this line of research has been revived recently as transformers have been applied to problems with longer sequence lengths. Various works [GGR22; GD24; Kat+20; Yan+24c; Pen+25] have attempted to design RNNs that benefit from parallelism similarly to the transformer but with reduced memory at inference time and potential advantages for recurrent state tracking. While these architectures have been developed under a patchwork of related frameworks and names (e.g., linear transformers, linear RNNs, state-space models, DeltaNet), they all generally share the property that they are architectures with a recurrent state whose update rule is a linear function, making it efficiently parallelizable. The following definition of a linear RNN captures many of these approaches:

Definition 4.16 (Linear RNN [MPS24]). *Given a sequence* $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^k$*, the* recurrent form *of a linear RNN layer defines a new sequence of states* $\mathbf{h}_1, \dots, \mathbf{h}_n \in \mathbb{R}^d$ *using projections* $\mathbf{A}_i \in \mathbb{R}^{d \times d}$ *and* $\mathbf{B}_i \in \mathbb{R}^{d \times k}$*, which can themselves depend on* $\mathbf{x}_i$*. For each* $1 \le i \le n$*,*

$$\mathbf{h}_i = \mathbf{A}_i \mathbf{h}_{i-1} + \mathbf{B}_i \mathbf{x}_i.$$

The layer outputs $\mathbf{y}_i = \mathbf{C}_i\mathbf{h}_i + \mathbf{D}_i\mathbf{x}_i \in \mathbb{R}^k$*, where* $\mathbf{C}_i \in \mathbb{R}^{k\times d}$ *and* $\mathbf{D}_i \in \mathbb{R}^{k\times k}$ *depend on* $\mathbf{x}_i$*.*

The *convolutional form* of Definition 4.16 computes the same $\mathbf{h}_1, \ldots, \mathbf{h}_n$ via:[2]

$$\mathbf{h}_i = \sum_{j=1}^{i} \left(\prod_{k=j+1}^{i} \mathbf{A}_k \right) \mathbf{B}_j \mathbf{x}_j. \tag{4.1}$$

The convolutional form is what allows these architectures to benefit from parallelism on the sequence length dimension, as efficient parallel algorithms exist for computing iterated sums and products [Ble90]. In contrast, the nonlinearity in traditional RNN architectures prevents them from being paralellized in this way. Definition 4.16 is general enough to capture S4, Mamba, and some other SSMs, though other related architectures including linear attention [Kat+20] and DeltaNet [Yan+24c] require a generalization where $\mathbf{h}_i$ is replaced by a matrix-valued state.

4.3.2 Limited expressivity of basic SSMs

Our first result shows that standard linear RNNs, including the SSM architecture S4 [GGR22], can be simulated in TC^0:

Theorem 4.17 ([MPS24], Theorem 4.2)**.** *Consider a linear RNN (e.g., S4) where each layer has the form* $\mathbf{h}_i = \mathbf{A}\mathbf{h}_{i-1} + \mathbf{B}\mathbf{x}_i$*. Then, the output of the network can be computed in* L*-uniform* TC^0*.*

Thus, despite the fact that S4 and other simple linear RNNs have a seemingly recurrent form, they can only implement highly parallelizable computation. The same applies to Mamba [GD24], a linear RNN that, unlike S4, allows input-dependent gating in the transition matrix $\mathbf{A}_i$:

Theorem 4.18 ([MPS24], Theorem 4.3)**.** *Consider a linear RNN (e.g., Mamba) where* $\mathbf{A}_i$ *is diagonal but input dependent. Let* $\mathbf{x}_i \mapsto \mathbf{A}_i$ *be computable in* L*-uniform* TC^0*. Then, the output of the network can be computed in* L*-uniform* TC^0*.*

[2] We define the cumulative product to unroll from greatest to least, e.g., $\prod_1^3 \mathbf{A}_i = \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1$. The order is important due to the non-commutativity of matrix multiplication.

Thus, S4, Mamba, and other other linear RNN variant whose transition matrix is non-input-dependent or diagonal can only solve problems in TC^0. Thus, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, these architectures cannot express inherently sequential NC^1-hard problems. In particular, this suggests that S4 and Mamba, alongside transformers, should not be able to recognize regular languages, a task which is NC^1-complete [MPS24]. These regular language tasks can be taken as a formal model of state tracking problems like composing chess moves or entity tracking (Section 5.4), suggesting potential limitations of fixed-depth basic SSMs on these tasks [MPS24].

4.3.3 Towards parallelizable and expressive architectures

Transformers, S4, and Mamba can all only recognize languages in TC^0. However, this is not true of all linear RNNs: it turns out there are minimal extensions to existing architectures that remain parallelizable on hardware while also gaining expressivity for NC^1-complete problems. In particular, as a minimal proof of concept, Merrill, Petty, and Sabharwal [MPS24] introduce input-dependent S4 (IDS4), where the transition matrix $\mathbf{A}_i$ is reparameterized as a linear projection of $\mathbf{h}_{i-1}$. They show IDS4 can recognize any regular language, thus gaining expressivity beyond TC^0 under standard conjectures:

Theorem 4.19 ([MPS24], Theorem 5.2)**.** *For any regular language, There exists a 1-layer IDS4 model that recognizes it.*

Due to this greater expressivity, IDS4 should be more capable of hard state tracking tasks compared to transformers, S4, or Mamba. Merrill, Petty, and Sabharwal [MPS24] test this empirically by training IDS4 and other architectures on several algebraic word problem tasks. As shown in Figure 4.2, whereas transformers, S4, and Mamba require depth to grow with sequence length to solve the NC^1-complete task A_5, both IDS4 and classical nonlinear RNNs can succeed with just one layer. This shows that, at least in this setting, IDS4 models can learn to use their greater expressivity to solve hard state tracking tasks.

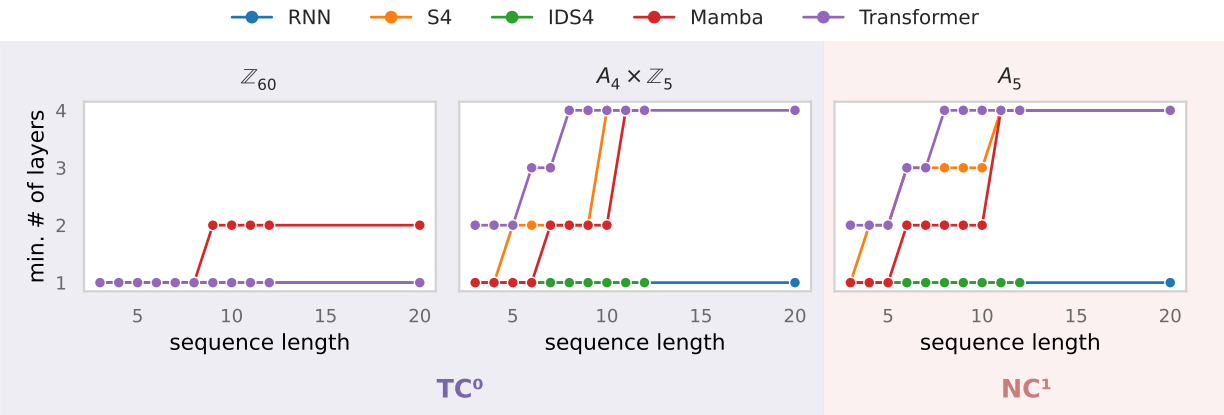

Figure 4.2: Figure taken from Merrill, Petty, and Sabharwal [MPS24]. The depth required to achieve strong accuracy by different architectures on three state tracking tasks, of which A_5 represents hard (NC^1-complete) state tracking. On A_5, classic RNNs and IDS4 can succeed with one layer for all sequence lengths, whereas transformers, S4, and Mamba require the number of layers to grow with the sequence length, as predicted by expressivity results (Theorems 3.1 and 4.17 to 4.19).

While IDS4 is efficienty parallelizable using the parallel SCAN algorithm [Ble90], it requires dense transition matrices, which translates to high memory overhead in practice. Inspired by Theorem 4.19, follow-up work has developed and scaled up more practical architectures that gain expressivity with less memory overhead. In particular, in the DeltaNet framework [Yan+24c], the transition matrix can be expressed as a sum of an identity and rank-1 matrix, which unlocks some NC^1-complete state tracking capabilities [Gra+25; Sie+25]. As this parameterization is convenient for reducing memory overhead, Grazzi et al. [Gra+25] successfully train an LM with this architecture at the 1.3B parameter scale, with the variant with expressivity guarantees outperforming a simpler baseline on math and code reasoning benchmarks. The RWKV-7 architecture, trained up to the 2.9B parameter scale, follows a similar idea of a diagonal-plus-rank-1 parameterization [Pen+25]. Notably, we show that RWKV-7 has strong expressivity guarantees for state tracking:

Theorem 4.20 ([Pen+25], Theorem 3)**.** *For any regular language, there exists a 4-layer RWKV-7 model that recognizes it.*

Similar to with IDS4, this enhanced expressivity translates to impressive performance of RWKV-7 on synthetic state tracking evaluations: as shown in Figure 4.3, RWKV-7 requires fewer

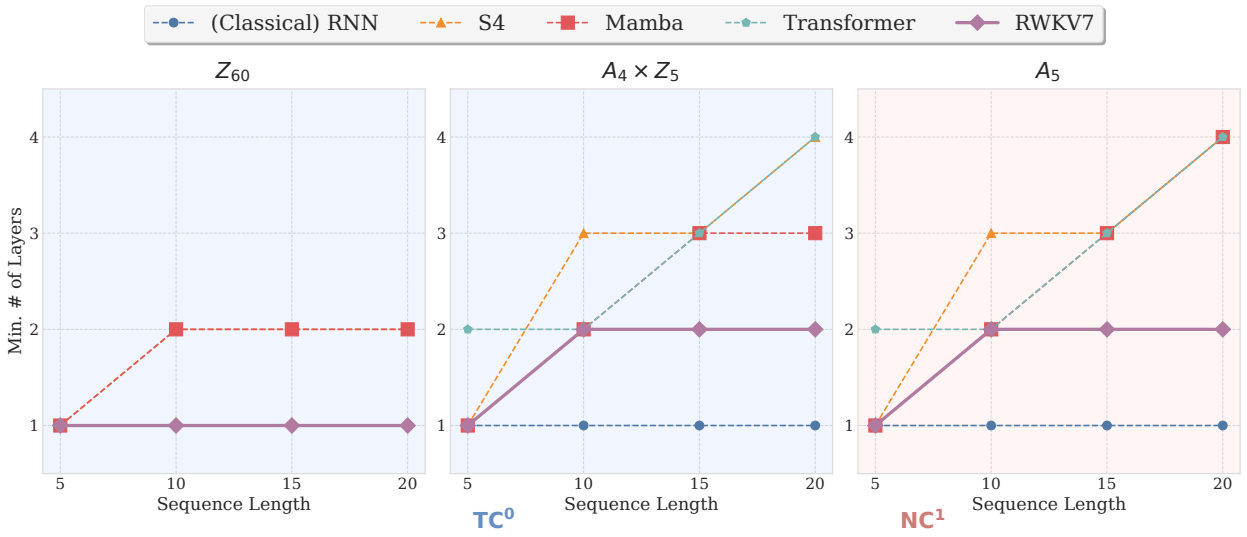

Figure 4.3: Figure taken from Peng et al. [Pen+25]. In line with Theorem 4.20, RWKV-7 requires fewer layers than S4 or Mamba to solve the hard state tracking task A_5 across sequence lengths.

layers than S4 and Mamba (at most 2) to solve the A_5 state tracking task across sequence lengths. Beyond this evaluation on a synthetic task, RWKV-7 shows strong compute efficiency on downstream multilingual benchmarks and comparable performance to Qwen 2.5, a competitive transformer baseline [Qwe+25], on English benchmarks [Pen+25, Figures 3 and 4].

In summary, Theorem 4.19 has inspired architectural innovations like the extension to DeltaNet, RWKV-7, and even more recent approaches like PaTH attention [Yan+25]. In general, these approaches are linear RNN architectures that balance enhanced expressivity with parallelism and memory overhead, making them feasibly scalable for large-scale language modeling.

5 Implications of expressivity results

The theoretical results on transformer expressivity discussed here, in particular, Theorem 3.1, help us understand the **computational model** of transformers. On one hand, expressivity results can help us understand what computation looks like inside transformers, giving us principled formalisms for representing the algorithms that can be implemented internally by transformers (Section 3.5) and revealing the different computational primitives enabled by different types of attention (Section 3.2). On the other hand, expressivity results are interesting because they help us reason about the fundamental limitations of LM architectures: Theorem 3.1 gives a formal characterization of many types of computation that transformer language models cannot express under standard complexity conjectures due to their inherent parallelism. I now turn to the broader implications of these formal results: what do they tell us about LMs in practice? First, I discuss the general notion of a tradeoff between parallelism and expressivity and its implications for architecture design. I then discuss the implications of these computational limitations for various capabilities we might want an LM to have: processing natural language, world modeling, in-context learning, and general-purpose mathematical reasoning. Finally, I turn to a discussion of limitations and open questions that I see as exciting opportunities for future research to address.

5.1 The parallelism tradeoff

The class TC^0 represents the class of problems that can be solved with a very high degree of parallelism (constant depth). Thus, in some sense, our result in Theorem 3.1 that transformers can be computed in TC^0 is a consequence of the fact that they were made to efficiently parallelizable for large-scale pretraining, in contrast to traditional RNNs [Vas+17]. On the other hand, transformers with CoT or traditional RNNs have expressive power beyond TC^0 under standard conjectures, but they cannot be parallelized to the same degree. These expressivity results suggest there is a fundamental **parallelism tradeoff** between the expressive power of a model and the degree to which it can be parallelized. As parallelism is essential for scaling models up, this suggests that some complexity class capturing parallel computation should be a fundamental limit for the expressive power of architectures that can be trained on the task of large-scale language modeling. In order to gain greater expressivity, these architectures must either sacrifice on parallelism, harming the scalability of pretraining, or perhaps be adapted after training to be less parallelizable at inference time.

This parallelism tradeoff resembles the more familar bias-variance tradeoff between expressivity and inductive bias, but differs in some respects. The bias-variance tradeoff suggests that more expressive models may be detrimental because they can overfit to noise in the training data, whereas less expressive models may have a better inductive bias towards the true underlying function. While this idea is attractive, it may not be the best mental model for thinking about expressivity of large language models or deep learning in general. Large language models are trained on massive amounts of data and appear to have some form of implicit regularization that prevent overfitting. Thus, in practice, their expressivity is generally not bottlenecked by concerns about overfitting, but, rather, the constraint that they must be parallelized to train on massive amounts of data. This makes the parallelism tradeoff an important consideration for evaluating and developing language modeling architectures (potentially more so than the bias-

variance tradeoff).

The parallelism tradeoff concerns parallelism on the token dimension, which is important for training LMs that can process long documents. A separate kind of parallelism that is also potentially important in practice is parallelism on the batch dimension: the more documents can be learned from in parallel during training without compromising performance, the more parallelizable training can be made. For this reason, various works have attempted to build out empirical and theoretical understanding of large-batch training, suggesting that potential for large-batch training increases as the training task becomes more difficult [McC+18; Zha+25]. In general, this line of work is complementary to the tradeoff between parallelism and discussed explored here.

5.2 The opportunity for more expressive architectures

From the point of view of the parallelism tradeoff, a natural question is whether transformer language models are at some frontier for balancing expressivity and parallelism, or whether it is possible to design a language modeling architecture that is more expressive while remaining just as parallelizable in practice. At some level, the parallelism tradeoff must fundamentally preclude gaining expressivity with no cost in parallelism: it is not possible to parallelize all efficient computation unless current conjectures ($\mathsf{NC} = \mathsf{P}$) in complexity theory are false. But perhaps the level of parallelism that transformers and other LM architectures attain (i.e., TC^0) is unnecessarily restrictive for models trained on GPUs, which, underlyingly, are AND/OR circuits lacking majority gates. This means that, in practice, the computation graph to implement a fixed-depth transformer or SSM on such hardware must end up having depth $O(\log n)$. Indeed, the SCAN algorithm [Ble90] uses $O(\log n)$ depth to compute a single layer of an SSM. In this sense, transformers are more parallelizable than they need to be: the constraint of being in TC^0 (as opposed to say, NC^1 or TC^1) does not translate to a gain in parallel runtime on today's hardware.

This suggests an opportunity: in principle, we could design model architectures that can solve problems thought to be outside TC^0 without sacrificing parallel parallelism. In particular, Merrill, Petty, and Sabharwal [MPS24] developed a linear RNN architecture that can solve NC^1-complete problems (including state tracking) while being parallelizable on hardware using the same parallel SCAN algorithm [Ble90] used to parallelize standard linear RNNs. This architecture is a proof-of-concept that we can design architectures with expressive power beyond transformers without sacrificing practical parallelism since transformers and related arhitectures are, in some sense, overly parallelizable compared to what today's hardware can exploit. In principle, there is opportunity to similarly extend expressivity to handle even more complex problems such as those that are NL-complete.

As discussed in Section 4.3.3, an additional constraint on linear RNNs is the memory overhead required by the parameterization of the transition matrix. Several works have already considered how to achieve expressivity of NC^1-complete problems with a memory-efficient parameterization [Gra+25; Pen+25], but figuring out how to best balance expressivity, parallelism, and memory overhead is an important guiding question for future research on linear RNN architectures.

Finally, a different interpretation of the fact that current hardware does not take full advantage of the parallelism of TC^0 architectures is that hardware advances could, in principle, make transformers and other LM architectures parallelizable. Partially inspired by this motivation, some recent theoretical work suggests there may be potential for classical or quantum hardware that could more efficiently compute majority gates compared to standard hardware today [CBM25; Oli+25]. This is an example of how theoretical understanding of architectures' expressivity might inform the development of specialized hardware for LMs.

5.3 Processing the structure of natural language

One of the goals originally motivating the analysis of deep learning architectures in terms of formal language and complexity theory was understanding whether deep learning models could theoretically build syntactic and semantic representations for natural language like those suggested in linguistic theory [Mer19]. We thus turn towards considering the implications for these questions from the complexity-theoretic analysis of transformers developed in this thesis. In particular, without approaches like CoT, is it possible for transformers to build representations of syntactic structure or carry out the computations required to evaluate the semantics of linguistic utterances?

5.3.1 Syntactic dependencies

What does Theorem 3.1 imply about the formal ability of transformers to process the syntax of natural language? Chomsky [Cho56] suggested that the hierarchical structure of natural language syntax could be approximately modeled by a context-free grammar (CFG), so we can take the circuit complexity characterization of context-free recognition as a formal model of processing natural language syntax. As discussed in Section 3.4.2, context-free recognition is in AC^1 [Ruz80; Ven91] Thus, at first glance, our characterization of transformers in TC^0 (Theorem 3.1) might be taken to suggest that transformers should struggle to represent the structure of natural language.

However, the characterization of context-free recognition in AC^1 is worst-case and does not take into account that the syntax of natural language is likely easier than an adversarial context-free grammar. For instance, for typical natural language, short sentences are much more likely than long ones, though there is a long tail of long sentences [Wil40; BK19]. Moreover, even for long sentences, we might imagine that syntactic center embedding depth is relatively bounded, or at least that sentences with less center embedding are preferred, though the degree of preference

may vary across languages. Futhermore, while natural language exhibits notable examples of syntactic ambiguities (*Time flies like an arrow*), ambiguity is not typically as widespread as is possible in some diabolical CFGs.

In general, these constraints on the types of context-free structure found in natural language likely make syntactic processing easier than worst-case CFG recognition. For instance, the pumping lemma implies that context-free languages with bounded center embedding are actually regular, and thus can be recognized by an $O(\log n)$-depth looped transformer (Section 3.4.1). As context-free languages become less ambiguous, recognition gets simpler from a circuit-complexity perspective: Dyck languages and structured CFLs, which are both special cases of deterministic CFLs, can both be recognized in TC^0 [MC89]. Thus, even if CFG recognition may be inexpressible by transformers in the worst case, constraints on the syntactic structures in natural language likely make the recognition problem for naturalistic grammars easier than for adversarial grammars (indeed, constraints or strategic avoidance of ambiguity may play a similar role in making parsing computationally efficient for humans as well [Alt88]; memory constraints, which are relevant for humans [Hah+22], are less relevant for transformers). On the other hand, it has been argued that some dependencies in natural language cannot be expressed in a context-free formalism [Shi85], so, in principle, certain aspects of natural-language syntax could be harder to model than the circuit complexity results about CFGs suggest.

5.3.2 Semantic evaluation

Semantically evaluating utterances can be harder than verifying their syntax: while recognizing well-formed boolean formulas is in TC^0, *evaluating* them is NC^1-complete [Bus87]. Thus, even if transformers can represent the hierarchical structure of natural language to an extent, evaluating natural-language expressions with a complex compositional structure is likely harder, potentially requiring CoT or dynamic depth for transformers in the worst case. On the other hand, if sentence length is typically bounded in practice, then a fixed-depth transformer may

suffice to evaluate complex compositional expressions up to a certain length.

Going beyond individual sentences, a question remains about LMs' ability to maintain semantic representations of a long document or discourse, especially when the document is long. We can address this question by considering a simple model of document-level language understanding inspired by truth-conditional formal semantics [HK98]. We take the meaning of each sentence s as its *truth conditions* $[\![s]\!] \subseteq W$: the set of possible worlds in which it is true. The meaning of a document is the set of worlds consistent with all its sentences:

Definition 5.1 (Truth-conditional denotation)**.** *Given a document (i.e., list of sentences)* $s_1 \dots s_n$*, its* truth-conditional denotation $[\![s_1 \dots s_n]\!] \subseteq W$ *is*

$$[\![s_1 \dots s_n]\!] = \bigcap_{i=1}^{n} [\![s_i]\!].$$

If we take the set of possible worlds W as finite, then evaluating the truth-conditional denotation of a document is a word problem over a commutative finite monoid (Section 3.4.1). Thus, at least in this simplified case, it seems truth conditional semantics should be close in complexity to what transformers are capable of, though technically we have not proved that transformers can express finite truth-conditional semantics—just that this problem is in TC^0. It is unclear how our characterization would change in the more general case where W is not finite, but rather finitely generated, which reflects the realistic setting where the sentences in the underlying language are compositional. It is conceivable that the structure theory of finitely generated commutative monoids could be applied to analyze this case, which we leave to future work.[1] For now, we conclude that truth-conditional semantics with a finite set of possible worlds is similar to complexity to what transformers are capable of expressing.

On the other hand, not all semantic phenomena in natural language can be captured by truth-conditional semantics (Definition 5.1). For example, presuppositions, pronoun anaphora, and di-

[1] https://mathoverflow.net/q/293883

alogues where speakers evolve over time require a semantic theory where the meaning of a sentence can change depending on its context in a discourse. To account for this, a research program of formal semantic theory called *dynamic semantics* [Nou+22] generalizes semantic evaluation to make each sentence a *relation* from possible to worlds to possible worlds. Formally, we recast $[\![s_i]\!]$ to be a relation in W^2, and semantically evaluating a document is equivalent to computing the *composition* of the relation represented by each sentence.

Definition 5.2 (Relation composition)**.** *For two relations $X, Y \subseteq W^2$, define the composition $X \circ Y$ such that $(x, y) \in X \circ Y$ if there exists $z \in W$ such that $(x, z) \in X$ and $(z, y) \in Y$.*

Definition 5.3 (Dynamic denotation)**.** *Let $\prod$ denote iterated relation composition ($\circ$). Given a document $s_1 \dots s_n$, its* dynamic denotation *$\langle s_1 \dots s_n \rangle \subseteq W^2$ is defined*

$$\langle s_1 \dots s_n \rangle = \prod_{i=1}^{n} \langle s_i \rangle.$$

The truth conditions can be recovered from a dynamic denotation via $[\![s_i]\!] = \text{diag}(\langle s_i \rangle)$, though, crucially, $\langle s_i \rangle$ contains more information than $[\![s_i]\!]$ and is not commutative. This means the truth conditions of a sentence can be context-dependent, which is important for handling phenomena like presupposition and pronoun anaphora.

As denotation composition is associative, dynamic denotations form a monoid with respect to composition. If the set of dynamic denotations is finite, this is a finite monoid, meaning composition is a regular language recognition problem (Section 3.4.1). Thus, computing it is NC^1-complete, suggesting a transformer with a fixed depth should not be able to solve this task over long documents. Moreover, due to the unbounded set of meanings expressible in natural language, it is probably not realistic to take W^2 to be finite. This means the complexity of document-level language understanding (formalized in terms of dynamic denotations) could be even more difficult than NC^1-complete, though it is hard to formalize this.

Sufficiency of distributional semantics. Beyond expressivity, a fundamental question about semantics in regards to LMs is the degree to which the meaning of sentences (or aspects of the world they refer to) can be reconstructed from the distributional pretraining objective of next-token prediction, where any information about the world or meaning behind a text must be implicit [Har54; BK20]. In a line of work orthogonal to analyzing architectures' expressivity, I have analyzed how the linguistic principles underlying human communication (such as a bias towards truthfulness or informativeness) might lead sentence co-occurrence probabilities in naturalistic text to encode abstract semantic information. Since LMs are trained to model such co-occurrences, this explains why they be able to reconstruct semantic information from training. For instance, in the context of programming languages, we showed formally and empirically that, if a code corpus has a bias towards *truthful* (i.e., correct) assertion statements, observing the distribution of these assertion statements is sufficient for learning how to execute some, but not all, programming languages [Mer+21b; Wu+23]. Similarly, if authors of a natural-language corpus have a bias towards *informativeness* (rather than truthfulness), we argued theoretically and empirically that this can lead co-occurrence patterns to closely reflect semantics in the sense of entailment relations [MWL22; Mer+24]. These works provide a preliminary understanding of how the communicative goals of speakers can create a correspondence between form and meaning in corpus data, which LMs can potentially leverage to learn semantic structure about language and the world. There is opportunity to develop and generalize this perspective, building out a theory of the types of semantic information that is easy to reconstruct from next-token prediction, and the types of semantics for which next-token prediction might be fundamentally insufficient.

5.4 World models and planning

Beyond representing the syntax and semantics of language, one cognitive capability that is interesting to investigate with LMs is their ability to maintain world models. That is, beyond the

apparent coherence of the text they generate, is it the case that LMs have a consistent model of the world while processing text about it? While this concept of world models is not entirely well defined, it is closely related to the notion of state tracking: being able to maintain a model of the state of the worlds as various actions or events take place. We have previously explored the state tracking capabilities of LMs in connection to regular languages (Section 3.4.1), and these results naturally bear on the question of LMs' world modeling capabilities. Concretely, the fact that constant-depth transformers cannot represent arbitrary regular languages suggests they should not be able to represent some kinds of world models, e.g., arbitrary chess games in source-target notation [MPS24]. However, it is an open question to what degree real world models correspond to regular languages that are hard to represent (i.e., NC^1-complete), and, deep transformers can express state tracking over exponentially long world histories, even if they cannot do so forever. These qualifications suggest that some practical, fuzzy notion of world modeling may be significantly easier to represent than the exact task; on the other hand, world modeling over large worlds may be significantly harder than the "finite" worlds captured by regular language problems. For both of these reasons, there are interesting opportunity for future work that refines our understanding of LMs' world modeling and state tracking capabilities in terms of formal language theory.

Beyond simply maintaining a model of the world, a higher target capability for LMs would be the ability to implement goal-oriented behavior, which is sometimes called planning. We can roughly define planning as, given a world model, using it to maximizing or approximately maximizing some notion of reward through interactions with the world. From a complexity perspective, computing the reward of a trajectory through the world is P-complete [GHR91]. Constant-depth transformers, or even $O(\log^k n)$ depth padded transformers, cannot solve P-complete problems unless $\mathsf{NC} = \mathsf{P}$, which suggests CoT is fundamentally necessary for LMs to achieve this notion of planning over a world model.

5.5 In-context capabilities

One capability of interest in LMs is the ability to adapt to new tasks specified in the context, either via instructions or examples provided by the user. We disentangle these two types of in-context behaviors as *instruction following* and *in-context learning*. Instruction following can be defined as executing some instruction or program specified in the input context over some data, also provided in the input context [Fin+22]. In-context learning [Bro+20] is similar, except that the behavior that should be executed is not specified via explicit instructions, but, rather, via input-output examples. These two capabilities are also potentially intertwined, as it has been suggested that one way LMs learn new tasks in-context is via inferring and executing compositional programs made out of tasks learned during pretraining [HG23; Li+24a]. Understanding the degree to which LMs can implement these behaviors reliably is important for evaluating to what degree general-purpose LMs can be adapted on the fly to solve new, domain-specific problems.

Our expressivity results suggest that transformer LMs must likely rely heavily on CoT to execute even simple instruction following tasks. In addition, we discuss how understanding transformer expressivity can potentially differentiate which learning algorithms can be executed in-context by transformers without CoT, which could possibly inform the analysis of how LMs carry out in-context learning in practice.

5.5.1 Instruction following

We formalize instruction following following Finlayson et al. [Fin+22]. The input contains some program P (specified for some model of computation) as well as an input w, and the goal of the task is to compute $P(w)$, which, for simplicity, we consider to be a single bit. Thus, instruction following can be thought of as an "in-context" variant of language recognition, where the underlying formal language varies depending on the program specified in the context.

The difficulty of the instruction following task depends on the complexity of the programs

that are allowed. However, even for very simple programs, instruction following is hard for transformers. Finlayson et al. [Fin+22] find empirically that transformer LMs struggle to solve instruction following where the programs are regular expressions. Theoretically, this task is NC^1-hard since it is at least as hard as fixed regular language recognition, so, Theorem 3.1 implies transformers likely cannot solve it without CoT, in line with Finlayson et al. [Fin+22]'s results. If we replace regular expressions with context-free grammars with ϵ productions, it is known that the instruction following task increases in difficulty to P-complete [GHR91], suggesting transformers require at least linear CoT to solve it unless $\mathsf{P} = \mathsf{NC}$.

Beyond regular expressions and context-free languages, another natural variant of the instruction following task is where the programs are boolean formulas (e.g., $x_1 \wedge \neg x_2$) and the inputs are bit strings. In this case, the task is NC^1-complete, as it is essentially the boolean formula value problem. Thus, fixed-depth transformers without CoT cannot solve it assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$. If we generalize the boolean formulas to be serializations of boolean circuits, this problem becomes P-complete [GHR91], so, as with the case of context-free grammars, transformers require at least linear CoT unless $\mathsf{NC} = \mathsf{P}$. Overall, these results suggest that transformers without CoT should be incapable of following complex instructions and provide guidance on how much CoT should be necessary for tranformers to execute instructions of different levels of complexity.

5.5.2 In-context learning

In-context learning [Bro+20] is a kind of inductive reasoning where an LM is expected to generalize a rule from some examples and apply it to a novel input. More formally, the input for an in-context learning problem is a sequence of labeled examples $D = \{\langle x_i, y_i \rangle\}_{i=1}^n$, as well as an unlabeled example x. The goal is to return $h^*(x)$, where h^* is some hypothesis computed from D via some learning algorithm. There has been significant empirical work evaluating LMs' in-context learning and inductive reasoning capabilities [Qiu+24; Hua+25a, inter alia] and work exploring the kind of learning algorithms that transformers might use for in-context learning [Aky+23;

Aky+24]. Our theoretical results about transformer expressivity can be leveraged to better understand the kinds of inductive reasoning that transformers could conceivably implement. In particular, we can compare the extent to which parallelizable transformers can implement sequential learning algorithms like gradient descent, and compare this to their ability to implement more parallelizable learning algorithms like Bayesian inference.

GRADIENT-LIKE DESCENT. One natural learning algorithm to use to select h^* in in-context learning would be with an algorithm resembling gradient descent. Past empirical and theoretical work has considered the ability for transformers to implement in-context gradient descent to solve linear regression and other in-context learning problem [Gar+22; Aky+23]. Inspired by this past work, we can define the following in-context learning algorithm that formalizes using a learner resembling gradient descent:

Definition 5.4 (In context gradient-like descent)**.** *Assume H is parameterized by θ. Let $G(x, y; \theta)$ be a function that returns new parameters θ'. In-context gradient-like descent initializes $\theta_0 = 0$ and then computes $\theta_{i+1} = G(x_i, y_i; \theta_i)$. The learner selects the hypothesis $h^* = h_{\theta_n}$.*

Standard gradient descent with batch size 1 can be recovered by making H differentiable with respect to θ and setting $G(x, y; \theta) = \theta - \eta \nabla L(x, y)$ for some loss function L. In the other direction, given a finite list of points $D = \{\langle x_i, y_i \rangle\}_{i=1}^n$, we can construct a loss function whose gradient matches G on D. Thus, gradient-like descent is equivalent to gradient descent with some loss and batch size 1, except that not all choices of G correspond to common loss functions.

Existing constructions for gradient descent with transformers require the depth to grow linearly with the context so that a sequence of steps can be applied sequentially, one per layer. Thus, transformers would not be able to implement gradient descent over long sequences assuming the depth of the network is relatively small, as is typically the case in practice. Intuitively, the linear depth of gradient construction feels necessary, since each gradient step depends on the steps taken previously. Interestingly, it follows from Theorem 3.1 that, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, trans-

formers require depth growing with the sequence length to implement in-context gradient-like descent:

Proposition 5.5. *There exists a gradient function G such that in-context gradient-like descent is* NC^1*-hard under* FO *reductions.*

Proof. For an arbitrary regular language $L \subseteq \Sigma^*$, we will construct G such that recognizing whether $w \in L$ can be reduced to in-context gradient-like descent with $D = \{\langle w_i, \$\rangle\}_{i=1}^n$ and $x = \$$ w.r.t. G, where $\$ \notin \Sigma$ is a special symbol. Since recognizing regular languages is NC^1-complete, it follows that in-context gradient-like descent is NC^1-hard.

Since L is regular, it can be recognized by a deterministic finite-state automaton with states Q and transition function δ. We identify H with Q, representing each state $q \in Q$ as a point θ_q in parameter space. For each hypothesis $h \in H$, we set $h(\$) = 1$ iff the state q associated with h is accepting. We set θ_0 to the initial state, i.e., θ_{q_0}. We then construct G as follows:

$$G(x, y; \theta_q) = \theta_{\delta(x,q)}.$$

By construction, the h^* returned by gradient-like descent on $\{\langle w_i, \$\rangle\}_{i=1}^n$ encodes the state reached by the automaton on input w. Thus, $h^*(\$)$ returns whether $w \in L$. □

Theorem 5.5 formalizes the inherently sequential nature of in-context gradient descent, suggesting it cannot be implemented by constant-depth transformers. This simple example is meant to give an intuitive proof of concept for how gradient descent in transformers requires depth growing with the sequence length. It is somewhat artificial in the sense that the loss implies by G is not necessarily natural. Further, it is possible that this argument could be strengthened to show that in-context gradient is actually more sequential than NC^1-hard: for example, it would be interesting to consider whether it could be shown that in-context gradient descent is P-complete, which would imply that it likely requires *linear* depth when implemented by transformers.

In-context Bayesian inference. Gradient descent is not the only approach to attempt to implement in-context learning. Another learning framework discussed in the literature is Bayesian inference: an LM could use in-context examples D as well as a prior to rank hypothesis by their likelihood [Xie+22; Qiu+25]. More formally, for each hypothesis h, we assume we are given a prior $p(h)$ and a way to assign probability to data $p(y, \mid x, h)$. In-context Bayesian inference selects h^* as the hypothesis that maximizes

$$p(h \mid D) = p(h) \prod_{i=1}^{n} p(x_i, y_i \mid x_i, h).$$

In contrast to gradient descent, where finding h^* is a sequential process over different data batches, much of the work in scoring $p(h \mid D)$ can be done in parallel over the data. Along these lines, it turns out that there is a natural TC^0 construction to implement in-context Bayesian inference. Assume the hypothesis class has size at most polynomial in the number of samples, i.e., $n = \Omega(\log|H|)$. Further, we will assume each hypothesis $h \in H$ is simple enough such that $\log p(h)$ and $\log p(x_i, y_i \mid h)$ are computable in TC^0. To perform in-context Bayesian inference in TC^0, we first compute the following in parallel for each h:

$$\log p(h) + \sum_{i=1}^{n} \log p(x_i, y_i, \mid h).$$

Next, we can use an AC^0 circuit to find h^* as the hypothesis that maximizes $\log p(h)$. This shows that, due to its inherent parallelism, in-context Bayesian inference can be implemented in TC^0. This likely makes it closer to the kinds of algorithms transformers can express without CoT compared to in-context gradient descent.

5.6 General mathematical reasoning

A central theme in the historical development of computer science from mathematics and philosophy has been the quest to develop an automated general-purpose system for reasoning and mathematics by some, and the establishment of fundamental limits on this enterprises by others. In some cases, the development of negative results frustrated ongoing projects towards automating mathematics: most crucially, Gödel's incompleteness theorems strongly suggested that the Hilbert Program to automate mathematical reasoning could not be achieved due to the undecidability of first-order logic [Göd31; Raa22]. As noted in a letter from Gödel to von Neumann [Har93], the P = NP question in complexity theory can be understood as a refinement of this earlier question: the simpler problem of determining if first-order formulas have a proof of size at most n is not possible in polynomial time unless P = NP [Har93].

Despite these foundational negative results, some have laid out a vision for an, allegedly, not so distant future where AI systems automate formal mathematical reasoning, or at least drastically surpass humans' abilities, driving mathematical advancements of immense practical and intellectual value. For example, a talk by Noam Brown of OpenAI at the Simons Institute justifies the steep cost of the o1 model because it might lead to new life-saving drugs or deep mathematical insights, posing the following question to an audience of theoretical computer scientists:[2]

What inference cost are you willing to pay for a proof of the Riemann Hypothesis?

This vision has led to the creation of hard mathematical benchmarks [Hen+21; Gla+24] that, to an extent, show progress for recent LMs [Dee+25; Ope25].[3] How do the results in this thesis on the computational power of LM architectures bear on the feasibility of this quest for automated mathematical reasoning with LMs and any fundamental barriers it might run into?

I do not claim to have much definitive to say about this deep question. But the computational

[2] https://simons.berkeley.edu/talks/noam-brown-openai-2024-09-26

[3] The related area of autonomous programming has undergone similar developments [Jim+24; Yan+24b].

characterization of LM architectures established here and in related work might provide a framework for addressing it and anticipating what architectural properties of LMs would be required for general mathematical reasoning. At the very least, many notions of worst-case general mathematical reasoning are well beyond the capabilities of transformers. Theorem 3.1 establishes that transformer LMs without CoT can only solve highly parallelizable problems in the class TC^0. In contrast, the determining whether a first-order formula is provable is undecidable and remains NP-complete even if we only care about proofs up to some maximum length [Har93].[4] So, transformers without CoT cannot generally prove mathematical results, even if we only care about proofs of bounded length (assuming $\mathsf{TC}^0 \neq \mathsf{NP}$, which is weaker than $\mathsf{P} \neq \mathsf{NP}$). Moreover, simpler forms of mathematical reasoning such as linear programming, circuit evaluation, or Horn satisfiability are P-complete [GHR91], which means transformers without CoT could not even express these unless $\mathsf{TC}^0 = \mathsf{P}$.

But, of course, the most sophisticated reasoning models today rely heavily on CoT: does this change the theoretical expectation for LMs' reasoning capabilities? Indeed, with CoT, Theorem 4.2 shows that the power of transformer LMs is expanded (likely) outside TC^0 with enough CoT steps, enabling transformer LMs with enough CoT steps to express linear programming, circuit evaluation, or Horn satisfiability. However, Theorem 4.6 shows, unsurprisingly, that transformer LMs would still require a superpolynomial number of CoT steps to solve hard mathematical reasoning problems like determining whether first-order statements have bounded-length proofs (NP-complete), propositional satisfiability (NP-complete), or propositional tautology (coNP-complete). Thus, while CoT extends the capabilities of transformers for inherently sequential computation, we should not expect CoT reasoning models to magically enable general mathematical reasoning.

A reasonable objection here is that, while these mathematical reasoning problems are hard in the worst case, restricted types of mathematical reasoning could very well be easier in practice, as

[4] Formally, the problem is $\{\langle \phi, 1^n \rangle \mid \phi \text{ has a proof of length at most } n\}$.

constraints on the types of theorems that are actually interesting could make this problem more tractable. A related objection is that the real target is not general theorem proving, but "just" surpassing human mathematical reasoning capabilities, which is a lower bar. In a sense, the more finegrained characterization of different LM architectures in terms of different complexity classes (Section 3.4) can be seen as a step towards addressing these objections. Rather than thinking about mathematical reasoning as a binary, understanding the computational power of LMs allows us to more qualitatively describe the kinds of reasoning they could express and compare the potential of different architectures and methods. In this way, this thesis provides a foundation for understanding the mathematical reasoning abilities of LMs.

5.7 Limitations and open questions

We now understand much more about the computational power of transformers than when the transformer architecture was originally proposed [Vas+17]. Vaswani et al. [Vas+17] famously claim that "attention is all you need" to process language, but Chapter 3 reveals that, because transformers (without CoT) rely solely on attention without recurrence, they can only express languages in the class TC^0 (Theorem 3.1). This means transformers are fundamentally limited in state tracking and other inherently sequential computation compared to recurrent architectures. Thus, while the parallelism of the transformer architecture (relative to recurrent architectures) has been important advantage for practical scalability, it comes with the tradeoff of reduced expressivity for sequential problems. In Chapter 4, we developed a finegrained theory for how CoT changes the computational power of a transformer LM by adding sequential dependencies into its computation graph. Linear CoT allows transformers to solve problems thought to be outside TC^0, and, with enough CoT, transformers can express any problem in P. But, on the other hand, CoT may not be the most efficient way to extend the computational power of LMs. Theoretically, looped and padded transformers or parallelizable recurrent architectures can gain some

expressive power beyond transformers while retaining the parallelism that CoT destroys.

There are many interesting theoretical questions around the expressive power of transformers and other LM architectures that would potentially offer guidance to the practice of building LMs. I offer some suggestions for high-level areas of future research below.

Tight characterization of soft-attention transformers. A longstanding quest in expressivity research is establishing a tight characterization of transformers, e.g., a variant of RASP [WGY21] that exactly defines the class of languages that transformers recognize. Practically, this would provide a principled "programming language" for interpretability research to use to represent transformer computation. For transformers with unique hard attention, such a characterization exists: B-RASP, or, equivalently, linear temporal logic [YCA24]. For soft-attention transformers, Theorem 3.7 [MS23a] establishes transformers can be "compiled into" the language $\mathsf{FO}[\mathsf{M}]$, and various works derive constrained logics that can be compiled into transformers [CCP23; YC24]. But there is opportunity to tighten both of these directions and potentially even obtain a single symbolic logic that fully captures soft-attention transformers.

Unconditional lower bounds for transformers. Additionally, there may be opportunity in the theory of transformer expressivity to establish unconditional power bounds for transformers on various reasoning problems. The characterization of transformers in TC^0 (Theorem 3.1) can be viewed as a lower bound on the depth or size required for transformers against NC^1-complete problems, conditional on the assumption that $\mathsf{TC}^0 \neq \mathsf{NC}^1$. This result leverages the parallelism of the transformer architecture, but there are other, more finegrained properties like the autoregressive structure of causal masking that could potentially be leveraged to prove unconditional lower bounds without running into fundamental complexity barriers [CPW24]. Such results could help us understand the computational differences between causally masked and unmasked transformers.

Attention that length generalizes. It would be interesting to more comprehensively understand the relations between different variants of attention and the types of attention patterns that can be represented and learned robustly. Research has documented that transformers struggle to robustly learn hard attention in a way that generalizes to longer strings [CC22; Liu+23a; Hua+25b], but is there a minimal change to softmax attention that fixes this? Solving this problem has the potential to improve transformers' length generalization capabilities.

The necessity of attention. More radically, there is an interesting question about whether we can build strong LMs *without* attention. We now understand that one key advantage of transformers compared to RNNs is that attention is easier to parallelize than (nonlinear) recurrence. On the other hand, the memory overhead of attention grows quasilinearly with sequence length, which becomes expensive at long context lengths, motivating a new generation of parallelizable recurrent architectures (cf. Section 4.3). Expressivity analysis reveals that transformers and these new recurrent have different limitations, suggesting it could be promising to combine them for greater computational power or a better tradeoff between expressivity and efficiency. Thus, it seems quite promising for theory work to continue to guide practitioners in pushing the limits of non-attention architectures for language modeling.

Beyond the worst case. One general caveat of our current understanding of expressivity is that it largely grounded in worst-case analysis. This is not necessary a limitation, as the worst case is a good model for assessing models to implement robust reasoning rather than relying on heuristics for particular data distributions. But it would be interesting to build out theoretical or empirical understanding of the distributional properties that make computational problems harder or easier for transformers. This would help the community better reconcile negative expressivity results with positive empirical results and could be useful in practice for constructing hard evaluations and diagnosing input domains where LMs might struggle.

Bridging expressivity and learnability. Finally, the expressivity viewpoint on LMs abstracts away optimization, instead focusing on what LMs are capable of with the "best possible" learning algorithm. However, it is now understood that there are problems that are possible for (some variant of) transformers to express but hard to learn, e.g., parity, due to an inductive bias of transformers towards low-sensitivity functions [HJF21; Bha+23; Vas+25]. It would thus be interesting to extend the theory of what transformers and other LMs can express to also account for what they can robustly learn via gradient-based methods. This is an ambitious endeavor, but there is a rich set of tools to draw from in classical learning theory (e.g., statistical query learning [Kea98]) and deep learning theory, with some notable examples of theoretical progress at this intersection [Bar+22; HR24; Hua+25b]. Continuing to bridge expressivity and learning has the potential to advance our fundamental scientific understanding of how LMs acquire complex computational behavior, clarify the limitations of the language modeling and reinforcement learning paradigms, and open the door to more performant, efficient, and interpretable AI systems down the road.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. 1st. USA: Cambridge University Press, 2009. ISBN: 0521424267.

[Aky+23] Ekin Akyürek et al. "What learning algorithm is in-context learning? Investigations with linear models". In: *The Eleventh International Conference on Learning Representations*. 2023.

[Aky+24] Ekin Akyürek et al. "In-context language learning: architectures and algorithms". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[Alt88] Gerry Altmann. "Ambiguity, parsing strategies, and computational models". In: *Language and Cognitive Processes* 3.2 (1988). DOI: 10.1080/01690968808402083.

[BAG20] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. "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)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 7096–7116. DOI: 10.18653/v1/2020.emnlp-main.576.

[Bar+22] Boaz Barak et al. "Hidden Progress in Deep Learning: SGD Learns Parities Near the Computational Limit". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022.

[Bha+23] Satwik Bhattamishra et al. “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 (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 5767–5791. DOI: 10.18653/v1/2023.acl-long.317.

[BK19] Gábor Borbély and András Kornai. “Sentence Length”. In: *Proceedings of the 16th Meeting on the Mathematics of Language*. Ed. by Philippe de Groote, Frank Drewes, and Gerald Penn. Toronto, Canada: Association for Computational Linguistics, July 2019, pp. 114–125. DOI: 10.18653/v1/W19-5710.

[BK20] Emily M. Bender and Alexander Koller. “Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 5185–5198. DOI: 10.18653/v1/2020.acl-main.463.

[Ble90] Guy E. Blelloch. *Prefix Sums and Their Applications*. Tech. rep. CMU-CS-90-190. School of Computer Science, Carnegie Mellon University, Nov. 1990.

[Bra+07] Thorsten Brants et al. “Large Language Models in Machine Translation”. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Ed. by Jason Eisner. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 858–867.

[Bra+17] James Bradbury et al. “Quasi-Recurrent Neural Networks”. In: *International Conference on Learning Representations*. 2017.

[Bro+20] Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[Bus87] S. R. Buss. “The Boolean formula value problem is in ALOGTIME”. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC ’87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 123–131. ISBN: 0897912217. DOI: 10.1145/28395.28409.

[CBM25] Anupam Chattopadhyay, Debjyoti Bhattacharjee, and Subhamoy Maitra. *Linear Decomposition of the Majority Boolean Function using the Ones on Smaller Variables*. 2025.

[CC22] David Chiang and Peter Cholak. “Overcoming a Theoretical Limitation of Self-Attention”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7654–7664. DOI: 10.18653/v1/2022.acl-long.527.

[CCP23] David Chiang, Peter Cholak, and Anand Pillay. “Tighter bounds on the expressivity of transformer encoders”. In: *Proceedings of the 40th International Conference on Machine Learning*. ICML’23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[Chi24] David Chiang. *Transformers in Uniform TC^0*. 2024.

[Cho+14] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Ed. by Dekai Wu et al. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012.

[Cho56] Noam Chomsky. “Three models for the description of language”. In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813.

[CM87] Stephen A Cook and Pierre McKenzie. “Problems complete for deterministic logarithmic space”. In: *Journal of Algorithms* 8.3 (1987), pp. 385–394. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(87)90018-6.

[CPW24] Lijie Chen, Binghui Peng, and Hongxun Wu. *Theoretical limitations of multi-layer Transformer*. 2024.

[Cra19] Jackson Crawford. *The Wanderer’s Havamal*. Hackett Publishing, 2019.

[CS59] N. Chomsky and M.P. Schützenberger. “The Algebraic Theory of Context-Free Languages**This work was supported in part by the U.S. Army Signal Corps, the Air Force Office of Scientific Research, and the Office of Naval Research; and in part by the National Science Foundation; and in part by a grant from the Commonwealth Fund.” In: *Computer Programming and Formal Systems*. Ed. by P. Braffort and D. Hirschberg. Vol. 26. Studies in Logic and the Foundations of Mathematics. Elsevier, 1959, pp. 118–161. DOI: https://doi.org/10.1016/S0049-237X(09)70104-1.

[Cyb89] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274.

[Dee+25] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025.

[Elm90] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: https://doi.org/10.1207/s15516709cog1402_1.

[Fin+22] Matthew Finlayson et al. “What Makes Instruction Learning Hard? An Investigation and a New Challenge in a Synthetic Environment”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg,

Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 414–426. DOI: 10.18653/v1/2022.emnlp-main.27.

[Gar+22] Shivam Garg et al. "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022.

[GD24] Albert Gu and Tri Dao. "Mamba: Linear-Time Sequence Modeling with Selective State Spaces". In: *First Conference on Language Modeling*. 2024.

[GGR22] Albert Gu, Karan Goel, and Christopher Re. "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations*. 2022.

[GHR91] Raymond Greenlaw, H James Hoover, and Walter L Ruzzo. *A compendium of problems complete for P*. Citeseer, 1991.

[Gla+24] Elliot Glazer et al. *FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI*. 2024.

[Göd31] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I". In: *Monatshefte für mathematik und physik* 38 (1931), pp. 173–198.

[Goy+24] Sachin Goyal et al. "Think before you speak: Training Language Models With Pause Tokens". In: *The Twelfth International Conference on Learning Representations*. 2024.

[Gra+25] Riccardo Grazzi et al. "Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[Gre73] Sheila A. Greibach. "The Hardest Context-Free Language". In: *SIAM Journal on Computing* 2.4 (1973), pp. 304–310. DOI: 10.1137/0202025.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. *Neural Turing Machines*. 2014.

[HAF22] Yiding Hao, Dana Angluin, and Robert Frank. "Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity". In: *Transactions of the Association for Computational Linguistics* 10 (2022). Ed. by Brian Roark and Ani Nenkova, pp. 800–810. DOI: 10.1162/tacl_a_00490.

[Hah+22] Michael Hahn et al. "A resource-rational model of human processing of recursive linguistic structure". In: *Proceedings of the National Academy of Sciences* 119.43 (2022), e2122602119. DOI: 10.1073/pnas.2122602119.

[Hah20] Michael Hahn. "Theoretical Limitations of Self-Attention in Neural Sequence Models". In: *Transactions of the Association for Computational Linguistics* 8 (Dec. 2020), pp. 156–171. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00306.

[Har54] Zellig S Harris. "Distributional structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[Har93] Juris Hartmanis. "Gödel, von Neumann and the P=? NP problem". In: *Current Trends in Theoretical Computer Science: Essays and Tutorials*. World Scientific, 1993, pp. 445–450.

[Hen+21] Dan Hendrycks et al. "Measuring Mathematical Problem Solving With the MATH Dataset". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021.

[Hes01] William Hesse. "Division Is In Uniform TC0". In: *Automata, Languages and Programming*. Ed. by Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 104–114. ISBN: 978-3-540-48224-6.

[HG23] Michael Hahn and Navin Goyal. *A Theory of Emergent In-Context Learning as Implicit Structure Induction*. 2023.

[HJF21] Michael Hahn, Dan Jurafsky, and Richard Futrell. "Sensitivity as a Complexity Measure for Sequence Classification Tasks". In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian Roark and Ani Nenkova, pp. 891–908. DOI: 10.1162/tacl_a_00403.

[HK98] Irene Heim and Angelika Kratzer. *Semantics in Generative Grammar*. Ed. by Angelika Kratzer. Malden, MA: Blackwell, 1998.

[HMU01] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. "Introduction to automata theory, languages, and computation". In: *ACM Sigact News* 32.1 (2001), pp. 60–65.

[HPV77] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. "On Time Versus Space". In: *J. ACM* 24 (1977), pp. 332–337.

[HR24] Michael Hahn and Mark Rofin. "Why are Sensitive Functions Hard for Transformers?" In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 14973–15008. DOI: 10.18653/v1/2024.acl-long.800.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[Hu+25] Michael Y. Hu et al. *Between Circuits and Chomsky: Pre-pretraining on Formal Languages Imparts Linguistic Biases*. 2025.

[Hua+25a] Wenyue Hua et al. "InductionBench: LLMs Fail in the Simplest Complexity Class". In: *Workshop on Reasoning and Planning for Large Language Models*. 2025.

[Hua+25b] Xinting Huang et al. “A Formal Framework for Understanding Length Generalization in Transformers”. In: *The Thirteenth International Conference on Learning Representations*. 2025.

[IL95] Neil Immerman and Susan Landau. “The complexity of iterated multiplication”. In: *Inf. Comput.* 116.1 (Jan. 1995), pp. 103–116. ISSN: 0890-5401. DOI: 10.1006/inco.1995.1007.

[Imm98] Neil Immerman. *Descriptive Complexity*. Springer Verlag, 1998.

[Jer+25] Selim Jerad et al. *Unique Hard Attention: A Tale of Two Sides*. 2025.

[Jim+24] Carlos E. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* 2024.

[JTX20] Ziwei Ji, Matus Telgarsky, and Ruicheng Xian. “Neural tangent kernels, transportation mappings, and universal approximation”. In: *International Conference on Learning Representations*. 2020.

[Kap+20] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020.

[Kat+20] Angelos Katharopoulos et al. “Transformers are RNNs: fast autoregressive transformers with linear attention”. In: *Proceedings of the 37th International Conference on Machine Learning*. ICML’20. JMLR.org, 2020.

[Kea98] Michael Kearns. “Efficient noise-tolerant learning from statistical queries”. In: *J. ACM* 45.6 (Nov. 1998), pp. 983–1006. ISSN: 0004-5411. DOI: 10.1145/293347.293351.

[KMR67] Kenneth Krohn, Richard Mateosian, and John Rhodes. “Methods of the algebraic theory of machines: I. Decomposition theorem for generalized machines; Properties preserved under series and parallel compositions of machines”. In: *Journal of Computer and System Sciences* 1.1 (1967), pp. 55–85. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(67)80007-2.

[KS23] Najoung Kim and Sebastian Schuster. “Entity Tracking in Language Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 3835–3855. DOI: 10.18653/v1/2023.acl-long.213.

[Kur64] Sige-Yuki Kuroda. “Classes of languages and linear-bounded automata”. In: *Information and Control* 7.2 (1964), pp. 207–223. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(64)90120-2.

[LC25] Jiaoda Li and Ryan Cotterell. *Characterizing the Expressivity of Transformer Language Models*. 2025.

[Lei+18] Tao Lei et al. “Simple Recurrent Units for Highly Parallelizable Recurrence”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 4470–4481. DOI: 10.18653/v1/D18-1477.

[Li+24a] Jiaoda Li et al. “What Do Language Models Learn in Context? The Structured Task Hypothesis.” In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 12365–12379. DOI: 10.18653/v1/2024.acl-long.669.

[Li+24b] Zhiyuan Li et al. “Chain of Thought Empowers Transformers to Solve Inherently Serial Problems”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[Liu+23a] Bingbin Liu et al. “Exposing Attention Glitches with Flip-Flop Language Modeling”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[Liu+23b] Bingbin Liu et al. “Transformers Learn Shortcuts to Automata”. In: *The Eleventh International Conference on Learning Representations*. 2023.

[MC89] David A. Mix Barrington and James Corbett. “On the relative complexity of some languages in NC1”. In: *Information Processing Letters* 32.5 (1989), pp. 251–256. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(89)90052-5.

[McC+18] Sam McCandlish et al. *An Empirical Model of Large-Batch Training*. 2018.

[Mer+21a] William Merrill et al. “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*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 1766–1781. DOI: 10.18653/v1/2021.emnlp-main.133.

[Mer+21b] William Merrill et al. “Provable Limitations of Acquiring Meaning from Ungrounded Form: What Will Future Language Models Understand?” In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian Roark and Ani Nenkova, pp. 1047–1060. DOI: 10.1162/tacl_a_00412.

[Mer+24] William Merrill et al. “Can You Learn Semantics Through Next-Word Prediction? The Case of Entailment”. In: *Findings of the Association for Computational Linguistics: ACL 2024*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 2752–2773. DOI: 10.18653/v1/2024.findings-acl.161.

[Mer19] William Merrill. “Sequential Neural Networks as Automata”. In: *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*. Ed. by Jason Eisner et al. Florence: Association for Computational Linguistics, Aug. 2019, pp. 1–13. DOI: 10.18653/v1/W19-3901.

[MIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. "On uniformity within NC1". In: *Journal of Computer and System Sciences* 41.3 (1990), pp. 274–306. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(90)90022-D.

[MP88] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 15–27. ISBN: 0262010976.

[MPS24] William Merrill, Jackson Petty, and Ashish Sabharwal. "The Illusion of State in State-Space Models". In: *Forty-first International Conference on Machine Learning*. 2024.

[MS23a] William Merrill and Ashish Sabharwal. "A Logic for Expressing Log-Precision Transformers". In: *Thirty-seventh Conference on Neural Information Processing Systems*. Dec. 2023.

[MS23b] William Merrill and Ashish Sabharwal. "The Parallelism Tradeoff: Limitations of Log-Precision Transformers". In: *Transactions of the Association for Computational Linguistics* 11 (June 2023), pp. 531–545. DOI: 10.1162/tacl_a_00562.

[MS24a] William Merrill and Ashish Sabharwal. *A Little Depth Goes a Long Way: The Expressive Power of Log-Depth Transformers*. Dec. 2024.

[MS24b] William Merrill and Ashish Sabharwal. "The Expressive Power of Transformers with Chain of Thought". In: *The Twelfth International Conference on Learning Representations*. 2024.

[MS25] William Merrill and Ashish Sabharwal. *Exact Expressive Power of Transformers with Padding*. 2025.

[MSS22] William Merrill, Ashish Sabharwal, and Noah A. Smith. "Saturated Transformers are Constant-Depth Threshold Circuits". In: *Transactions of the Association for Computational Linguistics* 10 (Aug. 2022). Ed. by Brian Roark and Ani Nenkova, pp. 843–856. DOI: 10.1162/tacl_a_00493.

[MWL22] William Merrill, Alex Warstadt, and Tal Linzen. “Entailment Semantics Can Be Extracted from an Ideal Language Model”. In: *Proceedings of the 26th Conference on Computational Natural Language Learning (CoNLL)*. Ed. by Antske Fokkens and Vivek Srikumar. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, Dec. 2022, pp. 176–193. DOI: 10.18653/v1/2022.conll-1.13.

[Nou+22] Rick Nouwen et al. “Dynamic Semantics”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Fall 2022. Metaphysics Research Lab, Stanford University, 2022.

[Nye+22] Maxwell Nye et al. *Show Your Work: Scratchpads for Intermediate Computation with Language Models*. 2022.

[Oli+25] Michael de Oliveira et al. “Unconditional advantage of noisy qudit quantum circuits over biased threshold circuits in constant depth”. In: *Nature Communications* (Mar. 2025). DOI: 10.1038/s41467-025-58545-4.

[Ols+22] Catherine Olsson et al. *In-context Learning and Induction Heads*. 2022.

[Ope25] OpenAI. *OpenAI o3 and o4-mini System Card*. Accessed: 2025-04-23. 2025.

[Pen+25] Bo Peng et al. “RWKV-7 "Goose" with Expressive Dynamic State Evolution”. In: *CoRR* abs/2503.14456 (Mar. 2025).

[PMB19] Jorge Pérez, Javier Marinković, and Pablo Barceló. “On the Turing Completeness of Modern Neural Network Architectures”. In: *International Conference on Learning Representations*. 2019.

[PMB24] Jacob Pfau, William Merrill, and Samuel R. Bowman. “Let’s Think Dot by Dot: Hidden computation in transformer language models”. In: *First Conference on Language Modeling*. 2024.

[Qiu+24] Linlu Qiu et al. “Phenomenal Yet Puzzling: Testing Inductive Reasoning Capabilities of Language Models with Hypothesis Refinement”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[Qiu+25] Linlu Qiu et al. *Bayesian Teaching Enables Probabilistic Reasoning in Large Language Models*. 2025.

[Qwe+25] Qwen et al. *Qwen2.5 Technical Report*. 2025.

[Raa22] Panu Raatikainen. “Gödel’s Incompleteness Theorems”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2022. Metaphysics Research Lab, Stanford University, 2022.

[Rei08] Omer Reingold. “Undirected connectivity in log-space”. In: *J. ACM* 55.4 (Sept. 2008). ISSN: 0004-5411. DOI: 10.1145/1391289.1391291.

[RT92] John H. Reif and Stephen R. Tate. “On Threshold Circuits and Polynomial Computation”. In: *SIAM Journal on Computing* 21.5 (1992), pp. 896–908. DOI: 10.1137/0221053.

[Ruz80] Walter L. Ruzzo. “Tree-size bounded alternation”. In: *Journal of Computer and System Sciences* 21.2 (1980), pp. 218–235. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(80)90036-7.

[Ruz81] Walter L. Ruzzo. “On uniform circuit complexity”. In: *Journal of Computer and System Sciences* 22.3 (1981), pp. 365–383. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(81)90038-6.

[Shi85] Stuart M. Shieber. “Evidence against the context-freeness of natural language”. In: *Linguistics and Philosophy* 8 (1985), pp. 333–343.

[Sie+25] Julien Siems et al. *DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products*. 2025.

[SS95] H.T. Siegelmann and E.D. Sontag. “On the Computational Power of Neural Nets”. In: *Journal of Computer and System Sciences* 50.1 (1995), pp. 132–150. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1995.1013.

[Str+24] Lena Strobl et al. “What Formal Languages Can Transformers Express? A Survey”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 543–561. DOI: 10.1162/tacl_a_00663.

[Vaf+24] Keyon Vafa et al. “Evaluating the World Model Implicit in a Generative Model”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[Vas+17] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[Vas+25] Bhavya Vasudeva et al. “Transformers Learn Low Sensitivity Functions: Investigations and Implications”. In: *The Thirteenth International Conference on Learning Representations*. 2025.

[Ven91] H. Venkateswaran. “Properties that characterize LOGCFL”. In: *Journal of Computer and System Sciences* 43.2 (1991), pp. 380–404. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(91)90020-6.

[Wei+22a] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022.

[Wei+22b] Jason Wei et al. “Emergent Abilities of Large Language Models”. In: *Transactions on Machine Learning Research* (2022). Survey Certification. ISSN: 2835-8856.

[WGY21] Gail Weiss, Yoav Goldberg, and Eran Yahav. *Thinking Like Transformers*. 2021.

[Wig92] Avi Wigderson. “The complexity of graph connectivity”. In: *Mathematical Foundations of Computer Science 1992*. Ed. by Ivan M. Havel and Václav Koubek. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 112–132. ISBN: 978-3-540-47291-9.

[Wil25] R. Ryan Williams. *Simulating Time With Square-Root Space*. 2025.

[Wil40] C. B. Williams. “A Note on the Statistical Analysis of Sentence-Length as a Criterion of Literary Style”. In: *Biometrika* 31.3/4 (1940), pp. 356–361. ISSN: 00063444.

[Wu+23] Zhaofeng Wu et al. “Transparency Helps Reveal When Language Models Learn Meaning”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 617–634. DOI: 10.1162/tacl_a_00565.

[Xie+22] Sang Michael Xie et al. “An Explanation of In-context Learning as Implicit Bayesian Inference”. In: *International Conference on Learning Representations*. 2022.

[Yan+24a] Andy Yang et al. *Simulating Hard Attention Using Soft Attention*. 2024.

[Yan+24b] John Yang et al. *SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering*. 2024.

[Yan+24c] Songlin Yang et al. “Parallelizing Linear Transformers with the Delta Rule over Sequence Length”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[Yan+25] Songlin Yang et al. *PaTH Attention: Position Encoding via Accumulating Householder Transformations*. 2025.

[Yao+21] Shunyu Yao et al. “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)*. Ed. by Chengqing Zong et al. Online: Association

for Computational Linguistics, Aug. 2021, pp. 3770–3785. DOI: 10.18653/v1/2021.acl-long.292.

[YC24] Andy Yang and David Chiang. "Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers". In: *First Conference on Language Modeling*. 2024.

[YCA24] Andy Yang, David Chiang, and Dana Angluin. "Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[Zha+25] Hanlin Zhang et al. *How Does Critical Batch Size Scale in Pre-training?* 2025.