# COMPLEXITY-THEORETIC TRADEOFFS BETWEEN PRECISION AND EXPRESSIVITY IN TRANSFORMERS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A common technique to accelerate inference and reduce the memory costs of transformers and related models is to restrict to low-precision weights and arithmetic. On the other hand, limiting numerical precision can constrain the set of functions that a transformer can represent. However, the precise relationship between precision and expressivity remains poorly understood. In this paper, we demonstrate a fine-grained theoretical tradeoff between expressivity and precision: For every $p$ we exhibit a function $\Gamma$ which tests equalities, and prove that a one-layer softmax transformer can compute $\Gamma$, with $p$ bits of precision, but not with $p-1$ bits of precision. Equality testing arises, both explicitly and implicitly, in many tasks that transformers are expected to perform, and our results show it is particularly sensitive to model precision. Our proofs combine explicit finite-precision transformer constructions with communication-complexity lower bounds, yielding a tight "one-bit" threshold.

## 1 INTRODUCTION

Transformers Vaswani et al. (2017) are the dominant architecture for modern sequence modeling, powering large language models and applications across multiple domains. However, their computational cost has motivated extensive efforts to reduce memory and inference requirements, including low-precision computation and quantization.

**Quantization.** Quantization reduces memory usage and computational cost by representing model parameters and intermediate computations using fewer bits. While empirical studies have demonstrated accuracy degradation under aggressive quantization, the fundamental relationship between numerical precision and transformer expressivity remains poorly understood.

In practical low-precision inference, a quantized value is typically represented by a low-bit integer together with a scaling factor that determines its numerical range. The same scale is generally shared by many values, for example across a tensor, channel, or quantization group, so the storage cost of the scale is amortized over a large number of low-precision entries. Consequently, the scaling metadata constitutes only a small overhead compared with storing an independent high-precision value for every model entry.

There are two commonly used formats to represent numbers: the *fixed-point precision* format and the *floating-point precision* format. In fixed-point precision format, a number is expressed as a $p$ bit binary value after scaling by an appropriate constant, while in floating-point precision format, the number is expressed in terms of a mantissa and an exponent. More details can be found in appendix A.

**Low-precision arithmetic.** Quantization reduces the memory and computational cost of transformers by representing values with fewer bits. In this work, we focus on low-precision arithmetic, where the primitive operations themselves are performed with bounded numerical precision, and study how this affects the functions that transformers can represent. Fixed-point quantization with INT8 and INT4 arithmetic has been widely explored for efficient transformer inference (Dettmers et al., 2022; Yao et al., 2022; Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2024). Floating-point formats such as FP8 have also been developed to enable efficient low-precision computation while maintaining a larger dynamic range (Micikevicius et al., 2022; Peng et al., 2023). More aggressive quantization



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

schemes, including one-bit and near one-bit representations, have further investigated the limits of low-precision models (Ma et al., 2024; Wang et al., 2025).

Empirical studies have shown that reducing precision can degrade model accuracy, particularly at very low precision or in the presence of activation outliers (Yao et al., 2022; Wei et al., 2023). Motivated by these observations, we study the representational limitations imposed by low-precision arithmetic from a theoretical perspective.

More specifically, we explore the effects of quantization with respect to accuracy and functionality from a theoretical perspective. Our goal is not to characterize the behavior of all practical quantized language models, which typically involve many layers, residual connections, and mixed-precision components. Instead, we identify explicit computational tasks for which numerical precision itself becomes a fundamental bottleneck. These results establish a precision-expressivity tradeoff in the transformer architecture and provide a theoretical framework for understanding one possible source of degradation under aggressive quantization. Our main result gives a sharp understanding of the number of bits of precision required to compute the Equality function with a one-layer, one-head transformer.

### 1.1 Our results: a precision tradeoff for Equality

In this paper, we focus on the expressive power of single-layer transformers. Although practical large language models are deep, the one-layer setting is important for several reasons. First, a single layer is the basic unit of attention based architectures like LLMs which perform one round of token-to-token information routing, and deeper transformers are obtained by composing many such rounds. Understanding what a single layer can or cannot compute identifies the primitive computational capabilities available to the architecture. Second, our results show that more layers are needed to compute the equality function in low-precision settings. However, recovering lost expressivity through additional architectural resources may introduce additional computational and memory costs, which motivates understanding the tradeoffs between precision and architectural complexity.

We prove that precision can be a fundamental computational bottleneck: even when the architecture and weights are allowed to change, reducing precision below a certain threshold prevents a one-layer transformer from computing some constructed equality tasks.

Equality is a fundamental computational primitive appearing in symbolic manipulation, retrieval-style operations, and other tasks requiring exact comparison of representations. We therefore use equality as a canonical task for studying precision-dependent expressivity. It has been observed that reducing precision does indeed degrade performance in similar instances, including mathematical reasoning, document classification, semantic similarity, approximate matching, etc., and some examples include (Li et al., 2025; Lin et al., 2024; Tissier et al., 2019; Shen et al., 2020; Wang et al., 2018).

We prove our tradeoff for the following representation of the equality function. For integers $m > 0$ and $n > 2m$, define $\mathcal{T}^n_{\mathsf{EQ}_m} : \{0,1\}^n \to \{0,1\}$, where, given $n$ bits as inputs, $\mathcal{T}^n_{\mathsf{EQ}_m}(x_1, \ldots, x_n)$ will output 1 if $x_i = x_{m+i}$ for all $i \in [m]$. This checks whether the first $m$ input bits are all equal to the next $m$ input bits (and does not care about the remaining input bits). As standard in the literature, a transformer computes a Boolean function $f : \Sigma^n \to \{0,1\}$ as follows. There are $n+1$ input tokens (instead of $n$) where the first $n$ tokens correspond to the $n$ inputs of the function, and the transformer should output the value of $f$ in the last $(n+1)$-th token. Each token is represented by a vector in $\mathbb{R}^d$, where $d$ is the embedding dimension. For more details, see appendix A.1.

Our main result characterizes exactly how many bits of precision are needed for testing equality in a one layer transformer. This should be interpreted as existential precision lower bounds: we identify tasks and representations where precision is a fundamental bottleneck, rather than claiming that all practical quantized language models fail under the same conditions.

**Theorem 1** (Fixed-point precision results, informal)**.** *For every integer $p \geq 2$ and $n > 4p$, there exists an input representation such that*

1. *no one-layer transformer that uses $\leq p-1$ bits of fixed-point precision can solve $\mathcal{T}^n_{\mathsf{EQ}_{2p-1}}$,*

2. *there exists a one-layer transformer that can solve $\mathcal{T}^n_{\mathsf{EQ}_{2p-1}}$ using $p$ bits of fixed-point precision.*

This gives a tight tradeoff for this equality representation: reducing precision by one bit prevents a one-layer transformer from computing this task under the specified computational model. A priori, it is not clear that losing one bit should matter: the parameters could be retrained, the embedding could be changed, and the attention weights could use the remaining precision in a different way. Our result shows that none of these adaptations can recover the lost capability for the constructed equality task. Note that contrary to prior works where precision has been considered as a coarse function of $n$ (either constant, logarithmic in $n$, or polynomial in $n$), we do not place any such restrictions on $p$. The tradeoff works for any value of $p$.

The techniques used to prove the above result are quite general, and such trade-offs are not limited to fixed-point precision formats. As an example to demonstrate this, we prove similar trade-offs for the floating-point precision format, another commonly used precision format.

**Theorem 2.** *(Floating-point precision results, informal) Given a function similar to $\mathcal{T}^n_{\mathsf{EQ}_{2p-1}}$: (1) no one-layer transformer using $\leq p-1$ bits of floating-point precision can solve $\mathcal{T}^n_{\mathsf{EQ}_{2p-1}}$; (2) a one-layer linear transformer with $p$ bits of floating-point precision, and a one-layer softmax transformer with $p + 2\log p + 2$ bits of floating-point precision, can solve $\mathcal{T}^n_{\mathsf{EQ}_{2p-1}}$.*

A more formal statement and discussions can be found in appendix C. In this case of floating-point precision, owing to the mantissa and exponent structure of this format, constructing an upper bound (a transformer that solves this variant of equality task) is more difficult. We discuss the challenges in more detail in section 2.3 below.

### 1.2 Comparison with related works

Our results concretely explain the widely observed phenomenon of empirical loss of expressivity when quantization is used. There are various other works related to this which study the expressivity limitations of transformers which have been described as follows.

**One-layer transformer lower bounds.** Representational limitations of one-layer transformers have been studied for a variety of functions and languages, including index lookup, nearest neighbor, Dyck languages, equality (Bhattamishra et al., 2024), parity and Dyck languages (Hahn, 2020), match3 and averaging problems (Sanford et al., 2023), induction heads task (Sanford et al., 2024b), function composition (Peng et al., 2024; Chakrabarti et al., 2026), counting related tasks (Yehudai et al., 2024), and function evaluation (Strobl et al., 2025).

Almost all of these one-layer lower bounds are based on communication complexity, and they establish that, for a one-layer transformer solving a particular task of interest, one must have $Hdp > \Omega(n)$, where $H$ is the number of heads, $d$ is the embedding dimension, and $p$ is the bits of precision. Hence precision is one factor in a broader resource bound. Such results do not isolate whether precision itself is the bottleneck, nor do they identify the exact precision level at which a task becomes computable. Recent work of (Kozachinskiy et al., 2025) using split-VC dimension removes precision dependence for several tasks. In contrast, we show that equality remains precision-sensitive even when $H = d = 1$.

In particular, this means the technique of (Kozachinskiy et al., 2025) cannot remove the dependence on $p$ for the Equality function.

Our bounds are also tight without any hidden constants (such as the asymptotic notation $\Omega$ prevalent in prior such lower bounds). We give a novel, explicit one-layer construction that matches our lower bound, using exactly one more precision bit. This distinguishes our result from prior lower bounds that show a task is hard for one-layer transformers but do not identify the precision threshold at which the task becomes computable. In this sense, our work gives a sharper, bit-level understanding of how numerical precision controls the expressivity of one-layer attention.

This should be also contrasted with (Bhattamishra et al., 2024), who use equality to study a different trade-off between one-layer and two-layer transformers. They show that equality can be solved by a two-layer transformer with logarithmic precision in the string length, while one-layer transformers

require a larger resource bound. Our result instead fixes the architecture to one layer transformer with $H = 1$ and $d = 1$, and asks a finer question: exactly how many bits of precision are necessary and sufficient for one attention layer to compute an equality-like function?

**Multilayer lower bounds.** In contrast, very little is proven about multi-layer transformers. Even proving that two-layer transformers cannot compute natural functions appears beyond current techniques, which largely rely on communication and circuit-complexity methods. The only unconditional lower bound to date was recently established by (Chen et al., 2025). However, their result is only for decoder-only transformers, and their proof critically relies on the intricacies of decoder units and one-sided information flow for a specific task that do not generalize to transformers with any encoder units. It will be an interesting open problem to understand multi-layer transformers better, and more ambitiously how the parameters of depth, number of heads, embedding dimension and precision relate to each other.

**Upper bounds on representational strength.** Prior work has studied the theory of numerical precision in transformers, but only at a coarser level than we consider here. For example, (Merrill & Sabharwal, 2023a;b; Strobl, 2023) showed that transformers using precision logarithmic in the number of tokens can be computed using TC0 circuits.[1] Later work of (Chiang, 2025) improved this result by showing that transformers that use polynomial precision can also be computed by TC0 circuits. More works on transformer expressivity along with their precision can be found in the comprehensive survey of (Strobl et al., 2024).

## 2 TECHNIQUE OVERVIEW

In this section, we give an overview of the main results of precision-expressivity tradeoffs for fixed-point arithmetic. The fixed-point precision format for $p$ bits works as: first, the values of the numbers are scaled to bring them within the range $-(2^{p-1}-1)$ to $(2^{p-1}-1)$, and then rounded off to the closest integer $(p-1)$-bit integer. This $(p-1)$-bit integer along with one sign bit gives the representation of the number in $p$-bit fixed-point precision format.

**Input representations.** As described before, the function for which we are proving the tradeoff is the Equality function. However, we now discuss how the inputs to the function are represented as input tokens of the transformer.

For an integer $n > 0$, a transformer is a function $\mathsf{Tf} : \Sigma^{n+1} \to \Sigma_{out}^{n+1}$, that takes as inputs $(n+1)$ input-tokens from the set of alphabets $\Sigma$, and outputs $(n+1)$ output-tokens from the alphabet set $\Sigma_{out}$.

Given a function, $f : \Lambda^N \to \Sigma_{out}$, a transformer $\mathsf{Tf} : \Sigma^{n+1} \to \Sigma_{out}^{n+1}$ is said to *compute* $f$ for an *input representation* $\mathcal{T} : \Lambda^N \to \Sigma^n$, for integers $N, n > 0$, if, when the transformer is given as input $n+1$ tokens, where the $i$-th token is $\mathcal{T}(x_1, \ldots, x_N)_i$ for $i \in [n]$, and the $(n+1)$-th token is $r$ (a placeholder which is a predefined constant and can be considered as zero), outputs the value of $f(x_1, \ldots, x_N)$ in the $(n+1)$-th token of the output. More specifically,

$$\mathsf{Tf}(\mathcal{T}(x_1, \ldots, x_N)_1, \ldots, \mathcal{T}(x_1, \ldots, x_N)_n, r)_{n+1} = f(x_1, \ldots, x_n).$$

For example, a simple input representation for $n = N$ and $\Lambda = \Sigma$ can be given by $\mathcal{T}(x_1, \ldots, x_n)_i = x_i$ for all $i \in [n]$, i.e., each of the $i$-th input token of the transformer contains the $i$-th input to the function $x_i$, and the transformer is expected to compute the value of $f(x_1, \ldots, x_N)$.

Using this definition, we will prove our main results. Here, a transformer is expected to compute the Equality function, $\mathsf{EQ}_m(\mathbf{y}, \mathbf{z})$, i.e., for given strings $\mathbf{y}, \mathbf{z} \in \{0,1\}^m$, the transformer will output 1 if $\mathbf{y} = \mathbf{z}$, otherwise 0. The input representation will be similar to the simple one described above, where the first $m$ tokens will contain one of each bit of $\mathbf{y}$, followed by the next $m$ tokens which will contain one of each bit of $\mathbf{z}$, followed by placeholder zeros to fill up the number of input tokens of the transformer to $n+1$. As is common in theoretical studies of transformer expressivity, our results are stated with respect to explicit input representations. The choice of representation is part of the

[1] A TC0 circuit is a Boolean circuit of polynomial size and constant depth consisting of AND, OR, NOT and threshold gates.

computational problem: different representations can change the resources required by a transformer. Our results therefore establish the existence of natural precision-sensitive representations rather than a universal lower bound over all possible encodings.

In all the results, the underlying function that the transformer computes is always Equality $\mathsf{EQ}_m$. However, the results are sensitive to input representations, due to which we modify the input representation somewhat so that testing Equality of two strings can be checked in the required precision format, and to prove lower and upper bounds.

The intuition behind both bounds is that bounded precision limits the amount of information that can be transferred through the attention computation. In the lower bound, we formalize this by converting the attention computation into a communication protocol: the intermediate state transmitted between two portions of the sequence contains only a bounded number of bits. The upper bound constructions show that this information bottleneck is tight by explicitly encoding equality information into the available precision budget.

**Lower bounds.** We first prove the lower bounds, i.e., transformers cannot compute the Equality function. At a high level, we use communication complexity based arguments to derive these lower bounds, which are tight and do not contain hidden constants. As discussed before, our focus is especially on areas where ideas of (Kozachinskiy et al., 2025) do not apply, and precision plays an important role.

**Upper bounds.** For the upper bounds, we give an explicit construction of a transformer to show that transformers can compute the Equality function (check if $\mathbf{y} = \mathbf{z}$) using query dimension $d_q = 3$ and value dimension $d_v = 1$. This construction has the basic idea: the numerator of the softmax self-attention is used to check if the first halves of the two strings are equal, and the denominator is used to check if the second halves are equal. This is performed by choosing the embedding function such that the value matrices of the tokens from the first halves of $\mathbf{y}, \mathbf{z}$ contribute to the numerator, while the remaining are zeros. For the denominator, the embedding function makes query and key matrices such that the values of $e^{\langle Q_i, K_j \rangle}$ are small, and ignored for the first half of the strings, while the only contribution to the value of the denominator is from the tokens corresponding to the second halves of $\mathbf{y}, \mathbf{z}$.

We use careful low-precision arithmetic to prevent information getting lost due to rounding. An overview of the proofs have been given in the following subsections, and the complete proofs can be found in Appendix B.

### 2.1 Tradeoff with fixed-point precision: simple equality function

First, we show that using a simple input representation, $\mathsf{EQ}_m$ can be computed by a transformer using $p$ bits, but cannot be computed using $p - 2$ bits, where $m := 2p - 3$.

Formally, we start with two strings $\mathbf{y}, \mathbf{z} \in \{0,1\}^m$, and are interested in computing the function $\mathsf{EQ}_m(\mathbf{y}, \mathbf{z})$ with the promise conditions:

- $m$ is odd.
- The binary values satisfy $\mathbf{y} \leq \mathbf{z}$.

The input representation $\mathcal{T}_m^0 : \{0,1\}^{2m} \to \{0,1\}^n$ is as follows:

1. $\mathcal{T}_m^0(\mathbf{y}, \mathbf{z})_i = y_i$ for $i \in [m]$.
2. $\mathcal{T}_m^0(\mathbf{y}, \mathbf{z})_{i+m} = z_i$ for $i \in [m]$.
3. $\mathcal{T}_m^0(\mathbf{y}, \mathbf{z})_i = 0$ for all $i \in [2m + 1 : n]$.

This input representation essentially implies that the first $m$ tokens contain each bit of $\mathbf{y}$, the next $m$ tokens contain each bit of $\mathbf{z}$, and the rest are zeros.

The output of the transformer (the value at the last output token) will be 1 if $\mathbf{y} = \mathbf{z}$ and $m$ is odd, 0 if $\mathbf{y} \neq \mathbf{z}$ and the promise conditions are met, and don't care otherwise

With this representation, we prove the following result:

**Theorem 3** (Tradeoff with fixed-point precision for equality)**.** *For every integer $p \geq 2$ and every integer $n > 4p$, choosing $m = 2p - 3$,*

1. *no one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head, can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^0$, using $p-2$ bits of precision,*
2. *there exists a one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head, that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^0$ using $p$ bits of precision in the fixed-point precision format.*

Even though this is not a tight tradeoff, this is enough to show a separation in real-world quantization regimes for fixed-point precision, where computations are usually often on $4, 8, 16$ bits. However, this gap can be bridged to make the tradeoff tight, as will be shown in the next section.

The proof of Theorem 3 will follow from the following two lemmas. Lemma 2.2 states the lower bound, showing that for our choice of parameter settings ($m = 2p - 3$), no single layer transformer with $p - 2$ bits of precision can compute $\mathsf{EQ}_m$. Lemma 2.3 states the near matching upper bound, showing that $\mathsf{EQ}_m$ can be computed using $p + 1$ bits of precision. (Note that since we have set $m = 2p - 3$ and $m$ is odd, $\lceil m/2 \rceil - 1 = p - 2$ and $\lceil m/2 \rceil + 1 = p$.)

**Lemma 1** (Fixed-point lower bound)**.** *Given the input representation $\mathcal{T}_m^0$, any one-layer transformer with $d_v = 1$ and $H = 1$, cannot compute $\mathsf{EQ}_m$ with $\leq \lceil m/2 \rceil - 1$ bits of precision.*

**Lemma 2** (Fixed-point upper bound)**.** *There exists a one-layer transformer with $d_v = 1$ and $H = 1$, that uses $\lceil m/2 \rceil + 1$ bits of fixed-point precision to compute $\mathsf{EQ}_m$ in the input representation $\mathcal{T}_m^0$.*

Lemma 1 is proven via a reduction to a communication complexity lower bound, similar at a high level to other communication complexity based lower bound proofs for transformers (Peng et al., 2024; Sanford et al., 2023). Next we give a proof overview for Lemma 2. The full proofs of both Lemmas can be found in Appendix B.

*Proof overview.* The main idea of this construction is inspired by the proof of the lower bound, where both the numerator and denominator of self-attention are used to transfer information. The construction uses the numerator to check equality in the first half (whether $\mathbf{y}_{1:\lceil m/2 \rceil} = \mathbf{z}_{1:\lceil m/2 \rceil}$), and then use the denominator to check the rest.

For the numerator part, we will choose the embedding such that the final numerator value is equal to the maximum possible value in this representation, $2^{\lceil m/2 \rceil} - 1$, if they are equal. We ensure that by making the summation value is

$$\sum_{i=1}^{n+1} e^{\langle Q_{n+1}, K_i \rangle} V_i = -\mathbf{y}_{1:\lceil m/2 \rceil} + \mathbf{z}_{1:\lceil m/2 \rceil} + \underbrace{1 \ldots 1}_{\lceil m/2 \rceil}.$$

In order to obtain this value, we design the embedding and the $W^Q, W^K, W^V$ matrices in a way such that

$$\sum_{i=1}^{\lceil m/2 \rceil} e^{\langle Q_{n+1}, K_i \rangle} V_i = -\mathbf{y}_{1:\lceil m/2 \rceil}, \qquad \sum_{i=m+1}^{m+\lceil m/2 \rceil} e^{\langle Q_{n+1}, K_i \rangle} V_i = \mathbf{z}_{1:\lceil m/2 \rceil}.$$

Along with this, there will be a dummy token in the $(2m + 1)$-th index such that

$$e^{\langle Q_{n+1}, K_{2m+1} \rangle} V_{2m+1} = 2^{\lceil m/2 \rceil} - 1,$$

and the rest will be zeros. The value of the numerator will be exactly $(2^{\lceil m/2 \rceil} - 1)$ if $\mathbf{y}_{1:\lceil m/2 \rceil} = \mathbf{z}_{1:\lceil m/2 \rceil}$, otherwise a large value (commonly used as $\mathsf{float}(\text{'inf'})$ in PyTorch), which follows from the promise on $\mathbf{y}, \mathbf{z}$.

For the denominator, we enforce $\sum_{i=1}^{\lceil m/2 \rceil} e^{\langle Q_{n+1}, K_i \rangle}$ and $\sum_{i=m+1}^{m+\lceil m/2 \rceil} e^{\langle Q_{n+1}, K_i \rangle}$ are small constants irrespective of $\mathbf{y}, \mathbf{z}$, and

$$\sum_{i=\lceil m/2 \rceil + 1}^{m} e^{\langle Q_{n+1}, K_i \rangle} + \sum_{i=m+\lceil m/2 \rceil + 1}^{2m} e^{\langle Q_{n+1}, K_i \rangle} = \bar{\mathbf{y}}_{\lceil m/2 \rceil + 1:m} + \mathbf{z}_{\lceil m/2 \rceil + 1:m} + 2^{-1}.$$

This is a fixed value if $\mathbf{y}_{\lceil m/2\rceil+1:m} = \mathbf{z}_{\lceil m/2\rceil+1:m}$, otherwise a larger value.

Summing up, if $\mathbf{y}_{1:m} = \mathbf{z}_{1:m}$, then the numerator will be $(2^{\lceil m/2\rceil} - 1)$ and the denominator will be a fixed value $2^{(\lceil m/2\rceil-1)} - 1 + 2^{-1}$, otherwise either the numerator make the output infinity (float('inf')), or the denominator will a larger value (therefore the output different from $\frac{2^{\lceil m/2\rceil}-1}{2^{\lceil m/2\rceil-1}-2^{-1}} = 2$), either of which can be detected with a simple MLP layer after the self-attention computation. □

### 2.2 Tradeoff with fixed-point precision: Theorem 1

In this section, we prove a tight one-bit tradeoff. We use the same function $\mathsf{EQ}_m$ as before, along with the same promise that $m$ is odd and $\mathbf{y} \leq \mathbf{z}$.

For ease of notation, we consider indexing from $-1$ up $n$. The input representation $\mathcal{T}^1_m : \{0,1\}^{2m} \to \{0,1\}^{n+2}$ is defined as follows:

1. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_{-1} = 0$.
2. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_0 = y_1$.
3. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_i = y_i$ for $i \in [m]$.
4. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_i = z_1$ for $i \in \{m+1, m+2\}$.
5. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_{m+3} = 0$.
6. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_i = z_{i-m-2}$ for $i \in [m+4 : 2m+2]$.
7. $\mathcal{T}^1_m(\mathbf{y},\mathbf{z})_i = 0$ for all $i \in [2m+3 : n]$.

Formally, Theorem 1 can be stated as follows.

**Theorem 4** (Tight tradeoff with fixed-point precision)**.** *For every integer $p > 2$ and every integer $n > 4p$, choosing $m = 2p - 1$,*

1. *no one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head, can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}^1_m$, using $p - 1$ bits of precision,*
2. *there exists a one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head, that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}^1_m$ using $p$ bits of precision in the fixed-point precision format.*

As we will see in the proof, padding tokens do increase the expressivity of the transformer and helps us achieve this exact trade off for this function. From a theoretical standpoint, using padding tokens do indeed improve the expressivity of transformers was also proved by (Merrill & Sabharwal, 2025). In our case, we use this representation to prevent overflow as we are dealing with the exact number of bits that can be used to compute the function. We first state the lower bound present in Appendix B.2.

**Lemma 3** (Lower bound)**.** *Given the input representation $\mathcal{T}^1_m$, any one-layer transformer with $d_v = 1$ and $H = 1$ that computes $\mathsf{EQ}_m$ requires precision $\geq \lceil m/2 \rceil$ bits.*

Now, we prove that there exists a transformer with $\lceil m/2 \rceil$ bits of precision which computes $T_m$.

**Lemma 4** (Upper bound)**.** *There exists a one-layer transformer with $d_v = 1$ and $H = 1$, that uses $\lceil m/2 \rceil$ bits of fixed-point precision to compute $\mathsf{EQ}_m$ in the input representation $\mathcal{T}^1_m$.*

*Proof overview.* The construction is similar to the proof of Lemma 2, where the numerator of the self-attention checks equality in the first half (whether $\mathbf{y}_{1:\lceil m/2\rceil} = \mathbf{z}_{1:\lceil m/2\rceil}$), and then the denominator checks the rest.

For the numerator part, we will choose the embedding such that the final numerator value is equal to one if they are equal. We make the summation value $\sum_{i=-1}^{n+1} e^{\langle Q_{n+1}, K_i\rangle} V_i = \mathbf{y}_{1:\lceil m/2\rceil} - \mathbf{z}_{1:\lceil m/2\rceil} + 1$, by ensuring $\sum_{i=-1}^{\lceil m/2\rceil-1} e^{\langle Q_{n+1}, K_i\rangle} V_i$ gives $\mathbf{y}_{1:\lceil m/2\rceil}$, and $\sum_{i=m}^{m+\lceil m/2\rceil} e^{\langle Q_{n+1}, K_i\rangle} V_i$ gives $\mathbf{z}_{1:\lceil m/2\rceil}$. However, this is not directly possible as before, since $\mathbf{y}_{1:\lceil m/2\rceil}, \mathbf{z}_{1:\lceil m/2\rceil}$ use $\lceil m/2 \rceil$ bits, but in $\lceil m/2 \rceil$ bits of fixed-point precision, one bit is allocated to the sign bit, giving us $\lceil m/2 \rceil - 1$ positions. To circumvent this difficulty, we use a dummy token in the $(-1)$-th position, such that

$e^{\langle Q_{n+1},K_{-1}\rangle}V_{-1} = -(2^{\lceil m/2\rceil-1}-1)$, and is used to add a negative shift to the value of $\mathbf{y}_{1:\lceil m/2\rceil}$ in order to bring it to the permissible range of $\lceil m/2\rceil$ precision bits.

There is one more difficulty, the value $\mathbf{y}_1 2^{\lceil m/2\rceil-1}$ might be infinity while computing $\mathbf{y}_{1:\lceil m/2\rceil}$. In order to prevent this, we split this value amongst two tokens corresponding to 0 and 1 positions, where each of them has the value $e^{\langle Q_{n+1},K_i\rangle}V_i = \mathbf{y}_1 2^{\lceil m/2\rceil-2}$. This will give the respective first and second part of the summation of the numerator as

$$\sum_{i\in[-1:m]} e^{\langle Q_{n+1},K_i\rangle}V_i = \mathbf{y}_{1,\lceil m/2\rceil} - \underbrace{1\ldots1}_{\lceil m/2\rceil-1}, \quad \sum_{i\in[m+1:2m+2]} e^{\langle Q_{n+1},K_i\rangle}V_i = -\mathbf{z}_{1,\lceil m/2\rceil} + \underbrace{1\ldots1}_{\lceil m/2\rceil-1}.$$

Finally, we choose the embedding such that $e^{\langle Q_{n+1},K_{2m+3}\rangle}V_{2m+3} = 1$. Therefore, the value of the numerator will be exactly 1 if $\mathbf{y}_{1:\lceil m/2\rceil} = \mathbf{z}_{1:\lceil m/2\rceil}$, otherwise $\leq 0$ from the promise on $\mathbf{y}$ and $\mathbf{z}$.

For the denominator, again we ensure that $\sum_{i=1}^{\lceil m/2\rceil} e^{\langle Q_{n+1},K_i\rangle}$ and $\sum_{i=m+1}^{m+\lceil m/2\rceil} e^{\langle Q_{n+1},K_i\rangle}$ are small constants irrespective of $\mathbf{y}, \mathbf{z}$ and ignored, while

$$\sum_{i=\lceil m/2\rceil+1}^{m} e^{\langle Q_{n+1},K_i\rangle} + \sum_{i=m+\lceil m/2\rceil+3}^{2m+2} e^{\langle Q_{n+1},K_i\rangle} = \bar{\mathbf{y}}_{\lceil m/2\rceil+1:m} + \mathbf{z}_{\lceil m/2\rceil+1:m}.$$

This is a fixed value for $\mathbf{y}_{\lceil m/2\rceil+1:m} = \mathbf{z}_{m+\lceil m/2\rceil+1:2m}$, otherwise a larger than that due to the promise.

Summing up, if $\mathbf{y} = \mathbf{z}$, then the numerator will be 1 and the denominator will be a fixed value $2^{(\lceil m/2\rceil-1)}-1$, otherwise either the numerator will be non-positive or the denominator a large value (i.e., float('inf')), either of which can be detected by the final MLP layer. □

*Proof of Theorem 1.* We choose $m := 2p-1$. From Lemma 3, $\mathsf{EQ}_m$ cannot be computed by a transformer using $\leq \lceil m/2\rceil - 1 = p-1$ bits for input representation $\mathcal{T}_m^1$, but can be computed by a transformer using $\lceil m/2\rceil = p$ bits (Lemma 4). □

### 2.3 Overview of floating-point results

Another commonly used precision format is the *floating-point precision* format. Unlike fixed-point arithmetic, floating-point numbers represent values using both a mantissa and an exponent. Specifically, a value is represented as

$$\pm 1.\mu \times 2^{\pm\varepsilon},$$

where $\mu$ denotes the mantissa bits and $\varepsilon$ denotes the exponent bits. This allows floating-point arithmetic to represent numbers with a much larger dynamic range than fixed-point arithmetic, but introduces additional challenges due to exponent manipulation and rounding.

We prove analogous precision-expressivity tradeoffs for floating-point transformers. The main difficulty is that the exponent information used to represent values can be amplified through the attention mechanism. In particular, softmax attention applies an exponential transformation to query-key inner products, meaning that the exponent will be exponentiated twice when computing $e^{\langle Q_i,K_j\rangle}$, and summing these values to produce another floating-point number can lead to overflow in the bounded-precision setting. To prevent overflow and preserve bounded-precision computation, our constructions use structured input representations where tokens contain tuples of bits. One component controls the floating-point exponent, while the remaining bits encode the mantissa. The constructions carefully separate exponent information from mantissa information so that intermediate computations remain within the representable range.

We first consider linear attention, where the absence of the softmax normalization allows an exact precision threshold.

**Theorem 5** (Tradeoff for linear transformers)**.** *For every integer $p > 6$ and every integer $n > 2p$, choosing $m = p$, there exists an input representation $\mathcal{T}_m^2$ such that:*

1. *No one-layer linear transformer with $n$ tokens having value dimension $d_v = 1$ and one head can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^2$ using $p-1$ bits of precision,*

2. *There exists a one-layer linear transformer with $n$ tokens having value dimension $d_v = 1$ and one head that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^2$ using $p$ bits of precision in the floating-point precision format.*

The lower bound follows from a communication complexity argument. Since the linear attention computation maintains a bounded-precision accumulator, the intermediate state passed between two parts of the input sequence contains only a limited number of bits. The resulting communication protocol requires at least $m$ bits, implying that precision below $m$ bits cannot solve equality. The upper bound constructs a floating-point representation where the two input strings are encoded as mantissa-exponent values whose difference is exactly zero if and only if the strings are equal.

For softmax attention, we obtain a similar tradeoff, but with an additional logarithmic precision overhead, that arises from the need to represent the attention weights accurately in this format.

**Theorem 6** (Tradeoff for softmax transformers)**.** *For every integer $p > 0$ and every integer $n > 4p$, choosing $m = 2p - 1$, there exists an input representation $\mathcal{T}_m^3$ such that:*

1. *No one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^3$ using $p - 1$ bits of precision,*

2. *There exists a one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^3$ using $p + 2\lceil \log_2 p \rceil + 2$ bits of precision in the floating-point precision format.*

The softmax construction approximates a power-of-two attention pattern. Integer key coordinates are chosen such that their exponentials generate powers of two, allowing the numerator and denominator of attention to separately encode different parts of the equality test. However, because the implemented query-key products use a finite-precision approximation of $\ln 2$, additional mantissa bits are required to guarantee that the computed attention weights remain separated after rounding. This introduces the additional $O(\log p)$ precision overhead.

The use of tuple-based input representations follows a common approach in theoretical studies of transformer expressivity (Strobl et al., 2025; Sanford et al., 2024a; Zhou et al., 2024), and input formatting can significantly affect representation capabilities. The complete proofs are provided in Appendix C.

Together, these results show that the precision-expressivity tradeoff extends beyond fixed-point arithmetic. Floating-point representations provide greater dynamic range, but precision remains a fundamental computational resource: reducing precision below the required threshold prevents a one-layer transformer from computing the constructed equality tasks.

## 3 Conclusion

This paper theoretically demonstrates that expressivity loss can arise from quantization: there exist computational tasks for which reducing numerical precision creates an unavoidable limitation for transformer representations. It further aims to develop a class of tasks on which quantization is difficult.

Our results focus on one-layer transformers and explicit input representations. Extending these precision tradeoffs to deep transformers, learned representations, and practical language modeling tasks remains an important open problem. We view the present results as a first step toward understanding how numerical precision interacts with architectural resources in transformer models.

The precision-expressivity trade-offs proved here can be extended beyond $H, d_v = 1$. For increasing $H$, the computations can be performed in parallel by dividing the strings $\mathbf{y}, \mathbf{z}$ into $H$ parts to compute $\mathsf{EQ}(\mathbf{y}, \mathbf{z})$ and obtain a tight trade-off. However, for $d_v \geq 2$, similar constructions will show that there exists a function that can be computed by the transformer with $p = {}^{m}/_{Hd_v}$, but cannot be computed by a transformer with $p \leq {}^{m}/_{2Hd_v} - 1$. However, finding such a trade-off for the multi-layer transformer remains elusive due to the lack of lower-bound techniques. Overcoming this challenge will help understand quantization methods better.

### AI USE STATEMENT

In this work, we used generative AI tools to assist with improving the clarity and organization of the manuscript, simplifying the presentation of proofs that were originally developed by the authors, and assisting with reference checking. We did not use generative AI tools to generate the core research ideas, theorem statements, mathematical arguments, experimental results, or scientific claims of this work. All AI-assisted modifications were reviewed and verified by the authors, who take full responsibility for the final content of the manuscript.

### REPRODUCIBILITY STATEMENT

This work presents theoretical results. All assumptions, definitions, computational models, and input representations used in our analysis are explicitly specified in the paper and appendices. Complete proofs of all stated theorems and lemmas are provided in Appendices B and C. The transformer constructions, precision models, rounding conventions, and parameter choices required to verify the upper bounds are fully described in the appendices. No external datasets or experimental resources are required to reproduce the results.

## REFERENCES

Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra. Linear attention is (maybe) all you need (to understand transformer optimization). In *International Conference on Learning Representations (ICLR)*, 2024.

Josh Alman and Zhao Song. Fast attention requires bounded entries. *Advances in Neural Information Processing Systems*, 36:63117–63135, 2023.

Satwik Bhattamishra, Michael Hahn, Phil Blunsom, and Varun Kanade. Separations in the representational capabilities of transformers and recurrent architectures. *Advances in Neural Information Processing Systems*, 37:36002–36045, 2024.

Sayak Chakrabarti, Toniann Pitassi, and Josh Alman. Poly-attention: a general scheme for higher-order self-attention. *International Conference on Learning Representations (ICLR), (to appear)*, 2026.

Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer. In *Foundations of Computer Science (FOCS)*, 2025.

David Chiang. Transformers in uniform TC$^0$. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=ZA7D4nQuQF.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

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

Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. Polysketchformer: fast transformers via sketching polynomial kernels. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International conference on algorithmic learning theory*, pp. 597–619. PMLR, 2023.

Soroush Abbasi Koohpayegani and Hamed Pirsiavash. Sima: Simple softmax-free attention for vision transformers. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2607–2617, 2024.

Alexander Kozachinskiy, Felipe Urrutia, Hector Ignacio Jimenez Orellana, Tomasz Steifer, Germán Pizarro, Matías Daniel Fuentes Álvarez, Francisco Meza Vásquez, Cristian Buc Calderon, and Cristobal Rojas. Strassen attention, split VC dimension and compositionality in transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=C9JMOzKD9k.

Eyal Kushilevitz. Communication complexity. In *Advances in Computers*, volume 44, pp. 331–360. Elsevier, 1997.

Zhen Li, Yupeng Su, Runming Yang, Congkai Xie, Zheng Wang, Zhongwei Xie, Ngai Wong, and Hongxia Yang. Quantization meets reasoning: Exploring llm low-bit quantization degradation for mathematical reasoning. *arXiv preprint arXiv:2501.03035*, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. *Advances in neural information processing systems*, 36:52453–52463, 2023a.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023b.

William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=O1abxStFcy.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the transformer architecture. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=KidynPuLNW.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.

Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, et al. Theory, analysis, and best practices for sigmoid self-attention. *arXiv preprint arXiv:2409.04431*, 2024.

Clayton Sanford, Daniel J Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. *Advances in Neural Information Processing Systems*, 36:36677–36707, 2023.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. *Advances in Neural Information Processing Systems*, 37:78320–78370, 2024a.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. One-layer transformers fail to solve the induction heads task. *arXiv preprint arXiv:2408.14332*, 2024b.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 8815–8821, 2020.

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

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Lena Strobl, Dana Angluin, and Robert Frank. Concise one-layer transformers can do function evaluation (sometimes). *arXiv preprint arXiv:2503.22076*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Julien Tissier, Christophe Gravier, and Amaury Habrard. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7104–7111, 2019.

Muhammed Ustaomeroglu and Guannan Qu. A theoretical study of (hyper) self-attention through the lens of interactions: Representation, training, generalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.

Hongyu Wang, Shuming Ma, Lingxiao Ma, Lei Wang, Wenhui Wang, Li Dong, Shaohan Huang, Huaijie Wang, Jilong Xue, Ruiping Wang, et al. Bitnet: 1-bit pre-training for large language models. *Journal of Machine Learning Research*, 26(125):1–29, 2025.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=sZEAMUizsd.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35:27168–27183, 2022.

Gilad Yehudai, Haim Kaplan, Asma Ghandeharioun, Mor Geva, and Amir Globerson. When can transformers count to n? *arXiv preprint arXiv:2407.15160*, 2024.

Cai Zhou, Rose Yu, and Yusu Wang. On the theoretical expressive power and the design space of higher-order graph transformers. In *International Conference on Artificial Intelligence and Statistics*, pp. 2179–2187. PMLR, 2024.

# A NOTATIONS AND PRELIMINARIES

For integers $n, m$, we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$, $[m : n]$ to denote the set of integers $\{m, m+1, \ldots, n\}$, where $m \leq n$, and $(m, n)$ (resp. $[m, n]$) to denote the set of real numbers between $m$ and $n$, end points excluded (resp. end points included). For a matrix $A \in \mathbb{R}^{n \times m}$ and for $i \in [n]$, $A_i$ denotes the $i$-th row vector $(A_{i,1}, \ldots, A_{i,m}) \in \mathbb{R}^m$. Similarly, for $j \in [m]$, $A_{:,j}$ denotes the column vector $(A_{1,j}, \ldots, A_{n,j}) \in \mathbb{R}^n$.

Strings are emboldened, and for a string $\mathbf{a} \in \{0,1\}^n$, $\mathbf{a}$ will denote the value of the number whose binary expression is $a_1 a_2 \ldots a_n$, and $\mathbf{a}_{p:q}$ represents the substring $a_p a_{p+1} \ldots a_q$, for $1 \leq p < q \leq n$.

## A.1 TRANSFORMERS

We define the multi-layer transformer architecture as follows.

**Definition 1.** *An $L$-layer transformer is a function $\Sigma^n \to \Sigma^n$, where $\Sigma$ is an alphabet set and $n$ is the number of tokens, also regarded as the sequence length. It consists of:*

1. *an embedding function $\mathbb{E} : \Sigma \to \mathbb{R}^d$, where $d$ is the embedding dimension,*
2. *several transformer units for each layer that are functions $\mathsf{Tf}_i : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ for $i \in [L]$,*
3. *and a final output projection function, also known as language modeling head, $f_{LM} : \mathbb{R}^d \to [0,1]^{|\Sigma|}$, which give the probabilities of every alphabet from the alphabet set $\Sigma$.*

*Each transformer layer consists of two components, a self-attention function $f^i_{SA} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d_v}$ and an MLP function $f^i_{MLP} : \mathbb{R}^{d_v} \to \mathbb{R}^d$, where $d_v$ is the value dimension. Now, an $L$-layer transformer is the composition of the functions:*

$$f_{LM} \circ f^L_{MLP} \circ f^L_{SA} \circ \ldots \circ f^1_{MLP} \circ f^1_{SA} \circ \mathbb{E} :\to \Sigma^n \to [0,1]^{|\Sigma|}.$$

The core operation in a transformer layer is the self-attention function, which is used to compute cross-token interactions. Soft-max self-attention is the most commonly used form of self-attention and is defined as follows.

**Definition 2** (Softmax self-attention)**.** *Given parameters $W^Q, W^K \in \mathbb{R}^{d \times d_q}$, and $W^V \in \mathbb{R}^{d \times d_v}$, where $d_q$ is the query dimension and $d_v$ is the value dimension, the soft-max self-attention is a function $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$, where the $i$-th row of the output is computed as*

$$SA_i = \frac{\sum_{j \in [n]} e^{(X_i W^Q (W^K)^T X_j^T)} X_j W^V}{\sum_{j \in [n]} e^{(X_i W^Q (W^K)^T X_j^T)}},$$

*for all $i \in [n]$.*

We will commonly refer to $XW^Q$ as the query matrix $Q$, $XW^K$ as the key matrix $K$ and $XW^V$ as the value matrix $V$. Also, $\sum_{j \in [n]} e^{(X_i W^Q (W^K)^T X_j^T)} X_j W^V$ will be called the numerator term and $\sum_{j \in [n]} e^{(X_i W^Q (W^K)^T X_j^T)}$ the denominator term.

We also define the linear self-attention mechanism, which is widely used as a more computationally efficient counterpart of the soft-max self-attention mechanism.

**Definition 3** (Linear self-attention)**.** *Given parameters $W^Q, W^K \in \mathbb{R}^{d \times d_q}$, and $W^V \in \mathbb{R}^{d \times d_v}$, the linear self-attention is a function $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$, where the $i$-th row of the output is computed as*

$$SA_i = \sum_{j \in [n]} X_i W^Q (W^K)^T X_j^T X_j W^V,$$

*for $i \in [n]$.*

In our constructions, given tokens $X_1, \ldots, X_n$ we will consider them to contain positional encodings, and the embedding as a function $[n] \times \Sigma \to \mathbb{R}^d$, where the inputs to the function are the index of the token and the value in $\Sigma$, and the output $f(i, X_i)$ is a vector in $\mathbb{R}^d$.

**Definition 4** (Final MLP layer)**.** *The final MLP layer is a function $f_{MLP} : \mathbb{R}^{d_v} \to \mathbb{R}^d$, which has $d_v$ input nodes, $d$ output nodes, and $d_{ff}$ nodes in one hidden layer in between them (usually $d_{ff} = 4d$). Each edge from nodes one layer to the next is associated with weights and biases.*

*The weights of the $i$-th input node to the $j$-th hidden node is denoted as $w^1_{i,j}$, and the bias of the $j$-th hidden node as $b^1_j$, for $i \in [d_v], j \in [d_{ff}]$. Subsequently, the weight of the $j$-th hidden node to the $k$-th output node is denoted as $w^2_{j,k}$, and the bias of the $k$-th output node as $b^2_k$, for $j \in [d_{ff}], k \in [d]$.*

*Given the values of the hidden nodes as $v_1, \ldots, v_{d_v}$, the $j$-th hidden node will have the value*

$$h_j = \max\{0, \sum_{i \in [d_v]} w^1_{i,j} v_i + b_j\},$$

*for $j \in [d_{ff}]$, and the $k$-th output node will have the value*

$$\sum_{j \in [d_{ff}]} w^2_{j,k} h_j + b_k,$$

*for $k \in [d]$.*

### A.2 Communication complexity

We use a model of communication with two players Alice and Bob, each of whom have unbounded computational power. In order to solve the problem, Alice has some bits, Bob has some bits and they want to communicate to compute a function known to all. Furthermore, we only allow one-way communication which is communication only from Alice to Bob. Interested readers can see (Kushilevitz, 1997) for more background on communication complexity.

**Definition 5** (Fooling sets)**.** *Given a function $f$, a set $F = \{(x_i, y_i) \mid i \in [\ell]\}$ is called a fooling set for $f$ of size $\ell$ if:*

- *for all $i \in [\ell]$, $f(x_i, y_i)$ has the same value,*
- *for all $i \neq j$, $i, j \in [\ell]$, either $f(x_i, y_j)$ or $f(x_j, y_i)$ has a value different from $f(x_i, u_i)$.*

**Lemma 5** (Folklore)**.** *If the function $f$ has a fooling set of size $\ell$, then the communication complexity of $f$ is at least $\lceil \log_2 \ell \rceil$.*

### A.3 Computations with bounded precision

In this article, we use a certain kind of precision defined as follows.

**Definition 6** (Fixed-point precision)**.** *We define the* fixed-point precision *by a triple $\langle s, a, b \rangle$, where $a, b$ are binary expression of integers and $s$ is the sign bit. The value is given by*

$$\langle 0, a, b \rangle = a + 2^{-|b|} b = a \cdot b \,,$$

*and*

$$\langle 1, a, b \rangle = -(a + 2^{-|b|} b) = -a \cdot b \,.$$

*We say that this requires $p$ bits of precision if $1 + |a| + |b| = p$.*

**Interpretation of the precision model.** The bounded-precision computation model used throughout this paper is intentionally explicit about arithmetic ordering, rounding, and scaling. These assumptions allow us to isolate the role of numerical precision in transformer expressivity. They should be interpreted as a theoretical model for studying precision-limited computation rather than an exact reproduction of every implementation detail of practical quantized inference systems.

Typically, fixed-point precision formats also have a scaling factor, say $\lambda$, associated with them, which brings the final value in the range $[-\lambda(2^{p-1} - 1) : \lambda(2^{p-1} - 1)]$.

We allow the scaling factor to depend on the tensor, coordinate, or other predetermined location in the computation, but never on the input being processed. This is consistent with practical quantization schemes, where scale parameters are shared across many numerical values, for example per tensor, per channel, or per quantization group. Thus, although a scale itself may be stored at higher precision, its metadata cost is amortized over a large collection of low-precision values and is negligible compared with assigning additional precision bits to every scalar. Accordingly, throughout our theoretical model, the bit complexity $p$ refers to the number of bits used for each quantized scalar, while the fixed scaling factors are treated as public metadata of the quantization scheme.

For addition or multiplication of two numbers, we first find the exact value of the two numbers, and then round off the answer to bring it back to the given precision bounds. Approximation occurs by rounding-off to the nearest number within the permissible set of numbers. Ties are broken by truncation toward zero.

**Example 1.** *To compute the sum of* $1 \cdot 1101$ *and* $10 \cdot 111$ *using* $p = 5$*, the answer becomes*

$$1 \cdot 1101 + 10 \cdot 111 = 100 \cdot 0011 = 100 \cdot 01.$$

*Similarly, for multiplying* $1 \cdot 101$ *with* $10 \cdot 11$ *using* $p = 4$*, we get*

$$1 \cdot 101 \times 10 \cdot 11 = 100 \cdot 01111 = 100 \cdot 1.$$

Note that in this model, $|a|, |b|$ do not need to be fixed separately, which gives it more representational power. When we perform iterated addition or multiplication, we use *left associativity*, i.e., given $a + b + c$ (resp. $a \times b \times c$), we compute it by first computing $(a + b)$ (resp. $a \times b$), find the approximation to the $p$ most significant bits, and then add (resp. multiply) that value to $c$, followed by another approximation to $p$ most significant bits. For summing $n$ elements, we iteratively perform this as

$$a_1 + a_2 + \ldots + a_n = ((((a_1 + a_2) + a_3) + \ldots) + a_n),$$

and similarly multiplication as

$$a_1 \times a_2 \times \ldots \times a_n = ((((a_1 \times a_2) \times a_3) \times \ldots) \times a_n).$$

We also define the floating-point precision format as follows. We make the zero and underflow conventions explicit because the constructions in appendix A.1 use exact cancellation at the bottom of the exponent range.

**Definition 7** (Floating-point precision)**.** *Fix integers* $t, e \geq 2$ *and put*

$$R := 2^{e-1} - 1, \qquad \delta := 2^{-R-(t-1)}.$$

*A floating-point word has* $t$ *mantissa bits (including one mantissa-sign bit) and* $e$ *exponent bits (including one exponent-sign bit), for a total precision of*

$$p = t + e.$$

*For a mantissa-sign bit* $s_a \in \{0, 1\}$*, a mantissa string* $a \in \{0, 1\}^{t-1}$*, an exponent-sign bit* $s_b \in \{0, 1\}$*, and an exponent string* $b \in \{0, 1\}^{e-1}$*, the ordinary normalized value is*

$$(-1)^{s_a}(1.a)2^{(-1)^{s_b}\,\mathrm{bin}(b)}.$$

*Thus the ordinary exponent range is* $[-R, R]$*. When the numerical exponent is zero we use the* $+0$ *exponent encoding for normalized numbers. The* $-0$ *exponent encoding is numerically redundant, so its* $2^t$ *codewords are instead used for zero and subnormal values. In particular, we represent*

$$0 \qquad \textit{and} \qquad \pm k\delta, \quad k \in [2^{t-1} - 1].$$

*These use* $1 + 2(2^{t-1} - 1) = 2^t - 1$ *codewords, so one redundant* $-0$ *codeword remains unused. Consequently, this convention introduces zero and the stated subnormal values without increasing the number of precision bits and without removing any numerical value from the normalized format.*

*Let* $\mathcal{F}_{t,e}$ *denote the resulting set of representable numerical values. For any real number* $x$*,* $\mathrm{Round}_{t,e}(x)$ *denotes a nearest element of* $\mathcal{F}_{t,e}$*; ties are broken toward zero. Every primitive arithmetic operation is first evaluated exactly over the reals and then replaced by* $\mathrm{Round}_{t,e}$ *of its exact value. Iterated addition and multiplication are evaluated left-associatively, with rounding after every primitive operation.*

Here, $1.a$ is called the *mantissa*. A useful consequence of the preceding definition is that every representable value is an integer multiple of $\delta$. Moreover, at exponent $E \in [-R, R]$, the spacing between adjacent normalized $t$-bit values is $2^{E-(t-1)}$; when this quantity is below the smallest normalized magnitude $2^{-R}$, the subnormal grid continues with spacing $\delta$. In particular, for every $E \in [-R, R]$ and $j \in [t-1]$, the quantity $2^{E-j}$ is representable (as a normalized or subnormal value).

Note that floating-point precision can simultaneously represent much larger and much smaller non-zero values than fixed-point precision because it stores a variable exponent. In contrast, a fixed-point format has uniform spacing over its representable range.

### A.4 Self-attention computation with bounded precision

In the bounded precision, we consider self-attention to be computed using left-associativity for both addition and multiplication. Denoting $Q := XW^Q$ as the query matrix, $K := XW^K$ as the key matrix, and the value matrix as $V = XW^V$, the output of self-attention can be seen as the value

$$D^{-1}AV,$$

where $A = e^{QK^T} \in \mathbb{R}^{n \times n}$, $D$ is a diagonal matrix in $\mathbb{R}^{n \times n}$ such that $D_{i,i} = A_i \mathbf{1}_n$. The computation is performed as:

1. First compute $Q := XW^Q$, $K = XW^K$ in bounded precision regime. For $i \in [n], j \in [d_q]$, this is performed as:
$$Q_{i,j} = \sum_{k \in [d]} X_{i,k} W^Q_{k,j},$$
where each of $X_{i,k}W^Q_{k,j}$ is first computed in bounded precision regime, and then the summation is performed using left associativity. A similar thing is done to compute $K$.
2. Similarly, we compute $QK^T \in \mathbb{R}^{n \times n}$.
3. Subsequently, we compute $A := e^{QK^T} \in \mathbb{R}^{n \times n}$.
4. After this, we compute the normalizing term $D$ from $A$.
5. We then matrix multiply as:
$$D^{-1}AV,$$
i.e., first compute $A$, store it and then compute $V$, followed by computing $AV$ which gives the numerator of the softmax, followed by computing $D$ from $A$ to give the final output matrix. Here, $AV$ is the numerator term and $D$ is the denominator term.

**Linear self-attention.** For the linear-attention lower and upper bounds, we use the computation order corresponding directly to definition 3. First compute $Q = XW^Q$, $K = XW^K$, and $V = XW^V$ in bounded precision, using left-associative dot products. Then, for each output position $i$ and input position $j$, compute the scalar score

$$s_{i,j} = \langle Q_i, K_j \rangle$$

in bounded precision, followed by the coordinatewise product $s_{i,j}V_j$. Finally sum the resulting $d_v$-dimensional vectors in token order, left-associatively, rounding after each addition. Thus, when $d_v = 1$,

$$SA_i = \mathrm{Round}_{t,e}(\cdots \mathrm{Round}_{t,e}(\mathrm{Round}_{t,e}(s_{i,1}V_1) + s_{i,2}V_2) + \cdots + s_{i,n}V_n)\,.$$

This convention is important for the communication reduction in lemma 8: if Alice owns a contiguous prefix of the input tokens, then the rounded scalar accumulator after her prefix is a single bounded-precision word, and Bob can continue the same left-associated computation on his suffix.

## B Proofs of Section 2

### B.1 Proofs of tradeoff with simple equality

We repeat the input representation again for the reader's convenience:

1. $\mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_i = y_i$ for $i \in [m]$.
2. $\mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_{i+m} = z_i$ for $i \in [m]$.
3. $\mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_i = 0$ for all $i \in [2m+1 : n]$.

With this notation, we prove the lower bound using communication complexity. First, we prove a lower bound on $\mathsf{EQ}_m$ using communication complexity, and use that to find a lower bound on the required size of a transformer that solves $\mathsf{EQ}_m$ using the above input representation.

**Lemma 6** (Communication lower bound)**.** *If Alice has* $\mathbf{y}$ *and Bob has* $\mathbf{z}$*, with* $\mathbf{y}, \mathbf{z} \in \{0,1\}^m$ *satisfying the promise* $\mathbf{y} \leq \mathbf{z}$*, in the one-way communication model where only Alice can send messages to Bob, Alice needs to send at least* $m$ *bits to Bob in order for Bob to compute the function* $g(\mathbf{y}, \mathbf{z})$.

*Proof.* Let us assume that the communication complexity of this problem (equal to the number of bits sent by Alice to Bob), is at most $m-1$. Since the number of possible $\mathbf{y}$'s is $2^m$, by pigeon hole principle, there exists a single message sent by Alice to Bob which corresponds to at least $\frac{2^m}{2^{m-1}} = 2$ distinct values of $\mathbf{y}$.

Let these two values of $\mathbf{y}$ be $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$. Assuming without loss of generality that $\mathbf{y}^{(1)} < \mathbf{y}^{(2)}$, if Bob has the string $\mathbf{y}^{(2)}$, then the correct answer can be either 1 or 0 for the same transcript sent by Alice. This is a contradiction, which implies the communication complexity is $\geq m$. □

Using this communication lower bound, we can prove Lemma 1, which is restated as follows.

**Lemma 1** (Fixed-point lower bound)**.** *Given the input representation* $\mathcal{T}^0_m$*, any one-layer transformer with* $d_v = 1$ *and* $H = 1$*, cannot compute* $\mathsf{EQ}_m$ *with* $\leq \lceil m/2 \rceil - 1$ *bits of precision.*

*Proof.* Consider a transformer with precision $p$ and value dimension $d_v = 1$ having one head, that correctly computes $\mathsf{EQ}_m$ in $\mathcal{T}^0_m$ representation. We use this transformer to create a communication protocol that uses $2p$ bits of communication for computing the equality function, $\mathsf{EQ}_m$. Subsequently, we will prove that any communication protocol solving $\mathsf{EQ}_m$ requires at least $m$ bits of communication, which would then imply $p \geq \lceil m/2 \rceil$ for a transformer to solve $\mathsf{EQ}_m$.

**Transformer to communication.** Recalling the problem definition, we have $n+1$ tokens. We devise the communication problem such that Alice has the contiguous set of tokens $\{\mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_{-1}, \mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_0, \mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_1, \ldots, \mathcal{T}^0_m(\mathbf{y}, \mathbf{z})_m\}$ (note that these tokens only depend on $\mathbf{y}_1, \ldots, \mathbf{y}_m$), and Bob has the rest (which depend on $\mathbf{z}$). For ease of notation, we define the contiguous set $S$ as the set of tokens with Alice.

Since there is a transformer computing $\mathsf{EQ}_m$ in the $\mathcal{T}^0_m$ representation, Alice and Bob can find the weight matrices of this transformer separately. If there are multiple such transformers, they just consider the lexicographically least setting of weights (a specific transformer which they agree upon). The communication proceeds as follows:

1. Alice computes the values of
$$r_{n+1,j} = e^{X_{n+1} W^Q (W^K)^T X_j},$$
for all $j \in S$ using left-associative multiplication.
2. Bob computes the values of
$$r_{n+1,j} = e^{X_{n+1} W^Q (W^K)^T X_j},$$
for all $j \in [n+1] \backslash S$ using left-associative multiplication.
3. Next, Alice sends Bob the values of
$$L_1 = \sum_{j \in S} r_{n+1,j},$$
$$L_2 = \sum_{j \in S} r_{n+1,j} X_j W^V,$$
using $p$ bits of precision each maintaining left-associative addition.

4. Bob computes the value of

$$L = \frac{L_2 + \sum_{j \in [n+1] \setminus S} r_{n+1,j} V_j}{L_1 + \sum_{j \in [n+1] \setminus S} r_{n+1,j}},$$

approximated upto $p$ bits of precision.

5. Then, Bob computes the value of $L$ applied to the final MLP layer, and outputs 1 if the output of the first entry of the MLP is 1, 0 otherwise.

The communication complexity of this protocol is $2p$.

**Wrap up.** Therefore, we need $2p \geq m$ as there exists a communication protocol of cost $2p$, which implies $p \geq \lceil m/2 \rceil$. □

We state the upper bound theorem again.

**Lemma 2** (Fixed-point upper bound)**.** *There exists a one-layer transformer with $d_v = 1$ and $H = 1$, that uses $\lceil m/2 \rceil + 1$ bits of fixed-point precision to compute $\mathsf{EQ}_m$ in the input representation $\mathcal{T}_m^0$.*

*Proof.* We give the construction of a transformer that uses precision $p = \lceil m/2 \rceil + 1$ to compute $\mathsf{EQ}_m$ with input representation $\mathcal{T}_m^0$. Let

$$q := \left\lceil \frac{m}{2} \right\rceil, \qquad B := 2^{q-1}, \qquad M := 2^q - 1 = 2B - 1.$$

Since $m$ is odd, we have $m = 2q - 1$ and $m - q = q - 1$. The hypothesis on $n$ provides a padding token at position $2m + 1$ and a distinct readout token at position $n + 1$. Empty index ranges are omitted, and empty bit strings have integer value zero. Throughout, we use the promise $\mathbf{y} \leq \mathbf{z}$.

**Overflow convention.** We use the same reserved-overflow convention as in the tight fixed-point construction. Of the two sign-magnitude codewords for zero, one represents the ordinary number 0 and the other represents $\mathsf{Inf}$. Thus, at scale $\lambda > 0$, the finite values are

$$-M\lambda, -(M-1)\lambda, \ldots, 0, \ldots, (M-1)\lambda, M\lambda.$$

Together with $\mathsf{Inf}$, these occupy exactly $2^p$ codewords.

For a finite value $x$, we first define rounding $x$ as $\mathrm{Round}_\lambda(x)$ for this operation. In particular, let $k$ be the nearest integer to $x/\lambda$, breaking ties toward zero. Return $k\lambda$ when $-M \leq k \leq M$, return $\mathsf{Inf}$ when $k > M$, and return $-M\lambda$ when $k < -M$.

In particular, at unit scale,

$$\mathrm{Round}_1\left(k + \frac{1}{2}\right) = k \qquad \text{for every integer } 0 \leq k \leq M.$$

This includes the endpoint $k = M$: tie-breaking is applied before overflow is tested.

The symbol $\mathsf{Inf}$ is preserved by every rounding operation. It is absorbing under addition of a finite number and multiplication by a positive finite number. Multiplication by zero gives zero. We also use

$$\frac{\mathsf{Inf}}{a} = \mathsf{Inf} \quad (a > 0 \text{ finite}), \qquad \mathrm{ReLU}(\mathsf{Inf}) = \mathsf{Inf}.$$

No division by zero or by $\mathsf{Inf}$ occurs in this construction. All scales are public, fixed in advance, and independent of the input.

**Embedding.** We define $\mathbb{E} : [n+1] \times \{0,1\} \to \mathbb{R}^d$, with $d = 3$, by Table 1. The readout position is treated as an additional zero token. Entries marked NA are not used by the input representation and may be defined to equal the corresponding zero-token embedding.

Table 1: Embedding function.

| Step | $i$ | $\mathcal{T}_m^0(\mathbf{y},\mathbf{z})_i = 0$ | $\mathcal{T}_m^0(\mathbf{y},\mathbf{z})_i = 1$ |
|---|---|---|---|
| 1 | $i \in [q]$ | $\begin{bmatrix}1 & -1 & 0\end{bmatrix}$ | $\begin{bmatrix}1 & -1 & -2^{q-i+1}\end{bmatrix}$ |
| 2 | $i = q + j,\ j \in [q-1]$ | $\begin{bmatrix}1 & q-1-j & 0\end{bmatrix}$ | $\begin{bmatrix}1 & -2 & 0\end{bmatrix}$ |
| 3 | $i = m + k,\ k \in [q]$ | $\begin{bmatrix}1 & -1 & 0\end{bmatrix}$ | $\begin{bmatrix}1 & -1 & 2^{q-k+1}\end{bmatrix}$ |
| 4 | $i = m + q + j,\ j \in [q-1]$ | $\begin{bmatrix}1 & -2 & 0\end{bmatrix}$ | $\begin{bmatrix}1 & q-1-j & 0\end{bmatrix}$ |
| 5 | $i = 2m + 1$ | $\begin{bmatrix}1 & -1 & 2M\end{bmatrix}$ | NA |
| 6 | $i \in [2m+2 : n+1]$ | $\begin{bmatrix}1 & -1 & 0\end{bmatrix}$ | NA |

**Construction of the weights.** Set $d_q = d_v = 1$ and choose

$$W^Q = (\ln 2)\begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad W^K = \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad W^V = \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

Write $X_i = [1, \kappa_i, v_i]$ for the embedding at position $i$. The scales below will ensure that the computed queries, keys, and values satisfy exactly

$$Q_i = \ln 2, \qquad K_i = \kappa_i, \qquad V_i = v_i.$$

**Computation order and scales.** We use the following fixed scales. Products and partial sums used to compute $Q$, $K$, and $V$ use the scale of their respective outputs.

| Quantity | Scale |
|---|---|
| $X_{:,1}, X_{:,2}, K, W^K, W^V$ | 1 |
| $W^Q, Q$, and the scores $\langle Q_j, K_i\rangle$ | $\ln 2$ |
| $X_{:,3}, V$ | 2 |
| Stored attention matrix $A$ | $1/2$ |
| Products $A_{n+1,i}V_i$ and numerator accumulator | 1 |
| Denominator addition outputs | 1 |
| Attention output, MLP biases, activations, and logits | $\lambda_{\text{out}} := 1/B$ |

The MLP weights use unit scale. by the finite index 1 at the public scale $\ln 2$.

First form and store $A$ using bounded-precision queries and keys:

$$A_{j,i} = \text{Round}_{1/2}\left(e^{\langle Q_j, K_i\rangle}\right).$$

Compute $V = XW^V$ separately, with scale 2. At the readout position, write

$$a_i := A_{n+1,i}, \qquad C_i := \text{Round}_1(a_i V_i).$$

The numerator $N$ is the left-associated sum of the $C_i$'s, with unit-scale rounding after every addition. The denominator $D$ is the left-associated sum of the stored $a_i$'s, again with unit-scale rounding after every addition. Specifically, its first operation is

$$D_2 := \text{Round}_1(a_1 + a_2), \qquad D_i := \text{Round}_1(D_{i-1} + a_i) \quad (3 \le i \le n+1),$$

and $D := D_{n+1}$. There is no preliminary unit-scale rounding of the individual $a_i$'s and no separately rounded addition $0 + a_1$. Finally, compute the quotient directly:

$$\text{SA}_{n+1} = \text{Round}_{\lambda_{\text{out}}}(N/D).$$

We verify that the displayed embeddings and weights are representable. When $q \geq 2$, all key coordinates belong to $[-2, q-2] \subseteq [-M, M]$; when $q = 1$, the tail ranges are empty and every key coordinate equals $-1$. Every third embedding coordinate is an even integer in $[-2M, 2M]$, so it is represented exactly at scale 2. Thus the computations of $Q$, $K$, and $V$ are exact. The scores are also exact at scale $\ln 2$, and hence

$$e^{\langle Q_{n+1}, K_i \rangle} = 2^{\kappa_i}.$$

In Steps 1, 3, 5, and 6, this gives the stored attention weight $a_i = 1/2$. In Steps 2 and 4, an active tail token gives $2^{q-1-j}$, whereas an inactive tail token gives

$$\mathrm{Round}_{1/2}(2^{-2}) = \mathrm{Round}_{1/2}(1/4) = 0.$$

For $q \geq 2$, the largest active tail weight is $2^{q-2} \leq M/2$, so every stored attention weight is finite and exactly as specified. For $q = 1$, all stored weights are $1/2$.

Consequently, the rounded products are exactly

$$C_i = \begin{cases} -2^{q-i} y_i, & i \in [q], \\ 2^{q-k} z_k, & i = m + k,\ k \in [q], \\ M, & i = 2m + 1, \\ 0, & \text{otherwise.} \end{cases}$$

Every product is an integer in $[-M, M]$. Thus no individual attention–value multiplication overflows or loses precision, even though $V$ is computed before attention aggregation.

**Numerator.** Let

$$Y := \mathrm{bin}(\mathbf{y}_{1:q}), \qquad Z := \mathrm{bin}(\mathbf{z}_{1:q})$$

denote the integer values of the first $q$ bits. Step 1 contributes

$$-\sum_{i=1}^{q} 2^{q-i} y_i = -Y,$$

Step 3 contributes

$$\sum_{k=1}^{q} 2^{q-k} z_k = Z,$$

and Steps 2 and 4 contribute zero. Step 5 then contributes $M$.

No overflow occurs before Step 5. During the $\mathbf{y}$-prefix, every partial sum lies in $[-M, 0]$. During the $\mathbf{z}$-prefix, every partial sum has the form $-Y + Z_{\mathrm{partial}}$, where $0 \leq Z_{\mathrm{partial}} \leq M$, and hence belongs to $[-M, M]$. All these additions are exact at unit scale.

The promise $\mathbf{y} \leq \mathbf{z}$ implies $Y \leq Z$. If $Y = Z$, Step 5 makes the accumulator exactly $M$. If $Y < Z$, Step 5 produces the exact value

$$M - Y + Z \geq M + 1,$$

which becomes $\mathsf{Inf}$. The remaining products are zero, so the final numerator satisfies

$$N = \begin{cases} M, & Y = Z, \\ \mathsf{Inf}, & Y < Z. \end{cases}$$

**Denominator.** The first two stored attention weights are $a_1 = a_2 = 1/2$. For $q \geq 2$, both positions belong to the $\mathbf{y}$-prefix; for $q = 1$, they carry $y_1$ and $z_1$, respectively. Thus the first denominator operation gives

$$D_2 = \mathrm{Round}_1\left(\frac{1}{2} + \frac{1}{2}\right) = 1.$$

Thereafter the accumulator is an integer. Each subsequent weight $1/2$ is discarded individually, since

$$\mathrm{Round}_1\left(k + \frac{1}{2}\right) = k \qquad (0 \leq k \leq M).$$

The only remaining nonzero contributions are the integer tail weights. Write

$$Y_{\text{tail}} := \text{bin}(\mathbf{y}_{q+1:m}), \qquad Z_{\text{tail}} := \text{bin}(\mathbf{z}_{q+1:m}).$$

Both values lie in $[0, B-1]$. Step 2 contributes

$$\sum_{j=1}^{q-1}(1-y_{q+j})2^{q-1-j} = B-1-Y_{\text{tail}},$$

while Step 4 contributes

$$\sum_{j=1}^{q-1} z_{q+j}2^{q-1-j} = Z_{\text{tail}}.$$

Including the initial contribution 1, the final denominator is

$$D = 1 + (B-1-Y_{\text{tail}}) + Z_{\text{tail}} = B - Y_{\text{tail}} + Z_{\text{tail}}.$$

In particular,

$$1 \le D \le 2B-1 = M.$$

Every retained contribution is nonnegative and every later half-unit term is discarded, including at the endpoint $M$. Therefore no intermediate denominator operation overflows, and $D$ is always positive and finite.

If $\mathbf{y} = \mathbf{z}$, then $D = B$. If $Y = Z$ but $\mathbf{y} \neq \mathbf{z}$, the promise implies $Y_{\text{tail}} < Z_{\text{tail}}$, and hence

$$D \ge B+1.$$

**Wrap up.** Let $\lambda := \lambda_{\text{out}} = 1/B$ and $S := \text{SA}_{n+1}$. If $\mathbf{y} = \mathbf{z}$, then $N = M$ and $D = B$, so

$$S = \frac{M}{B} = M\lambda.$$

This is exactly representable and is the largest positive finite value at the output scale.

If $Y < Z$, then $N = \mathsf{Inf}$ and $D$ is positive and finite, so $S = \mathsf{Inf}$.

It remains to consider $Y = Z$ but $\mathbf{y} \neq \mathbf{z}$. This case requires $q \ge 2$, and therefore $B \ge 2$. Before rounding the quotient, we have

$$\frac{N}{D} \le \frac{M}{B+1} \le \frac{M-1}{B} = (M-1)\lambda.$$

Indeed, using $M = 2B-1$,

$$\frac{M-1}{B} - \frac{M}{B+1} = \frac{B-2}{B(B+1)} \ge 0.$$

The number $(M-1)\lambda$ is representable. Since rounding to the nearest grid point is monotone, the rounded quotient also satisfies

$$S \le (M-1)\lambda.$$

Consequently, after all bounded-precision operations,

$$\begin{aligned} \mathbf{y} = \mathbf{z} \quad &\Longrightarrow \quad S = M\lambda, \\ \mathbf{y} \neq \mathbf{z} \quad &\Longrightarrow \quad S \in [0, (M-1)\lambda] \cup \{\mathsf{Inf}\}. \end{aligned}$$

Thus the separation persists after the final division is rounded.

**MLP layer.** Use two hidden ReLU nodes

$$h_1(x) = \text{ReLU}(x - (M-1)\lambda), \qquad h_2(x) = \text{ReLU}(x - M\lambda).$$

Both biases are representable at scale $\lambda$, the weights are 1, and all hidden-node arithmetic uses scale $\lambda$. The MLP passes $h_1, h_2$ to two output coordinates using weights in $\{0, 1\}$ and sets its third output coordinate to zero. The language-model head uses logits

$$\ell_0(x) = h_2(x), \qquad \ell_1(x) = h_1(x),$$

with ties broken in favor of the symbol 0.

On a YES instance, $x = M\lambda$, and all readout arithmetic is exact:

$$(\ell_0, \ell_1) = (0, \lambda).$$

Hence the output is 1. On a finite NO instance, $x \le (M-1)\lambda$, so both hidden values and both logits are zero, and the output is 0. Finally, for $x = \mathsf{Inf}$, adding either finite bias preserves $\mathsf{Inf}$, and hence

$$(\ell_0, \ell_1) = (\mathsf{Inf}, \mathsf{Inf}).$$

The tie rule again gives output 0.

Therefore the transformer outputs 1 exactly when $\mathbf{y} = \mathbf{z}$ and outputs 0 on every promised NO instance. It has one layer, one head, and $d_v = 1$, and uses $p = q + 1 = \lceil m/2 \rceil + 1$ bits of fixed-point precision while computing $V = XW^V$ before $AV$. □

### B.2 Tight tradeoff for fixed-point precision

In this section, we achieve a tight tradeoff with precision. For the reader's convenience, we repeat the input representation. The map $\mathcal{T}^1_m : \{0,1\}^{2m} \to \{0,1\}^n$ is defined as follows:

1. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_{-1} = 0$.
2. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_0 = y_1$.
3. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_i = y_i$ for $i \in [m]$.
4. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_i = z_1$ for $i \in \{m+1, m+2\}$.
5. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_{m+3} = 0$.
6. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_i = z_{i-m-2}$ for $i \in [m+4 : 2m+2]$.
7. $\mathcal{T}^1_m(\mathbf{y}, \mathbf{z})_i = 0$ at every remaining input position.

Thus, up to the final readout token, the input tokens are

$$0, y_1, y_1, y_2, \ldots, y_m, z_1, z_1, 0, z_2, z_3, \ldots, z_m, 0, 0, \ldots, 0.$$

We assume, as before, that $m$ is odd and that the binary integers satisfy $\mathbf{y} \le \mathbf{z}$. Put

$$q := \left\lceil \frac{m}{2} \right\rceil, \qquad r := q - 1, \qquad M := 2^{q-1} - 1.$$

Since the theorem is stated for $p \ge 2$ and we set $m = 2p - 1$, we have $q = \lceil m/2 \rceil = p \ge 2$.

For $p \ge 2$, we fix $m := 2p - 1$, and prove the lower and upper bounds for theorem 4.

**Lemma 3** (Lower bound)**.** *Given the input representation $\mathcal{T}^1_m$, any one-layer transformer with $d_v = 1$ and $H = 1$ that computes $\mathsf{EQ}_m$ requires precision $\ge \lceil m/2 \rceil$ bits.*

*Proof.* Suppose a one-layer, one-head transformer with $d_v = 1$ and precision $b$ computes $\mathsf{EQ}_m$ in the representation $\mathcal{T}^1_m$. Alice owns the contiguous prefix of positions $S = \{-1, 0, \ldots, m\}$, and Bob owns all remaining positions. The model parameters, fixed readout embedding, scales, and rounding schedule are public.

Under the $AV$ computation order, the value $V_j = (XW^V)_j$ and the readout attention weight $a_j = A_{i_{\text{out}},j}$ depend only on token $j$ and public information. Alice therefore computes these quantities for her tokens and executes the prescribed left-associated numerator and denominator accumulations through position $m$. She sends the resulting two $b$-bit words to Bob. Bob resumes both accumulations on his suffix in the original token order, using the prescribed rounding after every operation. He then performs normalization and applies the final MLP and language-model head. In particular, he continues the existing accumulators rather than adding separately rounded suffix sums.

This simulates the transformer exactly using $2b$ communication bits. The reserved exceptional word, when present, is already included in the $b$-bit encoding. As in the proof of Lemma 1, promised equality on $m$ bits requires at least $m$ bits of one-way communication. Hence

$$2b \ge m, \qquad b \ge \left\lceil \frac{m}{2} \right\rceil = q.$$

□

Before proving the upper bound, we define the notation of overflow. This is necessary for proving a bound as tight as a one-bit tradeoff, since information from all possible bit sequences need to be accounted for.

**Fixed-point code with a reserved overflow word.** The fixed-point definition in appendix A.4 is a sign–magnitude representation. Consequently, the two words

$$0\,0^{q-1} \qquad \text{and} \qquad 1\,0^{q-1}$$

would ordinarily both encode zero. We use the first word for the ordinary number $0$ and reserve the second word

$$\omega := 1\,0^{q-1} \tag{1}$$

as the positive-overflow symbol $\mathsf{Inf}$. All other words retain their usual finite values. Thus, at scale one, the finite values are exactly

$$-M, -M+1, \ldots, -1, 0, 1, \ldots, M,$$

and none of the input values, embedding entries, or weights used below is encoded by $\omega$. In particular, reserving (3) removes neither a positive finite level nor a negative finite level. Equivalently, if the format is presented using two's-complement bit patterns, one may reserve the word $1\,0^{q-1}$ (the usual most-negative word) and bijectively relabel the remaining $2^q - 1$ words by the finite levels $-M, \ldots, M$; the proof below is unchanged.

Because only one exceptional word is available, it represents *positive* overflow. After an exact primitive operation, we first apply round-to-nearest at the scale of the output tensor. If the nearest finite grid index would be larger than $M$ (equivalently, the exact value is at least one half-unit above the largest positive finite value), the result is $\mathsf{Inf}$. At the negative endpoint we use saturating rounding: a value below the smallest negative finite value is clipped to that value. Hence a value in the interval between $M$ and the positive-overflow threshold may still round to $M$, exactly as in the original fixed-point convention.

The symbol $\mathsf{Inf}$ is absorbing under subsequent additions and under multiplication by a positive finite number. Multiplication by an exact zero annihilates it. For normalization we use

$$\frac{a}{\mathsf{Inf}} = 0, \qquad \frac{\mathsf{Inf}}{a} = \mathsf{Inf} \quad (a > 0 \text{ finite}), \qquad \frac{\mathsf{Inf}}{\mathsf{Inf}} = 0. \tag{2}$$

The last convention is necessary when only one exceptional codeword is available: overflow in both the numerator and denominator is treated as a rejecting value.

We state the upper bound theorem again.

**Fixed-point code with a reserved overflow word.** The fixed-point definition in appendix A.4 uses a sign–magnitude representation. The two words $0\,0^{q-1}$ and $1\,0^{q-1}$ would ordinarily both encode zero. We use the first for the ordinary number $0$ and reserve the second word

$$\omega := 1\,0^{q-1} \tag{3}$$

as the positive-overflow symbol $\mathsf{Inf}$. At scale $\lambda > 0$, the finite values are

$$\{k\lambda : k \in \mathbb{Z},\ -M \le k \le M\}.$$

Together with $\mathsf{Inf}$, these occupy exactly $2M + 2 = 2^q$ codewords. Thus reserving (3) removes neither a positive nor a negative finite level.

For a finite real number $x$, first round $x/\lambda$ to the nearest integer $k$, breaking ties toward zero, and define

$$\mathrm{Round}_\lambda(x) := \begin{cases} \mathsf{Inf}, & k > M, \\ -M\lambda, & k < -M, \\ k\lambda, & -M \le k \le M. \end{cases}$$

Thus the grid index is rounded *before* overflow is tested. In particular,

$$\mathrm{Round}_\lambda\big((M + \tfrac{1}{2})\lambda\big) = M\lambda.$$

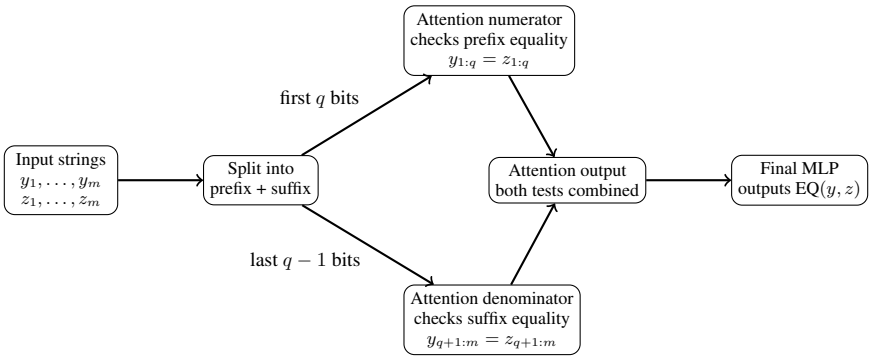


Figure 1: Information flow of the fixed-point upper bound. The numerator stores equality information for the prefix bits, while the denominator stores equality information for the remaining suffix bits. The final MLP separates the equality and inequality cases.

The symbol $\mathsf{Inf}$ is preserved by every rounding operation. It is absorbing under addition and under multiplication by a positive finite number, while multiplication by zero gives zero. For normalization, we use

$$\frac{a}{\mathsf{Inf}} = 0 \quad (a \text{ finite}), \qquad \frac{\mathsf{Inf}}{a} = \mathsf{Inf} \quad (a > 0 \text{ finite}), \qquad \frac{\mathsf{Inf}}{\mathsf{Inf}} = 0. \tag{4}$$

We also set $\mathrm{ReLU}(\mathsf{Inf}) = \mathsf{Inf}$. The construction below never divides by zero. All scales are public, fixed in advance, and independent of the input.

We state the upper bound again.

**Lemma 4** (Upper bound)**.** *There exists a one-layer transformer with $d_v = 1$ and $H = 1$, that uses $\lceil m/2 \rceil$ bits of fixed-point precision to compute $\mathsf{EQ}_m$ in the input representation $\mathcal{T}^1_m$.*

*Proof.* Put

$$q := \lceil m/2 \rceil \geq 3, \qquad r := q - 1, \qquad B := 2^{q-2}, \qquad M := 2^{q-1} - 1 = 2B - 1.$$

Since $m$ is odd, $m = 2q - 1$. Write

$$Y := \mathsf{bin}(\mathbf{y}_{1:q}), \qquad Z := \mathsf{bin}(\mathbf{z}_{1:q}),$$

and

$$Y_{\mathrm{tail}} := \mathsf{bin}(\mathbf{y}_{q+1:m}), \qquad Z_{\mathrm{tail}} := \mathsf{bin}(\mathbf{z}_{q+1:m}).$$

Thus

$$Y, Z \in [0, 2M + 1], \qquad Y_{\mathrm{tail}}, Z_{\mathrm{tail}} \in [0, M].$$

The promise $\mathbf{y} \leq \mathbf{z}$ implies $Y \leq Z$. We use the reserved-overflow arithmetic defined above.

Along with these notation, the proof proceeds as follows. The construction splits the equality test into two independent comparisons. The numerator checks equality of the first $q$ bits, while the denominator checks equality of the remaining $q - 1$ bits.

We now describe the embedding.

**Embedding.** Use embedding dimension $d = 3$. Let $i_{\mathrm{out}}$ denote the final readout position, which contains a fixed zero token and comes after position $2m + 3$. The embedding at each relevant position is given in table 2. Entries marked NA never occur and may be defined to equal the zero-token embedding in the same row.

Table 2: Embedding function for the tight fixed-point construction. Here $j \in [r]$ and $k \in [2:q]$.

| Position $i$ | $\mathcal{T}_m^1(\mathbf{y},\mathbf{z})_i = 0$ | $\mathcal{T}_m^1(\mathbf{y},\mathbf{z})_i = 1$ |
|---|---|---|
| $i = -1$ | $\begin{bmatrix}0 & -2 & -M\end{bmatrix}$ | NA |
| $i \in \{0,1\}$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & B\end{bmatrix}$ |
| $i \in [2:q]$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & 2^{q-i}\end{bmatrix}$ |
| $i = q + j$ | $\begin{bmatrix}0 & r-j & 0\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ |
| $i = m + 1$ | $\begin{bmatrix}0 & -2 & B\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ |
| $i = m + 2$ | $\begin{bmatrix}0 & -2 & B-1\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & -1\end{bmatrix}$ |
| $i = m + 3$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | NA |
| $i = m + k + 2$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | $\begin{bmatrix}0 & -2 & -2^{q-k}\end{bmatrix}$ |
| $i = m + q + j + 2$ | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | $\begin{bmatrix}0 & r-j & 0\end{bmatrix}$ |
| $i = 2m + 3$ | $\begin{bmatrix}0 & -2 & 1\end{bmatrix}$ | NA |
| $i = i_{\text{out}}$ | $\begin{bmatrix}1 & 1 & 0\end{bmatrix}$ | NA |
| remaining padding | $\begin{bmatrix}0 & -2 & 0\end{bmatrix}$ | NA |

**Weights and scales.** Set $d_q = d_v = 1$ and choose

$$W^Q = (\ln 2)\begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad W^K = \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad W^V = 4\begin{bmatrix}0\\0\\1\end{bmatrix}.$$

Write $X_i = [\chi_i, \kappa_i, c_i]$ for the embedding at position $i$. Use the following fixed scales:

| Quantity | Scale |
|---|---|
| $X$, $W^K$, $K$ | 1 |
| $W^Q$, $Q$, attention scores | $\ln 2$ |
| $W^V$, $V$ | 4 |
| Attention–value products and numerator accumulator | 1 |
| Denominator addition outputs before the readout | 1 |
| Final denominator addition output | 2 |
| Attention output, MLP activations, and logits | $\lambda_{\text{out}} := 1/(2B)$ |

Products and partial sums within each projection use the scale of that projection's output. Attention-storage scales are specified below. The MLP weights use unit scale, and all its biases are zero.

Every embedding coordinate lies in $[-M, M]$. In particular, $-2 \le \kappa_i \le q - 2 \le M$ and $|c_i| \le M$. Consequently, all three bounded-precision projections are exact:

$$Q_i = \chi_i \ln 2, \qquad K_i = \kappa_i, \qquad V_i = 4c_i.$$

Here $V = XW^V$ is computed before attention aggregation. Since $V$ uses scale 4, each $4c_i$ is represented by the integer index $c_i \in [-M, M]$. Thus computing $V$ first causes neither overflow nor a loss of precision. At the readout, $\chi_{i_{\text{out}}} = 1$, so the score $\langle Q_{\text{out}}, K_i\rangle$ is exactly $\kappa_i \ln 2$ at scale $\ln 2$.

**Stored attention weights.** Define the numerator-designated positions by

$$\mathcal{I}_{\text{num}} := \{-1, 0, \ldots, q\} \cup \{m+1, \ldots, m+q+2\} \cup \{2m+3\}.$$

In the readout row, use attention-storage scale $1/4$ on $\mathcal{I}_{\text{num}}$ and unit scale elsewhere. Thus

$$a_i := A_{i_{\text{out}},i} = \begin{cases}\text{Round}_{1/4}(2^{\kappa_i}), & i \in \mathcal{I}_{\text{num}},\\ \text{Round}_1(2^{\kappa_i}), & i \notin \mathcal{I}_{\text{num}}.\end{cases}$$

The other attention rows may use unit attention-storage scale. For the readout row, the resulting weights are

$$\begin{aligned} a_i &= \frac{1}{4} && (i \in \mathcal{I}_{\mathrm{num}}), \\ a_{q+j} &= (1 - y_{q+j})2^{r-j} && (j \in [r]), \\ a_{m+q+j+2} &= z_{q+j}2^{r-j} && (j \in [r]), \\ a_{i_{\mathrm{out}}} &= 2. \end{aligned}$$

All remaining padding weights are zero. Indeed, inactive tail and padding positions have $2^{\kappa_i} = 1/4$, which rounds to zero at unit scale. Active tail weights are integers at most $B \le M$, and the readout weight 2 is representable since $M \ge 3$.

**Numerator.** First compute the attention–value products

$$C_i := \mathrm{Round}_1(a_i V_i).$$

At every numerator-designated position,

$$C_i = \mathrm{Round}_1\left(\frac{1}{4} \cdot 4c_i\right) = c_i.$$

At every other position, $c_i = V_i = 0$, so again $C_i = c_i = 0$. Thus every individual product is exact and finite. The numerator $N$ is therefore the left-associated, unit-scale accumulation of the third coordinates in table 2, with rounding after every addition. This realizes the correct computational order $AV$ directly, without reassociating any rounded matrix product.

The $\mathbf{y}$-prefix starts with $-M$ and then adds only nonnegative terms. Its exact total is

$$-M + 2By_1 + \sum_{i=2}^{q} 2^{q-i} y_i = Y - M.$$

If $Y \le 2M$, every partial sum lies in $[-M, M]$, so these additions are exact. If $Y = 2M + 1$, the accumulator becomes $\mathsf{Inf}$. The promise then forces $Z = 2M + 1$ as well, and all subsequent additions preserve $\mathsf{Inf}$.

Now suppose $Y \le 2M$. The two tokens carrying $z_1$ contribute

$$B(1 - z_1) \qquad \text{and} \qquad B(1 - z_1) - 1.$$

Their combined contribution is $M - 2Bz_1$.

If $z_1 = 0$, the promise gives $Y \le Z \le M$. These two contributions move the accumulator from $Y - M$ to $Y$. The remaining $\mathbf{z}$-prefix tokens subtract $Z$ one bit at a time. Every intermediate value lies in $[-M, M]$, and the final constant token adds 1, giving $N = Y - Z + 1$.

If $z_1 = 1$, the two contributions are 0 and $-1$. All later nonzero contributions before the final $+1$ token are nonpositive, so no positive overflow occurs. If no negative clipping occurs, the accumulator immediately before that token is exactly $Y - Z$. If negative clipping occurs, the accumulator stays at $-M$ until the final $+1$, after which it is $-M + 1 \le 0$. When $Y = Z \le 2M$, no clipping occurs: the accumulator decreases from $Y - M \ge 1$ to zero before the final $+1$.

Consequently,

$$N = \begin{cases} 1, & Y = Z \le 2M, \\ \mathsf{Inf}, & Y = Z = 2M + 1, \\ \text{a finite value in } [-M, 0], & Y < Z. \end{cases} \tag{5}$$

The subsequent padding and readout positions have zero value and do not change $N$.

**Denominator.** Accumulate the stored attention weights in token order, with unit scale for every addition output before the readout. The first operation gives

$$\mathrm{Round}_1(a_{-1} + a_0) = \mathrm{Round}_1\left(\frac{1}{4} + \frac{1}{4}\right) = 0.$$

Thereafter, each numerator-designated weight $1/4$ is discarded when added to a finite nonnegative integer accumulator. If the accumulator has already become $\mathsf{Inf}$, it stays $\mathsf{Inf}$. The only effective contributions before the readout are therefore the tail weights, whose exact sum is

$$\sum_{j=1}^{r}(1-y_{q+j})2^{r-j}+\sum_{j=1}^{r}z_{q+j}2^{r-j}=M-Y_{\mathrm{tail}}+Z_{\mathrm{tail}}.$$

All these contributions are nonnegative integers. Hence the accumulator immediately before the readout satisfies

$$D_{\mathrm{pre}}=\begin{cases}M-Y_{\mathrm{tail}}+Z_{\mathrm{tail}}, & Y_{\mathrm{tail}}\geq Z_{\mathrm{tail}},\\ \mathsf{Inf}, & Y_{\mathrm{tail}}<Z_{\mathrm{tail}}.\end{cases}$$

In particular, every finite $D_{\mathrm{pre}}$ is an integer in $[0,M]$.

At the readout position, add $a_{i_{\mathrm{out}}}=2$ and round this final addition at scale 2:

$$D:=\mathrm{Round}_2(D_{\mathrm{pre}}+2).$$

For a finite $D_{\mathrm{pre}}$, ties toward zero give

$$D=2\left\lfloor\frac{D_{\mathrm{pre}}+2}{2}\right\rfloor\in[2,M+1]\subseteq[0,2M].$$

Thus this final addition does not overflow and its result is strictly positive. In particular,

$$D_{\mathrm{pre}}=M\quad\Longrightarrow\quad D=\mathrm{Round}_2(2B+1)=2B.$$

If $D_{\mathrm{pre}}=\mathsf{Inf}$, the final addition preserves $\mathsf{Inf}$. Therefore the denominator is never zero.

Conditioned on $Y=Z$, the promise implies $Y_{\mathrm{tail}}\leq Z_{\mathrm{tail}}$, and hence

$$D=\begin{cases}2B, & Y_{\mathrm{tail}}=Z_{\mathrm{tail}},\\ \mathsf{Inf}, & Y_{\mathrm{tail}}<Z_{\mathrm{tail}}.\end{cases}\tag{6}$$

The readout token changes only the denominator because its value $V_{i_{\mathrm{out}}}$ is zero. The scale change at this fixed position does not remove an earlier overflow: a stored $\mathsf{Inf}$ remains $\mathsf{Inf}$.

**Normalization and output layer.** Use output scale

$$\lambda_{\mathrm{out}}:=\frac{1}{2B}$$

and compute

$$S:=\mathrm{Round}_{\lambda_{\mathrm{out}}}(N/D),$$

using (4).

If $\mathbf{y}=\mathbf{z}$ and $Y\leq 2M$, then $N=1$ and $D=2B$, so $S=\lambda_{\mathrm{out}}$ exactly. If $\mathbf{y}=\mathbf{z}$ and $Y=2M+1$, then $N=\mathsf{Inf}$ and $D=2B$, so $S=\mathsf{Inf}$.

If $Y<Z$, then $N$ is finite and nonpositive, while $D$ is positive finite or $\mathsf{Inf}$. Thus $S\leq 0$. Finally, if $Y=Z$ but the tails differ, then $D=\mathsf{Inf}$ and $N\in\{1,\mathsf{Inf}\}$, so $S=0$. Consequently,

$$\mathbf{y}=\mathbf{z}\quad\Longleftrightarrow\quad S\in\{\lambda_{\mathrm{out}},\mathsf{Inf}\},$$

and every NO instance gives a finite nonpositive output.

The same separation holds if normalization first rounds the reciprocal and then multiplies:

$$R:=\mathrm{Round}_{\lambda_{\mathrm{out}}}(1/D),\qquad S:=\mathrm{Round}_{\lambda_{\mathrm{out}}}(NR).$$

Indeed, $D\geq 2$ whenever it is finite, so $R$ is finite and nonnegative. On a YES instance, $D=2B$ and $R=\lambda_{\mathrm{out}}$ exactly. On an unequal-prefix instance, $N\leq 0$, so the rounded product is nonpositive. On an equal-prefix, unequal-tail instance, $D=\mathsf{Inf}$ gives $R=0$ and $NR=0$, including when $N=\mathsf{Inf}$ by the zero-multiplication convention.

Use one hidden ReLU node

$$h(S)=\mathrm{ReLU}(S),\qquad\mathrm{ReLU}(\mathsf{Inf})=\mathsf{Inf}.$$

Pass $h(S)$ to one MLP output coordinate and set the other coordinates to zero. Let the language-model head use logits

$$(\ell_0, \ell_1) = (0, h(S)),$$

with ties broken in favor of symbol 0. All weights are in $\{0, 1\}$ and all biases are zero, so these operations are exact at the specified scales. The output is 1 exactly when $\mathbf{y} = \mathbf{z}$.

Every stored scalar uses one $q$-bit word at a predetermined scale. The construction has one layer, one head, and $d_v = 1$, and computes $V = XW^V$ before $AV$. It therefore computes $\mathsf{EQ}_m$ in representation $\mathcal{T}_m^1$ using $q = \lceil m/2 \rceil$ bits of fixed-point precision. □

# C TRADEOFF WITH FLOATING-POINT PRECISION

In this section, we prove the remaining results. We first show a tight tradeoff for linear self-attention transformers (see Definition 3), a simplified version of the usual self-attention transformer. They have been widely used in theoretical analyses of transformers (Keles et al., 2023; Ustaomeroglu & Qu, 2025; Ahn et al., 2024; Schlag et al., 2021). This simplified structure, even though less expressive, sometimes tolerating a loss in accuracy. Yet, they are quite efficient and have a linear running time, and have been widely used to devise efficient approximation algorithms for self-attention transformers (Choromanski et al., 2021; Katharopoulos et al., 2020; Koohpayegani & Pirsiavash, 2024; Kacham et al., 2024; Ramapuram et al., 2024; Alman & Song, 2023; Wang et al., 2020). Recently, Kimi 3 (Team et al., 2026) released an architecture which a series of large language models where linear-attention is used as Delta Attention, and achieved close to baseline performances.

The following are the main tradeoff results for floating-point precision quantizations.

**Theorem 5** (Tradeoff for linear transformers)**.** *For every integer $p > 6$ and every integer $n > 2p$, choosing $m = p$, there exists an input representation $\mathcal{T}_m^2$ such that:*

1. *No one-layer linear transformer with $n$ tokens having value dimension $d_v = 1$ and one head can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^2$ using $p - 1$ bits of precision,*
2. *There exists a one-layer linear transformer with $n$ tokens having value dimension $d_v = 1$ and one head that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^2$ using $p$ bits of precision in the floating-point precision format.*

We can show a similar result of expressivity tradeoff with precision for soft-max transformers as well, but with a logarithm gap.

**Theorem 6** (Tradeoff for softmax transformers)**.** *For every integer $p > 0$ and every integer $n > 4p$, choosing $m = 2p - 1$, there exists an input representation $\mathcal{T}_m^3$ such that:*

1. *No one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^3$ using $p - 1$ bits of precision,*
2. *There exists a one-layer transformer with $n$ tokens having value dimension $d_v = 1$ and one head that can compute $\mathsf{EQ}_m$ with the input representation $\mathcal{T}_m^3$ using $p + 2\lceil \log_2 p \rceil + 2$ bits of precision in the floating-point precision format.*

## C.1 TRADEOFF WITH FLOATING-POINT PRECISION: LINEAR SELF-ATTENTION

In this section, we prove an exact precision tradeoff for linear self-attention. The function depends on strings $\mathbf{y}, \mathbf{z} \in \{0, 1\}^m$ satisfying

- $\mathbf{y} \leq \mathbf{z}$ as binary integers,
- $\mathbf{y}_{2:e+1} > 0$,
- $y_{m-1} y_m \neq 10$.

We fix integers $t \geq 4$ and $e > 1$ with $t + e = m$, and assume $4t - 4 \leq n$ so that the token layout below fits in the input sequence. The restriction $t \geq 4$ is only used so that the layout is nondegenerate.

We define the input representation

$$\mathcal{T}_m^2 : \{0,1\}^{2m} \longrightarrow \big(\{0,1\} \times \{0,1\}^e \times \{0,1\}\big)^n$$

as follows:

- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_0 = (y_1, (y_2, \ldots, y_{e+1}), y_{e+3})$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (y_1, (y_2, \ldots, y_{e+1}), 0)$ for $i \in \{1, 3, \ldots, 2t-7\}$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (y_1, (y_2, \ldots, y_{e+1}), y_{e+3+i/2})$ for $i \in \{2, 4, \ldots, 2t-8\}$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_{2t-6} = (y_1, (y_2, \ldots, y_{e+1}), \bar{y}_{e+t})$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_{2t-5} = (y_1, (y_2, \ldots, y_{e+1}), y_{e+2})$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (0, 0^e, 0)$ for $i \in [2t-4, 2t-1]$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_{2t} = (z_1, (z_2, \ldots, z_{e+1}), z_{e+t})$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (z_1, (z_2, \ldots, z_{e+1}), z_{e+(i-2t+3)/2})$ for $i \in \{2t+1, 2t+3, \ldots, 4t-5\}$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (z_1, (z_2, \ldots, z_{e+1}), 0)$ for $i \in \{2t+2, 2t+4, \ldots, 4t-6\}$.
- $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i = (0, 0^e, 0)$ for $i \in [4t-4 : n]$.

We write $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i^1$, $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i^2$, and $\mathcal{T}_m^2(\mathbf{y},\mathbf{z})_i^3$ for the three components of the token. The second component has length $e$.

The first component will determine the sign of the encoded quantity, the second component will determine its binary exponent, and the third component will be used to construct the mantissa. We fix $m := p$ and use the following lemmas to prove theorem 5.

**Lemma 7** (Communication bound)**.** *If Alice has* $\mathbf{y}$ *and Bob has* $\mathbf{z}$*, then every deterministic one-way protocol that computes promised equality on the above domain requires at least* $m$ *bits of communication from Alice to Bob.*

*Proof.* Let

$$F := \{(x, x) : x \in \{0,1\}^m,\ x_{2:e+1} > 0,\ x_{m-1}x_m \neq 10\}.$$

Every pair in $F$ is a YES instance. For two distinct strings $x \neq x'$, assume without loss of generality that $x < x'$. Then the crossed pair $(x, x')$ satisfies the promise $x \leq x'$ and is a NO instance. Hence $F$ is a fooling set.

Because $t \geq 4$, the exponent block $x_{2:e+1}$ and the last two bits are disjoint. Therefore

$$|F| = 2(2^e - 1) \cdot 3 \cdot 2^{t-3} = 3 \cdot 2^{m-2}(1 - 2^{-e}).$$

For $e > 1$,

$$3(1 - 2^{-e}) > 2,$$

so $|F| > 2^{m-1}$. By lemma 5, the one-way communication complexity is therefore

$$\lceil \log_2 |F| \rceil = m.$$

□

**Lemma 8** (Transformer lower bound)**.** *Given the input representation* $\mathcal{T}_m^2$*, any one-layer linear transformer with* $d_v = 1$ *and* $H = 1$ *that computes* $\mathsf{EQ}_m$ *requires at least* $m$ *bits of precision.*

*Proof.* Suppose a one-layer linear transformer of precision $b$ computes the promised equality function. Alice owns the contiguous prefix of positions

$$S := \{0, 1, \ldots, 2t-1\},$$

which contains all tokens depending on $\mathbf{y}$, while Bob owns the remaining tokens, including all tokens depending on $\mathbf{z}$ and the fixed readout token. The model parameters and the fixed readout embedding are public.

By the bounded-precision computation convention for linear self-attention in appendix A.4, Alice can run the left-associated scalar accumulator through all positions in $S$ and send the resulting $b$-bit floating-point word to Bob. Bob then resumes exactly the same left-associated computation on his suffix, applies the final MLP and language-model head, and obtains the transformer's answer. This gives a one-way communication protocol of cost $b$ bits. By lemma 7, $b \geq m$. □

**Lemma 9** (Upper bound)**.** *There exists a one-layer linear transformer with $d_v = 1$ and $H = 1$ that uses floating-point precision $(t, e)$, where $t + e = m$, and computes $\mathsf{EQ}_m$ in the input representation $\mathcal{T}^2_m$.*

*Proof.* Put

$$R := 2^{e-1} - 1, \qquad q := 2^{e-1}, \qquad u := 2^{-(t-1)}, \qquad \delta := 2^{-R-(t-1)}.$$

The floating-point convention is the one in definition 7; in particular, zero and all multiples $\pm k\delta$, $k \in [2^{t-1} - 1]$, are represented exactly at the bottom of the exponent range.

Define

$$E_y := \mathbf{y}_{2:e+1} - q.$$

Since $\mathbf{y}_{2:e+1} \in [1, 2^e - 1]$,

$$-R \le E_y \le R.$$

For $\mathbf{z}$, define

$$\widehat{\mathbf{z}}_{2:e+1} := \max\{1, \mathbf{z}_{2:e+1}\}, \qquad E_z := \widehat{\mathbf{z}}_{2:e+1} - q,$$

so again $E_z \in [-R, R]$. If $\mathbf{z}_{2:e+1} = 0$, then the assumptions $\mathbf{y}_{2:e+1} > 0$ and $\mathbf{y} \le \mathbf{z}$ force $y_1 = 0$ and $z_1 = 1$; this is already a NO instance and will be handled by the different-sign case below.

For a bit string $w$ with the same block structure, define

$$A(w) := 1 + \sum_{j=1}^{t-2} w_{e+1+j} 2^{-j} + \bar{w}_{e+t} u.$$

Thus

$$A(w) = 1 \cdot w_{e+2} \dots w_{m-1} \bar{w}_m$$

as a binary significand. The quantity encoded by the $\mathbf{y}$-tokens will be

$$F(\mathbf{y}) := (-1)^{y_1} A(\mathbf{y}) 2^{E_y}.$$

When $\mathbf{z}_{2:e+1} > 0$, the $\mathbf{z}$-tokens will contribute exactly $-F(\mathbf{z})$.

**Embedding.** We use embedding dimension $d = 3$. The embedding at each position is given in table 3. The only difference from a naive cancellation construction is the order of each $\mathbf{z}$-pair: the bit-dependent token adds $1 - 2^{-j}$ and the following dummy token subtracts 1. The net contribution is still $-2^{-j}$ when the bit is one, but this ordering keeps all temporary magnitudes below the upper floating-point endpoint.

Table 3: Embedding for the linear-attention floating-point construction.

| Step | $i$ | $\mathcal{T}^2_m(\mathbf{y}, \mathbf{z})^3_i = 0$ | $\mathcal{T}^2_m(\mathbf{y}, \mathbf{z})^3_i = 1$ |
|---|---|---|---|
| 1 | $i = 0$ | $\begin{bmatrix} 1 & 2^{E_y} & (-1)^{y_1}(1 + 2^{-1}) \end{bmatrix}$ | $\begin{bmatrix} 1 & 2^{E_y} & (-1)^{y_1}(1 + 2^{-1} + 2^{-2}) \end{bmatrix}$ |
| 2 | $i \in \{1, 3, \dots, 2t - 7\}$ | $\begin{bmatrix} 1 & 2^{E_y} & -(-1)^{y_1} \end{bmatrix}$ | NA |
| 3 | $i \in \{2, 4, \dots, 2t - 6\}$ | $\begin{bmatrix} 1 & 2^{E_y} & (-1)^{y_1} \end{bmatrix}$ | $\begin{bmatrix} 1 & 2^{E_y} & (-1)^{y_1}(1 + 2^{-(i+4)/2}) \end{bmatrix}$ |
| 4 | $i = 2t - 5$ | $\begin{bmatrix} 1 & 2^{E_y} & -(-1)^{y_1} 2^{-1} \end{bmatrix}$ | $\begin{bmatrix} 1 & 2^{E_y} & 0 \end{bmatrix}$ |
| 5 | $i \in [2t - 4, 2t - 1]$ | $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ | NA |
| 6 | $i = 2t$ | $\begin{bmatrix} 1 & 2^{E_z} & -(-1)^{z_1}(1 + u) \end{bmatrix}$ | $\begin{bmatrix} 1 & 2^{E_z} & -(-1)^{z_1} \end{bmatrix}$ |
| 7 | $i \in \{2t + 1, 2t + 3, \dots, 4t - 5\}$ | $\begin{bmatrix} 1 & 2^{E_z} & (-1)^{z_1} \end{bmatrix}$ | $\begin{bmatrix} 1 & 2^{E_z} & (-1)^{z_1}(1 - 2^{-(i-2t+1)/2}) \end{bmatrix}$ |
| 8 | $i \in \{2t + 2, 2t + 4, \dots, 4t - 4\}$ | $\begin{bmatrix} 1 & 2^{E_z} & -(-1)^{z_1} \end{bmatrix}$ | NA |

Every remaining padding token, as well as the final readout token, is embedded as

$$[1 \quad 0 \quad 0].$$

All entries in the table are representable. The powers $2^{E_y}$ and $2^{E_z}$ are normalized floating-point values. Each third coordinate is a signed dyadic number with at most $t - 1$ fractional bits, and is therefore representable after normalization. The products appearing below are also representable or round to the subnormal grid specified in definition 7.

**Weights.** Set

$$W^Q = \begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad W^K = \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad W^V = \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

Thus, for a token $[a_1\ a_2\ a_3]$,

$$Q = a_1, \qquad K = a_2, \qquad V = a_3.$$

At the readout token, $Q_{\mathrm{out}} = 1$, and therefore the linear self-attention output is the left-associated bounded-precision sum

$$L = \sum_i K_i V_i.$$

No exponential is used.

**The y-block.** Let $s_y := (-1)^{y_1}$. Step 1 creates

$$s_y(1 + 2^{-1} + y_{e+3}2^{-2})2^{E_y}.$$

Each Step 2–Step 3 pair removes a temporary leading 1 and then restores it while inserting one further mantissa bit. All these operations are exact: every intermediate quantity is a multiple of $u2^{E_y}$, has magnitude at most $(2-u)2^{E_y}$, and, when $E_y = -R$, the subnormal grid represents the corresponding multiples of $\delta$ exactly. Immediately before Step 4 the accumulator equals

$$s_y\left(1 + 2^{-1} + \sum_{j=2}^{t-2} y_{e+1+j}2^{-j} + \bar{y}_m u\right)2^{E_y}.$$

Step 4 removes the temporary $2^{-1}$ precisely when $y_{e+2} = 0$. Hence, after Step 4,

$$L = F(\mathbf{y}) = s_y A(\mathbf{y})2^{E_y}.$$

Step 5 contributes zero.

**The z-block.** Let $s_z := (-1)^{z_1}$. Step 6 contributes

$$-s_z(1 + \bar{z}_m u)2^{E_z}.$$

For each $j \in [t-2]$, the corresponding Step 7–Step 8 pair contributes

$$s_z(1 - z_{e+1+j}2^{-j})2^{E_z} \qquad \text{and} \qquad -s_z 2^{E_z}.$$

Its exact net contribution is therefore

$$-s_z z_{e+1+j}2^{-j}2^{E_z}.$$

Hence, when $\mathbf{z}_{2:e+1} > 0$, the exact total contribution of Steps 6–8 is $-F(\mathbf{z})$.

It remains to show that the left-associated bounded-precision computation preserves the required separation. We use that $\mathrm{Round}_{t,e}$ is monotone and sign-symmetric.

We also use the following bound:

$$A(\mathbf{y}) \le 2 - 2u. \tag{7}$$

Indeed, if $y_m = 1$, then $\bar{y}_m = 0$ and the largest possible significand is $2-2u$. If $y_m = 0$, the promise forces $y_{m-1} = 0$, giving the stronger bound $A(\mathbf{y}) \le 2 - 3u$.

We now distinguish three cases.

**Case 1: $y_1 \neq z_1$.** The promise $\mathbf{y} \le \mathbf{z}$ forces $y_1 = 0$ and $z_1 = 1$. Hence $F(\mathbf{y}) > 0$.

After Step 6, the accumulator is

$$F(\mathbf{y}) + (1 + \bar{z}_m u)2^{E_z}.$$

Define

$$\Phi_j(X) := \mathrm{Round}_{t,e}\left(\mathrm{Round}_{t,e}\left(X + s_z(1 - b2^{-j})2^{E_z}\right) - s_z 2^{E_z}\right),$$

where $b = z_{e+1+j}$.

We claim that if $X > 0$, then

$$\Phi_j(X) > 0.$$

Indeed, after Step 7, the exact value differs from $X$ by at most $2^{E_z}$ in magnitude. The following Step 8 restores exactly this quantity. Since both operations are performed on the floating-point grid and the final value is separated from zero by at least the positive contribution already stored in the accumulator, rounding cannot map it to zero. Therefore every complete Step 7–Step 8 pair preserves positivity.

By induction over all $t-2$ pairs, the final accumulator satisfies

$$L > 0.$$

Hence $L \neq 0$. This also covers the exceptional case $\mathbf{z}_{2:e+1} = 0$.

**Case 2: $y_1 = z_1$ and $E_y < E_z$.** Put

$$s := (-1)^{y_1} = (-1)^{z_1}.$$

By sign symmetry, multiply the accumulator by $s$ and analyze the transformed quantity. Let

$$d := E_z - E_y \geq 1.$$

Immediately after Step 6,

$$A(\mathbf{y})2^{-d} - (1 + \bar{z}_m u)$$

is the value divided by $2^{E_z}$. By (7),

$$A(\mathbf{y})2^{-d} - (1 + \bar{z}_m u) \leq (2 - 2u)2^{-1} - 1 = -u.$$

Therefore, by monotonicity,

$$X \leq -u2^{E_z}$$

after Step 6.

For a Step 7–Step 8 pair with bit $b$ at level $j$,

$$X \mapsto \mathrm{Round}_{t,e}\left(\mathrm{Round}_{t,e}(X + (1 - b2^{-j})2^{E_z}) - 2^{E_z}\right).$$

The largest possible value occurs at $X = -u2^{E_z}$. At this point both operations are exact, giving

$$-(u + b2^{-j})2^{E_z} \leq -u2^{E_z}.$$

Thus the invariant is preserved and the final output is nonzero.

**Case 3: $y_1 = z_1$ and $E_y = E_z$.** Write the common exponent as $E$. After multiplying by the common sign, all quantities in the $\mathbf{z}$-block are integer multiples of

$$u2^E = 2^{E-(t-1)}.$$

If $E = -R$, these are multiples of the subnormal grid spacing $\delta$.

The reordered Step 7–Step 8 pairs keep every temporary significand strictly below 2 in magnitude. Consequently, every addition in the $\mathbf{z}$-block is exact. Therefore the final output is exactly

$$L = (-1)^{y_1}(A(\mathbf{y}) - A(\mathbf{z}))2^E.$$

The map

$$(w_{e+2}, \ldots, w_{m-1}, w_m) \mapsto (w_{e+2}, \ldots, w_{m-1}, \bar{w}_m)$$

is a bijection. Therefore, under equality of the first $e+1$ bits,

$$L = 0 \iff \mathbf{y} = \mathbf{z}.$$

Combining the three cases gives

$$L = 0 \iff \mathbf{y} = \mathbf{z}.$$

Finally, use two ReLU hidden nodes

$$h_1 = \mathrm{ReLU}(L), \qquad h_2 = \mathrm{ReLU}(-L).$$

Then $h_1 + h_2 = 0$ if and only if $L = 0$. The language-model head uses logits

$$(h_1 + h_2, 0)$$

for output symbols 0 and 1, respectively, and breaks a tie in favor of symbol 1. It therefore outputs 1 exactly when $\mathbf{y} = \mathbf{z}$ and outputs 0 otherwise. Hence the one-layer linear transformer computes $\mathsf{EQ}_m$ using floating-point precision $(t, e)$ with $t + e = m$. □

**Consequence for the tradeoff theorem.** For every parameter choice with $m = p$, $t + e = p$, $t \geq 4$, $e > 1$, and $4t - 4 \leq n$, lemmas 8 and 9 give the desired exact $p - 1$ versus $p$ separation. In particular, when $p \geq 6$ and $n > 2p$, one may choose, for example, $t = 4$ for $6 \leq p \leq 7$ and $t = \lfloor p/2 \rfloor$ for $p \geq 8$, with $e = p - t$; these choices satisfy the stated constraints. If theorem 5 is stated for the finitely many cases $p < 6$ as well, those cases require a separate constant-size construction and are not covered by the token layout above.

### C.2 Tradeoff with floating-point precision: softmax self-attention

We now show that using fixed-point precision $(t, e)$ where $t + e = \lceil m/2 \rceil + 2 \log \lceil m/2 \rceil + 2$ bits (one sign bit each), we can also solve a function, that cannot be solved using any less number of bits. We assume the inputs

- The binary values satisfy $\mathbf{y} \leq \mathbf{z}$.
- $z_1, \ldots z_e \neq 1 \ldots 1$.

The input tokens are again similar to the previous case, where one part of the input token will be used to create the exponent in the floating-point precision format. For a given $(y_1, \ldots, y_m), (z_1, \ldots, z_m)$, where $m$ is odd:

- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_0 = ((y_1, \ldots, y_{e-1}), 0)$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_i = ((y_1, \ldots, y_{e-1}), y_{e+i-1})$ for $i \in [t-1]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{t-1+i} = ((y_{t+e-1}, \ldots, y_{t+2e-2-\log \lceil m/2 \rceil}), y_{t+2e-2-\log \lceil m/2 \rceil + i})$ for $i \in [t-1]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{2t-1} = ((y_{t+e-1}, \ldots, y_{t+2e-2-\log \lceil m/2 \rceil}), 0)$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_i = ((0), 0)$ for $i \in [2t, m-1]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_m = ((z_1, \ldots, z_{e-1}), 0)$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{m+i} = ((z_1, \ldots, z_{e-1}), z_{e+i-1})$ for $i \in [t-1]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{m+t-1+i} = ((z_{t+e-1}, \ldots, z_{t+2e-2-\log \lceil m/2 \rceil}), z_{t+2e-2-\log \lceil m/2 \rceil + i})$ for $i \in [t-1]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{m+i} = ((0), 0)$ for $i \in [2t-1, m]$.
- $\mathcal{T}^3_m(\mathbf{y}, \mathbf{z})_{2m+1} = ((z_{t+e-1}, \ldots, z_{t+2e-2-\log \lceil m/2 \rceil}), 0)$

For ease of notation, we write $\mathbf{y}_{1:e-1}$ as $e^1_y$, $\mathbf{y}_{t+e+1:t+2e-\log \lceil m/2 \rceil}$ as $e^2_y$, $\mathbf{z}_{1:e-1}$ as $e^1_z$ and $\mathbf{z}_{t+e+1:t+2e-\log \lceil m/2 \rceil}$ as $e^2_z$.

Again, we choose $m := 2p - 1$ and prove the following results.

**Lemma 10** (Communication lower bound)**.** *If Alice has* $\mathbf{y}$ *and Bob has* $\mathbf{z}$ *in the one-way communication where only Alice can send messages to Bob, Alice needs to send at least* $m$ *bits to Bob in order for Bob to compute the function* $g(\mathbf{y}, \mathbf{z})$.

*Proof.* The proof is exactly the same as that of lemma 7. □

**Lemma 11** (Transformer lower bound)**.** *Any linear transformer having* $d_v = 1$ *and* $H = 1$*, that computes* $T_m$ *requires precision* $p \geq \lceil m/2 \rceil$ *bits.*

*Proof.* This lower bound is exactly the same as the proof of Lemma 3, but Alice's set of tokens, $S$, is now $\{0, 1, \ldots, m-1\}$. Using this, we can show the transformer requires precision at least $\lceil m/2 \rceil$. □

**Lemma 12** (Floating-point softmax upper bound)**.** *Let* $p \geq 2$*,* $m = 2p - 1$*, and* $n > 4p$ *be integers. There exists an input representation* $\widetilde{T}^3_m$ *such that a one-layer, one-head softmax transformer with embedding dimension* $d = 3$ *and query and value dimensions* $d_q = d_v = 1$ *computes* $\mathrm{EQ}_m(\mathbf{y}, \mathbf{z})$ *on all inputs satisfying* $\mathbf{y} \leq \mathbf{z}$*, using*

$$P = p + 2\lceil \log_2 p \rceil + 2$$

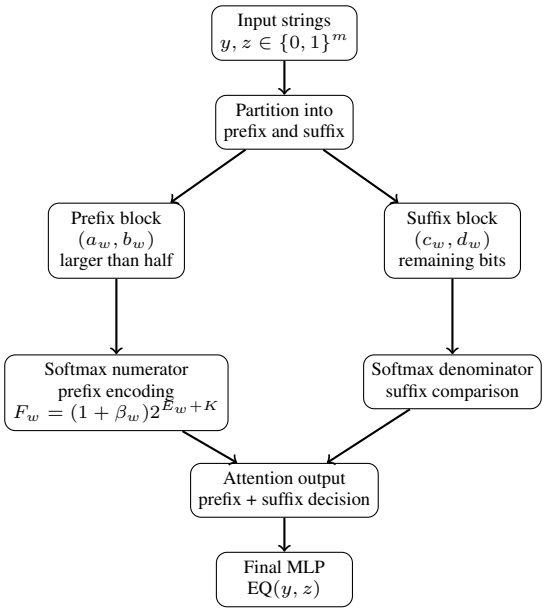


Figure 2: Information flow of the floating-point softmax construction. The numerator encodes the larger prefix block, while the denominator checks equality of the remaining suffix information. The additional precision overhead comes from preserving separation between softmax weights under floating-point rounding.

*bits of floating-point precision. Each input token depends only on* $\mathbf{y}$*, only on* $\mathbf{z}$*, or is constant.*

*We use the floating-point format of the preliminaries and the assumed power-of-two attention convention: an integer key coordinate* $\kappa$ *produces the stored attention weight* $\mathrm{Round}_{t,e}(2^\kappa)$*, where* $t + e = P$*.*

*Proof.* We first give the construction for $p \geq 3$. The case $p = 2$ is handled separately at the end. All logarithms below are to base two.

**Parameters and bit partition.** We split each input string into two conceptual parts: a prefix, which will be checked by the numerator of attention, and a suffix, which will be checked by the denominator. Define

$$\ell := \lceil \log_2 p \rceil, \qquad t := p - 1, \qquad e := 2\ell + 3, \qquad k := t - 1 = p - 2,$$

and put

$$r := \min\{2, p - \ell\}, \qquad h := p - \ell - r.$$

Then $r \in \{1, 2\}$, $h \geq 0$, and $h + 2 \leq t$. Moreover, $t + e = P$.

For each string $\mathbf{w} \in \{\mathbf{y}, \mathbf{z}\}$, partition its bits into four consecutive blocks

$$\mathbf{w} = (\mathbf{a}_w, \mathbf{b}_w, \mathbf{c}_w, \mathbf{d}_w),$$

where

$$|\mathbf{a}_w| = \ell + 1, \qquad |\mathbf{b}_w| = k, \qquad |\mathbf{c}_w| = r, \qquad |\mathbf{d}_w| = h.$$

The pair $(\mathbf{a}_w, \mathbf{b}_w)$ forms the prefix used by the numerator, while $(\mathbf{c}_w, \mathbf{d}_w)$ forms the suffix used by the denominator. These lengths satisfy

$$(\ell + 1) + k + r + h = 2p - 1 = m.$$

Define the integer represented by the first block and the binary fractions represented by the remaining blocks as

$$E_w := \mathrm{bin}(\mathbf{a}_w), \qquad J_w := \mathrm{bin}(\mathbf{c}_w),$$

and

$$\beta_w := \sum_{j=1}^{k} b_{w,j} 2^{-j}, \qquad \gamma_w := \sum_{j=1}^{h} d_{w,j} 2^{-j}, \qquad \eta := 2^{-h}.$$

The prefix $(\mathbf{a}_w, \mathbf{b}_w)$ is therefore represented by the real number $(1 + \beta_w)2^{E_w}$, while the suffix information is stored through the exponent-like quantity $J_w$ and the fractional remainder $\gamma_w$.

In particular,

$$0 \le E_w \le 2^{\ell+1} - 1, \qquad 0 \le J_w \le 3.$$

**Overview.** The construction separates the equality test into two independent checks. The numerator checks whether the prefixes $(\mathbf{a}_y, \mathbf{b}_y)$ and $(\mathbf{a}_z, \mathbf{b}_z)$ are equal, while the denominator checks whether the suffixes $(\mathbf{c}_y, \mathbf{d}_y)$ and $(\mathbf{c}_z, \mathbf{d}_z)$ are equal.

For the prefix check, define the numerical code

$$K := k + 1 = t, \qquad F_w := (1 + \beta_w)2^{E_w + K}.$$

This converts the binary prefix into a floating-point-like value. The important property is that different prefixes map to values separated by a gap of at least two. Indeed, for prefixes with the same exponent, the smallest possible positive difference is

$$2^{E_w + K - k} = 2^{E_w + 1} \ge 2,$$

and the gap between consecutive exponent ranges is

$$2^{E+K+1} - (2 - 2^{-k})2^{E+K} = 2^{E+1} \ge 2.$$

Hence,

$$(\mathbf{a}_y, \mathbf{b}_y) < (\mathbf{a}_z, \mathbf{b}_z) \quad \Longrightarrow \quad F_z - F_y \ge 2. \tag{8}$$

The numerator accumulates the difference between these two prefix codes. After this comparison, it adds a small positive term $\varepsilon 2^{-J_z}$ that encodes the suffix exponent. The prefix gap ensures that this additional term cannot change the decision when the prefixes are different. The denominator then performs the remaining suffix comparison.

**Input representation.** We use $n$ positions indexed by $0, \dots, n-1$, with position $n-1$ reserved for the constant readout token. Every token is a tuple

$$\tau_i = (\xi_i, \rho_i) \in \{0,1\}^{e-1} \times \{0,1\}.$$

Pad the exponent blocks with leading zeros by setting

$$\widehat{\mathbf{a}}_w := 0\mathbf{a}_w, \qquad \widehat{\mathbf{c}}_w := 0^{e-1-r}\mathbf{c}_w.$$

Both padded blocks have length $e-1$. Put $u := t + h + 1$ and define

$$\widetilde{T}^3_m(\mathbf{y}, \mathbf{z})_i := \begin{cases} (\widehat{\mathbf{a}}_y, 0), & i = 0, \\ (\widehat{\mathbf{a}}_y, b_{y,j}), & i = j, \quad j \in [k], \\ (\widehat{\mathbf{c}}_y, 0), & i = t, \\ (\widehat{\mathbf{c}}_y, d_{y,j}), & i = t + j, \quad j \in [h], \\ (\widehat{\mathbf{a}}_z, 0), & i = u, \\ (\widehat{\mathbf{a}}_z, b_{z,j}), & i = u + j, \quad j \in [k], \\ (\widehat{\mathbf{c}}_z, d_{z,j}), & i = u + t - 1 + j, \quad j \in [h], \\ (\widehat{\mathbf{c}}_z, 0), & i = 2u - 1, \\ (0^{e-1}, 0), & 2u \le i \le n - 1. \end{cases}$$

Thus all $\mathbf{y}$-dependent tokens form a contiguous prefix, followed by the $\mathbf{z}$-dependent tokens and constant padding. The construction uses $2u$ nonpadding positions and one readout position, and

$$2u + 1 = 2t + 2h + 3 = 4p - 2\ell - 2r + 1 < 4p < n.$$

Hence the token layout fits within the prescribed sequence length.

**Embedding.** Let

$$R := 2^{e-1} - 1, \qquad S := R - 3, \qquad \varepsilon := 2^{-S}, \qquad B := 2^{e}.$$

We define the position-dependent embedding $E(i, \tau_i) \in \mathbb{R}^3$ by Table 4. Write each embedded token as $X_i = [1, \kappa_i, \nu_i]$.

Table 4: Embedding for the floating-point softmax construction.

| Step | Position $i$ | $\rho_i = 0$ | $\rho_i = 1$ |
|---|---|---|---|
| 1 | $i = 0$ | $\begin{bmatrix} 1 & -S & 2^{E_y+K} \end{bmatrix}$ | NA |
| 2 | $i = j, j \in [k]$ | $\begin{bmatrix} 1 & -S & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & -S & 2^{E_y+K-j} \end{bmatrix}$ |
| 3 | $i = t$ | $\begin{bmatrix} 1 & -J_y & 0 \end{bmatrix}$ | NA |
| 4 | $i = t + j, j \in [h]$ | $\begin{bmatrix} 1 & -J_y - j & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & -B & 0 \end{bmatrix}$ |
| 5 | $i = u$ | $\begin{bmatrix} 1 & -S & -2^{E_z+K} \end{bmatrix}$ | NA |
| 6 | $i = u + j, j \in [k]$ | $\begin{bmatrix} 1 & -S & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & -S & -2^{E_z+K-j} \end{bmatrix}$ |
| 7 | $i = u + t - 1 + j, j \in [h]$ | $\begin{bmatrix} 1 & -B & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & -J_z - j & 0 \end{bmatrix}$ |
| 8 | $i = 2u - 1$ | $\begin{bmatrix} 1 & -S & 2^{-J_z} \end{bmatrix}$ | NA |
| 9 | $2u \le i \le n - 1$ | $\begin{bmatrix} 1 & -B & 0 \end{bmatrix}$ | NA |

**Construction of the weights.** For $d_q = d_v = 1$, choose

$$W^Q = \widetilde{\ln 2} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad W^K = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad W^V = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

where $\widetilde{\ln 2}$ is a public constant stored in the floating-point format. It is chosen as the nearest representable approximation of $\ln 2$ with $t$ mantissa bits.

Write

$$\mathsf{fl} := \mathrm{Round}_{t,e}, \qquad \Delta := \widetilde{\ln 2} - \ln 2.$$

Since $\widetilde{\ln 2}$ is rounded to $t$ mantissa bits,

$$|\Delta| \le 2^{-t}.$$

The largest key magnitude appearing in the construction is bounded by

$$|\kappa_i| \le B_0,$$

where

$$B_0 \le 2^{\ell+1} + p = O(p).$$

We use the mantissa precision defined above,

$$t = p - 1,$$

and the exponent precision

$$e = 2\ell + 3,$$

where $\ell = \lceil \log_2 p \rceil$. Hence the total floating-point precision is

$$P = t + e = p + 2\lceil \log_2 p \rceil + 2.$$

Let

$$\widetilde{\ln 2}$$

denote the nearest representable approximation of $\ln 2$ using $t$ mantissa bits, and write

$$\Delta := \widetilde{\ln 2} - \ln 2.$$

Since $\widetilde{\ln 2}$ is rounded to $t$ mantissa bits,

$$|\Delta| \le 2^{-t}.$$

The largest key magnitude appearing in the construction satisfies

$$|\kappa_i| \le B_0,$$

where

$$B_0 \le 2^{\ell+1} + p = O(p).$$

Therefore,

$$|\kappa_i \Delta| \le B_0 2^{-t} \le O(p) 2^{-(p-1)}.$$

For all sufficiently large $p$, this quantity is small enough that

$$|e^x - 1| \le 2|x|$$

applies. Therefore,

$$\left| e^{\kappa_i \widetilde{\ln 2}} - 2^{\kappa_i} \right| = 2^{\kappa_i} \left| e^{\kappa_i \Delta} - 1 \right| \le 2^{\kappa_i + 1} |\kappa_i \Delta|.$$

Because the error is smaller than one quarter of the spacing between adjacent floating-point values at exponent $\kappa_i$, we obtain

$$\mathrm{Round}_{t,e}\left( e^{\kappa_i \widetilde{\ln 2}} \right) = \mathrm{Round}_{t,e}\left( 2^{\kappa_i} \right).$$

Hence the bounded-precision softmax computation produces exactly the same stored attention weights as the idealized power-of-two construction:

$$\alpha_i = \mathsf{fl}\left( e^{\kappa_i \widetilde{\ln 2}} \right) = \mathsf{fl}(2^{\kappa_i}).$$

We next compute

$$V = XW^V$$

using bounded-precision left-associated dot products. Since $W^V$ selects the third embedding coordinate,

$$V_i = \nu_i$$

exactly.

The scalar numerator of $AV$ is computed as

$$U_{-1} := 0, \qquad U_i := \mathsf{fl}\left( U_{i-1} + \mathsf{fl}(\alpha_i V_i) \right), \qquad N := U_{n-1}.$$

The denominator is computed as

$$D_{-1} := 0, \qquad D_i := \mathsf{fl}(D_{i-1} + \alpha_i), \qquad D := D_{n-1}.$$

All sums below refer to these left-associated rounded computations. The rounding map is monotone, fixes representable values, and is sign-symmetric.

**Representability.** All displayed embedding entries and parameters other than the assumed $\ln 2$ factor are representable. For the largest prefix exponent,

$$E_w + K \le 2^{\ell+1} - 1 + p - 1 \le 3 \cdot 2^\ell - 2 < R = 4 \cdot 2^\ell - 1.$$

The integer $S = 4(2^\ell - 1)$ has $\ell \le t$ significant binary digits. The other nonconstant key magnitudes satisfy $J_w + j \le h + 3 \le t + 1 \le 2^t - 1$, and $B = 2^e$ is a representable power of two.

Let $\delta := 2^{-R-(t-1)}$ be the smallest positive subnormal. Since $B > R + t$,

$$2^{-B} < 2^{-R-t} = \frac{\delta}{2},$$

so every key $-B$ produces attention weight zero. All other attention weights are representable powers of two. In particular,

$$S + J_z \le R,$$

so the normalization contribution

$$\varepsilon 2^{-J_z} = 2^{-S-J_z}$$

is an exactly representable attention–value product. The nonzero prefix products have exponents $E_w + K - j - S \ge 1 - S > -R$ for $j \in \{0, \dots, k\}$, and their largest exponent is also below $R$. Hence every individual attention–value product is exact. Moreover, $V_i = \nu_i$ is representable before this multiplication; there is no potentially overflowing intermediate $2^S \nu_i$.

**Numerator.** Steps 1 and 2 accumulate exactly

$$U = \varepsilon 2^{E_y+K} + \sum_{j=1}^{k} \varepsilon b_{y,j} 2^{E_y+K-j} = \varepsilon F_y.$$

Every intermediate significand has at most $k + 1 = t$ bits and stays within the exponent range. Steps 3 and 4 have third coordinate zero and do not change this accumulator.

**Case 1: $E_y = E_z = E$.** Step 5 subtracts $\varepsilon 2^{E+K}$, and Step 6 subtracts the remaining $\mathbf{z}$-prefix bits. Every intermediate value after Step 5 is a multiple of $2^{E+1-S}$ with magnitude less than $2^{E+K-S}$, and is therefore exactly representable. After Step 6,

$$U = \varepsilon(F_y - F_z).$$

If the prefixes are equal, this is zero. Otherwise, the promise $\mathbf{y} \le \mathbf{z}$ and (8) imply $U \le -2\varepsilon$.

**Case 2: $E_y < E_z$.** Immediately after the exact subtraction at Step 5, before rounding,

$$\begin{aligned} \varepsilon F_y - \varepsilon 2^{E_z+K} &\le \varepsilon(2 - 2^{-k}) 2^{E_z+K-1} - \varepsilon 2^{E_z+K} \\ &= -\varepsilon 2^{E_z} \le -2\varepsilon. \end{aligned}$$

Since $-2\varepsilon$ is representable, monotonicity of rounding preserves this upper bound. All subsequent Step 6 contributions are nonpositive, so the rounded accumulator remains at most $-2\varepsilon$.

These two cases exhaust the possibilities under $\mathbf{y} \le \mathbf{z}$. Step 7 contributes zero to the numerator. Step 8 adds the exact quantity $\varepsilon 2^{-J_z} \le \varepsilon$. Thus, for equal prefixes, the final accumulator is exactly $\varepsilon 2^{-J_z}$; for unequal prefixes, it is at most $-\varepsilon$, even after rounding. Since $N = U_{n-1}$, this gives

$$N = \begin{cases} \varepsilon 2^{-J_z}, & (\mathbf{a}_y, \mathbf{b}_y) = (\mathbf{a}_z, \mathbf{b}_z), \\ \text{a value at most } -\varepsilon, & (\mathbf{a}_y, \mathbf{b}_y) < (\mathbf{a}_z, \mathbf{b}_z). \end{cases} \tag{9}$$

This is the rounded numerator of $AV$, where $V = XW^V$ has been already computed, without any rescaling after the accumulation.

**Denominator.** The first $t$ attention weights are all $\varepsilon$ and sum exactly to $t\varepsilon$. Our parameters give

$$S - (t+3) \geq 3p - 6 > \log_2 t,$$

so

$$t\varepsilon < 2^{-(t+3)}. \tag{10}$$

Step 3 adds $2^{-J_y} \geq 2^{-3}$. Half the upward spacing at $2^{-J_y}$ is $2^{-J_y-t} \geq 2^{-(t+3)}$. Therefore,

$$\mathsf{fl}(t\varepsilon + 2^{-J_y}) = 2^{-J_y}.$$

Thereafter the denominator accumulator is at least $2^{-J_y}$, and the upward spacing cannot decrease. Each subsequent individual $\varepsilon$-term is smaller than half this spacing and is therefore discarded. This includes the $\mathbf{z}$-prefix tokens and the normalization token.

Consequently, the only effective denominator contributions are

$$2^{-J_y}, \qquad (1 - d_{y,j})2^{-J_y-j}, \qquad d_{z,j}2^{-J_z-j}, \qquad j \in [h].$$

All of them are nonnegative, so

$$D \geq 2^{-J_y} > 0. \tag{11}$$

If $J_y = J_z = J$, every intermediate sum of these effective contributions is a multiple of $2^{-J-h}$ with magnitude less than $3 \cdot 2^{-J}$. Such a value has at most $h + 2 \leq t$ significant bits. Hence these additions are exact, and

$$\begin{aligned} D &= \left(1 + \sum_{j=1}^{h}(1 - d_{y,j})2^{-j} + \sum_{j=1}^{h} d_{z,j}2^{-j}\right) 2^{-J} \\ &= \big((2-\eta) - \gamma_y + \gamma_z\big)2^{-J}. \end{aligned} \tag{12}$$

Comparing the attention weights coordinatewise with those for $J_y = J_z = 0$, $d_{y,j} = 0$, and $d_{z,j} = 1$, and using monotonicity, also gives

$$D \leq 3 - 2\eta < 3.$$

Thus the denominator is always finite and strictly positive.

**Wrap up.** Set the decision threshold to

$$\theta := \frac{\varepsilon}{2} = 2^{-S-1} = 2^{-R+2}.$$

This is a normalized, exactly representable value. We first analyze the numerator and denominator and then check the rounded normalization.

**Case 1: $\mathbf{y} = \mathbf{z}$.** The prefixes and suffixes agree. By (9) and (12),

$$N = \varepsilon 2^{-J}, \qquad D = (2-\eta)2^{-J}.$$

Consequently,

$$\frac{N}{D} = \frac{\varepsilon}{2-\eta} > \theta.$$

**Case 2: The prefixes differ.** Equation (9) gives $N \leq -\varepsilon$, whereas (11) gives $D > 0$. Thus the attention output is nonpositive, including after rounding.

**Case 3: The prefixes agree but the suffixes differ.** Here $N = \varepsilon 2^{-J_z}$. If $J_y < J_z$, then

$$D \geq 2^{-J_y} \geq 2^{1-J_z}.$$

If $J_y = J_z = J$, the promise $\mathbf{y} \leq \mathbf{z}$ forces $\gamma_y < \gamma_z$. Since these fractions have $h$ bits,

$$\gamma_z - \gamma_y \geq 2^{-h} = \eta.$$

Equation (12) therefore gives

$$D \geq 2 \cdot 2^{-J} = 2^{1-J_z}.$$

The latter subcase is absent when $h = 0$. Hence every unequal-suffix instance with equal prefixes satisfies

$$D \geq 2^{1-J_z} = \frac{2N}{\varepsilon}, \qquad \frac{N}{D} \leq \theta.$$

**Rounded normalization.** Allow even the reciprocal to be rounded separately, so that the stored attention output is

$$\widehat{x} := \mathsf{fl}\big(N\,\mathsf{fl}(1/D)\big).$$

For unequal prefixes, $N \le 0$ and $\mathsf{fl}(1/D) \ge 0$, so $\widehat{x} \le 0$. For equal prefixes but unequal suffixes, $D \ge 2^{1-J_z}$ implies

$$\frac{1}{D} \le 2^{J_z-1}.$$

The right-hand side is representable. By monotonicity of rounding,

$$\mathsf{fl}(1/D) \le 2^{J_z-1}.$$

Since $N = \varepsilon 2^{-J_z}$ and $\theta$ is representable,

$$\widehat{x} \le \mathsf{fl}\big(\varepsilon 2^{-J_z} 2^{J_z-1}\big) = \theta.$$

For equality, put $x_* := 1/(2-\eta) \in (1/2, 1]$. Then

$$\frac{1}{D} = 2^J x_*, \qquad N = 2^{-S-J}.$$

The values $x_*$ and $2^J x_*$ are within the normal range, away from the exponent endpoints. Hence rounding commutes with this power-of-two shift:

$$\mathsf{fl}(2^J x_*) = 2^J \mathsf{fl}(x_*).$$

Multiplication by $N = 2^{-S-J}$ then gives $\varepsilon\mathsf{fl}(x_*)$. This product is exactly representable: it has the same significand as $\mathsf{fl}(x_*)$ and normalized exponent either $-S-1$ or $-S$, both in $[-R, R]$. Therefore,

$$\widehat{x} = \varepsilon\mathsf{fl}(x_*) = \mathsf{fl}(\varepsilon x_*).$$

Moreover,

$$x_* - \frac{1}{2} = \frac{\eta}{2(2-\eta)} > 2^{-h-2} \ge 2^{-t}.$$

The spacing immediately above $1/2$ is $2^{-t}$. Since $1/2 + 2^{-t}$ is representable, monotonicity yields

$$\widehat{x} \ge \varepsilon\left(\frac{1}{2} + 2^{-t}\right) = \theta + \varepsilon 2^{-t}.$$

Crucially, this separation is not lost to underflow. Indeed,

$$\varepsilon 2^{-t} = 2^{-S-t} = 2^{-R+3-t} = 4\delta.$$

Thus the YES output is separated from the threshold by at least four units of the subnormal grid. We have established

$$\mathbf{y} = \mathbf{z} \implies \widehat{x} \ge \theta + 4\delta, \qquad \mathbf{y} \ne \mathbf{z} \implies \widehat{x} \le \theta \tag{13}$$

on every promised input.

The same separation holds if normalization is implemented by a single rounded division, $\widehat{x} = \mathsf{fl}(N/D)$. On a YES instance, this again equals $\mathsf{fl}(\varepsilon x_*) = \varepsilon\mathsf{fl}(x_*)$. On a NO instance, $N/D \le \theta$, and monotonicity together with the representability of $\theta$ preserves this inequality.

**Final MLP layer.** Use one ReLU hidden node

$$h_{\mathrm{out}}(x) := \mathrm{ReLU}\big(\mathsf{fl}(x-\theta)\big).$$

The bias $-\theta$ is exactly representable. On a YES instance, $\widehat{x} \in (\theta, \varepsilon]$, and $\widehat{x} - \theta$ is an integer multiple of $\varepsilon 2^{-t} = 4\delta$ lying in $(0, \varepsilon/2]$. It has at most $t$ significant binary digits and, when subnormal, lies on the $\delta$ grid. Hence the subtraction is exact and $h_{\mathrm{out}}(\widehat{x}) > 0$. On a NO instance, monotonicity of rounding and (13) give $h_{\mathrm{out}}(\widehat{x}) = 0$.

Pass this hidden value to one output coordinate of the MLP and set the other coordinates to zero. Let the language-model head use logits

$$(\ell_0, \ell_1) = (0, h_{\mathrm{out}}(\widehat{x}))$$

for output symbols 0 and 1, respectively, with ties broken in favor of 0. The required weights are in $\{0, 1\}$, so these readout operations are exact. The output is 1 exactly when $\mathbf{y} = \mathbf{z}$, using precisely $t + e = p + 2\lceil \log_2 p \rceil + 2$ precision bits.

**The case $p = 2$.** Here $m = 3$ and $P = 5$. Choose $(t, e) = (2, 3)$ and, for a three-bit integer $w \in \{0, \dots, 7\}$, define

$$G(w) := \left(1 + \frac{w \bmod 2}{2}\right) 2^{\lfloor w/2 \rfloor - 1}.$$

Its eight values are

$$\frac{1}{2}, \quad \frac{3}{4}, \quad 1, \quad \frac{3}{2}, \quad 2, \quad 3, \quad 4, \quad 6.$$

They are representable and are separated by at least $1/4$. Use two data tokens, each containing its string's first two bits and its last bit as a tuple in $\{0, 1\}^2 \times \{0, 1\}$. Embed them as $[1, 0, G(y)]$ and $[1, 0, -G(z)]$, respectively. Embed every padding and readout token as $[1, -8, 0]$. Choose

$$W^Q = (\ln 2) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad W^K = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad W^V = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Computing $V = XW^V$ first gives $G(y)$, $-G(z)$, and zeros, all exactly representable. The two data attention weights are exactly one. All other attention weights round to zero because $2^{-8} < \delta/2$, where $\delta = 2^{-4}$ in this format. Thus the left-associated computation of $AV$ gives

$$N = \mathsf{fl}(G(y) - G(z)), \qquad D = 2.$$

If $y = z$, the stored attention output is zero. Otherwise, the exact difference has magnitude at least $1/4$. Monotonicity and sign symmetry of rounding preserve magnitude at least $1/4$, and the subsequent division by two preserves magnitude at least $1/8$. Both bounds are representable. The separately rounded reciprocal also gives the same output because $1/D = 1/2$ is exact. Hence the stored output $x$ is zero if and only if $y = z$.

Use two ReLU nodes $h_1 = \mathrm{ReLU}(x)$ and $h_2 = \mathrm{ReLU}(-x)$, followed by logits $(h_1 + h_2, 0)$ for symbols 0 and 1, with ties broken in favor of 1. This computes equality with $P = 5$ bits and completes the proof. □

*Proof of Theorem 6.* We choose $m := 2p - 1$ and define $\Gamma_n^{(t,e)}$ as $T_m$. From Lemma 11, $\Gamma_n^{(p)}$ cannot be computed by a transformer using $\leq \lceil m/2 \rceil - 1 = p - 1$ bits, but can be computed by a transformer using at most

$$\left\lceil \frac{m}{2} \right\rceil + 2 \left\lceil \log_2 \left\lceil \frac{m}{2} \right\rceil \right\rceil + 2 = p + 2\lceil \log_2 p \rceil + 2$$

bits (Lemma 12). □