The Exact Expressive Power of Fixed-Precision Looped Padded Transformers

Anej Svete[1,2*] **Will Merrill**[2] **Ashish Sabharwal**[2]
[1]ETH Zürich [2]Allen Institute for AI
`asvete@inf.ethz.ch`

Abstract

We establish the exact expressivity of *fixed-precision* looped padded transformers. With $\mathcal{O}(\log N)$ width, $\log^d N$ depth, and $\mathtt{poly}(N)$ padding (in input length N), they are equivalent to L-uniform $\mathtt{AC}^{\mathtt{d}}$ circuits. This extends recent results showing the equivalence of transformers with $\mathcal{O}(\log N)$ *precision*, $\log^d N$ depth, and $\mathtt{poly}(N)$ padding to FO-uniform $\mathtt{TC}^{\mathtt{d}}$. Our result exposes fixed precision's expressivity cost: A drop from $\mathtt{TC}^{\mathtt{d}}$ to $\mathtt{AC}^{\mathtt{d}}$ due to inability to compute threshold functions.

1 Introduction

Transformers are the foundational architecture of modern language models. Understanding their computational expressivity—what they can and cannot compute—is important for both theoretical insights and practical applications. Recent work has described transformer expressivity with circuit complexity, revealing tight connections between transformer architectures and parallel computation models (Merrill & Sabharwal, 2023; Li et al., 2024; London & Kanade, 2025; Merrill & Sabharwal, 2025a, *inter alia*).

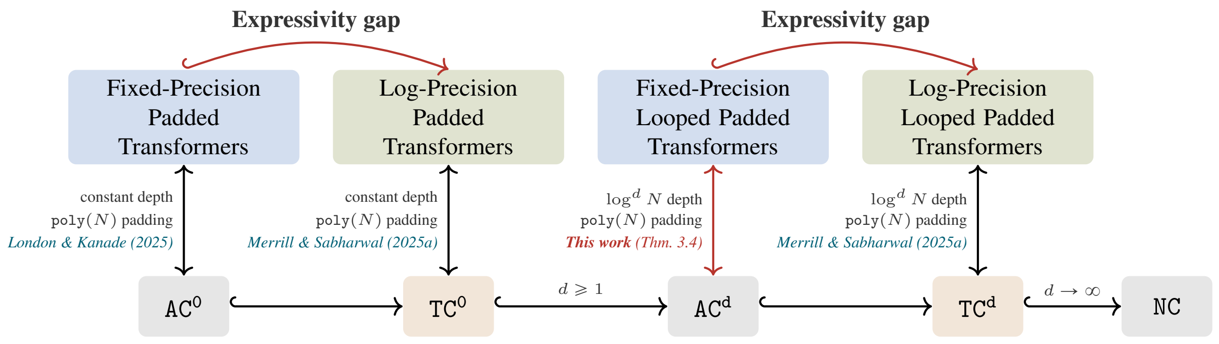

Figure 1: The landscape of transformer expressivity across precision and depth regimes. Colored boxes represent transformer computational models (fixed-precision transformers are L-uniform); brown boxes represent FO-uniform complexity classes while gray boxes represent L-uniform subclasses. Two-sided arrows ($\mathcal{X} \leftrightarrow \mathcal{Y}$) indicate equality; hooked arrows ($\mathcal{X} \hookrightarrow \mathcal{Y}$) indicate (non-strict) inclusion of $\mathcal{X}$ in $\mathcal{Y}$.

Different choices in how numerical precision, width, and depth scale with context length can fundamentally affect the computational power of transformers. To distinguish N input positions, a transformer must compute at least N distinct representations, requiring $\mathtt{p} \cdot D = \Omega(\log N)$, where $\mathtt{p}$ is numerical precision

[*] This research was conducted while interning at the Allen Institute for AI.

(bits per value) and D is width. Different ways of satisfying this constraint lead to different computational models, and it remains unclear which best characterizes practical transformers. Prior work has characterized *logarithmic-precision* transformers with constant width, i.e., $\mathtt{p} = \Theta(\log N)$ and $D = \Theta(1)$. Merrill & Sabharwal (2025a) show that such transformers with $\mathcal{O}(\log^d N)$ depth and polynomial padding are equivalent to $\mathtt{FO}$-uniform $\mathtt{TC^d}$ for $d \geqslant 1$, the class of (log-space uniform) threshold circuits of depth $\mathcal{O}(\log^d N)$.[1]

These results do not directly carry over to the *fixed-precision* regime, i.e., $\mathtt{p} = \Theta(1)$ and $D = \Theta(\log N)$. Fixed precision requires information such as counts and pointers to positions in the input string to be stored in a "distributed" manner across dimensions, rather than concentrated into individual values. This makes it difficult for the model to perform basic arithmetic with counts and pointer values, and limits expressivity compared to log-precision transformers: While log-precision transformers can compute certain $\mathtt{TC^0}$ functions, constant-precision transformers with logarithmic width (with or without polynomial padding) fall within $\mathtt{AC^0}$ (Li et al., 2024; London & Kanade, 2025). However, a complete characterization of the expressivity of fixed-precision *looped* padded transformers remains open, and is the focus of this work.

In this work, we describe the exact expressivity of fixed-precision looped padded transformers, focusing on the role of precision and width. The dependence of model width on context size means that we construct a separate transformer for each input length. This calls for some caution: in principle, we could hard-code a different transformer for each input length, leading to unrealistic models.[2] Therefore, we study the scaling of transformer width with $\mathtt{L}$-uniform transformers (London & Kanade, 2025). This enforces that the transformers can be built by a realistic computational model—a logarithmic-space Turing machine. Our main result shows that such transformers with $\log^d N$ looping and $\mathtt{poly}(N)$ padding are equivalent to $\mathtt{L}$-uniform $\mathtt{AC^d}$ circuits, which is believed to be a strictly weaker class than the $\mathtt{FO}$-uniform $\mathtt{TC^d}$ characterized by Merrill & Sabharwal (2025a) for the log-precision case. We find the inclusion of the $\mathtt{L}$-uniform *positional encodings* to be particularly important, as they enable fixed-precision transformers to use positional information that they themselves could not compute. As d increases, fixed-precision transformers approach $\mathtt{NC} = \bigcup_{d \geqslant 0} \mathtt{AC^d}$, but remain strictly weaker than their log-precision counterparts matched for depth (assuming $\mathtt{AC}^d \neq \mathtt{TC}^d$). Fig. 1 summarizes this landscape across different precision and depth regimes, positioning our contribution alongside prior work showing that log-precision transformers achieve $\mathtt{FO}$-uniform $\mathtt{TC^d}$ (Merrill & Sabharwal, 2025a) and that constant-depth fixed-precision transformers achieve $\mathtt{L}$-uniform $\mathtt{AC^0}$ (London & Kanade, 2025).

2 Preliminaries

2.1 Notation

Let Σ be an alphabet, a finite set of **symbols**. A **language** $\mathcal{L}$ is a subset of $\Sigma^* \overset{\text{def}}{=} \bigcup_{N \in \mathbb{N}} \Sigma^N$, the set of all strings. We denote the concatentation of two strings $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \Sigma^*$ as $\boldsymbol{w}_1 \circ \boldsymbol{w}_2$ or simply $\boldsymbol{w}_1\boldsymbol{w}_2$. A **language recognizer** is a function $R\colon \Sigma^* \to \{\mathtt{0}, \mathtt{1}\}$, where $\mathtt{0}$ and $\mathtt{1}$ are designated reject and accept symbols. R's language is $\mathcal{L}(R) \overset{\text{def}}{=} \{\boldsymbol{w} \in \Sigma^* \mid R(\boldsymbol{w}) = \mathtt{1}\}$. Two recognizers R_1 and R_2 are **equivalent** if and only if $\mathcal{L}(R_1) = \mathcal{L}(R_2)$. We use $\mathtt{poly}(N) \overset{\text{def}}{=} \{f\colon \mathbb{N} \to \mathbb{N} \mid \exists K > 0, f(N) = \mathcal{O}(N^K)\}$ to denote the set of functions with at most polynomial growth rate.

[1]In contrast, fixed-depth ($d = 0$) log-precision transformers are equivalent to $\mathtt{FO}$-uniform $\mathtt{TC^0}$ (Merrill & Sabharwal, 2025a).

[2]Without a uniformity constraint, circuit families (and thus transformer families) become unrealistically powerful. For example, consider the unary language $\{1^N \mid \text{the } N^{\text{th}} \text{ Turing machine halts}\}$ under some fixed enumeration of Turing machines. This undecidable language is recognizable by a non-uniform $\mathtt{AC^0}$ circuit family, since we can hard-code the correct answer for each input length N into the circuit C_N. Uniformity conditions prevent such pathological cases by requiring a single, feasible algorithm to construct all circuits in the family.

2.2 Circuit Complexity

Computational circuits are a model of parallel computation. They have been widely used in the study of the expressivity of neural networks. Circuits process binary input strings through a series of logical operations to produce binary outputs.[3] Formally, a **boolean circuit** is a directed acyclic graph where source nodes represent the N-bit input, and a single sink node represents the output. Non-source vertices are called **gates** and are labeled with logical operations (e.g., AND, OR, NOT). The **size** of a circuit is the number of gates, and its **depth** is the longest path from any input to the output.

Circuit families process input strings of variable length. A circuit family is a sequence of circuits $\mathcal{C} \overset{\text{def}}{=} \{C_N\}_{N \in \mathbb{N}}$ where C_N processes inputs of length N. A circuit family is said to recognize a language if for any given input string, the corresponding circuit outputs 1 if and only if the string is in the language.

A **circuit complexity class** is a set of circuit families that satisfy certain constraints on size, depth, and the types of gates used. This paper focuses on two common classes:

- $\mathtt{AC}^{\mathtt{d}}$: Circuits with NOT, AND, and OR gates that have unbounded fan-in and depth $\mathcal{O}(\log^d N)$.
- $\mathtt{TC}^{\mathtt{d}}$: The extension of $\mathtt{AC}^{\mathtt{d}}$ that adds **threshold gates**, which output 1 if the sum of their inputs exceeds a given threshold. It is known that $\mathtt{AC}^0 \subsetneq \mathtt{TC}^0$ and $\mathtt{AC}^{\mathtt{d}} \subseteq \mathtt{TC}^{\mathtt{d}}$. For example, PARITY, the language of binary strings with an even number of 1s, is in $\mathtt{TC}^0$ but not in $\mathtt{AC}^0$ (Furst et al., 1984).

Without additional constraints, circuit families can recognize undecidable languages by having arbitrary, "hard-coded" solutions for each input length. To avoid this and ensure the model of computation is realistic, we can impose a **uniformity** condition. A circuit family is **uniform** if there exists a Turing machine that, given an input of 1^N, can generate a description of the circuit C_N. In particular, a circuit class is **L-uniform** if a Turing machine using $\mathcal{O}(\log N)$ space can construct its description from the input 1^N. This ensures the circuits for different input lengths are related by a systematic procedure.

2.3 Finite-precision Fixed-point Arithmetic

Our computation models perform operations with finite-precision **fixed-point arithmetic** (Li et al., 2024; Saunshi et al., 2025; London & Kanade, 2025).

Definition 2.1 (Fixed-point representation)**.** *Let* $\mathsf{p} \in \mathbb{N}$ *be the number of bits devoted to each of the integer and fractional parts. We use* $\mathbb{F}_{\mathsf{p}}$ *to denote the set*

$$\mathbb{F}_{\mathsf{p}} \overset{\text{def}}{=} \{x_{\pm} \cdot a \cdot 2^{-\mathsf{p}} \mid x_{\pm} \in \{-1, 1\}, a \in \{0, 1, \ldots, 2^{2\mathsf{p}} - 1\}\} \tag{1}$$

We define $B_{\mathbb{F}} \overset{\text{def}}{=} \max \mathbb{F}_{\mathsf{p}} = 2^{\mathsf{p}} - 2^{-\mathsf{p}}$. All values exceeding $B_{\mathbb{F}}$ are considered out of range and are rounded to $B_{\mathbb{F}}$. Note, however, that $B_{\mathbb{F}}$ does *not* behave like infinity—it does not "consume" all subsequent operations. For example, for any non-negative $x \in \mathbb{F}_{\mathsf{p}}$, $B_{\mathbb{F}} - x \neq B_{\mathbb{F}}$ is a valid number.

To handle the results of arithmetic operations that may not be exactly representable in the fixed-point format, we define a standard for rounding.

Definition 2.2 (Rounding)**.** *For any* $x \in \mathbb{R}$ *and any closed subset* $\mathbb{F}$ *of* $\mathbb{R}$ *containing 0, we define* $\mathtt{round}(x, \mathbb{F})$ *as the closest number to* x *in* $\mathbb{F}$*. In case of a tie, the value with the smaller absolute value is chosen.*

We denote the rounding operation as $[\cdot]_{\mathsf{p}} \overset{\text{def}}{=} \mathtt{round}(\cdot, \mathbb{F}_{\mathsf{p}})$. This operation is applied to vectors and matrices element-wise. All binary operations are defined by first performing the ideal mathematical operation and then rounding the result to the nearest representable value in $\mathbb{F}_{\mathsf{p}}$. Division by zero is considered an error condition resulting in an incorrect output. We also note that ${}^{B_{\mathbb{F}}}\!/\!{}_{2} = 2^{\mathsf{p}-1} - 2^{-\mathsf{p}}$.

For operations involving more than two numbers, rounding is applied iteratively.

[3]By representing symbols from any alphabet with binary encodings, circuits (or circuit functions) can be used to process strings over any finite alphabet. We focus on binary strings for simplicity.

Definition 2.3 (Summation with iterative rounding)**.** *For* $\mathsf{p}, N \in \mathbb{N}$ *and* $\boldsymbol{x} \in \mathbb{R}^N$*, we define summation with iterative rounding to* p *fractional bits as the function* $\mathrm{SUM}_{\mathsf{p}} \colon \bigcup_{N \in \mathbb{N}} (\mathbb{F}_{\mathsf{p}})^N \to \mathbb{F}_{\mathsf{p}}$*, where for any* $N \in \mathbb{N}^+$ *and* $\boldsymbol{x} \in (\mathbb{F}_{\mathsf{p}})^N$*:*

$$\mathrm{SUM}_{\mathsf{p}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \left[\dots \left[[x_1 + x_2]_{\mathsf{p}} + x_3 \right]_{\mathsf{p}} + \dots + x_N \right]_{\mathsf{p}} \tag{2}$$

This iterative rounding process is not associative and the order of operations can affect the final result. Based on this, we can also define more complex operations such as the **fixed-point inner product** $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathsf{p}} \stackrel{\text{def}}{=} \mathrm{SUM}_{\mathsf{p}}(\boldsymbol{x} \odot \boldsymbol{y})$, where $\odot$ denotes the element-wise product of two vectors, and **fixed-point matrix product** for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, where $(\boldsymbol{A} \times_{\mathsf{p}} \boldsymbol{B})_{i,j} \stackrel{\text{def}}{=} \langle (\boldsymbol{A}_{i,:})^\top, \boldsymbol{B}_{:,j} \rangle_{\mathsf{p}}$. These operations will be used in the definition of fixed-precision transformers in §2.4.

2.4 Transformers and Transformer Families

For a fixed input length $N \in \mathbb{N}$, a transformer $\mathcal{T}_N$ consists of:

(1) a **symbol embedding** $\boldsymbol{e} \colon \Sigma \to \mathbb{F}^D$ for $w \in \Sigma$,
(2) a **positional encoding** $\boldsymbol{p} \colon \mathbb{N} \times \mathbb{N} \to \mathbb{F}^D$,
(3) L **layers** $\boldsymbol{\tau}^{(1)}, \dots, \boldsymbol{\tau}^{(L)}$, each of which consists of two sub-layers: A self-attention layer and a position-wise fully-connected feed-forward network f,[4] and
(4) a classification **output layer** $\boldsymbol{o}$ of the form $\boldsymbol{o} \colon \mathbb{F}^D \to \{\mathtt{0}, \mathtt{1}\}$.

Each layer has its own parameters and is indexed by the layer name and the depth for attention and feedforward layers. We use D to denote the **width** of a transformer. A transformer with layers $\boldsymbol{\tau}^{(1)}, \dots, \boldsymbol{\tau}^{(L)}$ computes $\boldsymbol{h}_n^{(l)} \in \mathbb{F}^D$ for $l \in \{1, \dots, L\}$ and each position $n \in [N]$ in the input string $\boldsymbol{w} = w_1 \cdots w_N \in \Sigma^*$ as follows:[5]

$$\boldsymbol{h}_n^{(0)} \stackrel{\text{def}}{=} \boldsymbol{e}(w_n) + \boldsymbol{p}(n, N) \in \mathbb{F}^D \text{ for } n \in [N] \tag{3a}$$

$$\boldsymbol{H}^{(l)} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{h}_1^{(l)\top} & \cdots & \boldsymbol{h}_N^{(l)\top} \end{pmatrix}^\top \in \mathbb{F}^{N \times D} \tag{3b}$$

$$\boldsymbol{Q}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_Q^{(l)}, \quad \boldsymbol{K}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_K^{(l)}, \quad \boldsymbol{V}^{(l)} \stackrel{\text{def}}{=} \boldsymbol{H}^{(l)} \boldsymbol{W}_V^{(l)} \quad \in \mathbb{F}^{N \times D} \tag{3c}$$

$$\boldsymbol{G}^{(l)} \stackrel{\text{def}}{=} \mathrm{softmax}(\boldsymbol{Q}^{(l)} \boldsymbol{K}^{(l)\top}) \boldsymbol{V}^{(l)} + \boldsymbol{H}^{(l)} \in \mathbb{F}^{N \times D} \tag{3d}$$

$$\boldsymbol{H}^{(l+1)} \stackrel{\text{def}}{=} \boldsymbol{G}^{(l)} + f(\boldsymbol{G}^{(l)}) \in \mathbb{F}^{N \times D} \tag{3e}$$

We say that the lth layer $\boldsymbol{\tau}^{(l)}$ computes the function $\boldsymbol{\tau}^{(l)} \colon \mathbb{F}^{N \times D} \to \mathbb{F}^{N \times D}$, defined as $\boldsymbol{\tau}^{(l)} \colon \boldsymbol{H}^{(l-1)} \mapsto \boldsymbol{H}^{(l)}$ for $l \in \{1, \dots, L\}$. We also denote with $\mathcal{T}$ the function $\mathcal{T} \colon \Sigma^* \to \mathbb{F}^{N \times D}$, defined as $\mathcal{T} \colon \boldsymbol{w} \mapsto \boldsymbol{H}^{(L)}$.

We focus on **constant-precision logarithmic-width** transformers. Thus, all operations in the transformer architecture are defined using fixed-point arithmetic from §2.3 for some fixed $\mathsf{p} \in \mathbb{N}$. This both limits what can be computed as well as enables the construction of various theoretical gadgets that leverage the bounded precision to implement logical operations and attention patterns. These are collected in §A. The proofs of our main results in §3 then use these higher-level constructions to implement the necessary computations for simulating circuit classes with fixed-precision transformers and vice versa.

Analogously to circuit families, each string length N is processed by a separate transformer model. To process all of Σ^*, we therefore define a **transformer family** $\{\mathcal{T}_N\}$ as a sequence of transformers where each $\mathcal{T}_N$ processes strings of length N. Further, we again impose a uniformity condition on the family, which will relate the transformers for different input lengths.

[4]We focus on single-head self-attention for simplicity.

[5]Note that we ignore layer normalization in our analysis, similar to London & Kanade (2025).

Definition 2.4 (Uniform transformer families; variant of London & Kanade, 2025, Def. 3.6)**.** *Let* $\mathtt{X}$ *be a computational complexity class. A transformer family* $\{\mathcal{T}_N\}$ *is* $\mathtt{X}$***-uniform*** *if there exist Turing machines* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *whose resource usage is constrained by the complexity class* $\mathtt{X}$ *such that:*

(1) $\mathcal{M}_1$ *takes input* 1^N *and outputs a description of* $\mathcal{T}_N$*, and*

(2) $\mathcal{M}_2$ *takes input* $(1^N, \mathtt{B}(n))$ *and outputs* $\boldsymbol{p}(n, N)$*.*

Def. 2.4 allows for size-dependent transformers while keeping them closely related (as the same Turing machines must construct them for all N). It also facilitates natural connections with uniform circuit classes (cf. §2.2) (London & Kanade, 2025). All our results concern $\mathtt{L}$-uniform transformer families, in which case, the Turing machines in Def. 2.4 operate in logarithmic space. We discuss other notions of uniformity in §4.

Information-rich positional encodings. Def. 2.4 defines two components of a transformer based on the notion of uniform computability: The transformer model itself and the positional encoding. Although superficially similar, these two components appear in distinctly different roles in our constructions. The transformer model intuitively defines the "algorithm" that processes the input string and is limited in terms of the operations to those implementable by the fixed-precision attention mechanism. For example, although it is *constructed* by a logarithmic space Turing machine, it cannot *compute* arbitrary logarithmic-space functions but rather (a subset of) $\mathtt{AC}^0$ functions (London & Kanade, 2025). The positional encodings, in contrast, provide a way to inject additional information into the model that it cannot compute itself. They can, for example, provide direct access to the binary representation of the position index n, and pre-compute arithmetic operations such as division and modulo. This way, positional encodings provide the transformer with information that it cannot necessarily compute itself, but that is still computable in logarithmic space.

Language recognition. We treat a transformer $\mathcal{T}$ as a **language encoder**—a length-preserving function $\Sigma^* \to (\mathbb{F}^D)^*$ (Chan et al., 2025; Cotterell et al., 2024)—and regard the output $\boldsymbol{H}^{(L)}$ on a (possibly padded) string $\boldsymbol{w}$ as a $|\boldsymbol{w}| \times D$ matrix, where each row corresponds to the contextual representation of the symbol at the corresponding position. To convert this into a language recognizer, we use the output layer $\boldsymbol{o}$ that maps the contextual representation of the *final symbol* to the membership (classification) decision `0` or `1`.

2.5 Looped Padded Transformers

Looped (or universal) transformers use a fixed block of transformer layers that is applied repeatedly to the input string (Dehghani et al., 2019; Giannou et al., 2023; Hao et al., 2024; Goyal et al., 2024; Chen et al., 2025; Zeng et al., 2025; Geiping et al., 2025). This increases the depth of the model, enabling more complex reasoning by applying layers multiple times, and does not increase the model size, as the same block is reused for each iteration, thus reducing the memory footprint and computational cost (Bae et al., 2025). We define looped transformers as follows.

Definition 2.5 (Looped transformer)**.** *Let* $L, T \in \mathbb{N}$ *and let* $1 \leqslant l_1 \leqslant l_2 \leqslant L$*. Given a depth-*$L$ *transformer, a* ***looped transformer*** *computes symbol contextual representations* $\boldsymbol{H}$ *by*

(1.) Computing the initial hidden states $\boldsymbol{H}^{(0)}$ *for the input string* $\boldsymbol{w} = w_1 \cdots w_N$ *and computing* $\boldsymbol{H}^{(l_1)}$ *as with the first* l_1 *layers of the transformer*

(2.) Applying the transformer layers $l_1{+}1, \ldots, l_2$ T *times to the hidden states* $\boldsymbol{H}^{(l_1)}$ *to obtain* $\boldsymbol{H}^{(l_1+T(l_2-l_1))}$*.*

(3.) Applying the transformer layers $l_2 + 1, \ldots, L$ *to the hidden states* $\boldsymbol{H}^{(l_1+T(l_2-l_1))}$ *to obtain the final representations* $\boldsymbol{H}$ *that are passed to the output layer.*

The dynamic computational depth of looped transformers endows them with the ability to perform more complex reasoning tasks by iteratively refining their hidden states over multiple timesteps. Importantly, these reasoning steps include both sequential and parallel processing of the input symbols, allowing for both parallel efficiency as well as depth in the reasoning process.

Padded transformers additionally pad the input string with padding (pause) symbols (Pfau et al., 2024; Goyal et al., 2024; London & Kanade, 2025).

Definition 2.6 (Padded Transformer)**.** *Given $P \in \mathbb{N}$, a **padded transformer** $\mathcal{T}$ transformer computes the contextual representations $\boldsymbol{H}$ of a string $\boldsymbol{w} \in \Sigma^*$ by processing the padded input $\boldsymbol{w} \circ \underbrace{\square \cdots \square}_{P}$ (possibly by looping, cf. Def. 2.5), where $\square \notin \Sigma$ is a designated padding symbol.*

Instead of being restricted to the contextual representations of the N input symbols, a padded transformer can determine string membership or symbol probabilities based on the contextual representations of the P additional padded symbols as well. This additional space can be used to perform more operations and is analogous to increasing the circuit width in circuit complexity.

Notation. We use the shorthand LPT for looped padded transformers. We denote the class of LPTs with (1) p bits of precision, (2) model width D, (3) P padding symbols, and (4) T iterations, as $\mathtt{LPT}[\mathsf{p}, D, P, T]$. We are particularly interested in the $\mathtt{L}$-uniform family (cf. Def. 2.4), i.e., $\mathtt{L}$-uniform $\mathtt{LPT}[\mathsf{p}, D, P, T]$. When it is clear from the context, $\mathtt{LPT}[\mathsf{p}, D, P, T]$ will also refer to the class of languages recognized by such transformers. Particularly important special cases for this paper are the following:

(1) the class of constant-depth ($T = \Theta(1)$) and polynomially padded ($P = \mathtt{poly}(N)$) transformers with constant precision ($\mathsf{p} = \Theta(1)$) and logarithmic width ($D = \mathcal{O}(\log N)$)—we denote this class with $\mathtt{LPT}^0 \overset{\text{def}}{=} \mathtt{LPT}[\Theta(1), \mathcal{O}(\log N), \mathtt{poly}(N), \Theta(1)]$ (this corresponds to the transformer family studied by London & Kanade (2025)), and,

(2) the class of polylogarithmically looped ($T = \mathcal{O}(\log^d N)$ for some $d \in \mathbb{N}$) and polynomially padded ($P = \mathtt{poly}(N)$) transformers with constant precision ($\mathsf{p} = \Theta(1)$) and logarithmic width ($D = \mathcal{O}(\log N)$)—we denote this class with $\mathtt{LPT}^{\mathsf{d}} \overset{\text{def}}{=} \mathtt{LPT}[\Theta(1), \mathcal{O}(\log N), \mathtt{poly}(N), \mathcal{O}(\log^d N)]$.

Padding and looping together increase the expressivity of transformers.

Remark 1 (LPT expressivity)**.** *The following characterizations of LPT expressivity are known:*

(1) Recognition of regular languages $\mathtt{Reg}$*:*
- $\mathtt{Reg} \subseteq \mathtt{L}$-*uniform* $\mathtt{LPT}[\Theta(1), \mathcal{O}(\log N), 0, \mathcal{O}(\log N)]$ *(Saunshi et al., 2025, Thm. 5.1),*
- $\mathtt{Reg} \subseteq$ *non-uniform* $\mathtt{LPT}[\mathcal{O}(\log N), \Theta(1), 0, \mathcal{O}(\log N)]$ *(Liu et al., 2023), and,*
- $\mathtt{Reg} \subseteq$ *Fully uniform* $\mathtt{LPT}[\mathcal{O}(\log N), \Theta(1), 0, \mathcal{O}(\log N)]$ *(Merrill & Sabharwal, 2025b),*

(2) $\mathtt{L}$-*uniform* $\mathtt{LPT}[\mathcal{O}(\log N), \mathcal{O}(\log N), \mathtt{poly}(N), \Theta(1)] = \mathtt{L}$-*uniform* $\mathtt{TC}^0$ *(London & Kanade, 2025, Thm. 4.5),*

(3) Fully uniform $\mathtt{LPT}[\mathcal{O}(\log N), \Theta(1), \mathtt{poly}(N), \mathcal{O}(\log^d N)] = \mathtt{FO}$-*uniform* $\mathtt{TC}^{\mathsf{d}}$ *for* $d \geqslant 0$ *(Merrill & Sabharwal, 2025a, Thms. 1 and 3),*

(4) $\mathtt{L}$-*uniform* $\mathtt{LPT}^0 = \mathtt{L}$-*uniform* $\mathtt{AC}^0$ *(London & Kanade, 2025, Thm. 4.1),*

3 Exact Characterization of $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathsf{d}}$

This section shows the equivalence between fixed-precision looped transformers with logarithmic depth and polynomial padding, and logarithmic depth circuits; $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathsf{d}} = \mathtt{L}$-uniform $\mathtt{AC}^{\mathsf{d}}$. The structure closely follows that of Merrill & Sabharwal (2025a), who show the analogous result for logarithmic-precision transformers and $\mathtt{FO}$-uniform $\mathtt{TC}^{\mathsf{d}}$. We break the proof into two parts:

1. We first show the inclusion of transformer class in the circuit class, namely $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathsf{d}} \subseteq \mathtt{L}$-uniform $\mathtt{AC}^{\mathsf{d}}$, based on existing expressivity results (§3.1).

2. The other direction—$\mathtt{L}$-uniform $\mathtt{AC^d} \subseteq \mathtt{L}$-uniform $\mathtt{LPT^d}$—is more involved. The case $d = 0$ is shown by London & Kanade (2025), so we focus on $d \geqslant 1$, which we show in four steps:

   (1) We first discuss the notion of *completeness* under *reductions* as first introduced by Merrill & Sabharwal (2025a) for the log-precision case. We extend to the constant-precision case their result that if a transformer can recognize a language complete for a class $\mathcal{C}$ under reductions in class $\mathcal{R}$ and can also compute every $\mathcal{R}$ reduction, then it can recognize every language in $\mathcal{C}$ (§3.2.1).

   (2) Second, we show that $\mathtt{NL} \subseteq \mathtt{L}$-uniform $\mathtt{LPT^1}$, which tells us that logaritmic-depth transformers can compute $\mathtt{L}$ reductions (§3.2.2).

   (3) Third, we introduce a version of the *circuit evaluation problem* and show that the evaluation of $\mathtt{AC^d}$ circuits is complete for $\mathtt{L}$-uniform $\mathtt{AC^d}$ under $\mathtt{L}$ reductions (Def. 3.4).

   (4) Finally, we show that $\mathtt{L}$-uniform $\mathtt{LPT^d}$ can solve the $\mathtt{AC^d}$ circuit evaluation problem, thus completing the proof that $\mathtt{L}$-uniform $\mathtt{AC^d} \subseteq \mathtt{L}$-uniform $\mathtt{LPT^d}$ (§3.2.3).

3.1 Upper Bound: $\mathtt{L}$-uniform $\mathtt{LPT^d} \subseteq \mathtt{L}$-uniform $\mathtt{AC^d}$

Theorem 3.1 (Analog of Merrill & Sabharwal, 2025a, Lem. 4)**.** *For any $d \geqslant 0$, $\mathtt{L}$-uniform $\mathtt{LPT^d} \subseteq \mathtt{L}$-uniform $\mathtt{AC^d}$.*

Proof. The proof is identical to the one of Merrill & Sabharwal (2025a), swapping $\mathtt{TC^d}$ with $\mathtt{AC^d}$ and using the fact that $\mathtt{L}$-uniform $\mathtt{LPT^0} = \mathtt{L}$-uniform $\mathtt{AC^0}$ (London & Kanade, 2025, Thm. 4.1). ■

3.2 Lower Bound: $\mathtt{L}$-uniform $\mathtt{LPT^d} \supseteq \mathtt{L}$-uniform $\mathtt{AC^d}$

This section shows the other direction of Thm. 3.1: $\mathtt{L}$-uniform $\mathtt{AC^d} \subseteq \mathtt{L}$-uniform $\mathtt{LPT^d}$. To do so, we leverage the recent technique of reductions within padded transformers (Merrill & Sabharwal, 2025a). This then allows us to use the known expressivity results on transformer and combine them into the desired result.

3.2.1 Transformers Can Make Use of Reductions

Merrill & Sabharwal (2025a) show how to use the standard notions of **completeness** of a language for a complexity class under certain types of **reduction** in order to reason about transformer expressivity. We first recall the formal framework for discussing reductions in the context of transformers based on their definitions, and then extend their results to our setting.

Definition 3.1 (Merrill & Sabharwal, 2025a, Def. 9)**.** *Let $\mathcal{R}$ be a class of languages. We say a transduction $\mathtt{t}\colon \Sigma^* \to \Sigma^*$ is an $\mathcal{R}$ **reduction** if $|\mathtt{t}(\boldsymbol{w})|$ is polynomial in $|\boldsymbol{w}|$ and the language $\mathcal{L}_{\mathtt{t}} = \{(\boldsymbol{w}, \mathtt{B}(n), w) \mid \mathtt{t}(\boldsymbol{w})_n = w\}$ is in $\mathcal{R}$.*[6]

A transformer computes a reduction $\mathtt{t}$ if it recognizes $\mathcal{L}_{\mathtt{t}}$. However, since transformers can output a symbol in Σ instead of $\mathtt{1}/\mathtt{0}$, it is more natural to require the transformer to compute a functional form of this language, namely compute $r_{\mathtt{t}}(\boldsymbol{w}, n)$ defined as $\mathtt{t}(\boldsymbol{w})_n$. Our constructions work under both views, though the latter is often more natural and efficient. Formally:

Definition 3.2 (Merrill & Sabharwal, 2025a, Def. 10)**.** *We say that a **transformer family computes an $\mathcal{R}$ reduction** $\mathtt{t}$ if it either recognizes the language $\mathcal{L}_{\mathtt{t}} = \{(\boldsymbol{w}, \mathtt{B}(n), w) \mid \mathtt{t}(\boldsymbol{w})_n = w\}$ in $\mathcal{R}$ or computes the function $r\colon 1^*\Sigma \times \{0,1\}^* \to \Sigma$ defined as $r_{\mathtt{t}}(\boldsymbol{w}, \mathtt{B}(n)) = \mathtt{t}(\boldsymbol{w})_n$.*

[6]Here $\mathtt{t}(\boldsymbol{w})_n$ denotes the n^{th} symbol of $\mathtt{t}(\boldsymbol{w})$ if $n \leqslant |\mathtt{t}(\boldsymbol{w})|$, and a special symbol $\square \notin \Sigma$ otherwise.

At a high level, the following lemma shows that transformers can make use of reductions to recognize any language in a complexity class if they can recognize a language complete for that class and can compute every reduction in the relevant class.

Lemma 3.1 (Analog of Merrill & Sabharwal, 2025a, Lem. 3)**.** *Let $\mathcal{C}, \mathcal{R}$ be classes of languages. Let language $\mathcal{L}$ be $\mathcal{C}$-complete under $\mathcal{R}$ reductions. If $\mathcal{L} \in$ $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathtt{d}}$ and $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathtt{d}}$ can compute every $\mathcal{R}$ reduction, then $\mathcal{C} \subseteq$ $\mathtt{L}$-uniform $\mathtt{LPT}^{\mathtt{d}}$.*

Proof. The proof is identical to that of Merrill & Sabharwal, 2025a, Lem. 3 with one difference that we highlight here. The construction with log-precision transformers divides the polynomially-sized padding space into *blocks* of size $B = N^{K+1}$ for some $K \in \mathbb{N}$. It then relies on the layer hash norm to compute, for each padding position n, the number r of the padding block n is in. Fixed-precision transformers, however, cannot compute the layer hash norm. Luckily, the information computed by the layer hash norm only depends on the positions, meaning that it can be *precomputed* in the positional encodings. We therefore provide the transformer with the logspace-computable positional encodings that include the binary encodings of n/N^{K+1}, which will allow the transformer to attend to individual blocks as per Lem. A.4.

■

3.2.2 Lower Bound: $\mathtt{NL} \subseteq$ L-uniform $\mathtt{LPT}^1$

We now show that padded log-depth transformers can recognize any language in $\mathtt{NL}$. To do so, we first show that the graph connectivity problem is solvable by logarithmic-depth transformers.

Theorem 3.2 (Analog of Merrill & Sabharwal, 2025b, Thm. 2)**.** *There exists an $\mathtt{L}$-uniform family of $\mathtt{LPT}^{\mathtt{d}}$ transformers $\{\mathcal{T}_N\}_{N \in \mathbb{N}}$ such that $\mathcal{T}_N$ solves connectivity on (directed or undirected) graphs over N vertices: Given the $N \times N$ adjacency matrix of a graph $\mathcal{G}$, N^3 padding positions, and $s, t \in [N]$ in binary, $\mathcal{T}_N$ checks whether $\mathcal{G}$ has a path from vertex s to vertex t.*

Proof. We consider a directed graph $\mathcal{G}$ over N vertices and follow the construction from Merrill & Sabharwal, 2025b, Thm. 2. We again only highlight the differences in the construction.

Let $\boldsymbol{A} \in \{0,1\}^{N \times N}$ be $\mathcal{G}$'s adjacency matrix. The input to the transformer is the adjacency matrix $\boldsymbol{A}$ represented using N^2 positions from $\{0,1\}$, followed by N^3 padding positions $\square$, and finally the source and target nodes $s, t \in \{1, \ldots, N\}$. In contrast to Merrill & Sabharwal, 2025b, Thm. 2, s and t are represented in binary with $\lceil \log N \rceil$ bits each:

$$A_{1,1} \ldots A_{N,N} \underbrace{\square \ldots \square}_{N^3} \ \& \ \mathsf{B}(s) \ \& \ \mathsf{B}(t) \tag{4}$$

In contrast to Merrill & Sabharwal, 2025b, Thm. 2, we cannot rely on the layer hash norm to identify positions. To account for that, we provide the transformer with the following positional encodings.

$$\mathsf{PE}(n, N) \overset{\text{def}}{=} \begin{pmatrix} \mathsf{B}^{\pm}(n) \\ \mathsf{B}^{\pm}(n \mod N) \\ \mathsf{B}^{\pm}(n/N) \\ \mathsf{B}^{\pm}(n') \\ \mathsf{B}^{\pm}(n' \mod N) \\ \mathsf{B}^{\pm}(n'/N^2) \end{pmatrix} \in \{0,1\}^{\mathcal{O}(\log(N))}. \tag{5}$$

where $n' \overset{\text{def}}{=} \max(0, n - N^2)$. $\mathsf{B}(n)$ denotes the binary encoding of n using $\lceil \log N \rceil$ bits and $\mathsf{B}^{\pm}(n)$ denotes the signed binary encoding $2\mathsf{B}(n) - \mathbf{1}$, where $\mathbf{1}$ is the $\lceil \log N \rceil$-dimensional vector of all ones. These positional encodings can be computed in logspace.

$\mathcal{T}_N$ first uses $\lceil \log N \rceil + 1$ layers to read the binary encodings of the source and target nodes s and t stored in the string into a single dimension in the residual stream with the $\mathtt{L}$-uniform construction from Lem. A.6. This provides $\mathcal{T}_N$ with enough information to attend to appropriate positions in the subsequent layers, meaning that the construction can follow that of the $\mathtt{L}$-uniform Merrill & Sabharwal, 2025b, Thm. 2 from here on.

■

We combine Thm. 3.2 with Lem. 3.1 for the following analog to Merrill & Sabharwal, 2025a, Thm. 2.

Theorem 3.3 (Analog of Merrill & Sabharwal, 2025a, Thm. 2)**.** $\mathtt{NL} \subseteq \mathtt{L}$*-uniform* $\mathtt{LPT}^1$.

Proof. We follow the proof of Merrill & Sabharwal, 2025a, Thm. 2. Let $\mathcal{L}$ be the graph connectivity problem, class $\mathcal{C}$ be $\mathtt{NL}$, and class $\mathcal{R}$ be $\mathtt{FO}$. We will show that the preconditions of Lem. 3.1 are met, which will give us $\mathtt{NL} \subseteq \mathtt{L}$-uniform $\mathtt{LPT}^1$:

1. First, graph connectivity is known to be $\mathtt{NL}$-complete under $\mathtt{FO}$ reductions (Immerman, 1999).
2. Second, Thm. 3.2 shows that log-depth transformers with cubic padding can recognize the graph connectivity problem $\mathcal{L}$, i.e., $\mathcal{L} \in \mathtt{L}$-uniform $\mathtt{LPT}^1$.
3. Finally, London & Kanade, 2025, Thm. 4.1 shows that $\mathtt{L}$-uniform $\mathtt{LPT}^0 = \mathtt{L}$-uniform $\mathtt{AC}^0$. Since $\mathtt{L}$-uniform $\mathtt{AC}^0 \supseteq \mathtt{FO}$-uniform $\mathtt{AC}^0 = \mathtt{FO}$ (Mix Barrington et al., 1990), $\mathtt{L}$-uniform $\mathtt{LPT}^0$ can recognize languages in $\mathtt{FO}$ and can therefore compute $\mathtt{FO}$ reductions.

Thus, Lem. 3.1 applies. ■

3.2.3 Lower Bound: $\mathtt{L}$-uniform $\mathtt{AC}^\mathtt{d} \subseteq \mathtt{L}$-uniform $\mathtt{LPT}^\mathtt{d}$

We are finally ready to conclude the proof of the equivalence $\mathtt{L}$-uniform $\mathtt{LPT}^\mathtt{d} = \mathtt{L}$-uniform $\mathtt{AC}^\mathtt{d}$ by showing that $\mathtt{L}$-uniform $\mathtt{AC}^\mathtt{d} \subseteq \mathtt{L}$-uniform $\mathtt{LPT}^\mathtt{d}$. To do so, we use Lem. 3.1 again, coupled with the fact that $\mathtt{NL} \subseteq \mathtt{L}$-uniform $\mathtt{LPT}^1$. Concretely, we use the completeness of the $\mathtt{AC}^\mathtt{d}$ circuit evaluation problem under $\mathtt{NL}$ reductions for the class $\mathtt{L}$-uniform $\mathtt{AC}^\mathtt{d}$ (Lem. 3.2) together with the fact that transformers can evaluate $\mathtt{AC}^\mathtt{d}$ circuits (Lem. 3.4). This section adapts Merrill & Sabharwal, 2025a, App. B to the $\mathtt{LPT}^\mathtt{d}$ and $\mathtt{AC}^\mathtt{d}$ setting.

Circuit evaluation. The **circuit simulation** problem receives a string and a description of a circuit and returns the value of that circuit on the input string. We use the following representation of an $\mathtt{AC}^\mathtt{d}$ circuit, which differs from that of Merrill & Sabharwal (2025a) in two aspects discussed shortly.

Definition 3.3 (Circuit encoding)**.** *Let C be an $\mathtt{AC}^\mathtt{d}$ circuit over N inputs. We define the following encoding*

$$\langle C \rangle \overset{\text{def}}{=} \underbrace{\mathtt{X}\ .\ .\ .\ \mathtt{X}}_{N \text{ times}} \circ \langle G_1 \rangle \circ \ .\ .\ .\ \circ \langle G_M \rangle \tag{6}$$

where $\langle G_m \rangle$ for $m \in \{1, \ldots, M\}$ is the encoding of the m^{th} gate in the circuit C, and $\mathtt{X}$ is a special placeholder symbol for holding the input of the circuit.[7] *The argument encodings of the gate G_m with K arguments is defined as*

$$\langle G_m \rangle \overset{\text{def}}{=} \mathtt{T}(G_m)\ \ \mathtt{\&}\ \mathtt{B}(g_1)\ \mathtt{\#}\ \mathtt{B}(m)\ \ldots\ \ \mathtt{\&}\ \mathtt{B}(g_K)\ \mathtt{\#}\ \mathtt{B}(m), \tag{7}$$

where $\mathtt{T}(G_m) \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$ for $m \in \{1, \ldots, M\}$ denotes the type of the gate G_m, $\mathtt{B}(g_N)$ is the binary encoding of the position of the i^{th} argument of the gate G_m, and $\mathtt{B}(m)$ is the binary encoding of the gate's index m.

[7]Note that, although the first N positions contain placeholders for input strigns, the circuit encoding is independent of the input string—since all compatible strings are of the same length, the circuit encoding does not change depending on the input string.

Example 3.1. *The encoding of the circuit $C(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee x_3$ is*

$$\langle C \rangle \overset{\text{def}}{=} \underbrace{\texttt{X X X}}_{\textit{input}} \ \texttt{AND} \ \underbrace{\texttt{\&000 \#010 \&001 \#010}}_{\textit{arguments to } \texttt{AND}} \ \texttt{OR} \ \underbrace{\texttt{\&011 \#100 \&010 \#100}}_{\textit{arguments to OR}} . \tag{8}$$

There are two main differences between Merrill & Sabharwal's (2025a) definitions and ours:

(1) We encode the argument pointers of each gate using their *binary encodings* instead of unary ones as Merrill & Sabharwal (2025a) do. We require this to ensure that a shallow (in our case, logarithmic-depth) finite-precision transformer can convert the circuit encoding into contents of the residual stream. It is not clear how to do this with unary encodings.

(2) Second, we replicate the positions of the gates after the pointer to each argument (prefixed by the special # symbol). This is needed to avoid the use of the layer hash norm to compute the pointer to the argument's gate, which is done by Merrill & Sabharwal (2025a).

Definition 3.4 ($\mathcal{F}$ circuit evaluation; Merrill & Sabharwal, 2025a, Def. 11)**.** *Let $\mathcal{F}$ be a class of (potentially non-uniform) circuit families. The $\mathcal{F}$ circuit evaluation problem is defined as follows:*

- ***Input:*** *$(\boldsymbol{w}, \langle C \rangle)$ where $\boldsymbol{w} \in \{0,1\}^*$ is a string and $\langle C \rangle$ is the serialization of a circuit C (cf. Def. 3.3) such that $C = C_{|\boldsymbol{w}|}$ for some circuit family $\{C_N\}_{N=0}^{\infty} \in \mathcal{F}$.*
- ***Output:*** *The value $C(\boldsymbol{w})$.*

We refer to the special case where $\mathcal{F} = \texttt{AC}^{\texttt{d}}$ as the $\texttt{AC}^{\texttt{d}}$ **circuit evaluation problem**. We further define **wide-$\texttt{AC}^{\texttt{d}}$** $\subseteq \texttt{AC}^{\texttt{d}}$ as the class of circuit families $\{C_N\}_{N=0}^{\infty}$ such that there exists some c such that, for large N, the depth of C_N is at most $c \log^d N$ and, crucially, the size is *at least* N^c. That is, the size (and hence the width) of the circuit is large relative to its depth. We define the corresponding **wide-$\texttt{AC}^{\texttt{d}}$ circuit evaluation problem** as a variant of circuit evaluation problem with $\mathcal{F} =$ wide-$\texttt{AC}^{\texttt{d}}$.

The following lemmata show that both $\texttt{AC}^{\texttt{d}}$ and wide-$\texttt{AC}^{\texttt{d}}$ circuit evaluation are hard for $\texttt{L}$-uniform $\texttt{AC}^{\texttt{d}}$ under $\texttt{NL}$ reductions. Their proofs are identical to the ones in Merrill & Sabharwal (2025a), except that we replace $\texttt{TC}^{\texttt{d}}$ with $\texttt{AC}^{\texttt{d}}$ (the proofs are not affected by the small differences in circuit serializations, since our serialization is also logspace-computable).

Lemma 3.2 (Analog of Merrill & Sabharwal, 2025a, Lem. 5)**.** *For $d \geqslant 1$, $\texttt{AC}^{\texttt{d}}$ circuit evaluation is hard for $\texttt{L}$-uniform $\texttt{AC}^{\texttt{d}}$ under $\texttt{L}$ reductions.*

Lemma 3.3 (Analog of Merrill & Sabharwal, 2025a, Lem. 6)**.** *For $d \geqslant 1$, wide-$\texttt{AC}^{\texttt{d}}$ circuit evaluation is hard for $\texttt{L}$-uniform $\texttt{AC}^{\texttt{d}}$ under $\texttt{L}$ reductions.*

The following lemma shows that wide-$\texttt{AC}^{\texttt{d}}$ circuit evaluation, using the modified circuit serialization we defined, can be solved by $\texttt{L}$-uniform $\texttt{LPT}^{\texttt{d}}$.

Lemma 3.4 (Analog Merrill & Sabharwal, 2025a, Lem. 7)**.** *There exists an $\texttt{L}$-uniform $\texttt{LPT}^1$ family of transformers $\{\mathcal{T}_N\}_{N \in \mathbb{N}}$ such that, on input $\boldsymbol{w} \circ \langle C \rangle$, where $\boldsymbol{w} \in \{0,1\}^*$ and $\langle C \rangle$ is the serialization of a depth L circuit with $N \overset{\text{def}}{=} |w \circ \langle C \rangle|$ inputs, $\mathcal{T}_N$ computes $C(\boldsymbol{w})$ when unrolled $\mathcal{O}(\log N) + L$ times.*

Proof sketch. The high-level idea of the construction is similar to that of Merrill & Sabharwal, 2025a, Lem. 7, but we need to adapt it to the fixed-precision setting. Let C be a circuit of depth L over N inputs. The transformer $\mathcal{T}$ simulates C in two stages:

1. Converting the input $\boldsymbol{w} \circ \langle C \rangle$ into internal representations that will allow the transformer to process it. This stage requires $\mathcal{O}(\log N)$ layers.

2. Iteratively applying the circuit operations to the internal representations to compute the final output. This stage requires L layers.

Stage 1 converts the binary encodings of the argument pointers stored in the input string $\langle C\rangle$ into binary encodings stored in the residual stream as per the L-uniform construction in Lem. A.6. These pointers the allow the model to retrieve the values stored in those positions (once they become available). Importantly, this conversion only has to happen once for all gates and inputs in parallel, even if the inputs have not been computed yet—this is possible since the encoding of the entire circuit is available at the beginning. This means that the computation of pointers adds a fixed overhead of $\mathcal{O}(\log N)$ layers to the simulation.

At the same time, the positions containing the *gate* addresses $\mathtt{B}(m)$ compute their binary encodings in the same way as the input arguments (cf. Lem. A.6), storing the binary encoding in another designated part of the residual stream. The final position of the input argument can then attend $\log N$ positions forward to retrieve the position of the gate it belongs to—this can be done by storing the (signed) binary encodings of both n and $n + \log N$ in the positional encodings. With this, each input argument g_N contains both the pointer to its value as well as the pointer to the gate it belongs to—this will be used at a later stage of the simulation, when each gate has to read its input argument values before computing its own value.

The rest of the proof (Stage 2) closely follows that of Merrill & Sabharwal, 2025a, Lem. 7. In contrast to Merrill & Sabharwal's (2025a) fully uniform transformer, ours is L-uniform; the only aspect of the transformer family that depends on N is the size of the matrices, which needs to grow with the growing positional encodings. The counters required to construct such matrices can be implemented in log-space, which is why the transformer family is L-uniform. The transformer uses L layers to simulate the circuit layer by layer, using the pointers constructed in Stage 1 to retrieve the values of the input arguments of each gate. The only difference is that, to avoid issues with fixed precision, the model computes the AND gates by detecting 0 among the inputs and the OR gates by detecting a 1 among the inputs as per Lem. A.5—this is enough to determine the truth value of the gate. ■

We have the following corollary, whose proof is identical to Merrill & Sabharwal, 2025b, Cor. 7.1, replacing $\mathtt{TC}^{\mathsf{d}}$ with $\mathtt{AC}^{\mathsf{d}}$.

Corollary 3.1 (Analog of Merrill & Sabharwal, 2025a, Cor. 7.1)**.** *For $d \geqslant 1$, the wide circuit evaluation problem is in* L*-uniform* $\mathtt{LPT}^{\mathsf{d}}$.

This leads us to the main result of this section.

Lemma 3.5. *For any $d \geqslant 1$,* L*-uniform* $\mathtt{LPT}^{\mathsf{d}} \supseteq$ L*-uniform* $\mathtt{AC}^{\mathsf{d}}$.

Proof. Let $\mathcal{L}$ be the wide-$\mathtt{AC}^{\mathsf{d}}$ circuit evaluation problem, $\mathcal{C}$ be the class L-uniform $\mathtt{AC}^{\mathsf{d}}$, and $\mathcal{R}$ be the class L. The preconditions of Lem. 3.1 are met:

1. Cor. 3.1 shows that $\mathcal{L} \in$ L-uniform $\mathtt{LPT}^{\mathsf{d}}$ for $d \geqslant 1$. Together with Thm. 3.1, this implies $\mathcal{L} \in$ L-uniform $\mathtt{AC}^{\mathsf{d}}$.
2. Lem. 3.3 shows that $\mathcal{L}$ is hard for $\mathtt{AC}^{\mathsf{d}}$ under L reductions. Together with the above, we get that $\mathcal{L}$ is complete for L-uniform $\mathtt{AC}^{\mathsf{d}}$.
3. Thm. 3.3 gives us L $\subseteq$ L-uniform $\mathtt{LPT}^{\mathsf{d}}$, meaning that L-uniform $\mathtt{LPT}^{\mathsf{d}}$ can compute any L reduction.

Thus, Lem. 3.1 applies, yielding L-uniform $\mathtt{AC}^{\mathsf{d}} \subseteq$ L-uniform $\mathtt{LPT}^{\mathsf{d}}$.

Note that, unlike Merrill & Sabharwal, 2025a, Thm. 3, which constructs a transformer with depth $\mathcal{O}(\log^d N)$ to simulate $\mathtt{TC}^{\mathsf{d}}$ circuits, our application of Lem. 3.4 yields a transformer with depth $\mathcal{O}(\log N + \log^d N)$ to simulate an $\mathtt{AC}^{\mathsf{d}}$ circuit. For $d \geqslant 1$, this reduces to $\mathcal{O}(\log^d N)$. ■

Lem. 3.5 and Thm. 3.1, together with London & Kanade's (2025) result for $d = 0$, characterize the relationship between L-uniform $\mathtt{LPT}^{\mathsf{d}}$ and L-uniform $\mathtt{AC}^{\mathsf{d}}$, analogous to Merrill & Sabharwal, 2025a, Thm. 3.

Theorem 3.4. *For any $d \geqslant 0$,* L*-uniform* $\mathtt{LPT}^{\mathsf{d}} =$ L*-uniform* $\mathtt{AC}^{\mathsf{d}}$.

4 Discussion and Conclusion

We establish that $\mathtt{L}$-uniform fixed-precision looped padded transformers with polylogarithmic depth and polynomial padding are exactly equivalent to $\mathtt{L}$-uniform $\mathtt{AC}^{\mathsf{d}}$ circuits. This adds to the growing characterization of transformer expressivity across different precision regimes, as summarized in Fig. 1.

Our results, combined with prior work, reveal how architectural choices—precision scaling, width scaling, looping depth, and padding—affect computational power. In particular, (i) **logarithmic precision, constant width** transformers achieve $\mathtt{FO}$-uniform $\mathtt{TC}^{\mathsf{d}}$ (Merrill & Sabharwal, 2025a), leveraging high-precision arithmetic to compute threshold gates, while (ii) **fixed precision, logarithmic width** transformers achieve $\mathtt{L}$-uniform $\mathtt{AC}^{\mathsf{d}}$, restricted to threshold-free computation due to bounded precision arithmetic.

The interplay of precision and width. These results shed additional light onto which architectural details matter for transformer expressivity. Precision arises as a major factor: Fixed-precision transformers achieve $\mathtt{AC}^{\mathsf{d}}$ while log-precision transformers achieve the strictly stronger $\mathtt{TC}^{\mathsf{d}}$ at every depth level d. This *cannot* be compensated for with a polynomial increase in model *width*: Despite having the same representation space volume ($\mathsf{p} \cdot D = \Omega(\log N)$), the precision–width trade-off fundamentally constrains expressivity. Fixed-precision transformers must distribute information across model dimensions without concentrating arbitrary amounts into individual values.

The role of transformer family uniformity. While Def. 2.4 defines transformer uniformity for any class $\mathtt{X}$, our results concern $\mathtt{L}$-uniform transformer and circuit families. We choose this class for multiple reasons. Firstly, $\mathtt{L}$-uniform transformer families have been characterized by prior work on finite-precision transformers (London & Kanade, 2025), allowing direct comparisons. More importantly, however, our constructions rely on the power of $\mathtt{L}$ for constructing positional embeddings. Recall that the definition of transformer uniformity requires positional encodings $\boldsymbol{p}(n, N)$ to be computable in class $\mathtt{X}$ from the input $(1^N, \mathtt{B}(n))$. Restricting the class $\mathtt{X}$ to $\mathtt{FO}$ would limit the positional encodings to first-order logic, which is insufficient for computing many useful functions of n and N (e.g., arithmetic operations such as division and modulo). Nevertheless, London & Kanade's (2025) results on the equivalence between $\mathtt{L}$-uniform $\mathtt{LPT}^{0}$ and $\mathtt{L}$-uniform $\mathtt{AC}^{0}$ should generalize to any uniformity class $\mathtt{X}$. However, our constructions that generalize the fixed-depth results to looping heavily rely on $\mathtt{L}$-computable functions for positional encodings, implying that $\mathtt{X}$ must at least contain $\mathtt{L}$ and suggesting the equivalence between $\mathtt{X}$-uniform $\mathtt{LPT}^{\mathsf{d}}$ and $\mathtt{X}$-uniform $\mathtt{AC}^{\mathsf{d}}$ for any class $\mathtt{X}$ at least as powerful as $\mathtt{L}$. This further motivates the study of transformer uniformity classes beyond $\mathtt{L}$-uniform. In particular, it raises the question of how the complexity of computing the positional encodings affects the expressivity of the resulting transformer family. Could we decouple the complexity of computing the positional encodings from the complexity of specifying the transformer? In particular, can we keep the construction of the transformer *completely* uniform (the same for all input lengths N; such transformers are studied by, e.g., Yang et al. (2024), Li & Cotterell (2025), and Jerad et al. (2025)) and increase the complexity of the positional encodings to achieve more expressivity?

Acknowledgments

Anej Svete is supported by the ETH AI Center fellowship. Some ideas for this work (for example, the notation in §2.5 and the motivation for unifying the results in §4) were developed during the 2025 Dagstuhl Seminar 25282 "Theory of Neural Language Models", particularly in the "Uniformity" working group attended by Satwik Bhattamishra, Michaël Cadilhac, David Chiang, Will Merrill, Ashish Sabharwal, Clayton Sanford, Howard Straubing, and Laura Strieker.

References

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025. URL https://arxiv.org/abs/2507.10524.

Robin S. M. Chan, Reda Boumasmoud, Anej Svete, Yuxin Ren, Qipeng Guo, Zhijing Jin, Shauli Ravfogel, Mrinmaya Sachan, Bernhard Schölkopf, Mennatallah El-Assady, and Ryan Cotterell. On affine homotopy between language encoders. In *NeurIPS*, 2025.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1369. URL https://aclanthology.org/2025.acl-long.1369/.

Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. Formal aspects of language modeling. *arXiv preprint arXiv:2311.04329*, 2024. URL https://arxiv.org/abs/2311.04329.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2019. URL https://arxiv.org/abs/1807.03819.

Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, Dec 1984. ISSN 1433-0490. doi: 10.1007/BF01744431. URL https://doi.org/10.1007/BF01744431.

Jonas Geiping, Sean Michael McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. URL https://openreview.net/forum?id=D6o6Bwtq7h.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024. URL https://arxiv.org/abs/2412.06769.

Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer, New York, 1999. ISBN 978-0-387-98600-5. doi: 10.1007/978-1-4612-0539-5. URL https://link.springer.com/book/10.1007/978-1-4612-0539-5.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. *arXiv preprint arXiv:2503.14615*, 2025. URL https://arxiv.org/abs/2503.14615.

Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of transformer language models. *arXiv*, 2025. URL https://arxiv.org/abs/2505.23623.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *ICLR*, 2024. URL https://openreview.net/forum?id=3EWTEy9MTM.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *ICLR*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

Charles London and Varun Kanade. Pause tokens strictly increase the expressivity of constant-depth transformers. *arXiv preprint arXiv:2505.21024*, 2025. URL https://arxiv.org/abs/2505.21024.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *TACL*, 11:531–545, 2023. URL https://aclanthology.org/2023.tacl-1.31/.

William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. *arXiv preprint arXiv:2505.18948*, 2025a. URL https://arxiv.org/abs/2505.18948.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *arXiv preprint arXiv:2503.03961*, 2025b. URL https://arxiv.org/abs/2503.03961.

David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(90)90022-D. URL https://www.sciencedirect.com/science/article/pii/002200009090022D.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *COLM*, 2024. URL https://openreview.net/forum?id=NikbrdtYvG.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025. URL https://arxiv.org/abs/2502.17416.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *NeurIPS*, 2024. URL https://openreview.net/forum?id=FBMsBdH0yz.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining language models to ponder in continuous space. *arXiv preprint arXiv:2505.20674*, 2025. URL https://arxiv.org/abs/2505.20674.

A Theoretical Gadgets

This section contains various (existing) theoretical gadgets that are used in the proofs of the main results. In the following, $N \in \mathbb{N}$ always refers to the length of the original input string. We use $[\![w]\!] \in \{0,1\}^{|\Sigma|}$ to denote the one-hot encoding of symbol $w \in \Sigma$. We define the **intereleaving** of the vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^D$ as $\boldsymbol{x}^\frown \boldsymbol{y} \in \mathbb{R}^{2D}$ where

$$\boldsymbol{x}^\frown \boldsymbol{y}_d \overset{\text{def}}{=} \begin{cases} x_{(d+1)/2} & \textbf{if } d \text{ is odd,} \\ y_{d/2} & \textbf{otherwise } d. \end{cases} \tag{9}$$

A.1 Positional Encodings

Uniquely identifying positions in a string requires the "volume" of the representation space to grow with the string length. In the case of finite-precision logarithmic-width transformers, this is achieved with positional encodings that encode the binary representation of the position in the string. The following lemma follows from the definition of fixed-point arithmetic, and the rounding and thresholding applied therein.

Lemma A.1. *Let $x \in \mathbb{F}_{\mathsf{p}}$ for some $\mathsf{p} \in \mathbb{N}$ such that $x > \log 2(\mathsf{p}+1)$. Then, it holds that*

$$\exp(x) = B_{\mathbb{F}}, \tag{10a}$$
$$\exp(-x) = 0. \tag{10b}$$

Lem. A.2 readily follows.

Lemma A.2 (Generalization of Li et al., 2024, Lem. E.3)**.** *For $N \in \mathbb{N}$, $n \in [N]$, define the vectors $\boldsymbol{q}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ and $\boldsymbol{k}_n \in \mathbb{R}^{2\lceil \log N \rceil}$ as follows:*

$$\boldsymbol{q}_n \overset{\text{def}}{=} {}^{B_{\mathbb{F}}}\!/\!_m \cdot \left(\mathsf{B}^{\pm}(n)^\frown \mathbf{1}_{\lceil \log N \rceil}\right) \tag{11a}$$
$$\boldsymbol{k}_{n'} \overset{\text{def}}{=} \mathsf{B}^{\pm}(n')^\frown(-\mathbf{1}_{\lceil \log N \rceil}). \tag{11b}$$

Then, it holds that

$$\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} = \begin{cases} 0 & \textbf{\textit{if }} n = n' \\ x & \textbf{\textit{otherwise}} \end{cases}. \tag{12}$$

where $x \leqslant -{}^{2B_{\mathbb{F}}}\!/\!_m$.

A.2 Useful Attention Patterns

The following lemmata describe how a transformer layer can either ignore or exclusively focus on specific positions in the input string.

Lemma A.3 (Ignoring Marked Positions with a Transformer)**.** *Let $N, D \in \mathbb{N}$, $\mathcal{N} \subseteq [N]$, and let $\boldsymbol{H} \in \mathbb{R}^{N \times D}$ be a matrix representing the residual stream such that*

$$[\![\square]\!] \in \boldsymbol{H}_{n,:} \iff n \in \mathcal{N}. \tag{13}$$

Here, the notation $[\![\square]\!] \in \boldsymbol{H}_{n,:}$ means that the vector $\boldsymbol{H}_{n,:}$ contains the one-hot encoding $[\![\square]\!]$ of the symbol $\square$ at position n. Further, let $\boldsymbol{G} \overset{\text{def}}{=} \boldsymbol{H}_{[N]\setminus\mathcal{N},:} \in \mathbb{R}^{(N-|\mathcal{N}|)\times D}$, where $\boldsymbol{H}_{[N]\setminus\mathcal{N},:}$ denotes the projection of the matrix $\boldsymbol{H}$ onto the rows not *in $\mathcal{N}$. Finally, let $\boldsymbol{\tau}$ be a transformer layer. Then, there exists a logarithmic-width transformer layer $\boldsymbol{\tau}'$ such that it holds for $\boldsymbol{G}' \overset{\text{def}}{=} \boldsymbol{\tau}(\boldsymbol{G}) \in \mathbb{R}^{(N-|\mathcal{N}|)\times D}$ and $\boldsymbol{H}' \overset{\text{def}}{=} \boldsymbol{\tau}'(\boldsymbol{H}) \in \mathbb{R}^{N \times D}$ that*

$$\boldsymbol{G}' = \boldsymbol{H}'_{[N]\setminus\mathcal{N},:}. \tag{14}$$

Informally, Lem. A.3 states that a transformer layer can ignore positions containing one-hot encodings of specific "marker" symbols, such as additional symbols not in the original alphabet.

Proof. Notice that, since $\boldsymbol{H}$ and $\boldsymbol{G}$ match on all positions not in $\mathcal{N}$, *ignoring* the positions in $\mathcal{N}$ (marked by $\square$) by $\boldsymbol{\tau}'$ will ensure that the outputs of the two layers $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$ are identical on the positions not in $\mathcal{N}$. We now construct a transformer layer that ignores the contributions of rows marked by $[\![\square]\!]$. To do so, we modify each attention head of $\boldsymbol{\tau}$ such that the head computes its attention scores with queries and keys of the form

$$\boldsymbol{q}'_n \stackrel{\text{def}}{=} \cdot \begin{pmatrix} \boldsymbol{q}_n \\ -B_{\mathbb{F}} \cdot [\![\square]\!] \\ -B_{\mathbb{F}} \cdot [\![\square]\!] \end{pmatrix} \tag{15a}$$

$$\boldsymbol{k}'_{n'} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{k}_{n'} \\ [\![w_{n'}]\!] \\ [\![w_{n'}]\!] \end{pmatrix} \tag{15b}$$

where $\boldsymbol{q}_n$ and $\boldsymbol{k}_{n'}$ are the original head's query and key vectors of $\mathcal{T}$ at position n and n', respectively, and $[\![w_{n'}]\!]$ is the one-hot encoding of the symbol at position n'. We can then compute the dot product of the two vectors as

$$\boldsymbol{q}'^{\top}_{n} \boldsymbol{k}'_{n'} = \boldsymbol{q}_n^{\top} \boldsymbol{k}_{n'} - B_{\mathbb{F}} \cdot \mathbb{1}\{w_{n'} = \square\} - B_{\mathbb{F}} \cdot \mathbb{1}\{w_{n'} = \square\}. \tag{16}$$

This is computed in finite-precision fixed-point arithmetic. Thus, if $w_{n'} \neq \square$, the attention score is $\boldsymbol{q}_n^{\top} \boldsymbol{k}_{n'}$ and the head behaves as it did in $\mathcal{T}$. If the symbol at position n' is $\square$, the last two components of the vectors $\boldsymbol{q}'_n$ and $\boldsymbol{k}'_{n'}$ ensure that the exponentiated value of the attention score becomes 0, thus not contributing to the attention weights. $\mathcal{T}'$ can thus simulate $\mathcal{T}$ on the rest of the positions. ■

Lemma A.4 (Focusing on Marked Positions with a Transformer)**.** *Let* $N, D \in \mathbb{N}$*,* $R\colon [N] \to [N]$*,* $r \in [N]$*,* $\mathcal{N} \stackrel{\text{def}}{=} R^{-1}(r) \subseteq [N]$*, and let* $\boldsymbol{H} \in \mathbb{R}^{N \times D}$ *be a matrix representing the residual stream such that*

$$\overline{\mathsf{B}}(R(n)) \in \boldsymbol{H}_{n,:} \quad \textit{for all } n \in [N] \tag{17}$$

Here, the notation $\overline{\mathsf{B}}(R(n)) \in \boldsymbol{H}_{n,:}$ *means that the vector* $\boldsymbol{H}_{n,:}$ *contains the signed binary encoding (cf. §2) of* $R(n)$*. Further, let* $\boldsymbol{G} \stackrel{\text{def}}{=} \boldsymbol{H}_{\mathcal{N},:} \in \mathbb{R}^{|\mathcal{N}| \times D}$*, where* $\boldsymbol{H}_{\mathcal{N},:}$ *denotes the projection of the matrix* $\boldsymbol{H}$ *onto the rows in* $\mathcal{N}$*. Finally, let* $\boldsymbol{\tau}$ *be a transformer layer. Then, there exists a logarithmic-width transformer layer* $\boldsymbol{\tau}'$ *such that it holds for* $\boldsymbol{G}' \stackrel{\text{def}}{=} \boldsymbol{\tau}(\boldsymbol{G}) \in \mathbb{R}^{|\mathcal{N}| \times D}$ *and* $\boldsymbol{H}' \stackrel{\text{def}}{=} \boldsymbol{\tau}'(\boldsymbol{H}) \in \mathbb{R}^{N \times D}$ *that*

$$\boldsymbol{G}' = \boldsymbol{H}'_{\mathcal{N},:}. \tag{18}$$

Informally, Lem. A.4 states that a transformer layer can focus on positions containing signed binary encodings of a number r computed as a function of the position while ignoring the rest of the positions.

Proof. The idea of the construction of $\boldsymbol{\tau}'$ is similar to that of Lem. A.3, but instead of ignoring the positions in $\mathcal{N}$, we want the transformer layer to focus on them while ignoring the rest of the positions. This can be done by including $\overline{\mathsf{B}}(R(n))$ in the positional encodings of the attention heads and then using the key and query vectors of the form

$$\boldsymbol{q}'_n \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{q}_n \\ {}^{B_{\mathbb{F}}}\!/_{2} \cdot (\mathsf{B}^{\pm}(r)^\frown \mathbf{1}_{\lceil \log N \rceil}) \\ {}^{B_{\mathbb{F}}}\!/_{2} \cdot (\mathsf{B}^{\pm}(r)^\frown \mathbf{1}_{\lceil \log N \rceil}) \end{pmatrix} \tag{19a}$$

$$\boldsymbol{k}'_{n'} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{k}_{n'} \\ \mathsf{B}^{\pm}(R(n'))^\frown(-\mathbf{1}_{\lceil \log N \rceil}) \\ \mathsf{B}^{\pm}(R(n'))^\frown(-\mathbf{1}_{\lceil \log N \rceil}) \end{pmatrix} \tag{19b}$$

where $\boldsymbol{q}_n$ and $\boldsymbol{k}_{n'}$ are the original head's query and key vectors of $\mathcal{T}$ at position n and n', respectively, and $B_{\mathbb{F}}$ is the maximal representable number in the fixed-point arithmetic (which might depend on the string length N). We can then compute the dot product of the two vectors as

$$\boldsymbol{q'}_n^\top \boldsymbol{k'}_{n'} = \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} + \underbrace{{}^{B_{\mathbb{F}}}\!/\!_2 \cdot (\mathsf{B}^\pm(r)^\frown \mathbf{1}_{\lceil \log N \rceil})^\top (\mathsf{B}^\pm(R(n'))^\frown(-\mathbf{1}_{\lceil \log N \rceil}))}_{\stackrel{\text{def}}{=} G} \tag{20a}$$
$$+ \underbrace{{}^{B_{\mathbb{F}}}\!/\!_2 \cdot (\mathsf{B}^\pm(r)^\frown \mathbf{1}_{\lceil \log N \rceil})^\top (\mathsf{B}^\pm(R(n'))^\frown(-\mathbf{1}_{\lceil \log N \rceil}))}_{\stackrel{\text{def}}{=} G}$$

Note that Eq. (20a) uses fixed-point arithmetic. We compute the inner product in Eq. (20a) by analyzing individual cases:

1. **Case 1:** $R(n') \neq r$.

   All intermediate computations of Eq. (20a) are thresholded at $\min(B_{\mathbb{F}}, \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} + {}^{B_{\mathbb{F}}}\!/\!_2)$. In particular, by Lem. A.2, the value after adding the first term G is at most $\min(B_{\mathbb{F}}, \boldsymbol{q}_n^\top \boldsymbol{k}_{n'} + {}^{B_{\mathbb{F}}}\!/\!_2) - 2{}^{B_{\mathbb{F}}}\!/\!_2 \leqslant B_{\mathbb{F}} - B_{\mathbb{F}} = 0$. After adding the second term G, the value is at most $-B_{\mathbb{F}}$, resulting in a 0 exponentiated attention score, as required.

2. **Case 2:** $R(n') = r$. We analyze three sub-cases based on the value of $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}$.

   1. **Sub-case 2a:** $\left|\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}\right| < \log 2(\mathsf{p}+1)$. All intermediate computations in Eq. (20a) are bounded by $\log 2(\mathsf{p}+1) + {}^{B_{\mathbb{F}}}\!/\!_2$ in absolute value, so they fall within the range of $\mathbb{F}_{\mathsf{p}}$. Moreover, addition of ${}^{B_{\mathbb{F}}}\!/\!_2$ can be exactly represented in $\mathbb{F}_{\mathsf{p}}$. This makes addition in Eq. (20a) associative and commutative. By Lem. A.2, the terms G in Eq. (20a) are 0, meaning that the final attention score equals $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'}$.
   2. **Sub-case 2b:** $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} \geqslant \log 2(\mathsf{p}+1)$. In this case, the intermediate computations of Eq. (20a) are either exact or thresholded at $B_{\mathbb{F}}$. In both cases, the exponent of the resulting attention score is $B_{\mathbb{F}}$ by Lem. A.1, preserving the attention score.
   3. **Sub-case 2c:** $\boldsymbol{q}_n^\top \boldsymbol{k}_{n'} \leqslant \log 2(\mathsf{p}+1)$. In this case, all intermediate computations are representable in $\mathbb{F}_{\mathsf{p}}$ analogously to the case 2a. The attention score is therefore preserved.

This means that the attention scores between positions n and n' of $\boldsymbol{\tau}'$ are identical to those of $\boldsymbol{\tau}$ on the positions in $\mathcal{N}$, while the attention scores on the rest of the positions are 0. This completes the proof. ■

Lemma A.5 (Detecting a symbol occurrence)**.** *There exists a* L*-uniform* LPT^0 *family of transformers* $\{\mathcal{T}_N\}_{N\in\mathbb{N}}$ *such that, for any* $N \in \mathbb{N}$*, on input* $\boldsymbol{w} \in \Sigma^*$ *of length* N *and* $w \in \Sigma$*,* $\mathcal{T}_N$*'s residual stream at position* 1 *contains the entry* $\mathbb{1}\{w \in \boldsymbol{w}\}$.

Proof sketch. Note that $\mathcal{T}_N$ cannot use the commonly-used *exact* uniform attention over all symbols to detect $\mathbb{1}\{w \in \boldsymbol{w}\}$ due to fixed precision. Nevertheless, rounded uniform attention suffices. By attending to all symbols in the string with weight 1, the denominator of the attention scores is at most $B_{\mathbb{F}}$. Using one-hot encodings of symbols w_n as the attention values $\boldsymbol{v}_n$, it is easy to see that the final contextual representation at the final position will have a positive value at the entry corresponding to w if and only if $w \in \boldsymbol{w}$, since ${}^{c}\!/\!_{B_{\mathbb{F}}} > 0$ for any $c \geqslant 1$. This condition can be checked by the MLP applied after the attention aggregation operation. This construction is clearly logspace computable. ■

Lemma A.6 (Converting a binary representation into a positional encoding)**.** *There exists a* L*-uniform* LPT^1 *family of transformers* $\{\mathcal{T}_N\}_{N\in\mathbb{N}}$ *such that, on input* $\&\ \mathsf{B}(n)$*,* $\mathcal{T}_N$*'s residual stream at position* $\lceil \log N \rceil + 1$ *contains the value* $\mathsf{B}(n)$ *for any* $N \in \mathbb{N}$ *and* $n \in [N]$.

Proof sketch. The transformer $\mathcal{T}_N$ has to convert the binary representation $\mathtt{B}(n)$ of n contained in across $\lceil \log N \rceil$ positions in the input string into a single $\lceil \log N \rceil$-dimensional binary vector in the residual stream. This is done as follows:

1. In the first layer, each symbol $w_{n'} \in \{0, 1\}$ checks if it is immediately preceded by the `&` symbol, which denotes the beginning of the pointer in the string. If it is, $w_{n'}$ stores $\boldsymbol{e}_1$ and $\boldsymbol{d}_1 \overset{\text{def}}{=} w_{n'}\boldsymbol{e}_1$ in designated parts of its residual stream. Here, $\boldsymbol{e}_1$ is the first unit vector of $\mathbb{R}^{\lceil \log N \rceil}$.
2. In the subsequent layers $l \in \{2, \ldots, \lceil \log N \rceil\}$, each symbol $w_{n'}$ checks if the entry $\boldsymbol{e}_{l-1}$ has already been written to the designated space of the previous symbol's residual stream. If it has, $w_{n'}$ copies and shifts $\boldsymbol{e}_{l-1}$ into $\boldsymbol{e}_l$, and stores $\boldsymbol{e}_l$ and $\boldsymbol{d}_l \overset{\text{def}}{=} \boldsymbol{d}_{l-1} + w_{n'}\boldsymbol{e}_l$ in designated parts of its residual stream.

After $\lceil \log N \rceil$ layers, the residual stream at position $\lceil \log N \rceil + 1$ thus contains $\mathtt{B}(n)$.

This construction only requires positional encodings that contain $\mathtt{B}(n)$ and $\mathtt{B}(N-1)$, which are logarithmically computable. Moreover, the parameters of the transformer $\mathcal{T}_N$ only change with N in terms of the size of the matrices, while their structure remains the same—they either project onto specific coordinates (whose indices can be computed with counters in a logarithmic-space Turing machine) or shift the coordinates of vectors, which can also be done in logarithmic space. Thus, the family $\{\mathcal{T}_N\}_{N \in \mathbb{N}}$ is in $\mathtt{L}$-uniform $\mathtt{LPT}^1$. ■