The Parallelism Tradeoff: Limitations of Log-Precision Transformers

William Merrill
Center for Data Science New York
University, New York, NY, USA
willm@nyu.edu

Ashish Sabharwal
Allen Institute for AI
Seattle, WA, USA
ashishs@allenai.org

Abstract

Despite their omnipresence in modern NLP, characterizing the computational power of transformer neural nets remains an interesting open question. We prove that transformers whose arithmetic precision is logarithmic in the number of input tokens (and whose feedforward nets are computable using space linear in their input) can be simulated by constant-depth logspace-uniform threshold circuits. This provides insight on the power of transformers using known results in complexity theory. For example, if $\mathsf{L} \neq \mathsf{P}$ (i.e., not all poly-time problems can be solved using logarithmic space), then transformers cannot even accurately solve linear equalities or check membership in an arbitrary context-free grammar with empty productions. Our result intuitively emerges from the transformer architecture's high parallelizability. We thus speculatively introduce the idea of a fundamental **parallelism tradeoff**: any model architecture as parallelizable as the transformer will obey limitations similar to it. Since parallelism is key to training models at massive scale, this suggests a potential inherent weakness of the scaling paradigm.

1 Introduction

This work aims to characterize the computational model implicit in transformer neural networks (Vaswani et al., 2017), which form the basis of recent breakthroughs in large language models such as BERT (Devlin et al., 2019), T5 (Raffel et al., 2020), and GPT-3 (Brown et al., 2020). What computational primitives can the transformer's components implement, and what problems can the full system solve in aggregate? These questions are important for interpreting transformers in a principled way, understanding potential limitations of their reasoning capabilities, and building trust in deployed transformer-based systems.

Early theoretical work on transformers established their Turing completeness, albeit with assumptions like infinite precision and arbitrarily powerful feedforward subnets (Pérez et al., 2019; Dehghani et al., 2019). On the other hand, a strand of more recent work uses techniques from circuit complexity theory to derive strong limitations on the types of problems transformers can solve given restrictions on the form of attention allowed in the transformer. Specifically, Hahn (2020) and Hao et al. (2022) showed that transformers restricted to hard attention are very limited: they can only solve problems in a weak complexity class (non-uniform AC^0) that doesn't even contain basic problems like majority of n bits. Merrill et al. (2022) extended this to a more general class of ''saturated attention'' transformers with a floating point datatype, and showed a larger class of problems (non-uniform TC^0) as an upper bound. This motivates analyzing a setting that strikes a middle ground: *Can we characterize transformers whose precision and feedforward nets' computational power are $\underline{\text{realistically bounded}}$, but where attention is also $\underline{\text{realistically expressive}}$?*

An important practical limitation of these prior results is the ''non-uniform'' nature of the considered circuit classes, which makes these classes non-realizable and the findings difficult to interpret. This is because non-uniform AC^0 and TC^0, while highly limited in computation, also contain some problems that are not even decidable, i.e., for which there doesn't exist any exact algorithm. Thus, non-uniform classes cannot be directly compared with standard algorithmic complexity classes such as P, NP, etc. This motivates our second key question: *Can we derive $\underline{\text{uniform}}$ upper bounds on transformers?*

We show that one can achieve both of these goals by making the modest assumption that all values in the transformer have $\mathrm{O}(\log n)$ precision (where n is the number of input tokens), and, similarly, that transformer's subnetworks are computable in $\mathrm{O}(\log n)$ space. Log precision is enough to represent the positional encodings at the input layer of the transformer, and to encode pointers to all other positions in the sequence at

Transactions of the Association for Computational Linguistics, vol. 11, pp. 531–545, 2023. https://doi.org/10.1162/tacl_a_00562
Action Editor: Daniel Gildea. Submission batch: 8/2022; Revision batch: 12/2022; Published 6/2023.

later transformer layers. Assuming log precision across all layers captures the idea that the hidden representations contain a constant number of hidden states whose precision (16 or 32 bits) is small relative to the length of the input (2048 in GPT-3). On long sequences, the precision will not be enough to losslessly encode the full input sequence into a single vector. Instead, the processing of the sequence must somehow be distributed in each layer and performed in parallel.

Upper Bound on Transformers. Our main contribution is proving that log-precision transformers can be simulated by *uniform* constant-depth threshold circuits. Thus, **such transformers can only solve problems in uniform** TC^0. This characterization is strikingly weak compared to the Turing-completeness of infinite-precision transformers. Since we believe log precision is more realistic for practical transformers than infinite precision, these results point to the conclusion that transformers are not Turing-complete in practice.

In contrast to past results, our upper bound on transformers is a *uniform* circuit class, enabling direct comparison of log-precision transformers to many natural complexity classes. These connections reveal specific problems that define the upper limits of log-precision transformers' capabilities, as discussed further in §2.

Intuitively, our upper bound says that log-precision transformers are computationally shallow, and that this shallowness can be understood to emerge from their parallelizability. Transformers' inherent parallelism is useful for training them efficiently at massive scale, but may limit the complexity of the computations they can express. We introduce the term **parallelism tradeoff** to capture this idea, which represents a potential fundamental weakness of the current paradigm of scaling language models. Formally characterizing reasoning capabilities relevant to language models and understanding whether they likely fall outside upper bounds implied by the tradeoff would clarify the practical implications of this limitation of scaling.

It could also be that the limitations of parallelism are not a curse but a blessing, if they constrain the hypothesis space in a way useful for learning. We have no evidence that this is true, but mention it as an alternate interpretation of the results that could be clarified in future work.

Instruction Following and Advice Transformers. We also consider an instruction following setting (Brown et al., 2020) where the transformer is provided the description of a task along with an input on which to execute the instruction. We construct a practically parameterizable transformer that can execute instructions perfectly if they are provided in the form of TC^0 circuits. This complements recent work that studies transformers' ability to follow other forms of instructions such as regular expressions (Finlayson et al., 2022).

Based on the fundamental property that transformers can correctly *evaluate* any given TC^0 circuit on a given input, we introduce the notion of *advice transformers* akin to advice-taking Turing machines. We show that transformers can recognize any (non-uniform) TC^0 language if provided appropriate poly-size advice.

In summary, our findings provide new insights on both the abilities and the limitations of transformers, and bring out bounded precision, threshold computations, and parallelism as key notions for understanding the implicit computational model of transformers in practice.

Roadmap. Before diving into technical details, we discuss in §2 the implications of our results on both fundamental as well as practical abilities of transformers. §3 provides a brief primer on circuits as a model of computation. It then discusses a way of serializing a circuit into a string; we later show how to generate such serializations using a resource-bounded algorithm, which is the key to proving containment of transformers in *uniform* circuit classes. §4 defines our formal model of bounded-precision transformers. §5 derives our first formal bound on log-precision transformers. This bound involves *non-uniform* circuit families, similar in spirit to prior results in this area. §6 proves our more technical main result: the first *uniform* circuit complexity upper bound for transformers (specifically, uniform TC^0). Finally, §7 provides a *lower bound* on transformers, introduces the notion of an Advice Transformer, and connects these to the machine learning problems of Instruction Learning and Following.

2 Implications of Our Findings

Before diving into technical details, we discuss the general implications of our findings on the abilities and limitations of transformers. We will focus here on our main result (Theorem 2), which shows that log-precision transformers are in the complexity class logspace-uniform TC^0.

The Parallelism Tradeoff. One interpretation of complexity classes such as NC^0, AC^0, and TC^0 is sets of poly-time solvable problems that are parallelizable to a very high degree—they can be solved in parallel in *constant* time with enough parallel processors. This gives some intuitive explanation of our result: log-precision transformers end up in TC^0 because they were designed to be highly parallelizable. Since parallelism is an important property of today's dominant paradigm of training models at massive scale, this points to the conclusion that any massively scaled up model—transformer or otherwise—will likely obey restrictions similar to the ones derived here for log-precision transformers. There is thus an important tradeoff between the massive parallelizability of today's networks and their representation power.

What Transformers Can/Cannot Compute. Our result places log-precision transformers in the complexity class logspace-uniform TC^0. This has immediate implications on the kinds of problems such transformers can and cannot accurately solve.

Consider any problem X that is complete for a complexity class C that contains logspace-uniform TC^0. By definition of completeness, every problem log-precision transformers can solve perfectly is efficiently reducible to X and is thus no harder than X. This implies that—despite their massive size—the computation performed by such transformers is, for instance, no harder than solving basic L-complete problems like **graph connectivity**: the problem of checking whether there is a path between two nodes in an undirected graph (Lewis and Papadimitriou, 1982; Reingold, 2008).

By the same token, if C is strictly larger than logspace-uniform TC^0, then such transformers *cannot* perfectly solve X. Thus, log-precision transformers cannot perfectly solve the following reasoning problems:

- **Linear equalities**: find x s.t. $Ax = b$[1]
- **Universal context-free recognition**[1,2]
- **Propositional satisfiability** (SAT)[3]
- **Horn-clause satisfiability** (HORN-SAT)[1]
- **AI planning** (Bylander, 1991)
- **Permanent computation**[4]

This highlights the limits of practical transformers with limited-precision arithmetic, indicating that they are far from being universal or all-powerful as suggested by some prior studies.

One important caveat about these negative results is that they are asymptotic in nature—they apply for ''large enough'' input size n. It's possible for log-precision transformers to solve such problems easily when n is small. Further, these negative results are about exact solutions, but they often also extend beyond this when formal hardness-of-approximation results are known.

Limitations of Our Formal Model. Prior formal characterizations of transformers either make unrealistically strong assumptions (Pérez et al., 2019; Dehghani et al., 2019) or place unrealistic restrictions (Hahn, 2020; Hao et al., 2022; Merrill et al., 2022). In contrast, we make only one assumption—namely, all intermediate values in the transformer are limited to $\mathrm{O}(\log n)$ bits, where n is the number of input tokens. We next discuss some implications of this assumption and what our findings mean for practical transformers.

As mentioned above, our bounds are asymptotic in nature and thus apply when n is sufficiently large. In practice, transformers use fixed precision at each computation node, which is more restrictive than precision growing with the input sequence length n, as $\mathrm{O}(\log n)$ bits. However, this constant could be large and thus, for relatively small n, our results do not rule out practical transformers solving difficult problems. Our results, however, do show that as n grows sufficiently large, log-precision transformers are fundamentally limited to problems within TC^0 and cannot accurately solve various commonly studied problems mentioned earlier under ''What Transformers Cannot Compute''. Extending our analysis to small n will help close the gap to practice.

[1]Assuming logspace-uniform $\mathsf{TC}^0 \neq \mathsf{P}$. Follows because these problems are P-complete (Greenlaw et al., 1991).

[2]Takes both a grammar and a string as input and return whether the grammar generates the string. Jones and Laaser (1976) demonstrate P-completeness.

[3]Assuming logspace-uniform $\mathsf{TC}^0 \neq \mathsf{NP}$. Follows because SAT is NP-complete (cf. Biere et al., 2009).

[4]Assuming logspace-uniform $\mathsf{TC}^0 \neq \#\mathsf{P}$. Follows because permanent is $\#\mathsf{P}$-complete (Valiant, 1979). Allender (1999) shows permanent is not in *logtime*-uniform TC^0.

Our formal model is based on a binary classification view of transformers. However, our results apply directly to multi-class classification as well and can be extended to generation problems by viewing, for instance, next word prediction in NLP as a multi-class classification problem. However, if the transformer decoder is allowed to condition on its previous output in a generation problem, then this would violate our formal setup.

2.1 Potential Applications

Extracting Circuits from Transformers. Elhage et al. (2021) propose extracting circuits[5] that capture the computational structure of transformers. Our results suggest threshold circuit families are a good formalism for expressing mechanisms extracted from transformers. Constructively converting transformers to threshold circuits is beyond the scope of the current paper, although we hope to explore this in more detail in future work.

Testing Separation Candidates in Complexity Theory. Theorem 2 also motivates a paradigm for quickly testing complexity theory conjectures. If a problem is believed to separate TC^0 and NC^1, a transformer can be trained on problem instances. If the transformer generalizes perfectly to harder instances than it was trained on, this gives an empirical hint that the problem is in TC^0, providing evidence against the conjecture.

3 Circuit Computation

Let $\{0,1\}^*$ be the set of finite binary strings. For $x \in \{0,1\}^*$, let $|x|$ be its length. We refer to a function from $\{0,1\}^*$ to $\{0,1\}^*$ as a Boolean function. Boolean functions can implement arithmetic operations if we define a semantics for binary strings as numbers. We will treat the intermediate values in a transformer as binary strings, and the internal operations as Boolean functions.

Circuits are a model of computation for computing Boolean functions of fixed-length binary strings.[6] Formally, a circuit is a directed acyclic computation graph. The leaf nodes represent binary variables and their negations. The internal nodes represent functions in some set $\mathcal{G}$, and the directed edges represent the flow of function outputs into inputs of other functions. One or more nodes in the circuit are marked such that their value is the output of the circuit.

Definition 1. For a set of functions $\mathcal{G}$, a $\mathcal{G}$-circuit is a directed acyclic computation graph where the internal nodes have labels from $\mathcal{G}$.

Complexity Measures. The *size* of a circuit is the total number of gates in it, including negation. The *depth* of a circuit is the length of the longest path from any input node to any output node.

Circuit Families. A *circuit family* generalizes a circuit to take variable-length binary strings as input. Formally, a circuit family is a sequence of circuits $C_n : \{0,1\}^n \to \{0,1\}$ for $n \in \mathbb{N}$. A circuit family implicitly recognizes a formal language defined as follows:

Definition 2. A circuit family C_n recognizes $L \subseteq \{0,1\}^*$ if, for all $x \in \{0,1\}^*$, $C_{|x|}(x) = 1$ if and only if $x \in L$.

We now define classes of languages by constraining the complexity of the circuit families needed to recognize them:

Definition 3. Let non-uniform AC^0 be the set of $L \subseteq \{0,1\}^*$ such that L is recognizable by a poly-size, constant-depth $\{\neg, \wedge, \vee\}$-circuit family.

For $k \in \mathbb{N}$, a *threshold gate* $\theta_{\leq k}$ takes m input bits and returns whether $\sum_{i=1}^m x_i \leq k$. We define $\theta_{\geq k}$ analogously. For example, $\theta_{\leq 3}(110011) = 0$.

Definition 4. Let TC^0 be the set of $L \subseteq \{0,1\}^*$ such that L is recognizable by a poly-size, constant-depth $\{\theta_{\leq k}, \theta_{\geq k}\}_{k \in \mathbb{N}}$-circuit.

The gates $\neg$, $\wedge$, and $\vee$ are all just special cases of thresholds, so we can imagine TC^0 circuits to have access to these as well. Thus, TC^0 circuits can implement AC^0 circuits.

Circuit Serialization. We identify a circuit with its serialization in a formal language that identifies each node's label and adjacency list. We will adopt a specific grammar for concreteness, but our construction can be adapted to other string representations of circuits.

We define a circuit serialization as a traversal of a circuit ordered by some topological sort. In this serialization, leaf nodes (variables) are represented by the string `X`. An internal node (gate) is represented in Polish notation by the function it

[5]Their sense of ''circuit'' is not exactly the formal sense we use in this paper, though the goal of capturing transformers' implicit computational mechanism is the same.

[6]For a mini-tutorial on circuit complexity theory and its relevance to transformers, see Merrill et al. (2022).

computes (`AND`, `OR`, or `NOT`) followed by a list of pointers to its arguments. Each argument $\texttt{\&1}^j$ of gate i encodes (in a unary) a zero-indexed pointer to the j-th gate in the circuit, where $j < i$. The final node is interpreted as the circuit output.

To serialize $\{\wedge, \vee\}$-circuits, we use the following grammar, where the i parameter is passed through Gate$[i]$ nonterminals to track the index of the gate in left-to-right order:

$$\begin{aligned} \text{Circuit} &\rightarrow \text{Gate[1] Gate[2]} \cdots \text{Gate}[g] \\ \text{Gate}[i] &\rightarrow \texttt{X} \mid \texttt{NOT}\ \text{Arg}[i] \mid \text{Op Arg}[i]^* \\ \text{Arg}[i] &\rightarrow \texttt{\&1}^j \quad \text{s.t. } j < i \\ \text{Op} &\rightarrow \texttt{AND} \mid \texttt{OR} \end{aligned}$$

In the Arg$[i]$ rule, we enforce that $j < i$ so that arguments must be pointers to already defined gates. As an example of this serialization language, the circuit for $x_1 \vee \neg x_2 \vee x_3$ is represented as[7]

```
X X X NOT &1 OR & &111 &11
```

By convention (cf. §3), negations in AC^0 circuits are usually taken to occur at the beginning of the circuit, rather than after $\wedge$ or $\vee$ nodes.[8] Our serialization grammar does not enforce this property, but of course any circuit with this property can be serialized by our grammar.

It is a bit more complicated to serialize threshold circuits. Formally, a threshold circuit serialization is generated by the following grammar:

$$\begin{aligned} \text{Circuit} &\rightarrow \text{Gate[1] Gate[2]} \cdots \text{Gate}[g] \\ \text{Gate}[i] &\rightarrow \texttt{X} \mid \text{Dir } \texttt{1}^k\texttt{0}^{m-k}\ \text{Arg}[i]^m \\ \text{Arg}[i] &\rightarrow \texttt{\&1}^j \quad \text{s.t. } j < i \\ \text{Dir} &\rightarrow \texttt{<=} \mid \texttt{>=} \end{aligned}$$

In the rewrite rule for Gate$[i]$, $m \in \mathbb{N}$ is the arity of the gate, and $k \leq m$ is its threshold. The span $\texttt{1}^k$ after Dir can be interpreted semantically as a unary encoding of the parameter k for a threshold gate, padded by `0`'s to the number of total arguments of gate i. For simplicity, we imagine $\neg$ gates are represented as unary $\theta_{\leq 0}$ gates. Thus, the circuit $\theta_{\geq 1}(x_1, \neg x_2)$ would be represented as

```
X X <= 00 &1 >= 10 & &11
```

[7]Spaces here (and in the grammar) are added for readability. We will ignore these spaces when passing circuit serializations as inputs to a transformer in Section 7.

[8]We can apply De Morgan's laws to force any AC^0 circuit to have this property.

Uniformity. The circuit families we have defined above are *non-uniform*, meaning that we do not enforce that the circuits processing different input sizes must be related in any way. In degenerate cases, non-uniform circuit families can solve undecidable problems[9] because they have infinite description length, making them a physically unrealizable model of computation. Complexity theorists have thus introduced *uniform* circuit families. Uniform circuit families are a realizable model of computation with relations to classes in computational complexity and formal language theory.

Intuitively, in a uniform circuit family, the circuits for different input sizes must be ''somewhat similar'' to each other. We formalize this (cf. Arora and Barak, 2009) by saying that there exists a resource-constrained Turing machine that maps the input 1^n to a serialization of circuit C_n.

Definition 5. A language L is $(S(n), I(n))$-space uniformly computable by a circuit model M iff there exists a Turing machine that, for all $n \geq 0$, uses $S(n)$ space to map 1^n to an M-circuit recognizing L on inputs of size $I(n)$.

This notion of uniformity is more general than the standard notion in that the input size $I(n)$ is a function of the problem complexity n. The reason for this is that we will apply uniformity to subcomputations with different input sizes $I(n)$ within a larger computation of input size n. The standard notion of uniformity corresponds to $I(n) = n$.

Furthermore, we will refer to a circuit family as *uniform* if it is uniformly computable with $S(n) = \mathrm{O}(\log n)$ (cf. Arora and Barak, 2009). We can define uniform versions of AC^0 and TC^0 by adopting the previous definitions exactly, but also enforcing uniformity. For the rest of the paper we will clarify whether we mean the uniform or non-uniform variant of TC^0 when unclear from context, since both classes will come up.

4 Bounded-Precision Transformers

A *transformer* (Vaswani et al., 2017) is a neural network architecture made up of a constant number of *transformer layers*. A transformer layer is a module that computes self-attention

[9]Consider the unary language 1^n such that Turing machine n (under some arbitrary enumeration) halts. This problem is in non-uniform AC^0 since we can hard-code the right answer for each n in C_n.

over a sequence followed by an elementwise transformation of the output vectors.

4.1 Precision and Space

We will assume that each transformer is resource bounded in terms of the *precision* of each value it computes and, for some of our results, the *space* it uses for the computation of key operations such as embedding, attention, and activation. Specifically, we will assume precision p, i.e., the values at all layers, as well as the outputs of all key intermediate operations in it (attention, activation, arithmetic operators, etc.), are represented using p bits. This is a realistic assumption as, in practice, today's transformers are typically limited to the 64-bit precision of the underlying hardware. Formally, we define p-precision as follows:

Definition 6. A k-ary function $f : x_1, \ldots, x_k \mapsto y$ is *p-precision* if $x_1, \ldots, x_k, y \in \{0,1\}^*$ have size at most p bits, and f can be computed by a p-space-bounded Turing machine.

This says the size of the function input and output are bounded below p. Similarly, the intermediate space used by the computation must also be bounded below p. Thus, higher precision computations cannot somehow be hidden inside f.

Definition 6 naturally applies to functions with bounded arity k. We will also need to define p precision for the summation operator in the transformer, which adds n different floats of size p.[10] Adding n floats can blow up the precision needed to represent their sum. For example, imagine adding the floating points $1 \cdot 2^0 + 1 \cdot 2^c$. We obtain $(2^c + 1) \cdot 2^0$, whose mantissa takes $c + 1$ bits to represent. In practice, computers do not preserve full precision in such situations: instead, small terms like $1 \cdot 2^0$ are discarded. Thus, we define the transformer's addition operation $\oplus$ to be similarly approximate (and thus preserve precision); see §A.

4.2 Transformer Definition

4.3 Attention Heads

The core building block of a transformer is an attention head. We define this at a high level of abstraction as follows:

[10]Our proof also goes through if the transformer weights are integers, as is sometimes done (Dettmers et al., 2022).

Definition 7. A p-precision attention head is specified by a binary p-precision *similarity* function $s : \{0,1\}^p \times \{0,1\}^p \to \{0,1\}^p$.

Let $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \{0,1\}^p$ be the input sequence to a p-precision attention head, and let $\oplus$ be approximate floating-point addition (§A).

Definition 8. For all $\ell \geq 0$, a p-precision attention head $H_h^{\ell+1}$ computes a vector $\mathbf{a}_{ih}^{\ell+1} \in \{0,1\}^p$ via

$$\mathbf{a}_{ih}^{\ell+1} = \bigoplus_{j=1}^{n} \frac{s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)}{Z_i} \cdot \mathbf{h}_j^\ell,$$

where $Z_i = \bigoplus_{j=1}^n s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$.

Standard transformer attention heads (Vaswani et al., 2017) are a special case of this definition where s is scaled dot-product similarity between keys and queries. Standard transformers also have a linear or affine value function applied to each $\mathbf{h}_j^\ell$ in the sum over j. By its affineness, the value function can, without loss of generality, be removed from the attention head and considered to be part of the transformer layer (i.e., applied to the output of the attention head).

4.4 Transformer Layers

A p-precision transformer layer is then a tuple of heads and a function f used to combine them.

Definition 9 (p-precision transformer layer)**.** A p-precision transformer layer is a tuple $L^{\ell+1} = \langle H_1, \cdots, H_k, f \rangle$, where each H_h is an attention head and $f : (\{0,1\}^p)^k \times \{0,1\}^p \to \{0,1\}^p$ is a p-precision *activation* function.

A p-precision transformer layer can be understood to define a sequence of vectors $\mathbf{h}_1^{\ell+1}, \ldots, \mathbf{h}_n^{\ell+1}$ in terms of an input sequence of vectors $\mathbf{h}_1^\ell, \ldots, \mathbf{h}_n^\ell$ (coming from the previous layer in the transformer) by first computing k attention heads in parallel and then combining their output using f. The first k inputs to f will correspond to the attention head outputs, and the additional input is the original input from the previous layer. Recall that $\mathbf{a}_{ih}^{\ell+1}$ is the output of head $H_{ih}^{\ell+1}$ on input $\mathbf{h}^\ell$ at position i. The function computed by a transformer layer can be described formally as follows.

Definition 10 (Transformer layer computation)**.** For $\ell \geq 0$, a p-precision transformer layer $L^{\ell+1}$ recurrently computes the output sequence $\mathbf{h}_1^{\ell+1}, \ldots, \mathbf{h}_n^{\ell+1}$ as a function of the inputs

$\mathbf{h}_1^\ell, \dots, \mathbf{h}_n^\ell$, where, for $1 \leq i \leq n$, the ith component is computed according to

$$\mathbf{h}_i^{\ell+1} = f(\mathbf{a}_{i1}^{\ell+1}, \dots, \mathbf{a}_{ik}^{\ell+1}, \mathbf{h}_i^\ell).$$

f can be understood to encapsulate layernorm, residual connections, and the feedforward sublayer of a standard transformer (Vaswani et al., 2017). $\mathbf{h}_i^\ell$ is given to f to allow residual connections. As mentioned in §4.3, f can also encapsulate the value function for each head.

4.5 Transformer Encoder

Finally, we define a transformer of depth d as a cascade of d transformer layers:

Definition 11 (p-precision transformer)**.** A p-precision transformer over alphabet Σ is a pair consisting of a p-precision position embedding function[11] $\phi : \Sigma \times \mathbb{N} \to \{0,1\}^p$ and a d-tuple of p-precision transformer layers $\langle L^1, \dots, L^d \rangle$.

For a position embedding function ϕ and $w \in \Sigma^n$, let $\phi(w)$ be the position-wise broadcasted embedding of w: for $1 \leq i \leq n$, $\phi_i(w) \triangleq \phi(w_i, i)$.

Definition 12 (Transformer computation)**.** A transformer $\big(\phi, \langle L^1, \cdots L^d \rangle\big)$ computes the following function of a string $w \in \Sigma^*$:

$$T(w) = (L^d \circ L^{d-1} \circ \cdots \circ L^1)(\phi(w)).$$

We will use n to denote the length of w, and take the transformer's depth d to be fixed w.r.t. n.

The **input** to the transformer can thus be represented with $N = n \log |\Sigma|$ bits using a binary encoding for the vocabulary. The circuits we construct in subsequent sections to simulate transformers will also have input size N. We will assume transformers have **log-precision** relative to the size of the input, specifically, $\mathrm{O}(\log N)$-precision. Since $|\Sigma|$ is fixed (typically 30000 in practice), we will think in terms of $\mathrm{O}(\log n)$-precision. Thus, by Definition 6, all of the intermediate functions of such transformers are computable in $\mathrm{O}(\log n)$ space and output (at most) these many bits. Note that this is enough precision to represent positional encodings and for each position to point to a constant number of other values, but not enough precision for non-lossy pooling of the entire input into a single value.

[11]To apply the normal notion of p-precision to inputs outside $\{0,1\}^*$, we imagine elements of Σ are encoded as integers $\leq |\Sigma|$ in binary, and natural numbers are represented as integers $\leq n$. Thus, we assume $\log |\Sigma| + \log n \leq p$.

Relationship to Practical Transformers. Our log-precision transformers do not enforce that s (Definition 7) and f (Definition 9) follow the transformer structure. However, a feedforward net whose primitive operations (e.g., scalar multiplication) are defined over $\mathrm{O}(\log n)$-size numbers can be computed in $\mathrm{O}(\log n)$ space. Thus, bounded-precision practical transformers are a special case of our log-precision transformers. This makes our setup appropriate for proving upper bounds on transformers, which is our main contribution.

5 Log-Precision Transformers as Non-Uniform Threshold Circuits

We first show that log-precision transformers can be simulated by *non-uniform* threshold circuits, before presenting the more technical *uniform* version of the results in §6. The initial non-uniform result extends the findings of Merrill et al. (2022), who showed that *saturated* attention transformers[12] can be simulated in TC^0. Here, we remove the simplifying saturated attention assumption and other restrictions on the underlying datatype. Instead, we show that our log-precision assumption is enough to prove that a transformer can be simulated in TC^0 with any attention function.

Hao et al. observed that any Boolean function of $\mathrm{O}(\log n)$ bits can be computed by a $\mathrm{poly}(n)$ size circuit. We extend this to m-bit outputs, which is both more convenient and more efficient than constructing m separate Boolean circuits:

Lemma 1 (Extended from Hao et al., 2022)**.** *Let $f : \{0,1\}^* \to \{0,1\}^m$ be a function. For all $c \in \mathbb{R}^+$ and $n \in \mathbb{N}$, there exists an AND/OR circuit of size at most $n^c + c \log n + m$ and depth 3 that computes f on inputs of size $c \log n$.*

Proof. Like Hao et al. (2022), we construct a circuit using a DNF representation of f on inputs of size $c \log n$, except we use a combined DNF representation for all output bits of f. The DNF formula has at most $2^{c \log n} = n^c$ terms. The circuit has a NOT gate for each input bit, an AND gate for each DNF term, and, for each of the m output bits, an OR gate combining the outputs of those AND gates (i.e., DNF terms) for which that bit is 1. $\square$

[12]Saturated attention is uniform attention over a subset of the prior layer nodes.

We now use Lemma 1 to prove the following non-uniform result. We note that the proof goes through even if the notion of p-precision (Definition 6) is relaxed to not require computability in space p. This requirement will, however, become important for our subsequent result in §6.

Theorem 1 (Non-uniform)**.** *Any $c \log n$-precision depth-d transformer operating on inputs in Σ^n can be simulated by a threshold circuit family of depth $3 + (9 + 2d_\oplus)d$.*

Proof. Let $w \in \Sigma^n$ be the input of a $c \log n$-precision transformer. We show by induction that we can construct a composition of constant-depth, poly-size threshold circuits to compute each layer of this transformer. Thus, any constant-depth transformer will be computable by a constant-depth threshold circuit.

In the base case of layer 0 and token i, we construct gates representing the constant i encoded in binary. We can then compute $\mathbf{h}_i^0 = \phi(w_i, i)$ using Lemma 1, yielding a poly-size depth-3 circuit.

In the inductive case of computing layer $\mathbf{h}_i^{\ell+1}$ for $1 \leq \ell + 1 \leq d$, we note that each vector output of layer $\mathbf{h}_i^\ell$ has size (at most) $c \log n$ bits because of the log-precision assumption.

We first fix a head $\mathbf{a}_{ik}^{\ell+1}$ (Definition 8) to simulate. Applying Lemma 1, we can compute $s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$ with a poly-size depth-3 circuit, in parallel for all j. Since n floats with $c \log n$ precision can be approximately added in TC^0 (§A), we can construct a TC^0 circuit of depth $d_\oplus$ to compute Z_j. Since $s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell), Z_i$, and $\mathbf{h}_i^\ell$ all have $c \log n$ bits, we can compute $\frac{s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)}{Z_i}\mathbf{h}_j^\ell$ with a poly-size depth-3 circuit;[13] we do this in parallel for all j. Next, we again use the fact that approximate addition of n floats is in TC^0 to compute $\mathbf{a}_{ih}^{\ell+1}$ as the approximate sum over j with a depth-$d_\oplus$ circuit.

We now simulate a layer $\mathbf{h}_i^{\ell+1}$ (Definition 10) in terms of its constituent heads. Since all arguments of g have size $c \log n$, we apply Lemma 1 to compute g with a poly-size depth-3 circuit, yielding $\mathbf{h}_i^{\ell+1}$. We repeat this in parallel for all i. This completes the inductive step new to compute all values in the $\ell + 1$-st layer with a circuit depth of $9 + 2d_\oplus$.

Aggregating the circuit over all d layers, the overall circuit depth is $3 + (9 + 2d_\oplus)d$. $\square$

[13]This may seem counterintuitive since multiplication of two n-precision numbers is outside AC^0. However, here we leverage the fact that the precision is $c \log n$.

Corollary 1.1 (Non-uniform)**.** *Any log-precision transformer can be simulated by a non-uniform TC^0 circuit family.*[14]

6 Log-Precision Transformers as *Uniform* Threshold Circuits

We will now extend the argument from the last section to show that $\mathrm{O}(\log n)$-precision transformers can be simulated by uniform constant-depth threshold circuits by capitalizing on the assumption that ϕ, s, and f are log-precision, and thus can be computed in $\mathrm{O}(\log n)$ space. The overall proof idea is similar, but due to the uniformity condition, the proof becomes substantially more technical. We must not just show the existence of a threshold circuit family computing a transformer, but also show that this circuit family can be generated by a log-space Turing machine.

We first extend Lemma 1 to respect uniformity:

Lemma 2. *Let $f : \{0,1\}^* \to \{0,1\}^m$ be a linear-space computable function. There exists a Turing machine that, for all $n \in \mathbb{N}$ and $c \in \mathbb{R}^+$, uses at most $c \log n + \log m$ space to map input 1^n to a circuit of size at most $n^c + c \log n + m$ and depth 3 that computes f on inputs of size at most $c \log n$.*

Proof. We give the proof in the form of an algorithm to construct a circuit as a function of n and then justify its correctness and space complexity.

Algorithm. We first print $2c \log n$ nodes representing unnegated and negated input nodes.[15]

Now, we need to show how to construct nodes corresponding to n^c DNF terms. To this end, we loop over all possible inputs $x \in \{0,1\}^{c \log n}$ by maintaining the $c \log n$ bit binary representation of x (initialized with $0^{c \log n}$) and incrementing it by 1 at each step of the loop. We create a new $\wedge$ node i with $c \log n$ arguments, defined as follows. For $j \in [c \log n]$, we create an argument pointer to (unnegated) node j if $x_j = 1$ and to (negated) node $c \log n + j$ otherwise.

[14]Here, a TC^0 circuit family is a constant-depth, poly-size circuit family computing some function $\{0,1\}^* \to \{0,1\}^*$. While we define TC^0 for decision problems in Definition 4, it is standard and well-defined to extend the same term to refer to circuit families computing functions as well (Hesse, 2001).

[15]We ignore the initial unnegated input nodes when considering the size of the circuit.

Now, we construct nodes computing each of the m output nodes. We loop over $k \in [m]$, constructing a single node for each k. We loop over all $x \in \{0,1\}^{c \log n}$ analogously above to construct a list of arguments. By our linear-space computability assumption and because x has $c \log n$ bits, we can compute $f(x)$ as a subroutine in $\mathrm{O}(\log n)$-space to obtain $f_k(x)$. If $f_k(x) = 1$, we print node $2c \log n + j$ as an argument of node k.

Correctness. We show that this Turing machine maps input n to a serialized circuit computing f on inputs of size n. The first layer simply produces unnegated and negated input values. The second layer then produce all possible DNF terms. Finally, node k of the third layer computes the disjunction over all terms x such that $f_k(x) = 1$. Thus, node k of the third layer computes f_k.

Log Space. To complete the proof, we justify that M uses $\mathrm{O}(\log n + \log m)$ space. Looping over $x \in \{0,1\}^{c \log n}$ is accomplished by treating x as a binary number initialized to 0 and incrementing it at each step. Thus, the loop pointer for building the DNF terms takes $c \log n$ space to store. For building the m output nodes, we maintain a similar loop pointer as well as an index $k \leq m$, taking $c \log n + \log m$ space. Thus, the overall algorithm uses $c \log n + \log m$ space.

Thus, M uses $c \log n + \log m$ space to map 1^n to a circuit of size at most $n^c + c \log n + m$ and depth 3 that computes f on size $c \log n$ inputs. $\square$

We can leverage this lemma to derive the *uniform* analog of Theorem 1, as follows.

Theorem 2 (Uniform, main result)**.** *Any $c \log n$-precision depth-d transformer operating on inputs in Σ^n can be simulated by a logspace-uniform threshold circuit family of depth $3 + (9 + 2d_\oplus)d$.*

Proof. We will provide a proof by induction over transformer layers ℓ that there is a Turing machine M operating in $\mathrm{O}(\log n)$ space that, on input 1^n, outputs a circuit that simulates the transformer's computation on inputs of size n. This circuit is identical to the one in the proof of Theorem 1, and thus has the same circuit depth.

In the base case, we use log space to track a counter maintaining the current token i (between 1 and n) throughout the circuit construction. We construct gates encoding the constant i in binary. We can then apply Lemma 2 to construct a Turing machine that maps 1^n to a constant-depth threshold circuit computing $\mathbf{h}_i^0 = \phi(w_i, i)$.

In the inductive case, we assume we can output in $\mathrm{O}(\log n)$ space a circuit computing every value $\mathbf{h}_i^\ell$ in the previous layer ℓ. We will show that we can, in $\mathrm{O}(\log n)$ space, now output a circuit computing every value in layer $\ell + 1$.

As in Theorem 1, we first fix a head $\mathbf{a}_{ih}^{\ell+1}$ to simulate. Recall (Definition 8) that

$$\mathbf{a}_{ih}^{\ell+1} = \bigoplus_{j=1}^{n} \frac{s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)}{Z_i} \cdot \mathbf{h}_j^\ell.$$

By Lemma 2, we can generate a depth-3 circuit of size at most $z = n^{c'} + c' \log n + 1$, where $c' = 2c$ (since the input to f is of size $2c \log n$) that computes $s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$ for specific i, j. We do this sequentially for $1 \leq j \leq n$ and $1 \leq h \leq k$, padding each circuit with unused nodes so that each one has size exactly z, and the z-th node corresponds to the output. Thus, the indices of the output nodes for each of the columns will be $w_\ell + z(jk + h)$ for $1 \leq j \leq n$, where w_ℓ is the index of the last output node $\mathbf{h}_n^\ell$ of the previous layer.

At this point, we use the fact that for $p = c \log n$, the p-precision approximate sum of n p-precision numbers can be computed by a uniform threshold circuit (§A). We can thus use a Turing machine as a sub-routine to generate, on input 1^n, a k threshold circuits, where each has size z' that computes an $\oplus$ gate over n items of precision p each. We set the inputs of circuit h to be nodes $w_\ell + z(jk + h)$ for $1 \leq j \leq n$. By construction, this yields the normalizing constants $Z_i = \bigoplus_{j=1}^n s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$, whose value is located at the node at index $w_\ell + znk + z'$ for head h.

Using p-precision arithmetic operator circuits, we can now also generate a circuit to compute $\frac{s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)}{Z_i} \mathbf{h}_j^\ell$ for each $1 \leq j \leq n$ and $1 \leq h \leq k$, by using index $w_\ell + z(jk + h)$ as before for the value of $s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$ and index $w_\ell + znk + z'h$ for the normalizing constant Z_i of head h. Here too we use circuits of identical size z'', making $w_\ell + k(zn + z' + z''i)$ the index of the output nodes of these n circuits. Next, we again employ a $\oplus$ circuit of size z', similar to the computation of Z_i, to compute the sum of these n values. Finally, we compute $h_i^{\ell+1}$ by applying f via Lemma 2.

Note that this requires keeping only ℓ, i, and n in memory, each of which takes $\mathrm{O}(\log n)$ bits.

We repeat this process for all $1 \leq i \leq n$ to compute the entire $\ell + 1$ layer, which finishes the inductive step: if we can output a circuit computing layer ℓ in $\mathrm{O}(\log n)$ space, then we can do the same for layer $\ell + 1$. □

Because the depth derived in Theorem 2 is constant with respect to n, it follows that:

Corollary 2.1 (Uniform, main result)**.** *Any log-precision transformer can be simulated by a uniform* TC^0 *circuit family.*

7 Lower Bounds for Instruction Following and Advice Transformers

So far, we have shown that uniform TC^0 is an upper bound for log-precision transformers. Is this upper bound tight, i.e., also a lower bound? While we do not answer this question here, we address a related question as a first step: we construct a transformer that can evaluate TC^0 circuits on binary inputs, showing that transformers can compute any TC^0 function when their input is augmented with the right ''instructions''.

More formally, we consider the **Circuit Value Problem (CVP)** (Ladner, 1975), also referred to as the Circuit Evaluation Problem, where the input is a Boolean circuit C and a string $x \in \{0,1\}^n$, and the task is to return the value of $C(x) \in \{0,1\}$. This problem is known to be complete for the class P under AC^0 reductions (Ladner, 1975). We will assume C is serialized as described in §3 and prove that log-precision transformers can evaluate any TC^0 circuit. Note that this is an extension of the typical CVP since the circuit has threshold gates, not just standard AND/OR gates.

It is known that LSTMs cannot evaluate Boolean formulae (Merrill, 2020), a special case of the CVP. In contrast, we show that transformers can.

To demonstrate the practicality of our lower bound construction, we will not just prove the existence of transformers that can evaluate TC^0 circuits but also specify concrete choices for the positional embedding scheme and the class of attention functions that are sufficient to do so.

Fractional Positional Embeddings. For a vector $\mathbf{x}$ and scalar y, let $\langle \mathbf{x}, y \rangle$ be the vector appending y onto $\mathbf{x}$.[16] For $\sigma \in \Sigma$, let $v(\sigma)$ be the one-hot encoding of σ into $\mathbb{R}^{|\Sigma|}$. For $w \in \Sigma^*$, we define the fractional positional embedding at token i as

$$\phi(w_i, i) = \langle v(w_i), i/n \rangle.$$

Saturated Attention. We imagine $f(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)$ is computed via saturated attention (cf. Merrill et al., 2022), which provides a simple model of the types of attention we can expect to be learned in transformers (Merrill et al., 2021). First, queries are computed as $\mathbf{q}_i = \mathbf{Q}\mathbf{h}_i^\ell$, and then keys $\mathbf{k}_j = \mathbf{K}\mathbf{h}_j^\ell$. Define the dot-product attention score $\sigma_{ij} = \mathbf{q}_i^\top \mathbf{k}_j$. We can then define saturated attention as

$$s(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell) = \begin{cases} 1 & \text{if } \sigma_{ij} = \max_k \sigma_{ik} \\ 0 & \text{otherwise.} \end{cases}$$

After normalization, saturated attention creates a distribution that is uniform over a subset of positions. Thus, it is capable of parameterizing hard attention, uniform attention over the full sequence, and various attention patterns in between.

Simple Pooling Functions. For simplicity, we assume pooling functions f are thresholded linear functions of their inputs. Thus, they could be implemented by a feedforward neural net. Without loss of generality, we let attention heads have a value function, which can be folded into the pooling function from the last layer (see §4).

Now, we are ready to present the main result. Our construction below is specific to the circuit serialization scheme discussed in §3, but can be extended to other serializations as well.

Lemma 3. *For all d, there exists a transformer with fractional positional embeddings, saturated attention, thresholded linear pooling functions, and depth $2d$ that, for any threshold circuit C of depth d, maps input $\langle C, x \rangle$ to the value $C(x)$.*

Proof. We will construct a pair of two transformer layers that evaluate all the nodes at depth ℓ in the threshold circuit, for any ℓ. It follows that a transformer of depth $2d$ can compute the value $C(x)$.

We refer to a token of type $\mathtt{X}$ as an *input node*. Similarly, we call a token of type Dir a *gate node*. Finally we call a token of type $\mathtt{\&}$ an *argument*.

Base Case: Input Nodes. We construct one attention layer that attends uniformly over all positions whose value returns 1 if $w_i = \mathtt{X}$ and 0 otherwise. Thus, this head computes $\#(\mathtt{X})/n$,

[16] I.e., $\langle \mathbf{x}, y \rangle_i = x_i$ for $1 \leq i \leq |\mathbf{x}|$, and y if $i = |\mathbf{x}| + 1$.

where $\#(\texttt{X})$ is the number of occurrences of $\texttt{X}$ in w. We then define a second layer that, at input node i, retrieves the token at position j defined by

$$j = \frac{1 - \#(\texttt{X}) + i}{n}.$$

j is the index of the ith input value. Thus, after this layer, each input node i stores its value x_i.

In the base case, we also construct an attention head that, at the ith gate node, counts the fraction of nodes (out of n) that are gate nodes to the left of the current node. Thus, this head computes i/n. Also at gate node i, we construct an attention head that counts the fraction of nodes to its right before the next $\texttt{\&}$ node that have value 1. This head thus has value k_i/m_i where k_i is the threshold value of the i-th gate and m_i is its arity. We apply the same construction at each argument $\texttt{\&}$ to count the $\texttt{1}$'s that follow until the next non-$\texttt{1}$ symbol.

Finally, using the first attention layer, we have each $\texttt{J}$ node attend to the first argument symbol $\texttt{\&}$ to its left and retrieve its index j/n. Then, in the second attention layer, each argument attends uniformly over all nodes with values j/n. The net effect is for each argument node to store j/n, i.e., the pointer it is encoding in unary as $\texttt{\&1}^j$.

Inductive Case: Gate Nodes. By our inductive assumption over prior layers, all tokens corresponding to circuit nodes at depth $\leq \ell$ contain their appropriate value. We now construct 2 transformer layers to evaluate gate nodes at depth $\ell + 1$.

In the first attention layer, each argument j attends to the the closest gate node i to its left, which is the gate it belongs to. Recall from the base case that argument j already stores j/n. Each argument $\texttt{\&0}^j$ attends with position key j/n to gate node j and retrieves its value in the previous layer.

The second attention layer applies at gate nodes, not arguments. At gate i of arity m_i, we set the attention $s(i, j)$ to indicate whether argument j belongs to gate node i, which holds for exactly m arguments. We set the attention value to be the binary value of the referent of argument j. Thus, the attention head computes c_i/m_i, where c_i is the number of arguments of node i that are 1. We repeat this for all gate nodes.

At this point, for the i-th gate node, we have computed both c_i/m_i and k_i/m_i. Thresholding $(c_i - k_i)/m_i$ at 0 allows us to decide, based on whether Dir is <= or >=, whether the current gate node should output a 0 or a 1. Repeating this for all gates at layer $\ell + 1$ completes the inductive step: We can evaluate all gate nodes in this layer. $\square$

Theorem 3. *Depth-$2d$ transformers can solve CVP for depth-d TC^0 circuits.*

7.1 Instruction Following

CVP is closely related to *instruction learning* (Brown et al., 2020) and *instruction following* tasks (Finlayson et al., 2022). The latter task setup provides a transformer two inputs: a regular expression r as an ''instruction'', and $z \in \{0,1\}^*$. The goal of the task is to return whether z belongs to the regular language represented by r. Viewed from this lens, the circuit evaluation setup asks: *Can transformers follow instructions provided in the form of a circuit?* As discussed below, our result says the answer is *yes* for all constant depth threshold circuits. This, to the best of our knowledge, provides the first non-trivial lower bound for transformers in the instruction learning setting.

Formally, an instruction I is any description[17] of a function f_I of $\{0,1\}^*$. We say a transformer correctly follows an instruction I if, for all $x \in \{0,1\}^*$, it correctly computes $f_I(x)$ on input $\langle I, x\rangle$. A non-uniform instruction description is a family of length-specific descriptions $\{I_n\}_{n=1}^{\infty}$. We say a transformer correctly follows a non-uniform instruction family $\{I_n\}$ if, for all n and all $x \in \{0,1\}^n$, it correctly computes $f_I(x)$ on input $\langle I_n, x\rangle$. The non-uniform description $\{I_n\}$ may take any form. When it forms a TC^0 circuit family, we refer to it as a TC^0 instruction description. Since Theorem 3 constructs a transformer that can evaluate any TC^0 circuit, it follows that:

Corollary 3.1. *There exists a depth-$2d$ transformer that can correctly follow any depth-d TC^0 instruction description.*

Thus, transformers with simple position embeddings, attention, and pooling functions can simulate any instruction provided in the form of a TC^0 circuit. We note that while it is unknown whether the class of regular languages, considered by Finlayson et al. (2022), is contained in TC^0, the other side is known: There *are* problems computable by TC^0 circuits that are not computable

[17]Formally, a function description is a fixed-size program to compute that function under some model of computation.

by a regular language. These include problems involving counting and arithmetic, which are beyond regular languages. Our results thus expand the known kinds of instructions transformers are able to follow, at least with hand-constructed weights.

7.2 Advice Transformers

We can also view circuit evaluation abilities of transformers (Lemma 3) from the lens of *advice-taking Turing machines* which, in addition to their usual input, are also provided an input length dependent (but input independent) advice string. For instance, P/poly is the class of problems decidable in polynomial time when the Turing machine is given an advice string of size polynomial in the input length (cf. Arora and Barak, 2009).

In the same vein, let T/poly be the class of log-precision, constant-depth transformers with polynomial advice strings. In other words, on an input of size n, we allow the transformer to receive an additional poly(n) bits of input that cannot depend on the standard input. Now let $\{C_n\}_{n=1}^{\infty}$ be a circuit family demonstrating that a problem is in non-uniform TC^0. Then, by passing the description of C_n as advice for input length n, it immediately follows from Lemma 3 that advice transformers can simulate non-uniform TC^0:

Corollary 3.2. *Non-uniform* $\mathsf{TC}^0 \subseteq \mathsf{T}/\mathsf{poly}$.

Since non-uniform TC^0 even contains some undecidable languages (Arora and Barak, 2009, Claim 6.8), T/poly is clearly a very powerful class and a strict superset of T, the class of decision problems recognized by transformers (which are all decidable). Thus, a problem in T/poly cannot always be solved by a transformer on its own. However, if given a description of *how* to do so (''advice'') in the form of a TC^0 circuit, our result shows that a transformer *could* solve that problem.

8 Conclusion

Answering two open questions from Merrill et al. (2022), we prove log-precision transformers with any (including soft) attention can be simulated by *uniform* constant-depth threshold circuits. This establishes *thresholded addition* as a fundamental operation for understanding the computational model of transformers: Any log-precision transformer can be re-expressed as a polynomial number of threshold gates stacked to a constant depth. This result also establishes potential limits on the computational power of log-precision transformers; e.g., if $\mathsf{L} \subset \mathsf{P}$, transformers cannot compute all poly-time functions. They are certainly very far from being universal. The intuition at the heart of this result is that forcing a model to be highly parallelizable likely sacrifices its expressiveness. Since parallelism seems essential to pretraining any massive model at scale, any large language model—transformer or otherwise—may suffer from a similar tradeoff.

Acknowledgments

The authors are grateful for the valuable feedback from the anonymous reviewers and the TACL action editor. They also thank Paul Beame and colleagues at AI2 including Kyle Richardson, Michal Guerquin, Peter Clark, Tushar Khot, and especially Matthew Finlayson, whose empirical findings about instruction learning inspired §7. Feedback from Sam Bowman, Arya McCarthy, Roma Patel, and Lena Strobl, and discussions with the FLaNN, ML for Code (MILA), and Foundations of Language Processing (Umeå) research groups helped improve earlier drafts. The authors also appreciate Rahul Santhanam's feedback. This work was funded in part by NSF award 1922658. William Merrill was supported by an NSF graduate research fellowship and by AI2.

References

Eric Allender. 1999. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*.

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

Arin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya

Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Tom Bylander. 1991. Complexity results for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*. https://doi.org/10.1016/B978-0-08-049944-4.50008-2

Andrew Chiu, George I. Davida, and Bruce E. Litow. 2001. Division in logspace-uniform nc1. *RAIRO Theoretical Informatics and Applications*, 35:259–275. https://doi.org/10.1051/ita:2001119

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Matthew Finlayson, Kyle Richardson, Ashish Sabharwal, and Peter Clark. 2022. 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*.

Raymond Greenlaw, James M. Hoover, and Walter L. Ruzzo. 1991. A compendium of problems complete for P. Technical Report TR91-11, University of Alberta.

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

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810. https://doi.org/10.1162/tacl_a_00490

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

Neil Immerman. 2012. *Descriptive Complexity*. Springer Science & Business Media.

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

Richard E. Ladner. 1975. The circuit value problem is log space complete for P. *ACM SIGACT News*, 7(1):18–20. https://doi.org/10.1145/990518.990519

Harry R. Lewis and Christos H. Papadimitriou. 1982. Symmetric space-bounded computation. *Theoretical Computer Science*, 19:161–187. https://doi.org/10.1016/03045-3975(82)90058-

William Merrill, Vivek Ramanujan, Yoav Goldberg, Roy Schwartz, and Noah A. Smith. 2021. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/2021.emnlp-main.133

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

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

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

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140).

Omer Reingold. 2008. Undirected connectivity in log-space. *Journal of the ACM*, 55:17:1–17:24. https://doi.org/10.1145/1391289.1391291

Leslie G. Valiant. 1979. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201. https://doi.org/10.1016/0304-3975(79)90044-6

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

A Iterated p-Precision Float Addition

We interpret a p-bit string x as a p-precision float by taking the first $p/2$ bits[18] of x as a signed integer m encoding the *mantissa* and the remaining $p/2$ bits of x as another signed integer e encoding the *exponent*. A float with mantissa m and exponent e, denoted $\langle m, e \rangle$, encodes $m \cdot 2^e$.

Computing the sum of n n-bit integers (known as iterated addition or simply summation) is well-known to be in uniform TC^0 (Hesse, 2001; Chiu et al., 2001). We leverage this fact to show that the same holds for the sum of n $\mathrm{O}(\log n)$-precision floats. A subtlety of adding p-precision floats is that their sum can require more than p bits to represent precisely as a float. For instance, while each of 2^r and 1 is representable with a (signed) mantissa of only 2 bits, their exact sum, $2^r + 1$, requires a mantissa of $r + 1$ bits. Hence, p-precision transformers must sacrifice some precision when performing summation.

We define float addition by mapping the floats to integers, adding the integers exactly, and then mapping the sum back to a float (with possible loss of precision). Let $I_q^{\max} = 2^q - 1$ be the greatest q-bit signed integer, and $I_q^{\min} = -I_q^{\max}$. Let $F_p^{\max}$ be the greatest value representable by a p-precision float. Since the exponent of a float ϕ can be negative and represent a fraction, we rescale ϕ by $2^{-I_{p/2}^{\min}}$ when mapping it to an integer $g_p(\phi)$:

Definition 13. The integer mapping of a p-bit float $\phi = \langle m, e \rangle$ is defined as $g_p(\phi) = m \cdot 2^{e - I_{p/2}^{\min}}$.

Definition 14. The p-truncated float mapping of an integer z is defined as $f_p(z) = \langle m, e \rangle$ where[19]

$$\begin{aligned} m &= \mathrm{rshift}(z, \max\{0, \mathrm{sizeof}(z) - p/2\}) \\ e &= \mathrm{sizeof}(z) - \mathrm{sizeof}(m) + I_{p/2}^{\min} \end{aligned}$$

when $e \leq I_{p/2}^{\max}$; otherwise (i.e., when $z > F_p^{\max}$), we set $m = e = I_{p/2}^{\max}$ to properly handle overflow.

Definition 15 (Iterated p-precision float addition)**.** We define the sum of k p-precision floats as

$$\bigoplus_{i=1}^{k} \phi_i = f_p\left(\sum_{i=1}^{k} g_p(\phi_i)\right).$$

We first verify that Definition 14 closely approximates exact addition.

Lemma 4. *Let $\phi = \langle e, m \rangle$ be a float such that $|\phi| \leq F_p^{\max}$ and $e \geq I_{p/2}^{\min}$. Then ϕ and $f_p(g_p(\phi))$ differ by a factor of at most* $1 \pm 2^{-p/2+2}$.

Proof. Let $z = g_p(\phi)$, which is well-defined because of the precondition $e \geq I_{p/2}^{\min}$ of the lemma. Let $\phi' = \langle m', e' \rangle = f_p(z)$.

[18]We assume w.l.o.g. that p is even.

[19]For $x \neq 0$, $\mathrm{sizeof}(x) = \lfloor \log |x| \rfloor + 2$; $\mathrm{sizeof}(0) = 2$. For $y \geq 0$, $\mathrm{rshift}(x, y)$ right-shifts x by y bits.

First consider the easy case where $\mathrm{sizeof}(z) \leq p/2$. Then $m' = z$ and $e' = I^{\min}_{p/2}$ from Definition 14. Since $z = m \cdot 2^{e - I^{\min}_{p/2}}$ by Definition 13, it follows that ϕ and ϕ' have exactly the same value.

Now assume $\mathrm{sizeof}(z) > p/2$. It follows from the precondition $|\phi| \leq F^{\max}_p$ of the lemma that there is no overflow when applying Definition 14 to compute $\langle m', e' \rangle$. Thus m' consists of the $p/2$ highest-order bits (including the sign bit) of z and $e' = \ell + I^{\min}_{p/2}$, where $\ell = \mathrm{sizeof}(z) - p/2$ is the number of bits truncated from z to obtain m'. Let δ denote the (non-negative) integer formed by the ℓ lowest-order bits of z that are truncated. Then $\delta \leq 2^\ell - 1 = 2^{\mathrm{sizeof}(z) - p/2} - 1 < z \cdot 2^{-p/2+2}$.

Recall that the value of ϕ is $g_p(\phi) \cdot 2^{-I^{\min}_{p/2}} = z \cdot 2^{-I^{\min}_{p/2}}$. By the above argument, we also have that the value of ϕ' is within $(z \pm \delta) \cdot 2^{-I^{\min}_{p/2}}$, which is within $z \cdot (1 \pm 2^{-p/2+2}) \cdot 2^{-I^{\min}_{p/2}}$. Thus, ϕ and ϕ' are within a factor of $1 \pm 2^{-p/2+2}$ of each other. □

Finally, we show that, with log precision, computing $\oplus$ (Definition 14) is in uniform TC^0.

Lemma 5. *Let $p \leq c \log n$ and $\phi = \bigoplus_{i=1}^k \phi_i$, where $k \leq n$ and each ϕ_i is p-precision. Then ϕ is computable by a constant-depth uniform threshold circuit of size poly(n).*

Proof. Let $N = c \log n + 2n^c$. We first use Lemma 1 to map each $\phi_i = \langle m_i, e_i \rangle$ to the integer $z_i = m_i \cdot 2^{e_i - I^{\min}_{p/2}}$, which has size $\mathrm{sizeof}(m_i) + (e_i - I^{\min}) \leq p/2 + 2 \cdot 2^{p/2} \leq c \log n + 2n^c = N$. For $1 \leq i \leq k$, we pad z_i to N bits, and for $k < i \leq N$, we create an N-bit integer $z_i = 0$. We can then compute $z = \sum_{i=1}^k z_i$ with a constant-depth uniform threshold circuit of size poly(N) using the classical construction to sum N N-bit integers (cf. Immerman, 2012, exercise 5.29). The size of this circuit is also polynomial in n by the definition of N. Finally, we compute $f^\dagger(z)$ using a constant-depth AND/OR circuit. □