The Expressive Power of Transformers with Chain of Thought

William Merrill
New York University
willm@nyu.edu

Ashish Sabharwal
Allen Institute for AI
ashishs@allenai.org

Abstract

Recent theoretical work has identified surprisingly simple reasoning problems, such as checking if two nodes in a graph are connected or simulating finite-state machines, that are provably unsolvable by standard transformers that answer immediately after reading their input. However, in practice, transformers' reasoning can be improved by allowing them to use a "chain of thought" or "scratchpad", i.e., generate and condition on a sequence of intermediate tokens before answering. Motivated by this, we ask: *Does such intermediate generation fundamentally extend the computational power of a decoder-only transformer?* We show that the answer is *yes*, but the amount of increase depends crucially on the amount of intermediate generation. For instance, we find that transformer decoders with a logarithmic number of decoding steps (w.r.t. the input length) push the limits of standard transformers only slightly, while a linear number of decoding steps adds a clear new ability (under standard complexity conjectures): recognizing all regular languages. Our results also imply that linear steps keep transformer decoders within context-sensitive languages, and polynomial steps make them recognize exactly the class of polynomial-time solvable problems—the first exact characterization of a type of transformers in terms of standard complexity classes. Together, our results provide a nuanced framework for understanding how the length of a transformer's chain of thought or scratchpad impacts its reasoning power.

1 Introduction

Recent theoretical results (Merrill & Sabharwal, 2023b,a; Merrill et al., 2022; Liu et al., 2023; Chiang et al., 2023; Hao et al., 2022) have unveiled surprising limits on realistic formal models of transformers. They have shown that standard transformers, even with ideal parameters, cannot perfectly solve many sequential reasoning problems at scale, such as simulating finite-state machines, deciding whether nodes in a graph are connected, or solving matrix equalities. Intuitively, the transformer computation graph lacks recurrent connections, which is required to solve these sequential reasoning problems. Empirically, cutting-edge transformer language models such as ChatGPT and GPT-4 struggle on reasoning problems inspired by these results (Zhang et al., 2023), and the reasoning performance of GPT-4 has been shown to be negatively correlated with the depth of the problem's computation graph (Dziri et al., 2023). These results show that certain kinds of sequential reasoning pose a challenge for the transformer architecture and motivate extensions to better handle sequential reasoning.

One method that has been empirically successful for improving sequential reasoning with transformers is adding a so-called *chain of thought* (Wei et al., 2022) or *scratchpad* (Nye et al., 2021). These methods allow the transformer to output a sequence of *intermediate tokens* before answering, rather than answering right away after reading the input. Intuitively, such methods could unlock greater expressive power on sequential reasoning problems because the model can use each intermediate token as a kind of recurrent state. Feng et al. (2023) recently showed how chain of thought lets

Mathematics of Modern Machine Learning Workshop at NeurIPS 2023.

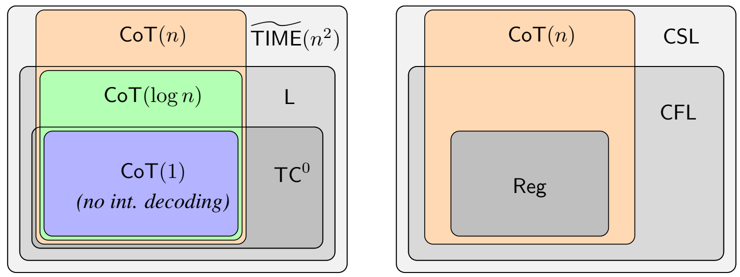

Figure 1: Summary of results: transformers with intermediate generation against various classes of formal languages. A logarithmic number of chain-of-thought steps remains in log-space (L). A linear number of steps adds more power, enabling recognizing all regular languages (Reg), but is contained within context-sensitive languages (CSL). We assume context-free languages (CFL) require $\tilde{\omega}(n^2)$ time to recognize. Some regions with area in the plot are not known to be non-empty.

transformers solve a specific modular arithmetic problem that they likely cannot solve without one. Yet there is no general characterization of the class of problems transformers can solve with chain of thought. Thus, the extent to which chain of thought alleviates transformers' weaknesses is unclear, as well as the number of chain of thought steps required to gain reasoning power.

In this work, we address these open questions by characterizing the reasoning power of transformer decoders that can take *intermediate steps* before generating an answer and comparing them against transformers without intermediate steps. A transformer with a chain of thought constitutes a special case of a transformer decoder with intermediate steps. Our fine-grained results give upper and lower bounds on transformers' power depending on $t(n)$: the number of allowed intermediate steps as a function of the input size n. We focus mainly on understanding three regimes: logarithmic steps (when $t(n) = \Theta(\log n)$), linear steps (when $t(n) = \Theta(n)$), and polynomial steps.

Prior Work: No Intermediate Steps. Recent work has shown transformer decoders without any intermediate steps can only solve problems that lie inside the fairly small circuit complexity class TC^0 (Merrill & Sabharwal, 2023b) and related logical classes (Merrill & Sabharwal, 2023a; Chiang et al., 2023). This implies basic transformers are far from Turing-complete: they cannot even solve problems complete for classes larger than TC^0 such as simulating automata (NC^1-complete), deciding directed graph connectivity (NL-complete), or solving linear equalities (P-complete).[1]

Logarithmic Steps. With a *logarithmic* number of intermediate steps, we show that the upper bound for transformers expands slightly from TC^0 to L. This means transformers with a logarithmic number of intermediate steps might gain power, but they still cannot solve NL-complete problems like directed graph connectivity or P-complete problems like solving linear equalities.[2]

Linear Steps. A *linear* number of intermediate steps allows transformers to simulate automata (NC^1-complete), which transformers without intermediate steps cannot do unless $\mathsf{TC}^0 = \mathsf{NC}^1$.

Polynomial Steps. With a *polynomial* number of decoding steps, we show that transformers are precisely equivalent to the class P. This, to the best of our knowledge, is the first equivalence (not only a one-sided bound) between a class of transformers and a standard complexity class.

Together, our results provide a framework for understanding how the length of a transformer's chain of thought or scratchpad affects its reasoning power: a logarithmic chain does not add much relative to no intermediate decoding, while a linear chain affords more power on inherently sequential problems.

2 Results: The Power of Transformers with Intermediate Decoding

We next discuss our findings in more detail, both on the capabilities and the limits of transformer decoders with intermediate generation. Due to limited space, we defer all formal definitions, theorem statements, and proofs to the appendix, focusing here instead on the key results and one novel

[1] Assuming NC^1, NL, and P do not collapse to TC^0, respectively.

[2] Assuming NL and P do not collapse to L, respectively.

technical tool (namely, *layer-norm hash*) used in our proofs. The formal models of transformer decoders and of multitape Turing machines underlying our results are discussed in Appendix A, lower bounds proved in Appendix B, and upper bounds proved in Appendix C.

Let $\mathsf{TIME}(t(n))$ be the class of languages L for which there exists a Turing machine that runs in time $\mathrm{O}(t(n))$ and accepts L.[3] Let $\widetilde{\mathsf{TIME}}(t(n))$ be the class of problems in $\mathsf{TIME}(t(n) \log^k n)$ for some k, which is meaningful for $t(n) \geq n$. Let $\mathsf{SPACE}(s(n))$ be the class of languages L for which there exists a Turing machine with tape size bounded by $\mathrm{O}(s(n))$ that accepts L. We show the following relationship between transformers with $t(n)$ steps and standard time/space complexity classes:

$$\mathsf{TIME}(t(n)) \subseteq \mathsf{CoT}(t(n)) \qquad \text{(Corollary 2.1)} \qquad (1)$$

$$\mathsf{CoT}(t(n)) \subseteq \mathsf{SPACE}(t(n) + \log n) \qquad \text{(Theorem 4)} \qquad (2)$$

$$\mathsf{CoT}(t(n)) \subseteq \widetilde{\mathsf{TIME}}(t(n)^2 + n^2) \qquad \text{(Theorem 3)} \qquad (3)$$

Both our time lower bound and space upper bound are fairly tight: improving either by a factor larger than $\log t(n)$ would result in a fundamental complexity theory advance (Hopcroft et al., 1977).

Capabilities of Transformers with CoT. Equation (1) implies that transformer decoders with $\Theta(n)$ steps can simulate real-time models of computation like automata or counter machines (Merrill, 2020). Under standard assumptions in complexity theory, transformers with no decoding steps *cannot* simulate all automata (Merrill & Sabharwal, 2023b; Merrill, 2023; Liu et al., 2023). Thus, a linear number of decoding steps makes transformers strictly more powerful. Similarly, Equation (1) implies transformers with a *quadratic* number of steps can express a linear-time algorithm (for a random access Turing machine) to solve directed graph connectivity (Wigderson, 1992), again a problem known to be beyond standard transformers. With *polynomial* decoding steps, transformers can solve linear equalities, Horn-clause satisfiability, and universal context-free recognition, all of which are P-complete and thus inexpressible by standard transformers (Merrill & Sabharwal, 2023b).

Our proof shows that transformer decoders can simulate t Turing machine steps with t intermediate steps. Similar prior results have assumed external memory (Schuurmans, 2023) or an encoder-decoder model with nonstandard-positional decodings (Pérez et al., 2021). Our construction adapts these ideas to work for a decoder-only model *without* external memory or extra positional encodings.[4]

Limitations of Transformers with CoT. Equations (2) and (3) establish two upper bounds on transformer decoders with $t(n)$ intermediate steps that depend on both $t(n)$ and n. We turn to the implications of this general result in different regimes for $t(n)$:

1. **Log Steps**: Transformer decoders with $\mathrm{O}(\log n)$ intermediate steps can only recognize languages in $\mathsf{L} = \mathsf{SPACE}(\log n)$. This implies that transformers with $\mathrm{O}(\log n)$ intermediate steps cannot solve NL- or P-complete problems[2] like directed graph connectivity, just like transformers with no intermediate decoding (Merrill & Sabharwal, 2023b).
2. **Linear Steps**: Transformer decoders with $\mathrm{O}(n)$ intermediate steps can only recognize languages that are in both $\widetilde{\mathsf{TIME}}(n^2)$ and $\mathsf{SPACE}(n)$. Since $\mathsf{SPACE}(n)$ falls within the context-sensitive languages (Kuroda, 1964), transformers with linear steps can recognize at most context-sensitive languages. Alongside our lower bound, this shows transformer decoders with $\Theta(n)$ steps fall somewhere between regular and context-sensitive languages in the Chomsky hierarchy. Further, transformers with $\mathrm{O}(n)$ steps cannot recognize all context-free languages unless context-free languages can be parsed in soft quadratic time.[5]
3. **Polynomial Steps**: If $t(n) = \mathrm{O}(n^c)$ for some c, we get an upper bound of $\mathsf{P} = \bigcup_{c=1}^{\infty} \mathsf{TIME}(n^c)$. Combined with our lower bound, this shows that transformer decoders with a polynomial number of steps recognize *exactly* the class P. Thus, a polynomial number of steps turns transformers into strong reasoners, though running a polynomial number of forward passes with a large transformer is likely intractable in practice.

[3]As we will define later in the appendix, this is a non-random-access multitape Turing machine.

[4]Our construction (Theorem 2) can be easily modified to work with an encoder-decoder model as well.

[5]The best known algorithms for context-free recognition run in time $\mathrm{O}(n^\omega)$, where ω is the matrix multiplication constant (Valiant, 1975); best lower bounds for context-free parsing are sub-quadratic (Lee, 2002).

Together, these results show that intermediate generation like chain of thought or scratchpad can add reasoning power to transformers and that the number of steps matters as a computational resource akin to time or space. Some of the limitations identified in prior work (Merrill & Sabharwal, 2023b; Chiang et al., 2023, etc.) can be overcome with a linear or quadratic number of steps, and a polynomial number of steps covers all problems in P. On the other hand, we have not identified any concrete reasoning problem where a logarithmic number of steps would help. These results provide a unified understanding of the power of transformer decoders across decoding lengths and problems.

2.1 The Layer-Norm Hash

The key idea behind our more general lower bound construction is the **layer-norm hash** (discussed in detail in Appendix B.1): a simple module for effectively storing memory in decoder-only transformers. We believe the layer-norm hash could be broadly useful for building algorithms in transformers. For example, Yao et al. (2021) used a related idea to construct transformers that recognize bounded-depth Dyck languages, although in a more ad hoc way.

The layer-norm hash is a mechanism that enables retrieval across different columns in the transformer based on query-key matching of numerical values. Exact-match retrieval is trivial when the query q_i and keys $k_1, \ldots k_i$ are items in a finite set: just one-hot encode q_i and k_j and the inner product will be maximized when q_i and k_j match. This, however, does not work when the keys and values are counts produced by uniform attention, which many transformer algorithms use (Weiss et al., 2021). In this case, the key is a fraction q_i/i and the queries are fractions k_j/j with *diferent* denominators.

The layer-norm hash helps by representing q_i/i and k_j/j such that hard attention retrieves the value j where $q_i = k_j$. The idea is to use layer-norm to project q_i and k_j to vectors ϕ_{q_i} and ϕ_{k_i} on the unit sphere satisfying this property. Let $\mathsf{layer\_norm}(\mathbf{x}) = \frac{\mathbf{x}'}{\|\mathbf{x}'\|}$, where $\mathbf{x}' = \mathbf{x} - \bar{x}$. Then, the layer-norm hash of $x \in \mathbb{R}$ at position $i \in \mathbb{N}$ is a unit vector in $\mathbb{R}^4$ defined as:

$$\phi(x/i, 1/i) = \mathsf{layer\_norm}\left(\frac{x}{i}, \frac{1}{i}, -\frac{x}{i}, -\frac{1}{i}\right).$$

A key feature of this representation is that it is invariant w.r.t. i in the sense that $\phi(x/i, 1/i) \triangleq \phi_x$ is only a function of x, independent of i. Further, the inner products of these representations of two scalars q, k is maximized to 1 if and only if $q = k$. We can thus look up key q_i/i in a sequence of keys $k_1/1, \ldots, k_i/i$ by attending with query $\phi(q_i/i, 1/i) = \phi_{q_i}$ at position i and key $\phi(k_j/j, 1/j) = \phi_{k_j}$ at each position j. This retrieves the value at j such that $q_i = k_j$.

3 Discussion

We have shown that intermediate decoding steps extend the formal power of transformers well beyond previously known upper bounds, such as TC^0 circuits and $\mathsf{FO}(\mathsf{M})$ logic, on transformers without intermediate decoding. Further, the amount of additional power is closely related to the number of decoding steps. In particular, transformers with a linear number of decoding steps have the capacity to recognize regular languages, but cannot recognize languages beyond context-sensitive. With a log number of decoding steps, such transformers can only recognize languages in L, which is a complexity class relatively close to TC^0. Thus, it appears that a linear number of intermediate decoding steps may be required to overcome the limitations of transformers on many sequential reasoning problems of interest. In future work, it may be possible to derive a strict separation between transformers with a log and a linear number of decoding steps and show that certain problems that currently have a quadratic bound can in fact be solved with a roughly linear number of steps.

Here we have focused on lower and upper bounds on expressive power, rather than analyzing learnability. Whereas our upper bounds directly reveal limitations on what transformers with intermediate generation can learn, one caveat is that our lower bounds *do not* directly imply transformers can learn to use intermediate steps effectively. It would be interesting to formally investigate transformers with CoT from a learning-theoretic lens, possibly along the lines of Malach (2023), and how different kinds of fine-tuning, such as reinforcement learning, might improve a model's ability to use the power of its chain of thought.

References

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. *ICML*, 2023.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *NeurIPS*, 2023.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective, 2023.

Sophie Hao, Dana Angluin, and Roberta Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *TACL*, 10:800–810, 2022.

John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space. *J. ACM*, 24: 332–337, 1977.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *ACM Sigact News*, 32(1):60–65, 2001.

S-Y Kuroda. Classes of languages and linear-bounded automata. *Information and control*, 7(2): 207–223, 1964.

Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, Jan 2002.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *ICLR*, 2023.

Eran Malach. Auto-regressive next-token predictors are universal learners, 2023.

William Merrill. On the linguistic capacity of real-time counter automata. *ArXiv*, abs/2004.06866, 2020.

William Merrill. Formal languages and neural models for learning on sequences. In François Coste, Faissal Ouardi, and Guillaume Rabusseau (eds.), *ICGI*, volume 217 of *PMLR*, Jul 2023.

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. In *NeurIPS*, 2023a.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *TACL*, 11:531–545, 2023b.

William Merrill, Vivek Ramanujan, Yoav Goldberg, Roy Schwartz, and Noah A. Smith. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. In *EMNLP*, 2021.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *TACL*, 10:843–856, 2022.

Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *ArXiv*, abs/2112.00114, 2021.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing complete. *JMLR*, 22(1), January 2021.

Dale Schuurmans. Memory augmented large language models are computationally universal. *ArXiv*, abs/2301.04589, 2023.

Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *NeurIPS*, 2022.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision RNNs for language recognition. In *ACL*, July 2018.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *ICML*, 2021.

Avi Wigderson. The complexity of graph connectivity. In *International Symposium on Mathematical Foundations of Computer Science*, pp. 112–132. Springer, 1992.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *ICML*, 2020.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *ACL*, 2021.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How language model hallucinations can snowball. *ArXiv*, abs/2305.13534, 2023.

A Preliminaries

We study the power of decoder-only transformers that can generate intermediate tokens between reading the input and generating an answer. On input $x \in \Sigma^n$, the transformer consumes tokens $x_1, \ldots, x_n$ for the first n steps, and then, for $t(n)$ *intermediate steps*, consumes the token generated by the previous step. At each step, the transformer can attend over all previous hidden states. This standard method of generating text from a decoder-only model can be described formally as follows. Let Σ be a finite alphabet and $f : \Sigma^* \to \Sigma$ be a function mapping a prefix to a next token (parameterized by a transformer). Let $\cdot$ be concatenation. We define the k-step extension of f as

$$f^0(x) = x, \qquad f^{k+1}(x) = f^k(x) \cdot f(f^k(x)).$$

We say we have run f on x with $t(n)$ (additional) decoding steps if we compute the function $f^{t(|x|)}(x)$. We consider f with $t(n)$ steps to recognize the language of strings such that $f^{t(|x|)}(x) = 1$, where $1 \in \Sigma$ is a special "accept" symbol. We denote by $\mathsf{CoT}(t(n))$ the set of languages that are recognized by $t(n)$ decoding steps for some transformer f.

A.1 Transformers

A transformer is a neural network parameterizing a function $\Sigma^* \to \Sigma$. Let $\mathbb{D}_p$ be the datatype of p-precision floats and define p-truncated addition ($+, \sum$), multiplication ($\cdot$), and division ($/$) over $\mathbb{D}_p$ as in Merrill & Sabharwal (2023b). We now define the high-level structure of the transformer in terms of its core components, with the details of those components in Appendix D.

Definition 1 (Merrill & Sabharwal 2023a)**.** A p-precision decoder-only transformer with h heads, d layers, model dimension m (divisible by h), and feedforward width w is specified by:

1. An embedding function $\phi : \Sigma \times \mathbb{N} \to \mathbb{D}_p^m$ whose form is defined in Appendix D.1;
2. For each $1 \le \ell \le d$ and $1 \le k \le h$, a head similarity function $s_k^\ell : \mathbb{D}_p^m \times \mathbb{D}_p^m \to \mathbb{D}_p$ whose form is defined in Appendix D.2;
3. For each $1 \le \ell \le d$ and $1 \le k \le h$, a head value function $v_k^\ell : \mathbb{D}_p^m \to \mathbb{D}_p^{m/h}$ whose form is defined in Appendix D.2;
4. For each $1 \le \ell \le d$, an activation function $f^\ell : (\mathbb{D}_p^{m/h})^h \times \mathbb{D}_p^m \to \mathbb{D}_p^m$ whose form is defined in Appendix D.3 and implicitly uses the feedforward dimension w;
5. An output function $\gamma : \mathbb{D}_p^m \to \Sigma$ parameterized as a linear transformation.

Definition 2. We define one decoding step $\Sigma^n \to \Sigma$ with a decoder-only transformer as follows:

1. Embeddings: For $1 \leq i \leq n$, $\mathbf{h}_i^0 = \phi(x_i, i)$.
2. Multihead Self Attention: For each layer $1 \leq \ell \leq d$, we compute h attention heads:

$$\mathbf{a}_{i,k}^{\ell} = \sum_{j=1}^{i} \frac{s_k^{\ell}(\mathbf{h}_i^{\ell-1}, \mathbf{h}_j^{\ell-1})}{Z_{i,k}^{\ell}} \cdot v_k^{\ell}(\mathbf{h}_j^{\ell-1}), \quad \text{where } Z_{i,k}^{\ell} = \sum_{j=1}^{i} s_k^{\ell}(\mathbf{h}_i^{\ell-1}, \mathbf{h}_j^{\ell-1}).$$

3. Activation Block: For $1 \leq \ell \leq d$, activation block ℓ maps the head outputs to $\mathbf{h}^{\ell}$:

$$\mathbf{h}_i^{\ell} = f^{\ell}(\mathbf{a}_{i,1}^{\ell}, \ldots, \mathbf{a}_{i,h}^{\ell}, \mathbf{h}_i^{\ell-1}).$$

4. Classifier Head: The transformer output is $\gamma(\mathbf{h}_n^d)$.

Transformer Precision. We consider log-precision transformers (Merrill & Sabharwal, 2023b), i.e., we allow the transformer at most $c \log m$ on mth decoding steps. As a transformer with intermediate generation runs for n input steps and $t(n)$ intermediate decoding steps, this means we have precision at most $c \log(n + t(n))$. Log precision is natural because it gives the the transformer just enough precision to represent indexes and sums across different positions, and it has then been analyzed in prior work (Pérez et al., 2021; Merrill & Sabharwal, 2023b,a).

Saturation. A saturated transformer (Merrill et al., 2021) is an idealized transformer with simple attention: all attention scores are either 0 or $1/v$ for some v. Crucially, saturated attention can express full uniform attention ($1/n$ over n positions) or hard attention (full weight on one position). Our upper bounds do not require saturated attention, but, following common practice (Pérez et al., 2021; Merrill & Sabharwal, 2023b) we construct all our lower bounds using saturated attention. This shows that the constructions can be implemented without complicated attention patterns.

A.2 Automata

A deterministic finite-state automaton is a tuple $\langle \Sigma, Q, q_0, \delta, F \rangle$ where:

1. Σ is a finite input vocabulary
2. Q is a finite set of states containing initial state q_0
3. δ is a transition function $Q \times \Sigma \to Q$
4. $F \subseteq Q$ is a set of final states

We define computation with an automaton on input string $\sigma \in \Sigma^*$ as follows. An automaton configuration is simply a current finite state $q \in Q$. We start with the initial state q_0. We process a string $\sigma \in \Sigma^n$ one token at a time, computing the next state $q_i = \delta(q_{i-1}, \sigma_i)$ recurrently for $1 \leq i \leq n$. We say that an automaton *accepts* a string if $q_n \in F$ and that it *rejects* it otherwise. The formal language that the automaton recognizes is the set of strings that it accepts.

A.3 Turing Machines

Adapting the notation of Hopcroft et al. (2001), 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 $(Q \setminus F) \times \Gamma^{k+2} \to Q \times \Gamma^{k+1} \times \{\pm 1\}$
7. $F \subseteq Q$ is a set of final states

We define computation with a Turing machine on input string $\sigma \in \Sigma^*$ as follows. 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:

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

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 $q_i \in F$, we say the Turing machine halts and take as its output the string of tokens on the output tape from the current head position on the left up to (but not including) the first b to its right. This shows how Turing machines parameterize functions $\Sigma^* \to \Sigma^*$. A Turing machine can also be viewed as a language recognizer if we set $\Sigma = \{0, 1\}$ and check whether the first token of the output string is 0 or 1, constituting a language membership decision.

B Lower Bounds for Transformer Decoders

Prior work (Merrill & Sabharwal, 2023a) has established strong upper bounds on the reasoning problems transformers can recognize. Specifically, accepting standard conjectures in complexity, transformers without intermediate decoding cannot recognize all regular languages.

In this section, we show that some of these shortcomings can be addressed with a suitable number of intermediate decoding steps. Specifically, a linear number of steps is sufficient to simulate an automaton. We also show how this construction can be extended so that a transformer with $t(n)$ intermediate decoding steps can simulate a Turing machine for $t(n)$ steps.

B.1 Introducing Layer-Norm Hash

We first introduce a useful building block for our results that we call the *layer-norm hash*. The layer-norm hash is a mechanism that enables retrieval across different columns in the transformer based on query-key matching of numerical values. Exact-match retrieval is trivial when the query q_i and keys $k_1, \dots k_i$ are items in a finite set: just one-hot encode q_i and k_j and the inner product will be maximized when q_i and k_j match. This does not work, though, when the keys and values are counts produced by uniform attention, which many transformer algorithms use (Weiss et al., 2021). In this case, the key is a fraction q_i/i and the queries are fractions k_j/j with *diferent* denominators.

The layer-norm hash helps by representing q_i/i and k_j/j such that hard attention retrieves the value j where $q_i = k_j$. The idea is to use layer-norm to project q_i and k_j to vectors ϕ_{q_i} and ϕ_{k_i} on the unit sphere satisfying this property. Let $\mathsf{layer\_norm}(\mathbf{x}) = \frac{\mathbf{x}'}{\|\mathbf{x}'\|}$, where $\mathbf{x}' = \mathbf{x} - \bar{x}$.

Definition 3 (Layer-norm hash)**.** Given a scalar $x \in \mathbb{R}$, its layer-norm hash at position $i \in \mathbb{N}$ is

$$\phi(x/i, 1/i) = \mathsf{layer\_norm}\left(\frac{x}{i}, \frac{1}{i}, -\frac{x}{i}, -\frac{1}{i}\right).$$

This is a unit vector in $\mathbb{R}^4$. A key feature of this representation is that it is invariant w.r.t. i in the sense that $\phi(x/i, 1/i)$ is only a function of x, independent of i. Let $\phi_x \triangleq \phi(x, 1)$:

Lemma 1 (Proof in Appendix E)**.** *For any $x \in \mathbb{R}$ and $i \in \mathbb{N}$, $\phi(x/i, 1/i) = \phi_x$.*

Further, the inner products of these representations of two scalars q, k checks equality of q and k:

Lemma 2 (Proof in Appendix E)**.** *For any $q, k \in \mathbb{R}$, $\phi_q \cdot \phi_k = 1$ if and only if $q = k$.*

We can look up key q_i/i in a sequence of keys $k_1/1, \dots, k_i/i$ by attending with query $\phi(q_i/i, 1/i)$ at position i and key $\phi(k_j/j, 1/j)$ at each j. By Lemmas 1 and 2 this retrieves the value at j such that $q_i = k_j$. The layer-norm hash can also be used to directly compare two values q_i, k_j without removing the denominator by computing $\phi(q_i, 1)$ and $\phi(k_j, 1)$.

The first use of the layer-norm hash requires having $1/i$ at position i. Lemma 3 shows that a transformer decoder can compute $1/i$ at each position i assuming start-separable positional encodings:

Definition 4. A positional encoding π is **start-separable** if there exists a transformer decoder that maps the position embedding sequence to $\langle 1, 0, \dots, 0 \rangle$ (i.e., tags whether each token is the first).

The initial token is always separable if there is a special `bos` token. Similarly, if the position embedding of 1 is unique, then the initial token is separable. Since one of these conditions will likely be met in practice, we believe any practical transformer variant should be start-separable.[6] By Lemma 3 can effectively "reduce" these other positional encodings to $1/i$ encodings: any construction with $1/i$ positional encodings can be adapted to work for any start-separable encoding.

Lemma 3 (Proof in Appendix F)**.** *For any language L, if some decoder-only saturated transformer with $1/i$ positional encodings recognizes L, then, for any start-separable positional encoding π, some decoder-only saturated transformer with π positional encodings recognizes L.*

B.2 Simulating Automata

We can use the layer-norm hash to simulate models of computation like automata or Turing machines with intermediate-generation transformers. To warm up, we first show how to use the layer-norm hash to simulate an automaton (i.e., recognize a regular language) and then extend it in Appendix B.3 to show how a transformer can simulate a Turing machine for a bounded number of steps.

Theorem 1. *Let L be a regular language. For any start-separable positional encoding, there is a decoder-only saturated transformer that recognizes whether $x \in L$ with $|x| + 1$ decoding steps.*

Proof. Without loss of generality, we construct a transformer with $1/i$ positional encodings (cf. Lemma 3). The idea is to simulate one step of the automaton with one transformer decoding step (after first reading n input tokens). At decoding step i, we will output a token q_i encoding the state transitioned to. After printing the final state q_n, we use one additional step to output 1 iff $q_n \in F$.

The base case of $i = 0$ corresponds to the final input token, at which we output the initial state q_0.

In the inductive case, we know that the sequence of intermediate tokens is $q_0, \ldots, q_i$. Our goal is to compute and output $q_{i+1} = \delta(q_i, \sigma_i)$, which first involves computing δ's two arguments. We already have q_i as the input to the current column of the transformer. We use hard attention to retrieve the current input symbol σ_{i+1}. We already have $1/j$ stored at each input position j. Decoding step $i + 1$ corresponds to the $(n + i + 1)$st input to the transformer. We attend uniformly over the left context with a value of 1 at decoding steps and 0 at input steps, yielding $1/(i + 1)$. We attend with query $\phi(1/(i + 1), 1)$, keys $\phi(1/j, 1)$, and values σ_j. By Lemma 2, attention is maximized at $j = i + 1$, so this head retrieves σ_{i+1}. At this point, we have the current finite state q_i and current input token σ_{i+1}. We conclude by using a feedforward network to output $q_{i+1} = \delta(q_i, \sigma_{i+1})$. □

Theorem 1 shows that a linear number of decoding steps gives additional reasoning power to log-precision transformers (assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$). This follows because log-precision transformers with no decoding steps are contained in uniform TC^0 (Merrill & Sabharwal, 2023b), which means they cannot recognize all regular languages. In contrast, Theorem 1 says a linear number of steps is sufficient for recognizing all regular languages, establishing a conditional separation.

Theorem 1 shows an example of simple and familiar additional computational power granted by additional decoding steps. The core challenge in simulatingan automaton is simulating recurrence, which cannot be done without decoding steps (Merrill & Sabharwal, 2023b). Intuitively, a linear number decoding steps is sufficient for simulating recurrence, which is where the additional power comes from. However, this additional power does not stop with finite-state machines: the layer-norm hash can be used to simulate more complex models of computations like Turing machines, which we will turn to in the next section.

B.3 Simulating Turing Machines

We now show that the layer-norm hash can be applied to construct a transformer decoder that uses $t(n)$ intermediate steps to simulate a Turing machine for $t(n)$ steps. Our decoder-only construction resembles the encoder-decoder construction of Pérez et al. (2021) for simulating Turing machines. However, it avoids a simplifying assumption from Pérez et al. (2021): they needed the transformer to have access to $1/i, 1/i^2$, and i in its positional embeddings. This assumption is problematic because transformers cannot represent unbounded scalars like i due to layer-norm and it is unclear how a transformer could compute a square like $1/i^2$. In contrast, our construction works for any start-separable positional encoding, including just $1/i$.

[6]Technically it will repeat for sinusoidal positional encodings, but not on inputs with in-domain length.

Theorem 2. *Let M be a Turing machine that, on input length n, runs for at most $t(n)$ steps. For any start-separable positional encoding, there exists a decoder-only saturated transformer that, on input x, computes $M(x)$ with $t(n) + |M(x)|$ decoding steps.*

Proof. Without loss of generality, we show a construction for transformers with $1/i$ positional encodings (cf. Lemma 3). We will construct a transformer decoder that simulates a Turing machine in real time, i.e., a single transformer decoding step will simulate one Turing machine step. This means the simulation will take $t(n)$ steps, followed by an additional $\ell(n)$ steps to reconstruct the output. The main difficulty will be constructing a distributed representation of a Turing machine tape in a sequence of transformer state vectors and showing how the transformer can both maintain this representation and correctly reconstruct the current status of the tape at the head position. The key idea to the construction will be to store "diffs" to the tape at each timestep and use the layer-norm hash to dynamically reconstruct the contents at the head position at every new timestep.

We introduce a new finite vocabulary Δ and identify each element of $Q \times \Gamma \times \{\pm 1\}$ with some $\delta \in \Delta$. A deterministic Turing machine run of length m induces a *diff sequence* $\delta_0, \ldots, \delta_m \in \Delta$ capturing the state entered, token written, and direction moved after each token. We show by induction over i that we can output δ_i at decoding step i:

Base Case: In the first layer of the transformer, we use a different head to reconstruct the current position on each tape. At each position i, the head attends uniformly over the full left context where the value at a decoding step j is the $(\tau + 1)$st component of δ_j, corresponding to d_j^τ, the move direction on tape τ at time j.[7] The value of this head is thus $h_i^\tau/(n + i)$, where h_i^τ is the position on tape τ after i steps.

Inductive Case: We can now apply k different layer-norm hashes to get $\phi_i^\tau = \phi(h_i^\tau/(n+1), 1/(n+i))$ in parallel at each i. By Lemma 2, the inner product $\phi_i^\tau \cdot \phi_j^\tau$ is maximized when h_i^τ and h_j^τ are equal. For each τ we construct a head that hard-attends with query $\langle \phi_i^\tau, 0 \rangle$ over keys $\langle \phi_j^\tau, -1/j \rangle$ and values $\delta_{h_j^\tau}$, for $j \leq i$. This returns the last update d_i^τ written to the head position at step i.

We now have all necessary inputs to compute the next Turing machine transition. For each τ, the second component of d_i^τ is γ_i^τ: the current contents of cell h_i^τ of tape τ. As input to column i of the transformer, we have δ_{i-1}, which contains q_{i-1}. We now compute $\delta_i = \delta(q_{i-1}, \gamma_i^0, \ldots \gamma_i^{k+1})$ using a feedforward network, which is possible because this computation is simply a finite table lookup. We write out δ_i as the output of the column, completing the inductive step.

Finally, we use at most $\ell(n)$ additional steps to reconstruct the Turing machine output. Analogously to above, we retrieve d_i^{k+1}, the last update written to the current head position on the output tape, and output γ_i^{k+1}. We then write a diff δ_i that moves the head to the right. We iterate in this state until b (blank) is read, at which point we halt. We conclude that we are able to simulate M in real-time, i.e., the transformer runs one decoding step per step of M followed immediately by M's output. $\square$

Corollary 2.1. $\mathsf{TIME}(t(n)) \subseteq \mathsf{CoT}(t(n))$.

This lets us reason about the power of transformer decoders in terms of Turing machines. We see simulating an automaton (cf. Theorem 1) is not the only new capability unlocked with $\mathrm{O}(n)$ steps: rather, we can solve *any* problem a Turing machine can solve in $\mathrm{O}(n)$ time, such as simulating real-time counter machines.[8] With $\mathrm{O}(n^2)$ steps, we can run an $\mathrm{O}(n^2)$-time Turing machine to solve directed graph connectivity using standard graph traversal methods like depth-first search. These methods run in $\mathrm{O}(n)$ time on a random access Turing machine (Wigderson, 1992), which can be simulated in $\mathrm{O}(n^2)$ time without random access. It is possible that transformer decoders can solve directed graph connectivity with fewer than $\mathrm{O}(n^2)$ steps, as results from Zhang et al. (2023) hint at.

C Upper Bounds for Transformer Decoders

Having shown lower bounds on transformer with $t(n)$ steps, we present two different upper bounds: one that relates transformer decoders to time complexity classes, and one that relates them to space complexity classes. The relative strength of the two different bounds will vary depending on $t(n)$.

[7] Let the attention value be 0 at an input step. This construction can be easily extended to allow no-op moves.

[8] A memory-augmented variant of finite automata similar to LSTMs (Weiss et al., 2018).

C.1 Time Upper Bound

A simple upper bound on transformers with chain of thought can be obtained based on the fact that transformers can be simulated using a quadratic number of arithmetic operations.

Theorem 3. $\mathsf{CoT}(t(n)) \subseteq \widetilde{\mathsf{TIME}}(n^2 + t(n)^2)$.

Proof. We sketch a multitape Turing machine that will simulate the transformer. Each forward pass i appends key i onto a work tape and value i onto another work tape. To simulate the forward pass at time i, it suffices to show how to simulate computing self-attention at time i.

To compute self attention, the Turing machine first computes the query at time i. It then iterates over pairs on the key and value work tapes. For each pair j, we compute the attention score between query i and key j and then multiply it by value j using additional work tapes. We then add this value to a running sum tape. We treat the final sum at the output of the attention mechanism.

For runtime, observe that we compute $n+t(n)$ forward passes, and each forward pass involves looping over $n + t(n)$ key-value pairs. This means we run at most $\mathrm{O}(n^2 + t(n)^2)$ inner loop calls. It remains to be shown that one inner loop runs in polylogarithmic time. An inner loop just involves adding and multiplying $\mathrm{O}(\log n)$-bit numbers. p-bit numbers can be added in time $\mathrm{O}(p) = \mathrm{O}(\log n)$. Similarly, p-bit numbers can be multiplied in time $\mathrm{O}(p \log p) \leq \mathrm{O}(p^2)$, which comes out to $\log^2(n + t(n))$ with log precision. Thus, one inner loop can be run in polylogarithmic time. We conclude that a transformer decoder with $t(n)$ intermediate steps can be simulated by a multitape Turing machine in time $\widetilde{\mathrm{O}}(n^2 + t(n)^2)$. □

C.2 Space Upper Bound

Our second upper bound relies on the TC^0 upper bound for transformers without intermediate steps.

Theorem 4. $\mathsf{CoT}(t(n)) \subseteq \mathsf{SPACE}(t(n) + \log n)$.

Proof. Since log-precision transformers can be simulated in uniform TC^0 (Merrill & Sabharwal, 2023b), 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 $\mathrm{O}(t(n))$. To compute the next token, we call the transformer with an input of size $\mathrm{O}(n + t(n))$ 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

$$\mathsf{CoT}(t(n)) \subseteq \mathsf{SPACE}(t(n) + c \log(n + t(n))).$$

Since $\log$ is subadditive, we can simplify:

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

□

With at least $\Omega(\log n)$ steps, this upper bound can be simplified to $\mathsf{SPACE}(t(n))$. The $t(n) = \Theta(n)$ case establishes the context-sensitive languages as an upper bound for transformers with linear steps. Given our $\mathsf{TIME}(t(n))$ lower bound (Theorem 2), the tightest possible space upper bound without making fundamental complexity advances would be $\mathsf{SPACE}(t(n)/\log t(n))$ (Hopcroft et al., 1977). Conversely, our lower bound can only be tightened to $\mathsf{TIME}(t(n) \log t(n))$.

On the other hand, if we have only $\mathrm{O}(\log n)$ decoding steps, we see that intermediate decoding does not increase expressive power much beyond TC^0, because the upper bound simplifies to $\mathsf{SPACE}(t(n)) = \mathsf{L}$. Thus, under standard assumptions, transformers with at most a logarithmic number of decoding steps cannot solve directed graph connectivity, Horn formula satisfiability, or other NL- or P-complete problems. On the other hand, they may be able to solve L-complete problems, unlike transformers without intermediate decoding.

D Additional Details: Transformer Components

This section recalls the definition from Merrill & Sabharwal (2023a) for the components of the transformer layer. We assume a pre-norm (Xiong et al., 2020) parameterization of the transformer for concreteness and because this is more standard in newer transformers. However, the results would also hold with the original post-norm (Vaswani et al., 2017).

D.1 Transformer Embeddings

For each position $1 \leq i \leq n$, the transformer embedding function represents token $\sigma_i \in \Sigma$ and its position i with a vector. Let $\mathbf{V}$ be an embedding matrix of size $|\Sigma| \times m$ where each row represents the embedding for some σ. Let $f : \mathbb{N} \to \mathbb{D}_p^m$ be computable in time $\mathrm{O}(\log n)$. Then,

$$\phi(\sigma_i, i) = \mathbf{v}_{\sigma_i} + f(i).$$

D.2 Self Attention

The two components of the self attention block are s, the similarity function, and v, the value function. Let $\mathbf{h}_i$ be the hidden state at the previous layer and $\bar{\mathbf{h}}_i = \mathsf{layer\_norm}(\mathbf{h}_i)$. Then, the similarity function first computes queries and keys, and then takes the scaled dot-product between them:

$$s(\mathbf{h}_i, \mathbf{h}_j) = \exp\left(\frac{\mathbf{q}_i^\top \mathbf{k}_i}{\sqrt{m/h}}\right), \qquad \text{where} \begin{array}{l} \mathbf{q}_i = \mathbf{W}_q \bar{\mathbf{h}}_i + \mathbf{b}_q \\ \mathbf{k}_i = \mathbf{W}_k \bar{\mathbf{h}}_i + \mathbf{b}_k \end{array}.$$

Then the value function is defined $v(\mathbf{h}_i) = \mathbf{W}_h \bar{\mathbf{h}}_i + \mathbf{b}_h$.

D.3 Activation Block

The activation function f encapsulates the aggregation of the attention head outputs and the feedforward subnetwork of the transformer. f takes as input attention head outputs $\mathbf{a}_{i,1}, \ldots, \mathbf{a}_{i,h} \in \mathbb{D}_p^{m/h}$ and the previous layer value $\mathbf{h}_i$.

The first part of the activation block simulates the pooling part of the self-attention sublayer. The head outputs are first concatenated to form a vector $\mathbf{a}_i$, which is then passed through an affine transformation $(\mathbf{W}_o, \mathbf{b}_o) : \mathbb{D}_p^m \to \mathbb{D}_p^m$ followed by residual connections to form the sublayer output $\mathbf{o}_i \in \mathbb{D}_p^m$:

$$\mathbf{o}_i = \mathbf{W}_o \mathbf{a}_i + \mathbf{b}_o + \mathbf{h}_i.$$

The second part of the activation block first applies layer-norm and then simulates the feedforward subnetwork to compute the next layer vector $\mathbf{h}'_i$. Let $\bar{\mathbf{o}}_i = \mathsf{layer\_norm}(\mathbf{o}_i)$. Let σ be a nonlinearity computable in linear time on its input (in the most standard transformer, ReLU). Then, for affine transformations $(\mathbf{W}_1, \mathbf{b}_1) : \mathbb{D}_p^m \to \mathbb{D}_p^w$ and $(\mathbf{W}_2, \mathbf{b}_2) : \mathbb{D}_p^w \to \mathbb{D}_p^m$, the feedforward subnetwork can be defined:

$$\mathbf{h}'_i = \mathbf{W}_2 \sigma(\mathbf{W}_1 \bar{\mathbf{o}}_i + \mathbf{b}_1) + \mathbf{b}_2 + \mathbf{o}_i.$$

E Additional Details: Layer-Norm Hash

Lemma 1 (Proof in Appendix E)**.** *For any $x \in \mathbb{R}$ and $i \in \mathbb{N}$, $\phi(x/i, 1/i) = \phi_x$.*

Proof. Let $\mathbf{v}_q = \langle q/i, 1/i, -q/i, -1/i \rangle$. $\mathbf{v}_q$ is constructed with mean 0, so layer-norm reduces to 2-normalization. Thus,

$$\begin{aligned} \phi(q/i) &= \mathbf{v}_q / \|\mathbf{v}_q\| \\ &= \mathbf{v}_q \cdot \frac{n}{\sqrt{2q^2 + 2}} \\ &= \frac{1}{\sqrt{2q^2 + 2}} \langle q, 1, -q, -1 \rangle. \end{aligned}$$ □

Lemma 2 (Proof in Appendix E)**.** *For any $q, k \in \mathbb{R}$, $\phi_q \cdot \phi_k = 1$ if and only if $q = k$.*

Proof. Following the notation and derivation from Lemma 1, we have

$$\begin{aligned}\phi(q/n) \cdot \phi(k/m) &= \frac{2qk + 2}{\sqrt{(2q^2 + 2)(2k^2 + 2)}} \\ &= \frac{qk + 1}{\sqrt{(q^2 + 1)(k^2 + 1)}}.\end{aligned}$$

The inner product of unit-norm vectors is maximized at 1. In this case, we show that it achieves 1 only when $q = k$, meaning that is the unique maximum:

$$\begin{aligned}1 &= \frac{qk + 1}{\sqrt{(q^2 + 1)(k^2 + 1)}} \\ (qk + 1)^2 &= (q^2 + 1)(k^2 + 1) \\ (qk)^2 + 2qk + 1 &= (qk)^2 + q^2 + k^2 + 1 \\ 2qk &= q^2 + k^2 \\ 0 &= (q - k)^2.\end{aligned}$$

We conclude that, for any i, j, $\phi(q/i) \cdot \phi(k/j)$ is maximized if and only if $q = k$. □

F Additional Details: Start-Separable Position Encodings

Lemma 3 (Proof in Appendix F)**.** *For any language L, if some decoder-only saturated transformer with $1/i$ positional encodings recognizes L, then, for any start-separable positional encoding π, some decoder-only saturated transformer with π positional encodings recognizes L.*

Proof. If the positional encoding is start-separable, we can construct a transformer decoder that attends uniformly over the previous tokens and set the value to be 1 at the initial token and 0 otherwise. The output of this head will be $1/i$. We can then compose this transformer with a transformer that uses $1/i$ positional encoding to recognize L, which exists as a premise. □